aws

使用者指南

Amazon EKS



Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon EKS: 使用者指南

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon 的商標和商業外觀不得用於任何非 Amazon 的產品或服務，也不能以任何可能造成客戶混淆、任從何貶低或使 Amazon 名譽受損的方式使用 Amazon 的商標和商業外觀。所有其他非 Amazon 擁有的商標均為其各自擁有者的財產，這些擁有者可能附屬於 Amazon，或與 Amazon 有合作關係，亦或受到 Amazon 贊助。

Table of Contents

..... xxix
..... xxx
什麼是 Amazon EKS？ 1
Amazon EKS：簡化的 Kubernetes 管理 1
使用 Kubernetes 建置和擴展：Amazon EKS 功能 2
Amazon EKS 的功能 3
相關服務 4
Amazon EKS 定價 5
常用案例 5
Architecture 7
控制平面 7
運算 7
EKS 功能 8
Kubernetes 概念 9
為什麼選擇 Kubernetes？ 10
叢集 14
工作負載 17
後續步驟 21
部署選項 22
了解 Amazon EKS 部署選項 22
雲端 Amazon EKS 22
您的資料中心或邊緣環境中的 Amazon EKS 23
適用於氣隙環境的 Amazon EKS Anywhere 24
Amazon EKS 工具 24
設定 25
後續步驟 25
設定 AWS CLI 25
建立存取金鑰 25
設定 AWS CLI 26
取得安全字符 26
若要驗證使用者身分 27
後續步驟 27
設定 kubectl 和 eksctl 28
安裝或更新 kubectl 28

步驟 1：檢查是否已安裝 kubectl 28
步驟 2：安裝或更新 kubectl 29
安裝 eksctl 41
後續步驟 41
快速指南 42
於本教學課程中 42
先決條件 42
設定叢集 42
建立 IngressClass 43
部署 2048 遊戲範例應用程式 44
藉助 Amazon EKS 自動模式讓資料持續保留 47
清除 49
了解 Amazon EKS 50
概觀 50
Amazon EKS 研討會 50
Amazon EKS 實作叢集設定教學 51
Amazon EKS 範例 52
AWS 教學課程 53
開發人員工作坊 53
Terraform 工作坊 53
AWS Amazon EKS 訓練 53
開始使用 55
建立叢集 (EKS 自動模式) 55
建立叢集 (eksctl) 56
先決條件 56
步驟 1：建立 Amazon EKS 叢集和節點 56
步驟 2：檢視 Kubernetes 資源 58
步驟 3：刪除您的叢集和節點 59
後續步驟 60
建立叢集 (主控台和 CLI) 60
先決條件 60
步驟 1：建立您的 Amazon EKS 叢集 61
步驟 2：連線至您的叢集 64
步驟 3：建立節點 65
步驟 4：檢視資源 67
步驟 5：刪除資源 67

後續步驟 68
EKS 自動模式 69
功能 69
自動化元件 70
組態 71
共同責任模型 72
建立叢集 73
eksctl CLI 74
AWS CLI 75
管理主控台 83
啟用現有叢集 85
移轉參考 86
移轉 EBS 磁碟區 87
移轉負載平衡器 87
在叢集上啟用 88
從 Karpenter 移轉 91
從 MNG 移轉 94
從 Fargate 移轉 95
執行工作負載 102
部署 inflate 工作負載 103
部署負載平衡器 105
部署具狀態工作負載 111
部署加速工作負載 116
設定 122
建立節點類別 126
建立節點集區 140
建立靜態節點集區 146
建立傳入類別 152
建立服務 160
多叢集目標群組 166
共用目標群組 167
將 RBAC 授予受管控制器 169
授予對 LBC 的秘密存取權 171
建立 StorageClass 174
停用 EKS 自動模式 181
更新 Kubernetes 版本 182

檢閱內建的節點集區 ... 183
控制部署 ... 186
執行關鍵附加元件 ... 188
使用網路政策 ... 190
標記子網路 ... 200
產生 CIS 報告 ... 202
加密根磁碟區 ... 205
更新 AMI 控制項 ... 208
控制容量保留 ... 212
使用本機區域 ... 217
進階安全性 ... 221
成本最佳化 ... 223
運作方式 ... 230
受管執行個體 ... 230
身分和存取 ... 235
聯網 ... 237
可觀測性 ... 240
EKS Auto Managed 元件日誌 ... 240
疑難排解 ... 243
節點監控代理程式 ... 244
使用 EC2 CLI 從 AWS EC2 受管執行個體取得主控台輸出 ... 244
使用偵錯容器和 kubectl CLI 取得節點日誌 ... 244
在 AWS 主控台中檢視與 EKS Auto Mode 相關聯的資源 ... 246
檢視您 AWS 帳戶中的 IAM 錯誤 ... 246
對 Pod 無法排程到自動模式節點進行疑難排解 ... 247
對未加入叢集的節點進行疑難排解 ... 247
跨 Pod 共用磁碟區 ... 249
在控制平面日誌中檢視 Karpenter 事件 ... 250
對自動模式中包含的控制器進行疑難排解 ... 251
相關資源 ... 173
版本備註 ... 252
2026 年 8 月 17 日 ... 252
2026 年 8 月 12 日 ... 252
2026 年 8 月 5 日 ... 253
2026 年 7 月 27 日 ... 253
2026 年 7 月 21 日 ... 254

2026 年 6 月 30 日 254
2026 年 6 月 9 日 256
2026 年 6 月 4 日 256
2026 年 6 月 3 日 256
2026 年 5 月 5 日 256
2026 年 4 月 10 日 256
2026 年 4 月 2 日 256
2026 年 2 月 2 日 256
2025 年 12 月 19 日 256
2025 年 11 月 19 日 257
2025 年 11 月 19 日 257
2025 年 10 月 23 日 257
2025 年 10 月 1 日 257
2025 年 9 月 30 日 257
2025 年 9 月 29 日 257
2025 年 9 月 10 日 257
2025 年 8 月 24 日 257
2025 年 8 月 15 日 258
2025 年 8 月 6 日 258
2025 年 6 月 30 日 258
2025 年 6 月 20 日 258
2025 年 6 月 13 日 258
2025 年 4 月 30 日 258
2025 年 4 月 18 日 258
2025 年 4 月 11 日 258
2025 年 4 月 4 日 259
2025 年 3 月 31 日 259
2025 年 3 月 21 日 259
2025 年 3 月 14 日 259
設定叢集 260
建立自動叢集 260
先決條件 42
建立叢集 - AWS 主控台 262
建立叢集 - AWS CLI 265
後續步驟 83
建立 叢集 270

先決條件 42
步驟 1：建立叢集 IAM 角色 271
步驟 2：建立叢集 272
步驟 3：更新 kubeconfig 281
步驟 4：叢集設定 281
後續步驟 83
建立佈建的控制平面 283
概觀 80
使用案例 134
控制平面擴展層 285
考量事項 87
佈建控制平面入門 290
進階控制平面組態 298
概觀 80
支援的 Kubernetes 控制平面參數 298
考量事項 87
後續步驟 83
設定控制平面參數 307
叢集洞察 317
叢集洞見類型 317
考量事項 317
使用案例 318
開始使用 319
檢視叢集洞察 320
更新 Kubernetes 版本 328
Amazon EKS 自動模式的考量事項 167
摘要 330
步驟 1：準備升級 330
步驟 2：檢閱升級考量 331
步驟 3：更新叢集控制平面 332
步驟 4：更新叢集元件 334
升級適用於 Amazon EKS 叢集的 Kubernetes 版本 335
復原 Kubernetes 版本 335
復原的內容 335
不會復原的內容 336
先決條件 336

摘要 337
步驟 1：檢閱轉返整備洞見 337
步驟 2：準備工作者節點 340
步驟 3：復原叢集控制平面 341
步驟 4：監控轉返進度 342
考量事項和警告 343
轉返和附加元件 344
相關資源 344
回復 EKS Auto Mode 叢集 345
刪除叢集 352
考量事項 87
先決條件步驟 352
刪除叢集 (eksctl) 353
刪除叢集AWS （主控台） 353
刪除叢集 (AWS CLI) 354
避免 EKS 叢集遭到意外刪除 356
叢集端點存取 357
IPv6 叢集端點格式 357
IPv4 叢集端點格式 358
叢集私有端點 358
修改叢集端點存取 359
存取僅限私有 API 伺服器 361
設定端點存取 362
啟用 Windows 支援 365
考量事項 87
先決條件 42
啟用 Windows 支援 367
部署 Windows Pod 369
在 Windows 節點上支援更多數量的 Pod 369
停用 Windows 支援 370
私有叢集 370
叢集架構要求 371
節點要求 371
Pod 要求 373
控制平面輸出 375
輸出路由模式 376

先決條件 376
建立具有客戶路由輸出的叢集 378
更新現有叢集 379
IPv6 考量因素 379
考量事項 380
IAM 條件索引鍵 380
OIDC 提供者組態 381
驗證連線 382
對輸出進行故障診斷 382
自動擴展 388
EKS 自動模式 389
其他解決方案 389
了解區域轉移 390
了解 Pod 間的東西網路流量 390
EKS 區域轉移要求 395
常見問答集 402
其他資源 405
啟用區域轉移 405
考量事項 405
什麼是 Amazon 應用程式復原控制器？ 406
什麼是區域轉移？ 406
什麼是區域自動轉移？ 406
EKS 在區域轉移期間會做什麼？ 406
向 Amazon Application Recovery Controller (ARC) 註冊 EKS 叢集AWS （主控台） 407
後續步驟 83
管理存取權 408
一般任務 408
背景介紹 200
EKS 自動模式的考量事項 352
Kubernetes API 存取 409
建立 IAM 身分與 Kubernetes 許可的關聯 410
設定叢集驗證模式 410
存取項目 412
aws-auth ConfigMap 463
連結 OIDC 提供商 473
取消連結 OIDC 提供商 479

存取叢集資源 479
所需的許可 480
CloudTrail 可見性 483
修改叢集資源 484
所需的許可 484
CloudTrail 可見性 483
透過 kubectl 存取叢集 485
使用 EKS 主控台連線 (CloudShell) 486
使用 CLI AWS 建立kubeconfig檔案 486
工作負載存取 AWS 487
服務帳戶字符 487
叢集附加元件 489
將 AWS Identity and Access Management 許可授予 Amazon Elastic Kubernetes Service 叢集上的工作負載 489
具有 IRSA 的憑證 492
Pod 身分識別 515
管理運算 547
對比運算選項 547
受管節點群組 552
受管節點群組概念 552
受管節點群組容量類型 554
建立 557
更新 569
更新行為詳細資訊 573
啟動範本 577
刪除 592
自我管理的節點 594
Amazon Linux 595
Bottlerocket 602
Windows 606
Ubuntu Linux 614
更新方法 616
AWS Fargate 627
AWS Fargate 考量事項 628
Fargate 比較表 630
開始使用 632

定義設定檔 637
刪除設定檔 641
Pod 組態詳細資訊 642
OS 修補事件 645
收集指標 647
記錄 649
Amazon EC2 執行個體類型 659
maxPods的判斷方式 661
EKS 自動模式的考量事項 352
預先建置的最佳化 AMI 662
AL2 AMI 棄用 663
Amazon Linux 668
Bottlerocket 677
Ubuntu Linux 691
Windows 691
節點運作狀態和修復 773
節點監控代理程式 774
自動節點修復 775
節點運作狀態偵測 775
自動節點修復 789
檢視節點運作狀態 794
取得節點日誌 796
擷取網路流量 799
混合節點 802
功能 803
限制 804
考量事項 804
其他資源 805
先決條件 805
執行混合節點 864
設定 884
運作方式 961
混合節點 nodeadm 1000
疑難排解 1015
混合節點閘道 1034
應用程式資料儲存 1070

Amazon EBS 1070
考量事項 1071
先決條件 1071
步驟 1：建立 IAM 角色 1071
步驟 2：取得 Amazon EBS CSI 驅動程式 1079
步驟 3：部署範例應用程式 1080
Amazon EC2 執行個體儲存體 1080
考量事項 1080
先決條件 1081
安裝 Amazon EC2 執行個體存放區 CSI 驅動程式 1081
部署範例應用程式 1081
I/O 限流 1082
新啟動的執行個體上的 NVMe 命名空間變更 1083
Amazon EFS 1085
考量事項 1085
先決條件 1085
步驟 1：建立 IAM 角色 1086
步驟 2：取得 Amazon EFS CSI 驅動程式 1092
步驟 3：建立 Amazon EFS 檔案系統 1092
步驟 4：部署範例應用程式 1092
Amazon S3 Files 1093
考量事項 1093
先決條件 1093
步驟 1：建立 IAM 角色 1094
步驟 2：取得 Amazon EFS CSI 驅動程式 1103
步驟 3：建立 Amazon S3 檔案系統 1103
步驟 4：部署範例應用程式 1103
Amazon FSx for Lustre 1103
部署驅動程式 1104
最佳化 (EFA) 1110
最佳化 (非 EFA) 1127
Amazon FSx for NetApp ONTAP 1134
Amazon FSx for OpenZFS 1135
Amazon File Cache 1135
適用於 Amazon S3 的掛載點 1136
考量事項 1136

部署驅動程式 1136
移除驅動程式 1146
CSI 快照器控制器 1147
設定聯網 1149
從管理主控台將現有 VPC 子網路新增至 Amazon EKS 叢集 1149
VPC 和子網要求 1149
VPC 要求和注意事項 1150
子網需求和注意事項 1151
共用子網路需求和注意事項 1157
建立 VPC 1157
先決條件 42
公有和私有子網路 1158
僅公有子網路 1160
一個私有子網路 1161
安全群組要求 1162
預設的叢集安全群組 1163
限制叢集流量 1164
共用安全群組 1165
管理聯網附加元件 1166
內建附加元件 1166
選用 AWS 的聯網附加元件 1167
Amazon VPC CNI 1167
替代 CNI 外掛程式 1281
Multus 1282
AWS Load Balancer控制器 1283
CoreDNS 1300
kube-proxy 1316
工作負載 1321
範例部署 (Linux) 1321
先決條件 42
建立命名空間。 1322
建立 Kubernetes 部署 1322
建立服務 1323
檢閱建立的資源 1324
在 Pod 上執行 Shell 1327
後續步驟 1328

範例部署 (Windows) 1328
先決條件 42
建立命名空間。 1322
建立 Kubernetes 部署 1322
建立服務 1323
檢閱建立的資源 1331
在 Pod 上執行 Shell 1327
後續步驟 1335
Vertical Pod Autoscaler 1335
部署 Vertical Pod Autoscaler 1335
測試您的 Vertical Pod Autoscaler 安裝 1337
Horizontal Pod Autoscaler 1340
執行 Horizontal Pod Autoscaler 測試應用程式 1341
網路負載平衡 1343
先決條件 42
考量事項 87
建立 Network Load Balancer 1346
(選用) 部署範例應用程式 1348
應用程式負載平衡 1352
先決條件 42
透過輸入群組重複使用 ALB 1354
(選用) 部署範例應用程式 1355
限制服務的外部 IP 1358
將映像複製到儲存庫 1360
檢視 Amazon 映像登錄檔 1363
Amazon EKS 附加元件 1366
考量事項 1367
附加元件的自訂命名空間 1368
Amazon EKS 自動模式的考量事項 1368
支援 1369
AWS 附加元件 1371
社群附加元件 1391
Marketplace 附加元件 1397
建立附加元件 1413
更新附加元件 1424
確認相容性 1431

移除附加元件 1432
IAM 角色 1436
您可自訂的欄位 1442
驗證容器映像 1445
EKS 功能 1446
可用的功能 1446
AWS Kubernetes (ACK) 的控制器 1446
Argo CD 1447
kro (Kube Resource Orchestrator) 1447
EKS 功能的優點 1447
定價 242
EKS 功能的運作方式 1448
常用案例 1449
使用 功能 1450
EKS 功能資源 1450
了解功能狀態 1451
建立 功能 1452
列出功能 1452
描述功能 1453
更新 功能的組態 1455
刪除功能 1455
後續步驟 83
功能控制器日誌 1457
支援的日誌類型 1457
設定日誌交付 241
範例：使用 CloudWatch Logs Insights 查詢日誌 1459
檢視您的日誌 242
定價 242
Kubernetes 資源 1460
Argo CD 資源 1460
kro 資源 1461
ACK 資源 1462
資源限制 1463
後續步驟 83
考量事項 1464
功能 IAM 角色和 Kubernetes RBAC 1464

多叢集架構模式 1464
比較 EKS 功能與自我管理解決方案 1466
AWS Kubernetes 的控制器 1468
ACK 的運作方式 1468
ACK 的優點 1469
支援 AWS 的服務 1469
與其他 EKS 受管功能整合 1470
ACK 入門 1470
建立 ACK 功能 1470
ACK 概念 1479
設定 許可 1486
考量事項 1493
疑難排解 1501
與自我管理的比較 1504
Argo CD 1508
Argo CD 的運作方式 1508
Argo CD 的優點 1509
與 AWS Identity Center 整合 1510
與其他 EKS 受管功能整合 1470
Argo CD 入門 1510
建立 Argo CD 功能 1510
Argo CD 概念 1525
設定 許可 1531
使用 Argo CD 1543
私有端點存取 1595
Argo CD 考量事項 1599
疑難排解 1608
與自我管理的比較 1619
kro 1624
kro 的運作方式 1624
kro 的優點 1624
與其他 EKS 受管功能整合 1470
kro 入門 1626
建立 kro 功能 1626
kro 概念 1637
設定 許可 1643

考量事項 1648
疑難排解 1653
與自我管理的比較 1659
故障診斷功能 1660
一般疑難排解方法 1660
使用控制器日誌進行故障診斷 1660
檢查功能運作狀態 1662
常見功能狀態 1663
驗證 Kubernetes 資源狀態 1663
檢閱 IAM 許可和叢集存取 1664
功能特定的疑難排解 1666
所有功能的常見問題 1666
後續步驟 83
叢集管理 1668
成本監控 1668
按 Pod 檢視成本 1669
安裝 Kubecost 1669
存取 Kubecost 儀表板 1671
深入了解 Kubecost 1672
指標伺服器 1682
考量事項 87
使用 Amazon EKS 附加元件部署為社群附加元件 1682
使用資訊清單進行部署 1683
使用 Helm 部署應用程式 1684
標記您的 資源 1685
標籤基本概念 1686
標記您的 資源 1686
標籤限制 1687
標記您的資源以便計費 1687
透過主控台使用標籤 1688
透過 CLI、API 或 eksctl 使用標籤 1689
Service Quotas 1690
在 中檢視 EKS 服務配額 AWS 管理主控台 1691
使用 CLI 檢視 EKS AWS 服務配額 1691
Amazon EKS 服務配額 1692
AWS Fargate 服務配額 1692

安全 1694
最佳實務 1695
分析漏洞 1695
Center for Internet Security (CIS) 的 Amazon EKS 基準 1695
Amazon EKS 平台版本 1696
作業系統漏洞清單 1696
使用 Amazon Inspector 進行節點偵測 1696
使用 Amazon GuardDuty 進行叢集與節點偵測 1696
驗證合規性 1696
EKS 的考量 1697
基礎架構安全 1697
恢復能力 1703
預防跨服務混淆代理人 1703
Kubernetes 的考量事項 1705
憑證簽署 1706
CA 輪換 1708
預設 角色和使用者 1739
啟用秘密加密 1743
AWS Secrets Manager 1747
針對所有 Kubernetes API 資料的預設封套加密 1748
RBAC 強化 1754
EKS 自動模式的考量事項 1756
API 安全性和驗證 1756
網路安全 1756
EC2 受管執行個體安全性 1757
自動化資源管理 1758
安全最佳實務 1759
EKS 功能的考量事項 1759
功能 IAM 角色 1759
EKS 存取項目 1760
其他 Kubernetes 許可 1761
功能所需的 IAM 許可 1763
安全最佳實務 1763
後續步驟 83
IAM 參考 1766
目標對象 1766

使用身分驗證 1766
使用政策管理存取權 1769
Amazon EKS 和 IAM 1771
身分型政策 1775
服務連結角色 1782
Pod 執行 IAM 角色 1795
連接器 IAM 角色 1800
AWS 受管政策 1804
疑難排解 1834
叢集 IAM 角色 1836
節點 IAM 角色 1840
自動模式叢集 IAM 角色 1845
自動模式節點 IAM 角色 1851
功能 IAM 角色 1854
監控叢集 1860
在 Amazon EKS 上記錄和監控 1860
在 Amazon EKS 中監控和記錄工具 1861
可觀測性儀表板 1864
摘要 1864
叢集運作狀態 1864
控制平面監控 1865
叢集洞察 1866
節點運作狀態問題 1867
EKS 功能 1867
容器網路可觀測性 1867
使用案例 134
功能 69
開始使用 319
先決條件和重要備註 1869
使用 AWS CLI、EKS API 和 NFM API 1870
使用基礎設施做為程式碼 (IaC) 1872
其運作方式？ 193
考量和限制 1733
Prometheus 指標 1881
步驟 1：開啟 Prometheus 指標 1882
步驟 2：使用 Prometheus 指標 1884

步驟 3：管理 Prometheus 湊集器 1884
使用 Helm 部署 1884
控制平面 1887
Amazon CloudWatch 1895
Amazon CloudWatch 中的基本指標 1895
Amazon CloudWatch 可觀測性運算子 1902
控制平面日誌 1903
啟用或停用控制平面日誌 1904
檢視叢集控制平面日誌 1906
AWS CloudTrail 1908
參考 1908
日誌檔案項目 1909
Auto Scaling 群組指標 1912
ADOT 運算子 1912
Amazon EKS 儀表板 1913
什麼是 Amazon EKS 儀表板？ 1913
儀表板如何使用 AWS Organizations？ 1914
使用 AWS 主控台啟用 EKS 儀表板 1915
檢視 EKS 儀表板 1916
設定儀表板 1916
使用 AWS 主控台停用 EKS 儀表板 1917
EKS 儀表板疑難排解 1917
使用其他 服務 1924
AWS 備份 1924
AWS CloudFormation 1925
Amazon EKS 與 AWS CloudFormation 範本 1925
進一步了解 AWS CloudFormation 1925
AWS CodeConnections 1926
Use AWS CodeConnections 與 Argo CD 1926
搭配 Argo CD 使用 CodeConnections 的考量事項 1927
Amazon Detective 1927
將 Amazon Detective 與 Amazon EKS 搭配使用 1928
Amazon GuardDuty 1928
AWS 本機區域 1929
AWS Resilience Hub 1930
AWS Secrets Manager 1930

搭配 Argo CD 使用 AWS Secrets Manager 1930
Amazon Security Lake 1932
搭配 Amazon EKS 使用 Security Lake 的優勢 1932
針對 Amazon EKS 啟用 Security Lake 1933
分析 Security Lake 中的 EKS 日誌 1933
Amazon VPC Lattice 1933
Amazon EKS 連接器 1934
Amazon EKS 連接器考量事項 1934
Amazon EKS 連接器的必要 IAM 角色 1935
連接至叢集 1935
考量事項 1935
先決條件 1935
步驟 1：註冊叢集 1936
步驟 2：安裝 eks-connector 代理程式 1938
後續步驟 1940
授予存取所有叢集 1940
先決條件 1940
取消註冊叢集 1942
取消註冊 Kubernetes 叢集 1942
清除 Kubernetes 叢集中的資源 1943
對 EKS 連接器進行疑難排解 1944
基本疑難排解 1944
Helm 問題：403 禁止 1946
主控台錯誤：叢集停留在待定狀態 1946
主控台錯誤：使用者 system:serviceaccount:eks-connector:eks-connector 無法在叢集範圍內模擬 API 群組內的資源使用者 1946
主控台錯誤：[...]被禁止：使用者 [...] 無法在叢集範圍中列出「在 API 群組中的 [...]」的資源 1947
主控台錯誤：Amazon EKS 無法與您的 Kubernetes 叢集 API 伺服器通訊。叢集必須處於作用中狀態，才能成功連接。請在幾分鐘後再試一次。 1947
Amazon EKS 連接器 Pod 正在迴圈當機 1948
無法初始化 eks-connector：InvalidActivation 1948
叢集節點遺漏對外連線 1949
Amazon EKS 連接器 Pod 處於 ImagePullBackOff 狀態 1949
常見問答集 1950
安全考量 1951

AWS 責任 1951
客戶責任 1951
AWS Outposts 上的 Amazon EKS 1953
使用每個部署選項的時機 1953
兩個本機叢集實作 1954
比較部署選項 1954
執行本機叢集 1958
支援 AWS 的區域 1958
部署本機叢集 1959
EKS 平台版本 1968
建立 VPC 和子網路 1978
準備中斷連線 1981
容量考量事項 1986
對叢集進行疑難排解 1987
執行本機叢集 (EC2 執行個體存放區） 1996
具有 EC2 執行個體存放區 Outposts 的 Amazon EKS 本機叢集 1997
開始使用 1997
部署本機叢集 (EC2 執行個體存放區） 1998
建立 VPC 和子網路 (EC2 執行個體存放區） 2003
容量考量 (EC2 執行個體存放區） 2005
準備中斷連線 (EC2 執行個體存放區） 2008
Amazon EKS 附加元件 (EC2 執行個體存放區） 2013
叢集疑難排解 (EC2 執行個體存放區） 2015
節點 2017
eksctl 2018
AWS 管理主控台 2020
EKS 上的 AI/ML 2026
為什麼要將 Amazon EKS 用於 AI/ML 2026
金鑰使用案例 2026
案例研究 2027
指南結構 2028
在 Amazon EKS 上開始使用 AI/ML 2028
設定叢集 2028
架構概觀 2028
叢集運算選項 2029
您將設定的內容 2029

開始使用 319
命令列界面 (CLI) 2030
Terraform 2079
Inference 2101
為什麼使用 Amazon EKS 進行推論 2102
詞彙表 2102
Amazon EKS 上的推論如何運作 2103
推論效能指標 2103
縮短冷啟動時間 2104
您將部署的內容 2104
載入並服務模型 2105
自動擴展 2120
加速模型載入 2139
叢集組態 2150
使用 EKS Linux GPU AMIs 2150
設定 Windows GPU AMI 2155
使用 EFA 設定訓練叢集 2160
使用 Inferentia 設定推論叢集 2170
容量管理 2176
從 EC2 執行個體類型中選擇 2176
了解 EC2 購買選項 2176
將購買選項與工作負載需求配對 2177
驗證您的 EC2 服務配額 2177
搭配 Amazon EKS 使用 EC2 購買選項 2178
混合策略：合併購買選項 2178
EKS 自動模式和 Karpenter 2179
受管節點群組和自我管理節點 2197
搭配 EKS 使用 P6e-GB200 2202
裝置管理 2215
動態資源分配與裝置外掛程式的比較 2215
Amazon EKS 的 DRA 驅動程式 2216
Amazon EKS 的裝置外掛程式 2216
考量事項 87
DRA ResourceClaim 與 ResourceClaimTemplate 2217
主題 2219
EFA 裝置 2219

Neuron 加速器 2234
NVIDIA GPU 2242
配方 2283
阻止在特定節點上排程 Pod 2283
Resources 2285
研討會 2285
最佳實務 2286
參考架構 2287
教學 2287
工具 2289
模型內容通訊協定 (MCP) 2289
概觀 80
整合範例 2290
開始使用 319
開始使用 2291
組態 2307
工具 2309
Amazon Q 2318
運作方式 133
使用 Amazon Q 進行疑難排解 2319
考量事項 87
進一步了解 2319
版本控制 2320
Kubernetes 版本 2320
標準支援的可用版本 2321
延長支援的可用版本 2321
版本轉返 2322
Amazon EKS Kubernetes 發佈日曆 2322
使用 CLI AWS 取得版本資訊 2323
Amazon EKS 版常見問題 2324
Amazon EKS 延長支援常見問答集 2326
標準支援版本 2327
延長支援版本 2332
檢視支援時段 2336
檢視升級政策 2336
啟用延長支援 2338

停用延伸支援 2339
平台版本 2340
Kubernetes 版本 1.36 2341
Kubernetes 版本 1.35 2342
Kubernetes 版本 1.34 2344
Kubernetes 版本 1.33 2347
Kubernetes 版本 1.32 2351
Kubernetes 版本 1.31 2357
Kubernetes 版本 1.30 2362
了解目前的平台版本 2369
變更平台版本 2370
疑難排解 2371
容量不足 2371
節點無法加入叢集 2371
未經授權或存取遭拒 (kubectl) 2373
hostname doesn't match 2374
getsockopt: no route to host 2374
Instances failed to join the Kubernetes cluster 2374
受管節點群組錯誤代碼 2374
Not authorized for images 2379
節點進入 NotReady 狀態 2379
EKS 日誌收集器 2379
容器執行階段網路尚未就緒 2380
TLS 交握逾時 2381
InvalidClientTokenId 2382
升級控制平面之前，節點群組必須符合 Kubernetes 版本 2382
啟動許多節點時，會出現 Too Many Requests 錯誤 2383
針對 Kubernetes API 伺服器請求回應的 HTTP 401 未經授權的錯誤 2383
Amazon EKS 平台版本比目前平台版本落後兩個版本以上 2383
叢集運作狀態常見問答集與包含解析路徑的錯誤代碼 2386
Amazon EKS 相關專案 2391
支援部署至 EKS 的軟體 2391
管理工具 2392
eksctl 2392
AWS Kubernetes 的控制器 2392
kro (Kube Resource Orchestrator) 2392

Argo CD 2392
Flux CD 2393
Kubernetes 專用 CDK 2393
聯網 2393
Kubernetes 專用 Amazon VPC CNI 外掛程式 2393
Kubernetes 的AWS Load Balancer控制器 2394
ExternalDNS 2394
機器學習 2394
Kubeflow 2394
Auto Scaling 2394
Cluster Autoscaler 2394
Karpenter 2395
Escalator 2395
監控 2395
Prometheus 2395
持續整合 / 持續部署 2395
Jenkins X 2396
新功能與藍圖 2397
文件歷史紀錄 2398
貢獻 2447
編輯單一頁面 2447
先決條件 42
程序 90
提取請求概觀 2449
使用 GitHub 編輯檔案 2450
先決條件 42
程序 90
檢視樣式意見回饋 2451
安裝 Vale 2452
安裝 VS Code Vale 延伸 2452
同步 Vale 2453
在 VS Code 中檢視樣式意見回饋 2453
建立頁面 2453
建立頁面 2453
將頁面新增至導覽 2454
插入連結 2454

《EKS 使用者指南》中的頁面或章節的連結 2454
連結至 AWS 文件中的其他指南 2455
外部網頁的連結 2455
使用 Amazon Q 建立 2456
透過 VS Code 安裝 Amazon Q 2456
登入 Amazon Q 2457
透過 Amazon Q 建立內容 2457
提示 2457
檢視 PR 預覽 2458
檢視預覽 2458
預覽限制 2459
AsciiDoc 語法 2459
新頁面 2459
標題 2459
文字格式 2460
清單 2460
連結 2461
程式碼範例 2461
映像 2461
表格 2461
標註 2462
包含其他檔案 2462
屬性 (類似實體) 2462
程序 2463

協助改進此頁面

本文為英文版的機器翻譯版本，如內容有任何歧義或不一致之處，概以英文版為準。

若要為本使用者指南貢獻內容，請點選每個頁面右側面板中的在 GitHub 上編輯此頁面連結。

本文為英文版的機器翻譯版本，如內容有任何歧義或不一致之處，概以英文版為準。

什麼是 Amazon EKS？

> **Tip**
>
> 註冊即將舉行的 Amazon EKS 研討會。

Amazon EKS：簡化的 Kubernetes 管理

Amazon Elastic Kubernetes Service (EKS) 提供一項全受管 Kubernetes 服務，能夠消除運作 Kubernetes 叢集的複雜性。您可藉助 EKS 實現：

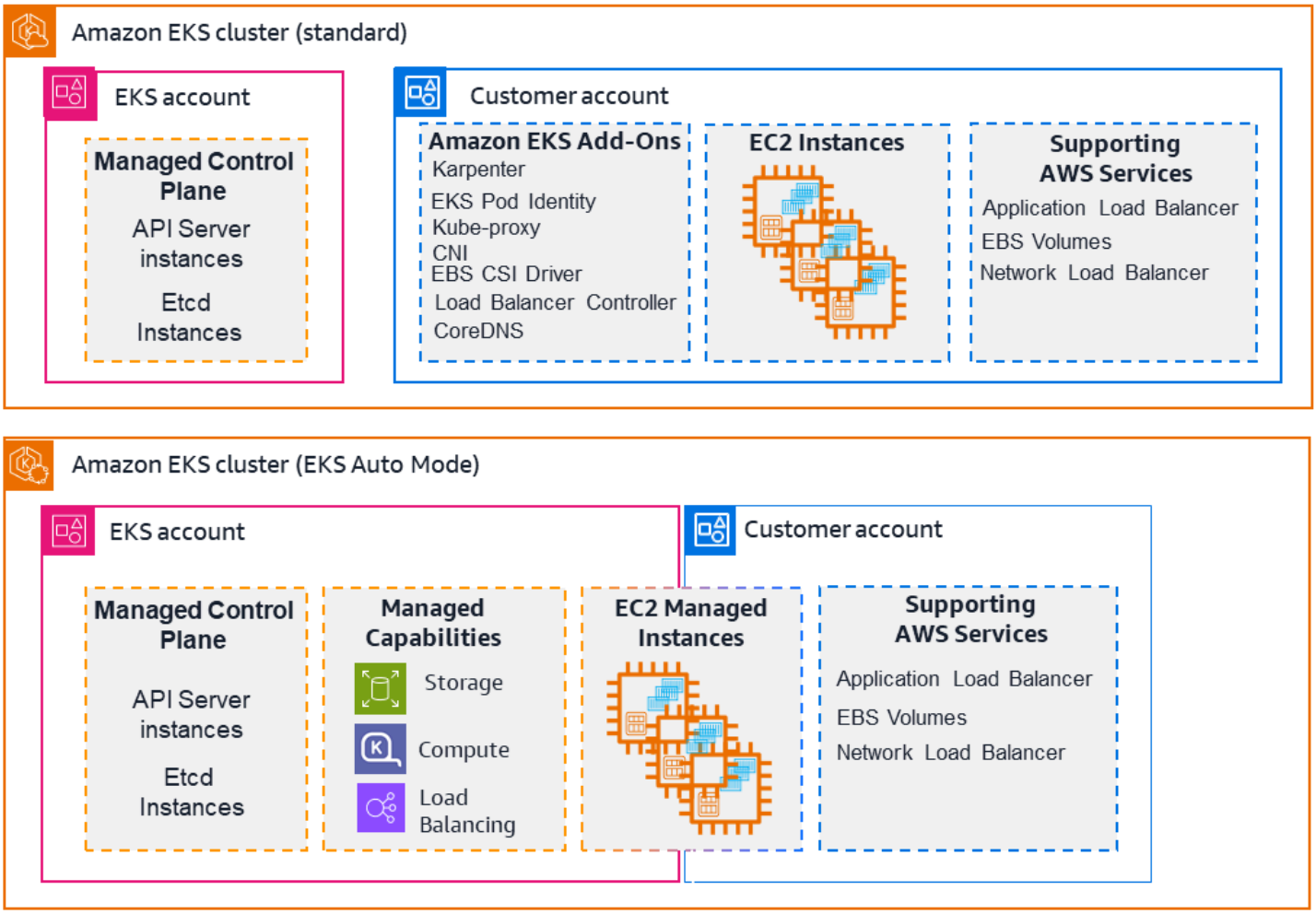
- 以較低營運開銷加速部署應用程式
- 順暢擴展以滿足日益變化的工作負載需求
- 透過 AWS 整合和自動更新來改善安全性
- 在標準 EKS 或全自動 EKS 自動模式間進行選擇

Amazon Elastic Kubernetes Service (Amazon EKS) 是在 Amazon Web Services (AWS) 雲端和您自己的資料中心 (EKS Anywhere 和 Amazon EKS 混合節點) 中執行 Kubernetes 叢集的首要平台。

Amazon EKS 讓 Kubernetes 叢集的建置、保護和維護得以簡化。相較於維護您自己的資料中心，該服務可提供充足的資源來滿足尖峰需求，更加經濟高效。使用 Amazon EKS 的兩種主要方法如下所示：

- 當您使用 EKS 建立叢集時，EKS standard： AWS 管理 Kubernetes 控制平面。系統會自動為您處理管理節點、排程工作負載、與 AWS 雲端整合，以及存放和擴展控制平面資訊的元件，以保持叢集正常運作。
- EKS 自動模式：EKS 使用 EKS 自動模式功能來延伸其控制，以同時管理節點 (Kubernetes 資料平面)。它透過自動佈建基礎設施、選取最佳運算執行個體、動態擴展資源、持續最佳化成本、修補作業系統，以及與 AWS 安全服務整合，來簡化 Kubernetes 管理。

下圖說明 Amazon EKS 如何根據您選擇的叢集建立方法，將 Kubernetes 叢集與 AWS 雲端整合：



Amazon EKS 可協助您消除摩擦並加速生產時間、改善效能、可用性和彈性，以及增強系統安全性。如需詳細資訊，請參閱 Amazon Elastic Kubernetes Service。

使用 Kubernetes 建置和擴展：Amazon EKS 功能

Amazon EKS 不僅可協助您建置和管理叢集，還可協助您使用 Kubernetes 建置和擴展應用程式系統。Amazon EKS 功能是全受管叢集服務，可使用免持 Kubernetes 原生工具擴展叢集的功能，包括：

- Argo CD：Argo CD 為您的工作負載、 AWS 資源和雲端基礎設施提供宣告式、以 GitOps 為基礎的持續部署。
- AWS Kubernetes (ACK) 控制器：ACK 支援 Kubernetes 原生 AWS 的資源建立和生命週期管理，整合工作負載協同運作和Infrastructure-as-code工作流程。
- kro (Kube Resource Orchestrator)：kro 擴展原生 Kubernetes 功能，以簡化自訂資源建立、協同運作和組成，為您提供建立自訂雲端建置區塊的工具。

EKS 功能是雲端資源，可將在叢集中安裝、維護和擴展這些基礎平台元件的操作負擔降至最低，讓您專注於建置軟體，而非叢集平台操作。

如需詳細資訊，請參閱 EKS 功能。

Amazon EKS 的功能

Amazon EKS 提供以下進階功能：

管理介面

EKS 提供多個介面來佈建、管理和維護叢集，包括 AWS 管理主控台 Amazon EKS API/SDKs、CDK、 AWS CLI、eksctl CLI、 AWS CloudFormation 和 Terraform。如需詳細資訊，請參閱開始使用及設定叢集。

存取控制工具

EKS 依賴 Kubernetes 和 AWS Identity and Access Management (AWS IAM) 功能來管理來自使用者和工作負載的存取。如需詳細資訊，請參閱the section called “Kubernetes API 存取”及the section called “工作負載存取 AWS ”。

運算資源

對於運算資源，EKS 允許完整範圍的 Amazon EC2 執行個體類型和 AWS 創新，例如 Nitro 和 Graviton 搭配 Amazon EKS，讓您最佳化工作負載的運算。如需詳細資訊，請參閱管理運算。

儲存

EKS 自動模式可使用 EBS 磁碟區來自動建立儲存類別。使用容器儲存介面 (CSI) 驅動程式，您也可以使用 Amazon S3、Amazon S3 檔案、Amazon EFS、Amazon FSx 和 Amazon File Cache 滿足您的應用程式儲存需求。如需詳細資訊，請參閱應用程式資料儲存。

安全性

採取共同的責任模型，因為該模型與 Amazon EKS 中的安全相關。如需詳細資訊，請參閱安全性最佳實務、基礎結構安全性與 Kubernetes 安全性。

監控工具

使用可觀測性儀表板來監控 Amazon EKS 叢集。監控工具包括 Prometheus、CloudWatch、CloudTrail 和 ADOT Operator。若要了解儀表板、指標伺服器及其他工具的相關詳細資訊，請參閱 EKS 叢集成本與 Kubernetes 指標伺服器。

叢集功能

EKS 提供受管叢集功能，以根據開放原始碼創新進行持續部署、雲端資源管理和資源合成。EKS 會在您的叢集中安裝 Kubernetes APIs，但控制器和其他元件會在 EKS 中執行並受到完整管理，提供自動化修補、擴展和監控。如需詳細資訊，請參閱EKS 功能。

Kubernetes 相容性與支援

Amazon EKS 具備 Kubernetes 合規認證，因此，您無須重構及使用 Kubernetes 社群工具與外掛程式，即可部署與 Kubernetes 相容的應用程式。EKS 提供 Kubernetes 標準支援與延長支援。如需詳細資訊，請參閱the section called “Kubernetes 版本”。

相關服務

與 Amazon EKS 搭配使用的服務

您可以使用其他 AWS 服務搭配您使用 Amazon EKS 部署的叢集：

Amazon EC2

藉助 Amazon EC2，獲取隨需、可擴展的運算容量。

Amazon EBS

藉助 Amazon EBS，連接可擴展、高效能的區塊儲存資源。

Amazon ECR

藉助 Amazon ECR，安全地存放容器映像。

Amazon CloudWatch

使用 Amazon CloudWatch 即時監控 AWS 資源和應用程式。

Amazon Prometheus

藉助 Amazon Managed Service for Prometheus，來追蹤容器化應用程式的指標。

Elastic Load Balancing

藉助 Elastic Load Balancing，將傳入流量分散至多個目標。

Amazon GuardDuty

藉助 Amazon GuardDuty，來偵測對 EKS 叢集的威脅。

AWS Resilience Hub

藉助 AWS Resilience Hub，來評估 EKS 叢集復原能力。

Amazon EKS 定價

Amazon EKS 提供三種定價：以 Kubernetes 叢集版本支援為基礎的每個叢集定價、Amazon EKS 自動模式定價，以及 Amazon EKS 混合節點的每個 vCPU 定價。

使用 Amazon EKS 時，您需要另外支付用於在 Kubernetes 工作者節點上執行應用程式 AWS 的資源費用。舉例來說，若以使用 Amazon EBS 磁碟區及公有 IPv4 位址的 Amazon EC2 執行個體執行 Kubernetes 工作節點，您需要為下列服務付費：透過 Amazon EC2 使用執行個體容量、透過 Amazon EBS 使用磁碟區容量，以及透過 Amazon VPC 使用 IPv4 位址。

叢集中的 Amazon EKS 控制平面和工作者節點之間的通訊會使用 VPC 中的申請者受管網路介面。系統會針對此連線之客戶端的流量向您收取標準 AWS 資料傳輸費率，特別是從控制平面傳入工作者節點，以及從工作者節點傳出至控制平面。Amazon EKS 會在連線的控制平面端吸收資料傳輸成本。如需資料傳輸定價的詳細資訊，請參閱資料傳輸定價。

請造訪您搭配 Kubernetes 應用程式使用 AWS 之服務的個別定價頁面，以取得詳細的定價資訊。

- 如需 Amazon EKS 叢集、Amazon EKS Auto Mode、Amazon EKS 功能和 Amazon EKS 混合節點定價，請參閱 Amazon EKS 定價。
- 如要了解 Amazon EC2 定價，請參閱 Amazon EC2 隨需定價與 Amazon EC2 Spot 定價。
- 如需 AWS Fargate 定價，請參閱 AWS Fargate 定價。
- 對於 Amazon EKS 叢集中使用的運算，您還可使用 Savings Plans。如需詳細資訊，請參閱透過 Savings Plans 定價。

Amazon EKS 中的常用案例

Amazon EKS 在 上提供強大的受管 Kubernetes 服務 AWS，旨在最佳化容器化應用程式。以下是一些 Amazon EKS 最常見的使用案例，協助您充分利用其強項來滿足您的特定需求。

部署高可用性應用程式

使用 Elastic Load Balancing，您可以確保您的應用程式在多個可用區域之中均高度可用。

建置微服務架構

搭配 AWS Cloud Map 或 Amazon VPC Lattice 使用 Kubernetes 服務探索功能來建置彈性系統。

自動化軟體版本程序

管理持續整合和持續部署 (CI/CD) 管道，簡化應用程式自動化建置、測試和部署的程序。如需宣告式持續部署，請參閱使用 Argo CD 持續部署。

執行無伺服器應用程式

搭配 Amazon EKS 使用 AWS Fargate 來執行無伺服器應用程式。這表示您可以專注於應用程式開發，而 Amazon EKS 和 Fargate 則可以處理基礎結構。

執行機器學習工作負載

Amazon EKS 與熱門的機器學習架構相容，例如 TensorFlow、MXNet 和 PyTorch。有了 GPU 支援，您甚至可以有效地處理複雜的機器學習任務。

在內部部署和雲端中一致地部署

為了簡化內部部署環境中的 Kubernetes 執行，您既可使用相同的 Amazon EKS 叢集、功能和工具在 AWS Outposts 上執行自我管理節點，也可使用 Amazon EKS 混合節點搭配您自己的基礎結構。對於可獨立運作、氣隙隔離的環境，您可使用 Amazon EKS Anywhere 在您自己的基礎結構上自動化 Kubernetes 叢集生命週期管理。

執行符合成本效益的批次處理和大數據工作負載

運用 Spot 執行個體來執行批次處理和巨量資料工作負載，例如 Apache Hadoop and Spark，僅需極少的成本。這可讓您以折扣價格充分利用未使用的 Amazon EC2 容量。

從 Kubernetes 管理 AWS 資源

使用 AWS Controllers for Kubernetes (ACK)，使用原生 Kubernetes APIs 直接從 Kubernetes 叢集建立和管理 AWS 資源。

建置平台工程抽象概念

建立自訂 Kubernetes APIs，使用 kro (Kube Resource Orchestrator) 將多個資源組成更高層級的抽象。

保護應用程式並確保合規性

實作強大的安全實務，並維持與 Amazon EKS 的合規性，Amazon EKS 與 AWS Identity and Access Management (IAM)、Amazon Virtual Private Cloud (Amazon VPC) 和 AWS Key Management Service (AWS KMS) 等 AWS 安全服務整合。這確保了資料隱私權和保護均依照產業標準實行。

Amazon EKS 架構

Amazon EKS 與 Kubernetes 的一般叢集架構一致。如需詳細資訊，請參閱 Kubernetes 文件中的 Kubernetes 原件。下列各節摘要了 Amazon EKS 一些額外架構的詳細資訊。

控制平面

Amazon EKS 能確保每個叢集都有自己獨一無二的 Kubernetes 控制平面。此設計會將每個叢集的基礎設施分開，而不會在叢集或 AWS 帳戶之間重疊。設定包含：

分散式元件

控制平面會在 AWS 區域內的三個 AWS 可用區域中，定位至少兩個 API 伺服器執行個體和三個等化執行個體。

最佳效能

Amazon EKS 會主動監控和調整控制平面執行個體，以維持尖峰效能。

彈性

如果控制平面執行個體淡出，Amazon EKS 會視需要使用不同的可用區域快速取代它。

一致的執行時間

透過在多個可用區域之中執行叢集的方式來達成可靠的 API 伺服器端點可用性服務水準協議 (SLA)。

Amazon EKS 會使用 Amazon Virtual Private Cloud (Amazon VPC) 來限制單一叢集內控制平面元件之間的流量。叢集元件無法檢視或接收來自其他叢集或 AWS 帳戶的通訊，除非 Kubernetes 角色型存取控制 (RBAC) 政策授權。

運算

除了控制平面之外，Amazon EKS 叢集還有一組稱為節點的工作者機器。選取適當的 Amazon EKS 叢集節點類型，對於滿足您特定的需求與最佳化資源使用率而言十分關鍵。Amazon EKS 提供下列主節點類型：

EKS 自動模式

EKS Auto Mode 將 AWS 管理延伸到控制平面之外，以包含資料平面，自動化叢集基礎設施管理。它整合了核心 Kubernetes 功能作為內建元件，包括運算自動擴展、聯網、負載平衡、DNS、儲存

和 GPU 支援。EKS 自動模式根據工作負載需求動態管理節點，使用具有增強安全功能的不可變 AMI。它在遵守 Pod 中斷預算的同時，自動化更新與升級，並包含原本需要附加元件管理的受管元件。此選項非常適合希望利用day-to-day操作 AWS 專業知識、將營運開銷降至最低，以及專注於應用程式開發而非基礎設施管理的使用者。

AWS Fargate

Fargate 是適用於容器的無伺服器運算引擎，無需管理基礎執行個體。使用 Fargate，您可以指定應用程式的資源需求，並 AWS 自動佈建、擴展和維護基礎設施。此選項非常適合優先考慮簡單易用的使用者，也相當適合希望專注於應用程式開發和部署，而不是管理基礎結構的使用者。

Karpenter

Karpenter 是一款靈活的高效能 Kubernetes 叢集自動擴展程式，可協助提升應用程式的可用性和叢集效率。Karpenter 會根據應用程式負載的變化啟動大小適中的運算資源。此選項可佈建符合工作負載要求的即時運算資源。

受管節點群組

受管節點群組是自動化和自訂的混合，用於管理 Amazon EKS 叢集內 Amazon EC2 執行個體的集合。 AWS 負責修補、更新和擴展節點等任務，讓操作方面更為輕鬆。同時，也支援自訂 `kubelet` 引數，為進階 CPU 和記憶體管理政策開啟了可能性。此外，它們透過服務帳戶的 AWS Identity and Access Management (IAM) 角色來增強安全性，同時減少每個叢集對個別許可的需求。

自我管理節點

自我管理節點提供對 Amazon EKS 叢集中的 Amazon EC2 執行個體的完全控制權。您負責管理、擴展和維護節點，讓您完全掌握基礎結構。此選項適合需要精細控制和自訂其節點的使用者，也適合已經準備好投入時間來管理和維護其基礎架構的使用者。

Amazon EKS 混合節點

透過 Amazon EKS 混合節點，您可使用內部部署和邊緣基礎結構作為 Amazon EKS 叢集中的節點。Amazon EKS 混合節點整合跨環境 AWS 的 Kubernetes 管理，並為內部部署和邊緣應用程式將 Kubernetes 控制平面管理卸載至 。

EKS 功能

Amazon EKS 提供全受管叢集功能，在叢集中安裝和管理 Kubernetes APIs （使用 Kubernetes Custom Resource Definitions)，同時操作控制器和其他元件 AWS的自有基礎設施，與您的叢集分開。EKS 提供這些功能的自動化修補、擴展和監控，完全管理其生命週期，以減輕為工作負載協同運作、 AWS 資源管理等操作叢集內服務的負擔。

EKS 提供下列功能類型：

AWS Kubernetes (ACK) 的控制器

AWS Kubernetes (ACK) 的控制器可讓您使用 Kubernetes APIs 管理 AWS 資源，讓您將 S3 儲存貯體、RDS 資料庫、IAM 角色和其他 AWS 資源定義為 Kubernetes 自訂資源。您可以使用相同的工具和工作流程來管理 AWS 資源與 Kubernetes 工作負載，並支援 50 種以上的 AWS 服務，包括 S3、RDS、DynamoDB 和 Lambda。

Argo CD

Argo CD 使用 Git 儲存庫作為事實來源，為您的應用程式工作負載、資源和叢集組態實作 GitOps 型持續部署。 AWS Argo CD 會自動同步您的叢集與 Git 儲存庫，並偵測偏離，持續協調以確保您部署的應用程式和資源符合版本控制中所需的狀態。您可以使用 Argo CD 來管理特定叢集上的應用程式，或從單一 Argo CD 資源跨多個叢集部署和管理應用程式，並在每次遞交變更時從 Git 儲存庫自動部署。

kro (Kube Resource Orchestrator)

kro (Kube Resource Orchestrator) 可讓您建立自訂 Kubernetes APIs，將多個資源組成更高階的抽象概念，讓平台團隊為常見的資源組合定義可重複使用的模式。這可讓平台團隊提供具有適當防護機制的自助式功能，讓開發人員能夠使用簡單的專用 APIs 佈建複雜的基礎設施，同時維護組織標準和最佳實務。

Kubernetes 概念

Amazon Elastic Kubernetes Service (Amazon EKS) 是以開放原始碼 Kubernetes 專案為基礎的 AWS 受管服務。雖然您需要了解 Amazon EKS 服務如何與 AWS Cloud 整合 （特別是當您首次建立 Amazon EKS 叢集時），但一旦 Amazon EKS 叢集啟動並執行，您使用 Amazon EKS 叢集的方式與任何其他 Kubernetes 叢集的方式大致相同。因此，若要開始管理 Kubernetes 叢集並部署工作負載，您至少需要對 Kubernetes 概念有基本的了解。

此頁面將 Kubernetes 概念分為三個部分：the section called “為什麼選擇 Kubernetes？”、the section called “叢集” 和 the section called “工作負載”。第一部分說明執行 Kubernetes Service 的價值，特別是 Amazon EKS 等受管服務。工作負載部分介紹了如何建置、存放、執行和管理 Kubernetes 應用程式。叢集部分則會配置組成 Kubernetes 叢集的不同元件，以及您建立和維護 Kubernetes 叢集的責任。

主題

- 為什麼選擇 Kubernetes？

- 叢集
- 工作負載
- 後續步驟

當您瀏覽此內容時，會看到一些連結，這些連結將會引導您進一步檢視 Amazon EKS 和 Kubernetes 文件中有關 Kubernetes 概念的說明，如果您想要深入了解此處涵蓋的任何主體，即可按一下這些連結進行查閱。如需 Amazon EKS 如何實作 Kubernetes 控制平面和運算功能的詳細資訊，請參閱 the section called “Architecture”。

為什麼選擇 Kubernetes？

Kubernetes 旨在提高執行關鍵任務、生產品質容器化應用程式時的可用性和可擴展性。Kubernetes 不僅是在單一機器上執行 Kubernetes (雖然有可能)，而且讓您能夠在可根據需求擴展或收縮的電腦組中執行應用程式，進而實現這些目標。Kubernetes 包含的功能讓您能更輕鬆地：

- 在多台機器上部署應用程式 (使用部署在 Pod 中的容器)
- 監控容器的運作狀態並重新啟動故障容器
- 根據負載向上和向下擴展容器
- 將容器更新至新版本
- 在容器之間配置資源
- 平衡跨機器的流量

讓 Kubernetes 自動執行這些類型的複雜任務，以便應用程式開發人員專注於建置和改善其應用程式工作負載，而不是擔心基礎結構。開發人員通常會建立格式為 YAML 檔案的組態檔案，以此描述應用程式所需的狀態。這可能包括要執行的容器、資源限制、Pod 複本數量、CPU/記憶體配置、親和性規則，等等。

Kubernetes 的屬性

為實現其目標，Kubernetes 具有下列屬性：

- 容器化：Kubernetes 是一種容器協同運作工具。若要使用 Kubernetes，您必須先將您的應用程式容器化。視應用程式類型而定，這可以表現為一組微服務、批次任務或其他形式。然後，您的應用程式可以利用包含龐大工具生態系統的 Kubernetes 工作流程，其中容器可以儲存為容器登錄檔中的映像，可部署到 Kubernetes 叢集，並且能在可用的節點上執行。您可以在本機電腦上使用 Docker 或其他容器執行時期建置和測試個別容器，然後再將其部署到您的 Kubernetes 叢集。

- 可擴展：如果您的應用程式的需求超過這些應用程式的執行中的執行個體的容量，那麼 Kubernetes 可以向上擴展。Kubernetes 可以視需要判斷應用程式是否需要更多 CPU 或記憶體，並透過自動擴展可用容量或使用更多現有容量來進行回應。如果有足夠的運算容量來執行更多應用程式執行個體，則可以在 Pod 層級進行擴展 (水平 Pod 自動擴展)；如果需要新增更多節點來處理增加的容量，則可以在節點層級進行擴展 (Cluster Autoscaler 或 Karpenter)。由於不再需要容量，這些服務可以刪除不必要的 Pod 並關閉不需要的節點。
- 可用：如果應用程式或節點運作狀態不佳或不可用，則 Kubernetes 可以將執行中的工作負載移動至另一個可用的節點。您只需刪除工作負載的執行中的執行個體或執行工作負載的節點，即可強制解決此問題。此時的底線是，如果工作負載無法再原地執行，則可以將它們移動到其他位置。
- 宣告：Kubernetes 使用主動調解，以持續檢查您為叢集宣告的狀態是否與實際狀態相符。例如，透過將 Kubernetes 物件套用至叢集 (通常是藉助 YAML 格式的組態檔案)，您可以要求啟動要在叢集上執行的工作負載。您稍後即可變更組態以執行某些動作，例如使用較新版本的容器或配置更多記憶體。Kubernetes 會執行一些必要動作，以建立所需狀態。這可能包括啟動或關閉節點、停止和重新啟動工作負載，或是提取更新的容器。
- 可組合：由於應用程式通常由多個元件組成，因此您希望能夠對這些元件進行統一管理 (通常由多個容器表示)。雖然 Docker Compose 提供了直接使用 Docker 執行此動作的方法，但 Kubernetes Kompose 命令可協助您使用 Kubernetes 執行此動作。如需如何執行此動作的範例，請參閱將 Docker Compose 檔案轉換為 Kubernetes 資源。
- 可擴展：與專屬軟體不同，開放原始碼 Kubernetes 專案旨在讓您以任何您想要的方式擴展 Kubernetes，進而滿足您的需求。開啟 API 和組態檔案，以直接進行修改。建議第三方編寫自己的控制器，以擴展基礎結構和最終使用者 Kubernetes 功能。Webhook 可讓您設定叢集規則，以強制執行政策並適應不斷變化的條件。如需如何擴展 Kubernetes 叢集的更多想法，請參閱擴展 Kubernetes。
- 可攜式：許多組織已在 Kubernetes 上將其操作標準化，因為它允許他們以相同的方式管理其所有應用程式需求。開發人員可以使用相同的管道來建置和儲存容器化應用程式。然後，這些應用程式可以部署到內部部署、雲端、餐廳point-of-sales終端機或分散在公司遠端網站的 IoT 裝置上執行的 Kubernetes 叢集。其開放原始碼性質可讓人員開發這些特殊的 Kubernetes 分佈，以及管理它們所需的工具。

管理 Kubernetes

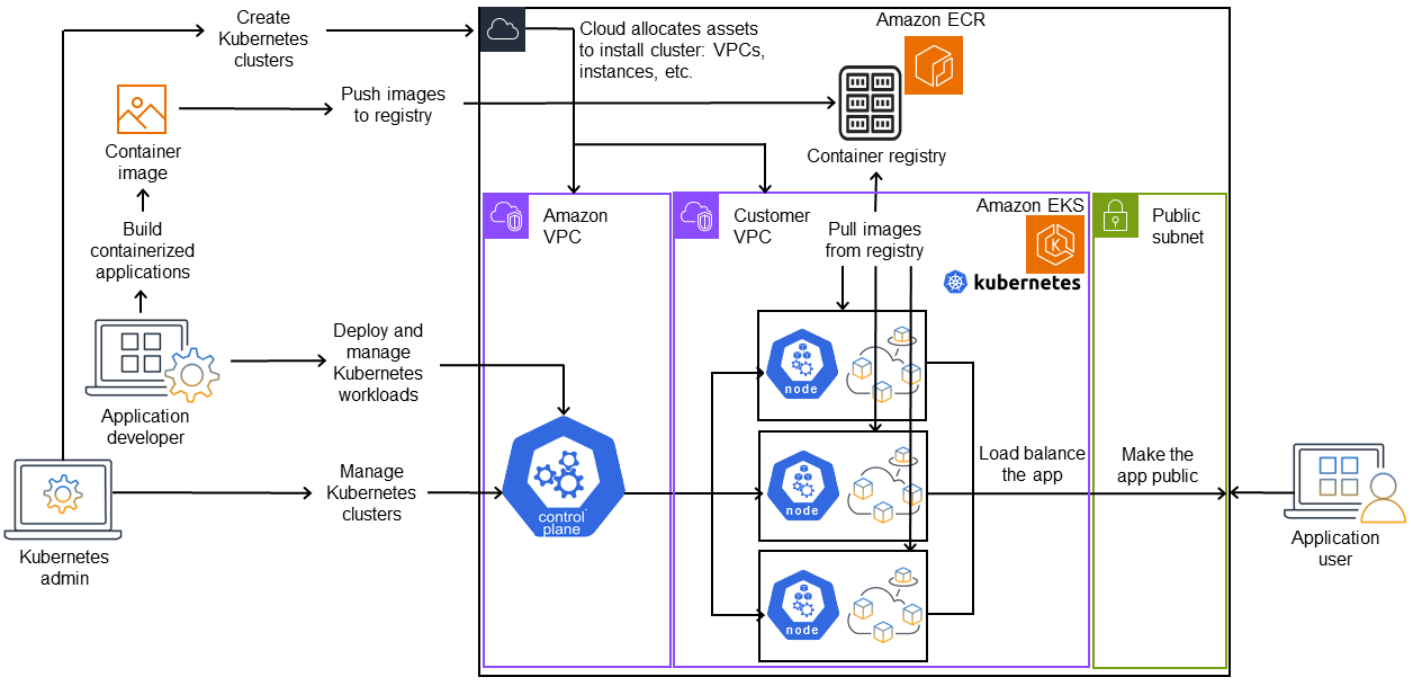
Kubernetes 原始程式碼會免費提供，因此您可以使用自己的設備自行安裝和管理 Kubernetes。不過，自我管理 Kubernetes 需要深入的操作專業知識，而且進行維護亦需要耗費時間和精力。基於這些原因，大多數部署生產工作負載的人員都會選擇雲端提供者 (例如 Amazon EKS) 或內部部署提供者 (例

如 Amazon EKS Anywhere)，而這些提供者皆擁有其自己經過測試的 Kubernetes 分佈和 Kubernetes 專家的支援。這可讓您卸載維護叢集所需的大部分無差別的繁重任務，包括：

- 硬體 ： 如果您沒有可根據您的需求執行 Kubernetes AWS 的硬體，Amazon EKS 等雲端供應商可以節省您的預付成本。使用 Amazon EKS，這表示您可以使用 提供的最佳雲端資源 AWS，包括運算執行個體 (Amazon Elastic Compute Cloud)、您自己的私有環境 (Amazon VPC)、中央身分和許可管理 (IAM) 和儲存 (Amazon EBS)。 AWS 管理電腦、網路、資料中心，以及執行 Kubernetes 所需的所有其他實體元件。同樣地，您不需要規劃資料中心，即可在需求最高的時日處理最大容量。對於 Amazon EKS Anywhere 或其他內部部署 Kubernetes 叢集，您需負責管理 Kubernetes 部署中使用的基礎設施，但您仍可依賴 AWS 來協助您將 Kubernetes 保持在最新狀態。
- 控制平面管理 — Amazon EKS 管理 AWS託管 Kubernetes 控制平面的安全性和可用性，負責排程容器、管理應用程式的可用性和其他關鍵任務，因此您可以專注於應用程式工作負載。如果您的叢集中斷， AWS 應該具有將叢集還原為執行中狀態的方法。對於 Amazon EKS Anywhere，您可自行管理控制平面。
- 經過測試的升級：升級叢集時，您可以依賴 Amazon EKS 或 Amazon EKS Anywhere 來提供其 Kubernetes 分佈的經測試版本。
- 附加元件：建置數百個專案，以擴展和使用 Kubernetes，而您可以將這些專案新增至叢集的基礎結構，或使用這些專案來協助工作負載的執行。 AWS 提供您可以與叢集搭配使用the section called “Amazon EKS 附加元件”的 ，而不是自行建置和管理這些附加元件。Amazon EKS Anywhere 提供精製套件，其中包含了眾多熱門的開放原始碼專案的建置版本。因此，您無需自行建置軟體或管理重要的安全修補程式、錯誤修正或升級。同樣地，如果預設符合您的需求，則通常只需要對這些附加元件進行極少的設定即可。如需使用附加元件擴展叢集的詳細資訊，請參閱 the section called “擴展叢集”。

執行中的 Kubernetes

下圖顯示您身為 Kubernetes 管理員或應用程式開發人員要建立和使用 Kubernetes 叢集的關鍵活動。在此過程中，它會說明 Kubernetes 元件如何彼此互動，使用 AWS 雲端做為基礎雲端供應商的範例。



Kubernetes 管理員會使用要建置叢集的此類提供者的專用工具來建立 Kubernetes 叢集。此範例使用 AWS 雲端做為提供者，提供稱為 Amazon EKS 的受管 Kubernetes 服務。受管服務會自動配置建立叢集所需的資源，包括為叢集建立兩個新的虛擬私有雲端 (Amazon VPC)、設定聯網，以及將 Kubernetes 許可與新的 VPC 直接進行對應，進而執行雲端資產管理。受管服務也會看到控制平面服務具有執行和配置零個或多個 Amazon EC2 執行個體做為執行工作負載的 Kubernetes 節點的位置。會 AWS 管理控制平面的一個 Amazon VPC 本身，而另一個 Amazon VPC 包含執行工作負載的客戶節點。

許多 Kubernetes 管理員的後續任務都需要使用 Kubernetes 工具來完成，例如 `kubectl`。該工具會直接向叢集的控制平面發出服務請求。然後，對叢集執行查詢和變更的方式與您在任何 Kubernetes 叢集上執行查詢和變更的方式非常類似。

想要將工作負載部署到此叢集的應用程式開發人員可以執行數個任務。開發人員需要將應用程式建置到一或多個容器映像中，然後將這些映像推送到可供 Kubernetes 叢集存取的容器登錄檔。 為此目的 AWS 提供 Amazon Elastic Container Registry (Amazon ECR)。

若要執行應用程式，開發人員可以建立 YAML 格式的組態檔案，告訴叢集如何執行應用程式，包括要從登錄檔提取哪些容器以及如何將這些容器包裝在 Pod 中。控制平面 (排程器) 會將容器排程至一或多個節點，而每個節點上的容器執行時期實際上會提取及執行所需的容器。開發人員也可以設定 Application Load Balancer，以平衡每個節點上執行的可用容器的流量並公開應用程式，以便這些應用

程式可在公有網路上供外界使用。完成後，想要使用應用程式的人員可以連接到應用程式端點，以進行存取。

以下各節會從 Kubernetes 叢集和工作負載的角度詳細介紹這些功能。

叢集

如果您的任務是啟動和管理 Kubernetes 叢集，則您應該知道如何建立、增強、管理和刪除 Kubernetes 叢集。您還應該知道組成叢集的元件有哪些，以及您需要做些什麼來維護這些元件。

用於管理叢集的工具可處理 Kubernetes Service 和基礎硬體提供者之間的重疊。因此，Kubernetes 供應商（例如 Amazon EKS 或 Amazon EKS Anywhere) 通常會使用供應商專用的工具來完成這些任務的自動化。例如，若要啟動 Amazon EKS 叢集，您可以使用 `eksctl create cluster`，而對於 Amazon EKS Anywhere，則可以使用 `eksctl anywhere create cluster`。請注意，當這些命令建立 Kubernetes 叢集時，它們專屬於提供商，而不屬於 Kubernetes 專案本身。

叢集建立和管理工具

Kubernetes 專案可提供手動建立 Kubernetes 叢集的工具。因此，如果您想要在單一機器上安裝 Kubernetes，或在機器上執行控制平面並手動新增節點，您可以使用 Kubernetes 安裝工具下所列的 CLI 工具，例如 kind、minikube 或 kubeadm。為了簡化和自動化叢集建立和管理的完整生命週期，使用受既有 Kubernetes 提供商支援的工具 (例如 Amazon EKS 或 Amazon EKS Anywhere) 要容易得多。

在 AWS 雲端中，您可以使用 CLI 工具建立 Amazon EKS 叢集，例如 eksctl，或更多宣告工具，例如 Terraform（請參閱適用於 Terraform 的 Amazon EKS 藍圖)。您還可以從 AWS 管理主控台建立叢集。如需使用 Amazon EKS 可取得的功能的清單，請參閱 Amazon EKS 功能。Amazon EKS 為您承擔的 Kubernetes 責任包括：

- 受管控制平面 — AWS 確保 Amazon EKS 叢集可用且可擴展，因為它會為您管理控制平面，並讓它跨 AWS 可用區域可用。
- 節點管理：您無需手動新增節點，而是可以使用受管節點群組 (請參閱 the section called “受管節點群組”) 或 Karpenter，讓 Amazon EKS 視需要自動建立節點。受管節點群組可與 Kubernetes Cluster Autoscaling 整合。使用節點管理工具，您可以利用 Spot 執行個體和節點合併等功能來實現成本節省，同時還能使用排程功能來設定部署工作負載及選取節點的方式，進而確保工具可用性。
- 叢集聯網：使用 CloudFormation 範本，`eksctl` 可在 Kubernetes 叢集中設定控制平面和資料平面 (節點) 元件之間的聯網。它也設定了可進行內部和外部通訊的端點。如需詳細資訊，請參閱解密 Amazon EKS 工作節點的叢集聯網。Amazon EKS 中的 Pod 之間的通訊是使用 Amazon EKS Pod

身分 （請參閱 the section called “Pod 身分識別”) 完成，其提供讓 Pod 利用 AWS 雲端方法來管理登入資料和許可。

- 附加元件：利用 Amazon EKS，您不必建置和新增通常用於支援 Kubernetes 叢集的軟體元件。例如，當您從 建立 Amazon EKS 叢集時 AWS 管理主控台，它會自動新增 Amazon EKS kube-proxy (the section called “kube-proxy”)、適用於 Kubernetes 的 Amazon VPC CNI 外掛程式 (the section called “Amazon VPC CNI”) 和 CoreDNS (the section called “CoreDNS”) 附加元件。如需有關這些附加元件 (包括可用清單) 的詳細資訊，請參閱 the section called “Amazon EKS 附加元件”。

若要在您自己的內部部署電腦和網路上執行叢集，Amazon 提供了 Amazon EKS Anywhere。您可以選擇使用自己的設備在 VMware vSphere 上執行 Amazon EKS Anywhere、裸機 (Tinkerbell 提供者)、Snow、CloudStack 或 Nutanix 平台，而不是選擇雲端 AWS 作為提供者。

Amazon EKS Anywhere 是以 Amazon EKS 所使用的相同 Amazon EKS Distro 軟體為基礎。不過，Amazon EKS Anywhere 依賴 Kubernetes 叢集 API (CAPI) 介面的不同實作來管理 Amazon EKS Anywhere 叢集中機器的完整生命週期 (例如適用於 vSphere 的 CAPV 和適用於 CloudStack 的 CAPC)。由於整個叢集都會在您的設備上執行，因此您須承擔一些額外的責任，例如管理控制平面以及備份其資料 (請參閱本文件後文的 `etcd`)。

叢集元件

Kubernetes 叢集元件可分為兩個主要領域：控制平面和工作節點。控制平面元件負責管理叢集並提供其 API 的存取。工作節點 (有時又稱為節點) 則負載提供實際工作負載執行的位置。節點元件包含在每個節點上執行的服務，以與控制平面進行通訊並執行容器。叢集的工作節點集稱為資料平面。

控制平面

控制平面包含一組管理叢集的服務。這些服務可能會全都在單部電腦上執行，也可能分散在多部電腦上。在內部，我們將其稱為控制平面執行個體 (CPI)。執行 CPI 的方式取決於叢集的大小和高可用性的需求。隨著叢集的需求增加，控制平面服務可以擴展以提供該服務的更多執行個體，並在執行個體之間負載平衡請求。

Kubernetes 控制平面的元件執行的任務包括：

- 與叢集元件 (API 伺服器) 通訊：API 伺服器 (kube-apiserver) 會公開 Kubernetes API，因此可以從叢集內部和外部對叢集提出請求。換言之，新增或變更叢集物件 (Pod、服務、節點等) 的請求可能來自外部命令，例如來自 `kubectl` 的執行 Pod 的請求。同樣地，也可以從 API 伺服器向叢集內的元件提出請求，例如查詢 `kubelet` Pod 狀態的服務。
- 儲存有關叢集的資料 (**`etcd`** 金鑰值存放區)：`etcd` 服務可提供追蹤叢集目前狀態的關鍵角色。如果 `etcd` 服務變得無法存取，您將無法更新或查詢叢集的狀態，但工作負載會繼續執行一段時間。

因此，關鍵叢集通常會一次同時執行多個負載平衡的 `etcd` 服務的執行個體，並在資料遺失或損毀時定期備份 `etcd` 金鑰值存放區。請記住，在 Amazon EKS 中，預設會自動為您處理全部任務。Amazon EKS Anywhere 提供了有關 etcd 備份和還原 的說明。請參閱 etcd 資料模型，以了解 `etcd` 如何管理資料。

- 排程 Pod 到節點 (排程器)：在 Kubernetes 中啟動或停止 Pod 的請求會導向至 Kubernetes 排程器 (kube-scheduler)。由於叢集可能有多個能夠執行 Pod 的節點，因此由排程器選擇 Pod 應執行的節點 (如有複本，則為多個節點)。如果沒有足夠的可用容量來在現有節點上執行請求的 Pod，除非您已執行其他佈建，否則請求將會失敗。這些佈建可能包括啟用受管節點群組 (the section called “受管節點群組”) 或 Karpenter 等服務，其中這些服務可以自動啟動新節點，進而處理工作負載。
- 將元件保持在所需狀態 (控制器管理員)：Kubernetes 控制器管理員會以常駐程式程序 (kube-controller-manager) 執行，以觀測叢集的狀態並變更叢集，進而重新建立預期狀態。特別是，有數個控制器觀測不同的 Kubernetes 物件，其中包括 `statefulset-controller`、`endpoint-controller`、`cronjob-controller`、`node-controller`，等等。
- 管理雲端資源 (雲端控制器管理員)：執行基礎資料中心資源請求的 Kubernetes 與雲端提供者之間的互動由雲端控制器管理員 (cloud-controller-manager) 處理。由雲端控制器管理員管理的控制器可包括路由控制器 (用於設定雲端網路路由)、服務控制器 (用於使用雲端負載平衡服務) 和節點生命週期控制器 (用於在其整個生命週期中與 Kubernetes 保持節點同步)。

工作節點 (資料平面)

對於單一節點 Kubernetes 叢集，工作負載會在與控制平面相同的機器上執行。不過，較標準的組態是設有一或多個專用於執行 Kubernetes 工作負載的獨立電腦系統 (節點)。

您首次建立 Kubernetes 叢集時，某些叢集建立工具可讓您設定要新增至叢集的特定數量節點 (可透過識別現有的電腦系統，亦可透過讓提供商建立新的節點)。在將任何工作負載新增至這些系統之前，都會將相關服務新增至每個節點以實作這些功能：

- 管理每個節點 (**`kubelet`**)：API 伺服器會與每個節點上執行的 kubelet 服務進行通訊，以確保節點已正確註冊，並且排程器請求的 Pod 正在執行中。kubelet 可以讀取 Pod 資訊清單，並在本機系統上設定 Pod 所需的儲存磁碟區或其他功能。它也可以檢查本機執行中容器的運作狀態。
- 在節點上執行容器 (容器執行時期)：每個節點上的容器執行時期會管理指派給節點的每個 Pod 所請求的容器。這表示它可以從適當的登錄檔提取容器映像、執行容器、停止容器，以及回應有關容器的查詢。預設容器執行時期是 containerd。從 Kubernetes 1.24 起，可用作容器執行時期的 Docker (`dockershim`) 的特殊整合功能已從 Kubernetes 中刪除。雖然您仍然可以使用 Docker 在本機系統上測試和執行容器，但若要將 Docker 與 Kubernetes 搭配使用，您現在必須在每個節點上安裝 Docker 引擎，以便與 Kubernetes 搭配使用。

- 管理容器之間的聯網 (**kube-proxy**)：為了支援 Pod 之間的通訊，Kubernetes 會使用稱為服務的功能來設定 Pod 網路，進而追蹤與這些 Pod 相關聯的 IP 位址和連接埠。kube-proxy 服務會在每個節點上執行，以允許在 Pod 之間進行通訊。

擴展叢集

有些服務可以新增至 Kubernetes 以支援叢集，但這些服務無法在控制平面中執行。這些服務通常會直接在 kube-system 命名空間或其自有命名空間的節點上執行 (如同第三方服務提供商所做的那樣)。CoreDNS 服務就是其中一個常見範例，它可為叢集提供 DNS 服務。如需如何查看叢集上的 kube-system 中正在執行哪些叢集服務的資訊，請參閱探索內建服務。

您可以考慮將不同類型的附加元件新增至叢集。為了讓您的叢集保持運作狀態良好作，您可以新增可觀測性功能 (請參閱 監控叢集)，以便能夠執行記錄、稽核和指標等動作。利用此資訊，您通常可以透過相同的可觀測性介面對發生的問題進行故障診斷。這些類型的服務範例包括 Amazon GuardDuty、CloudWatch (請參閱 the section called “Amazon CloudWatch”)、AWS Ditro for OpenTelemetry、Kubernetes 專用 Amazon VPC CNI 外掛程式 (請參閱 the section called “Amazon VPC CNI”) 和 Grafana Kubernetes Monitoring。對於儲存 (請參閱 應用程式資料儲存)，Amazon EKS 的附加元件包括 Amazon Elastic Block Store CSI 驅動程式 (請參閱 the section called “Amazon EBS”)、Amazon Elastic File System CSI 驅動程式 (請參閱 the section called “Amazon EFS”)，以及數個第三方儲存附加元件，例如 Amazon FSx for NetApp ONTAP CSI 驅動程式 the section called “Amazon FSx for NetApp ONTAP”)。

如需可用的 Amazon EKS 附加元件的更完整清單，請參閱 the section called “Amazon EKS 附加元件”。

工作負載

Kubernetes 將工作負載定義為「在 Kubernetes 上執行的應用程式」。該應用程式可以由一組在 Pod 中作為容器執行的微服務組成，也可以作為批次任務或其他類型的應用程式執行。Kubernetes 的任務是確保您對要設定或部署的物件提出的請求已執行。如果有人部署應用程式，您應該了解容器的建置方式、Pod 的定義方式以及您可使用的部署方式。

容器

您在 Kubernetes 中部署和管理的應用程式工作負載最基本的元素是 Pod。Pod 代表一種保存應用程式元件的方式，同時也是定義描述 Pod 屬性的規格。與 RPM 或 Deb 套件等內容進行對比，該套件會將 Linux 系統的軟體封裝在一起，但本身不會以實體的形式執行。

由於 Pod 是最小的可部署單位，因此通常會保留單一容器。不過，在容器緊密耦合的情況下，多個容器可以在 Pod 中。例如，Web 伺服器容器可能封裝在 Pod 中，其附屬類型的容器則可能會提供與 Web 伺服器容器緊密連結的記錄、監控或其他服務。在此情況下，位於相同的 Pod 中即可確保對於每個執行中的 Pod 執行個體，兩個容器一律會在相同的節點上執行。同樣地，Pod 中的所有容器都會共用相同的環境，而 Pod 中的容器會像在相同的隔離主機中一樣執行。如此一來，容器會共用單一 IP 位址，提供 Pod 的存取權，而且容器可以彼此通訊，就像在自己的 localhost 上執行一樣。

Pod 規格 (PodSpec) 會定義 Pod 的所需狀態。您可以使用工作負載資源來管理 Pod 範本，以部署個別 Pod 或多個 Pod。工作負載資源包括部署 (管理多個 Pod 複本)、StatefulSets (部署需要是唯一的 Pod，例如資料庫 Pod) 和 DaemonSets (其中 Pod 需要在每個節點上持續執行)。稍後會詳細說明這些內容。

雖然 Pod 是您部署的最小單位，但容器是您建置和管理的最小單位。

建置容器

Pod 實際上只是一或多個容器周圍的結構，而每個容器本身都會保存檔案系統、可執行檔、組態檔案、程式庫和其他元件，以便實際執行應用程式。由於一家名為 Docker Inc. 的公司率先推廣普及了容器，因此有人將容器稱為 Docker 容器。不過，Open Container Initiative 此後已為業界定義了容器執行時期、映像和分發方法。除此之外，容器是基於許多現有的 Linux 功能建立的，因此其他人會將容器稱為 OCI 容器、Linux 容器或簡稱為容器。

當您建置容器時，通常會以 Dockerfile (原名) 開頭。在該 Dockerfile 中，您可以識別：

- 基礎映像：基礎容器映像就是一種容器，且其建置基礎通常是作業系統的檔案系統的最低版本 (例如 Red Hat Enterprise Linux 或 Ubuntu) 或經過增強以執行特定類型的應用程式 (例如 nodejs 或 python 應用程式) 的軟體的最小系統。
- 應用程式軟體：您可以將應用程式軟體新增至容器，其方式與將其新增至 Linux 系統的方式大致相同。例如，您可以在 Dockerfile 中執行 npm和 `yarn` 安裝 JavaScript 應用程式，或執行 `dnf` `yum`和安裝 RPM 套件。換言之，在 Dockerfile 中使用 RUN 命令，您可以執行基礎映像的檔案系統中可用的任何命令，進而在產生的容器映像內安裝軟體或設定軟體。
- 指示：Dockerfile 參考說明您在設定 Dockerfile 時可新增至 Dockerfile 的指示。這些包括用於建置容器本身 (本機系統中的 `ADD` 或 `COPY` 檔案) 的指示、識別容器執行時要執行的命令 (`CMD` 或 `ENTRYPOINT`)，以及將容器連接到其執行所在的系統 (透過識別要執行的 `USER`、要掛載的本機 `VOLUME` 或要 `EXPOSE` 的連接埠)。

雖然 docker 命令和服務歷來皆用於建置容器 (docker build)，但可用於建置容器映像的其他工具包括 Podman 和 nerdctl。請參閱建置更好的容器映像或 Docker Build 的概觀，以了解建置容器的相關資訊。

儲存容器

建置容器映像後，您可以將其儲存在工作站的容器分佈登錄檔或公有容器登錄檔。在工作站上執行私有容器登錄檔可讓您將容器映像以本機方式儲存，以便您隨時可用。

若要以更公開的方式儲存容器映像，您可以將它們推送至公有容器登錄檔。公有容器登錄檔提供集中位置，可用於儲存和分發容器映像。公有容器登錄檔的範例包括 Amazon Elastic Container Registry、Red Hat Quay 登錄檔和 Docker Hub 登錄檔。

在 Amazon Elastic Kubernetes Service (Amazon EKS) 上執行容器化工作負載時，我們建議提取儲存在 Amazon Elastic Container Registry 中的 Docker 官方映像複本。自 2021 以來，Amazon ECR 一直儲存著這些映像。您可以在 Amazon ECR 公共映像庫中搜尋熱門容器映像，特別是 Docker Hub 映像；另外，您也可以搜尋 Amazon ECR Docker Gallery。

執行容器

由於容器是以標準格式建置的，因此容器可以在可執行容器執行時期 (例如 Docker) 且其內容符合本機電腦架構 (例如 x86_64 或 arm) 的任何機器上執行。若要測試容器或在本機桌面上執行容器，您可以使用 docker run 或 podman run 命令，以在 localhost 上啟動容器。不過，對於 Kubernetes，每個工作節點都有部署的容器執行時期，並且可以由 Kubernetes 請求節點執行容器。

指派容器在節點上執行後，節點會查看請求的容器映像版本是否已存在於節點上。如果沒有，Kubernetes 會告知容器執行時期從適當的容器登錄檔中提取該容器，然後以本機方式執行該容器。請記住，容器映像是指在筆記型電腦、容器登錄檔和 Kubernetes 節點之間移動的軟體套件。容器是指該映像的執行中的執行個體。

Pod

容器準備就緒後，即可處理 Pod，包括設定、部署和讓 Pod 可存取。

設定 Pod

當您定義 Pod 時，您可為其指派一組屬性。這些屬性必須至少包含 Pod 名稱和要執行的容器映像。不過，您還需要使用 Pod 定義來設定許多其他項目 (有關哪些項目可以進入 Pod 的詳細資訊，請參閱 PodSpec 頁面)。其中包含：

- 儲存：當停止和刪除執行中的容器時，除非您設定更永久的儲存，否則該容器中的資料儲存會消失。Kubernetes 支援許多不同的儲存類型，並在磁碟區保護下對其進行抽象化。儲存類型包括

CephFS、NFS、iSCSI 等。您甚至可以從本機電腦使用本機區塊型儲存設備。透過叢集提供的其中一種儲存類型，您可以將儲存磁碟區掛載到容器的檔案系統中選取的掛載點。持久性磁碟區是在刪除 Pod 後會繼續存在的磁碟區，而暫時性磁碟區則會在刪除 Pod 時被刪除。如果您的叢集管理員為叢集建立不同的儲存類別，您可以選擇使用的儲存體屬性，例如使用後是否刪除或回收磁碟區、需要更多空間時是否會擴展，甚至是否符合特定的效能需求。

- 秘密 — 透過將秘密提供給 Pod 規格中的容器，您可以提供這些容器存取檔案系統、資料庫或其他受保護資產所需的許可。金鑰、密碼和字符是可儲存為秘密的項目之一。使用秘密，您即不必將此資訊儲存在容器映像中，而只需要讓秘密可供執行中的容器使用即可。類似於秘密是 ConfigMaps。`ConfigMap` 往往會保留較不重要的資訊，例如用於設定服務的鍵值對。
- 容器資源：用於進一步設定容器的物件可以採用資源組態的形式。對於每個容器，您可以請求可使用的記憶體容量和 CPU 數量，以及容器可使用的那些資源總數的限制。如需範例，請參閱 Pod 和容器的資源管理。
- 中斷：Pod 可能會非自願中斷 (節點故障) 或自願中斷 (需要升級)。透過設定 Pod 中斷預算，您可以對應用程式在中斷發生時保持的可用性施加一些控制。如需範例，請參閱為您的應用程式指定中斷預算。
- 命名空間：Kubernetes 會提供不同的方法來隔離 Kubernetes 元件和工作負載。在相同命名空間中執行特定應用程式的所有 Pod，是統一時保護和管理這些 Pod 的常見方式。您可以建立自己的命名空間來使用，或選擇不指示命名空間 (這會導致 Kubernetes 使用 `default` 命名空間)。Kubernetes 控制平面元件通常會在 kube-system 命名空間中執行。

剛才描述的組態通常會集中在一個 YAML 檔案中，以套用至 Kubernetes 叢集。對於個人 Kubernetes 叢集，您可能只會將這些 YAML 檔案儲存在您的本機系統上。不過，透過更關鍵的叢集和工作負載，GitOps 是自動化儲存和更新工作負載和 Kubernetes 基礎結構資源的熱門方式。

用來收集和部署 Pod 資訊的物件可由下列其中一種部署方式進行界定。

部署 Pod

您選擇部署 Pod 的方法取決於您打算搭配這些 Pod 執行的應用程式類型。以下是一些可供您選擇的方案：

- 無狀態應用程式：無狀態應用程式不會儲存用戶端的工作階段資料，因此另一個工作階段不需要參考先前工作階段發生的情況。如果 Pod 運作狀態不佳或四處移動而不儲存狀態，則您可更輕鬆地將 Pod 取代為新的 Pod。如果您執行的是無狀態應用程式（例如 Web 伺服器），您可以使用部署來部署 Pod 和 ReplicaSets。ReplicaSet 會定義您希望同時執行的 Pod 的執行個體數量。雖然您可以直接執行 ReplicaSet，但通常會直接在部署內執行複本，以定義一次應執行的 Pod 複本數量。

- 具狀態應用程式：在具狀態應用程式中，Pod 的身分和 Pod 啟動的順序很重要。這些應用程式需要穩定的持久性儲存，並且需要以一致方式進行部署和擴展。若要在 Kubernetes 中部署具狀態的應用程式，您可以使用 StatefulSets。資料庫就是一個典型的以 StatefulSet 執行的應用程式範例。在 StatefulSet 中，您可以定義複本、Pod 及其容器、要掛載的儲存磁碟區，以及儲存資料所在容器中的位置。如需部署為 StatefulSet 的資料庫範例，請參閱執行複寫狀態應用程式。
- 每個節點應用程式：有時您想要在 Kubernetes 叢集的每個節點上執行應用程式。例如，您的資料中心可能需要每部電腦執行監控應用程式或特定的遠端存取服務。對於 Kubernetes，您可以使用 DaemonSet 來確保選取的應用程式會在您叢集中的每個節點上執行。
- 應用程式執行到完成：您想為完成特定特務而執行的一些應用程式。這可能包括執行每月狀態報告或清除舊資料。任務物件可用來設定應用程式啟動並執行，然後在任務完成時結束。CronJob 物件可讓您使用 Linux crontab 格式定義的結構，將應用程式設定為在某個特定小時、分鐘、日期、月份或星期幾執行。

讓應用程式可從網路存取

隨著應用程式通常部署為一組移動到不同位置的微服務，Kubernetes 需要一種方法來讓這些微服務能夠互相尋找。此外，若要讓其他人存取 Kubernetes 叢集以外的應用程式，Kubernetes 需要一種方式，以在外部地址和連接埠上公開該應用程式。這些聯網相關功能分別由服務和輸入物件完成：

- 服務：由於 Pod 可以移動到不同的節點和地址，另一個需要與第一個 Pod 通訊的 Pod 可能會發現很難找到相關位置所在。為了解決此問題，Kubernetes 可讓您將應用程式表示為服務。使用服務時，您可以識別具有特定名稱的一個 Pod 或一組 Pod，然後指出哪個連接埠會公開該應用程式的 Pod 服務，以及另一個應用程式可以使用哪些連接埠來聯絡該服務。叢集中的另一個 Pod 可以直接依名稱請求服務，而 Kubernetes 會將該請求導向執行該服務的 Pod 執行個體的適當連接埠。
- 傳入：傳入可讓 Kubernetes Service 代表的應用程式可供叢集外部的用戶端使用。傳入的基本功能包括負載平衡器 (由傳入管理)、傳入控制器以及將請求從控制器路由到服務的規則。Kubernetes 中有數個傳入控制器可供您選擇。

後續步驟

了解 Kubernetes 的基本概念及其與 Amazon EKS 的關聯性，將協助您導覽 Amazon EKS 文件和 Kubernetes 文件，進而找到管理 Amazon EKS 叢集和將工作負載部署到這些叢集所需的資訊。若要開始使用 Amazon EKS，請選擇下列選項：

- the section called “建立叢集 (eksctl)”
- the section called “建立 叢集”

- the section called “範例部署 (Linux)”
- 叢集管理

在整個雲端和內部部署環境中部署 Amazon EKS 叢集

了解 Amazon EKS 部署選項

Amazon Elastic Kubernetes Service (Amazon EKS) 是全受管的 Kubernetes 服務，可讓您在雲端和內部部署環境中無縫執行 Kubernetes。

在雲端中，Amazon EKS 為 Kubernetes 控制平面與節點自動化 Kubernetes 叢集基礎結構管理。這對於排程容器、管理應用程式可用性、動態擴展資源、最佳化運算、儲存叢集資料以及執行其他關鍵功能至關重要。透過 Amazon EKS，您可以取得 AWS 基礎設施的強大效能、可擴展性、可靠性和可用性，以及與 AWS 聯網、安全性、儲存和可觀測性服務的原生整合。

為了簡化在內部部署環境中執行 Kubernetes 的流程，您可以使用相同的 Amazon EKS 叢集、功能與工具，在自有基礎結構上部署 EKS 節點至 the section called “節點” 或使用 Amazon EKS 混合節點，或者對於獨立氣隙環境，可以使用 Amazon EKS Anywhere。

雲端 Amazon EKS

您可以在 AWS 區域、 AWS 本機區域和 Wavelength 區域中搭配運算使用 AWS Amazon EKS。在雲端中使用 Amazon EKS，Kubernetes 控制平面的安全性、可擴展性和可用性由 AWS 區域中的 完全管理 AWS 。在 AWS 區域中使用運算執行應用程式時，您可以取得 AWS 和 Amazon EKS 功能的完整廣度，包括 Amazon EKS Auto Mode， AWS 只要按一下，即可完全自動化 上的運算、儲存和聯網的 Kubernetes 叢集基礎設施管理。在 AWS Local Zones 和 AWS Wavelength Zones 中使用運算執行應用程式時，您可以使用 Amazon EKS 自我管理節點來連接叢集運算的 Amazon EC2 執行個體，並且可以使用 AWS Local Zones 和 AWS Wavelength Zones 中的其他可用 AWS 服務。如需詳細資訊，請參閱 AWS 本機區域功能與 AWS Wavelength 區域功能。

	AWS 區域中的 Amazon EKS	本機/Wavelength 區域中的 Amazon EKS
Kubernetes 控制平面管理	AWS受管	AWS受管
Kubernetes 控制平面位置	AWS 區域	AWS 區域

	AWS 區域中的 Amazon EKS	本機/Wavelength 區域中的 Amazon EKS
Kubernetes 資料平面	• Amazon EKS 自動模式 • Amazon EKS 受管節點群組 • Amazon EC2 自我管理節點 • AWS Fargate	• Amazon EKS 受管節點群組 (僅限本機區域) • Amazon EC2 自我管理節點
Kubernetes 資料平面位置	AWS 區域	AWS 本機或 Wavelength 區域

您的資料中心或邊緣環境中的 Amazon EKS

若您需要在自己的資料中心或邊緣環境中執行應用程式，可使用 AWS Outposts 上的 Amazon EKS 或 Amazon EKS 混合節點。您可以將自我管理節點與 AWS Outposts 上的 Amazon EC2 執行個體搭配使用，以進行叢集運算，也可以將 Amazon EKS 混合節點與您自己的現場部署或邊緣基礎設施搭配使用，以進行叢集運算。 AWS Outpost 是在資料中心或主機代管設施中執行的 AWS受管基礎設施，而 Amazon EKS 混合節點會在您在現場部署或邊緣環境中管理的實體或虛擬機器上執行。Amazon EKS on AWS Outposts 和 Amazon EKS 混合節點需要從現場部署環境到 AWS 區域的可靠連線，而且您可以使用與雲端中執行應用程式相同的 Amazon EKS 叢集、功能和工具。在 AWS Outposts 上執行時，您也可以在 AWS Outposts 上使用 Outposts 上的 Amazon EKS 本機叢集部署整個 AWS Kubernetes 叢集。

	Amazon EKS 混合節點	AWS Outposts 上的 Amazon EKS
Kubernetes 控制平面管理	AWS受管	AWS受管
Kubernetes 控制平面位置	AWS 區域	AWS 區域或 AWS Outpost
Kubernetes 資料平面	客戶管理的實體或虛擬機器	Amazon EC2 自我管理節點
Kubernetes 資料平面位置	客戶資料中心或邊緣環境	客戶資料中心或邊緣環境

適用於氣隙環境的 Amazon EKS Anywhere

Amazon EKS Anywhere 透過自動化基礎結構設定與 Kubernetes 叢集生命週期操作等繁重工作，簡化了在內部部署和邊緣環境中的 Kubernetes 叢集管理。與 Amazon EKS 不同，Amazon EKS Anywhere 是一項客戶管理的產品，客戶需負責 Amazon EKS Anywhere 叢集的叢集生命週期操作與維護。Amazon EKS Anywhere 建置在 Kubernetes 子專案叢集 API (CAPI) 上，並支援各種基礎設施，包括 VMware vSphere、裸機、Nutanix、Apache CloudStack 和 AWS Snow。Amazon EKS Anywhere 可以在氣隙隔離的環境中執行，並提供與區域 AWS 服務的選用整合，以實現可觀測性和身分管理。若要取得 Amazon EKS Anywhere 的支援和對 AWS Kubernetes 附加元件的存取，您可以購買 Amazon EKS Anywhere Enterprise Subscriptions。

	Amazon EKS Anywhere
Kubernetes 控制平面管理	客戶自管
Kubernetes 控制平面位置	客戶資料中心或邊緣環境
Kubernetes 資料平面	客戶管理的實體或虛擬機器
Kubernetes 資料平面位置	客戶資料中心或邊緣環境

Amazon EKS 工具

您可以使用 Amazon EKS 連接器來註冊任何符合標準的 Kubernetes 叢集，並將其連接到 ， AWS 並在 Amazon EKS 主控台中檢視。將叢集連線後，您可以在 Amazon EKS 主控台中查看叢集的狀態、組態和工作負載。您可以使用此功能在 Amazon EKS 主控台中檢視已連接的叢集，但 Amazon EKS Connector 不允許透過 Amazon EKS 主控台對已連接的叢集進行管理或變更操作。

Amazon EKS Distro 是支援所有 Amazon EKS 產品的基礎 Kubernetes 元件的 AWS 分佈。它包含運作中 Kubernetes 叢集所需的核心元件，例如 Kubernetes 控制平面元件 (etcd、kube-apiserver、kube-scheduler、kube-controller-manager) 和聯網元件 (CoreDNS、kube-proxy、CNI 外掛程式)。Amazon EKS Distro 可與您選擇的工具搭配使用，以自行管理 Kubernetes 叢集。 AWS Support Plans 不涵蓋 Amazon EKS Distro 部署。

設定以使用 Amazon EKS

如需為 Amazon EKS 叢集的命令列管理做準備，需要安裝多種工具。藉助以下各項來設定憑證，建立及修改叢集，以及在叢集執行後使用叢集：

- 設定 AWS CLI – 取得 AWS CLI 來設定和管理使用 Amazon EKS 叢集所需的服務。特別是，您需要 AWS CLI 來設定登入資料，但您也需要使用其他 AWS 服務。
- 設定 kubectl 和 eksctl – CLI 會與 `eksctl` 互動， AWS 以建立、修改和刪除 Amazon EKS 叢集。一旦叢集啟動，則使用開放原始碼 `kubectl` 命令，來管理 Amazon EKS 叢集內的 Kubernetes 物件。
- 設定開發環境 （選用） – 考慮新增下列工具：
  - 本機部署工具 – 若您不熟悉 Kubernetes，則考慮安裝 minikube 或 kind 等本機部署工具。這些工具可讓您在本機電腦上，使用 Amazon EKS 叢集來測試應用程式。
  - 套件管理員 – Helm 是 Kubernetes 的熱門套件管理員，可簡化複雜套件的安裝和管理。透過 Helm，您可以更輕鬆地在 Amazon EKS 叢集上安裝和管理套件，例如 AWS Load Balancer控制器。

後續步驟

- 設定 AWS CLI
- 設定 kubectl 和 eksctl
- 快速入門：部署 Web 應用程式並存放資料

設定 AWS CLI

AWS CLI 是一種命令列工具，可用於 AWS 服務，包括 Amazon EKS。它也用於驗證 IAM 使用者或角色，以便從本機電腦存取 Amazon EKS 叢集和其他 AWS 資源。若要 AWS 從命令列在 中佈建資源，您需要取得 AWS 存取金鑰 ID 和私密金鑰，以便在命令列中使用。然後，您需要在 CLI AWS 中設定這些登入資料。如果您尚未安裝 AWS CLI，請參閱《 AWS 命令列界面使用者指南》中的安裝或更新最新版本的 AWS CLI。

建立存取金鑰

1. 登入 AWS 管理主控台。
2. 對於單一使用者或多個使用者帳戶：

- 單一使用者帳戶 – : : 在右上角，選擇您的 AWS 使用者名稱以開啟導覽功能表。例如，選擇 **webadmin** 。
- 多個使用者帳戶：從服務清單中選擇 IAM。從 IAM 儀表板中，選取使用者，然後選擇使用者的名稱。

3. 選擇 Security credentials (安全登入資料)。
4. 在存取金鑰之下選擇建立存取金鑰。
5. 選擇命令列介面 (CLI)，然後選擇下一步。
6. 選擇 Create access key (建立新的存取金鑰)。
7. 選擇下載 .csv 檔案。

設定 AWS CLI

安裝 CLI AWS 之後，請執行下列步驟進行設定。如需詳細資訊，請參閱《 AWS 命令列界面使用者指南》中的設定 AWS CLI。

1. 在終端機視窗中，輸入以下命令：

```
aws configure
```

   您也可以選擇設定具名設定檔，例如 `--profile cluster-admin`。如果您在 CLI AWS 中設定具名設定檔，則一律必須在後續命令中傳遞此旗標。

2. 輸入您的 AWS 登入資料。例如：

```
Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Default region name [None]: region-code
Default output format [None]: json
```

取得安全字符

如有需要，請執行下列命令以取得 CLI AWS 的新安全字符。如需詳細資訊，請參閱 AWS CLI 命令參考中的 get-session-token。

依預設，該字符的有效期為 15 分鐘。若要變更預設工作階段逾時，請傳遞 `--duration-seconds` 旗標。例如：

```
aws sts get-session-token --duration-seconds 3600
```

此命令會傳回 CLI AWS 工作階段的臨時安全登入資料。您應該會看到下列回應輸出：

```
{
    "Credentials": {
        "AccessKeyId": "ASIA5FTRU3LOEXAMPLE",
        "SecretAccessKey": "JnKgvwfqUD9mNsPoi9IbxAYEXAMPLE",
        "SessionToken": "VERYLONGSESSIONTOKENSTRING",
        "Expiration": "2023-02-17T03:14:24+00:00"
    }
}
```

若要驗證使用者身分

如有需要，請執行下列命令來驗證終端機工作階段的 IAM 使用者身分 AWS 憑證 （ 例如 *ClusterAdmin*)。

```
aws sts get-caller-identity
```

此命令會傳回為 CLI 設定的 IAM 實體的 Amazon Resource Name AWS (ARN)。您應該會看到類似以下內容的回應輸出：

```
{
    "UserId": "AKIAIOSFODNN7EXAMPLE",
    "Account": "01234567890",
    "Arn": "arn:aws: iam::01234567890:user/ClusterAdmin"
}
```

後續步驟

- 設定 kubectl 和 eksctl
- 快速入門：部署 Web 應用程式並存放資料

設定 kubectl 和 eksctl

> **Tip**
>
> 註冊即將舉行的 Amazon EKS 研討會。

安裝 CLI AWS 後，您應該安裝其他兩個工具來建立和管理 Kubernetes 叢集：

- kubectl：kubectl 命令列工具是您用來管理 Kubernetes 叢集內資源的主要工具。此頁面說明如何下載和設定與 Kubernetes 叢集版本相符的 kubectl 二進位檔。請參閱安裝或更新 kubectl。
- eksctl：eksctl命令列工具用於在 AWS 雲端或內部部署 （使用 EKS Anywhere) 中建立 EKS 叢集，以及修改和刪除這些叢集。請參閱安裝 eksctl。

安裝或更新 kubectl

本主題將協助您在裝置上下載並安裝或更新 kubectl 二進位檔案。二進位檔案與上游社群版本相同。二進位對 Amazon EKS 或 來說不是唯一的 AWS。使用下列步驟來取得您需要的 kubectl 的特定版本，雖然許多建置器只需執行 brew install kubectl 即可安裝。

> **Note**
>
> 您所使用的 kubectl 版本，必須與 Amazon EKS 叢集控制平面的版本差距在一個版本以內。例如，1.35 kubectl 用戶端搭配 Kubernetes 1.34、1.35 和 1.36 叢集運作。

> **Tip**
>
> 您也可以直接從 EKS kubectl 主控台透過 AWS CloudShell 使用 。在任何叢集詳細資訊頁面上選擇連線，以取得預先設定的工作階段。如需詳細資訊，請參閱the section called “透過 kubectl 存取叢集”。

步驟 1：檢查是否已安裝 kubectl

判斷您是否已在裝置上安裝 kubectl。

```
kubectl version --client
```

若您已在裝置路徑中安裝 kubectl，則範例輸出包含下列類似資訊。若您要更新目前使用較新版本安裝的版本，請完成下一個步驟，確定將新版本安裝在目前版本所處的位置。

```
Client Version: v1.31.X-eks-1234567
```

若您未收到輸出，則表示您尚未安裝 kubectl，或其未安裝在裝置路徑中的位置。

步驟 2：安裝或更新 kubectl

在下列其中一個作業系統上安裝或更新 kubectl：

- the section called “macOS”
- the section called “Linux (amd64)”
- the section called “Linux (arm64)”
- the section called “Windows”

> **Note**
>
> 如果從本節使用的 AWS 區域下載到 AWS 區域的速度緩慢，請考慮設定 CloudFront 來前置內容。如需詳細資訊，請參閱開始使用基本 CloudFront 分佈。

macOS

請遵循下列步驟以在 macOS 上安裝 kubectl。步驟包括：

- 選擇並下載您想要的 Kubernetes 版本的二進位檔。
- 選擇性地檢查二進位檔的檢查總和。
- 將執行套用至二進位檔的許可。
- 將二進位檔複製到 PATH 中的資料夾。
- 選擇性地將二進位檔的目錄新增至 PATH。

程序：

1. 從 Amazon S3 下載適用於您叢集 Kubernetes 版本的二進位檔。
   - Kubernetes 1.36

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.36.2/2026-07-05/bin/darwin/amd64/kubectl
     ```

   - Kubernetes 1.35

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.35.6/2026-07-05/bin/darwin/amd64/kubectl
     ```

   - Kubernetes 1.34

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.34.9/2026-07-05/bin/darwin/amd64/kubectl
     ```

   - Kubernetes 1.33

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.33.13/2026-07-05/bin/darwin/amd64/kubectl
     ```

   - Kubernetes 1.32

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.13/2026-07-05/bin/darwin/amd64/kubectl
     ```

   - Kubernetes 1.31

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.14/2026-07-05/bin/darwin/amd64/kubectl
     ```

   - Kubernetes 1.30

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.14/2026-07-05/bin/darwin/amd64/kubectl
     ```

2. (選用) 使用您的二進位檔案 SHA-256 檢查總和，驗證下載的二進位檔案。
   a. 下載適用於您叢集 Kubernetes 版本的 SHA-256 檢查總和。
      - Kubernetes 1.36

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.36.2/2026-07-05/bin/
darwin/amd64/kubectl.sha256
```

- Kubernetes `1.35`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.35.6/2026-07-05/bin/
darwin/amd64/kubectl.sha256
```

- Kubernetes `1.34`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.34.9/2026-07-05/bin/
darwin/amd64/kubectl.sha256
```

- Kubernetes `1.33`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.33.13/2026-07-05/bin/
darwin/amd64/kubectl.sha256
```

- Kubernetes `1.32`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.13/2026-07-05/bin/
darwin/amd64/kubectl.sha256
```

- Kubernetes `1.31`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.14/2026-07-05/bin/
darwin/amd64/kubectl.sha256
```

- Kubernetes `1.30`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.14/2026-07-05/bin/
darwin/amd64/kubectl.sha256
```

b. 檢查適用您下載之二進位檔案的 SHA-256 檢查總和。

```
openssl sha1 -sha256 kubectl
```

c. 確定輸出中產生的檢查總和符合下載`kubectl.sha256`檔案中的檢查總和。

3. 將執行許可套用至二進位檔。

```
chmod +x ./kubectl
```

4. 將二進位檔複製到 PATH 中的資料夾。如果您已安裝某一版本的 kubectl，建議您建立 $HOME/bin/kubectl 並確認您的 $PATH 中會先出現 $HOME/bin。

```
mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$HOME/bin:$PATH
```

5. (選用) 將 $HOME/bin 路徑新增到 Shell 初始化檔案，因此當您開啟 shell 時，該組態已設定完畢。

```
echo 'export PATH=$HOME/bin:$PATH' >> ~/.bash_profile
```

Linux (amd64)

請遵循下列步驟以在 Linux 上安裝 kubectl (amd64)。步驟包括：

- 選擇並下載您想要的 Kubernetes 版本的二進位檔。
- 選擇性地檢查二進位檔的檢查總和。
- 將執行套用至二進位檔的許可。
- 將二進位檔複製到 PATH 中的資料夾。
- 選擇性地將二進位檔的目錄新增至 PATH。

程序：

1. 從 Amazon S3 下載適用於您叢集 Kubernetes 版本的 kubectl 二進位檔。
   - Kubernetes 1.36

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.36.2/2026-07-05/bin/linux/amd64/kubectl
```

   - Kubernetes 1.35

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.35.6/2026-07-05/bin/linux/amd64/kubectl
```

   - Kubernetes 1.34

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.34.9/2026-07-05/bin/linux/amd64/kubectl
```

- Kubernetes `1.33`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.33.13/2026-07-05/bin/linux/amd64/kubectl
```

- Kubernetes `1.32`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.13/2026-07-05/bin/linux/amd64/kubectl
```

- Kubernetes `1.31`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.14/2026-07-05/bin/linux/amd64/kubectl
```

- Kubernetes `1.30`

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.14/2026-07-05/bin/linux/amd64/kubectl
```

2. (選用) 使用您的二進位檔案 SHA-256 檢查總和，驗證下載的二進位檔案。

   a. 使用裝置硬體平台的 命令，從 Amazon S3 下載叢集 Kubernetes 版本的SHA-256檢查總和。

   - Kubernetes `1.36`

   ```
   curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.36.2/2026-07-05/bin/linux/amd64/kubectl.sha256
   ```

   - Kubernetes `1.35`

   ```
   curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.35.6/2026-07-05/bin/linux/amd64/kubectl.sha256
   ```

   - Kubernetes `1.34`

   ```
   curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.34.9/2026-07-05/bin/linux/amd64/kubectl.sha256
   ```

- Kubernetes 1.33

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.33.13/2026-07-05/bin/linux/amd64/kubectl.sha256
```

- Kubernetes 1.32

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.13/2026-07-05/bin/linux/amd64/kubectl.sha256
```

- Kubernetes 1.31

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.14/2026-07-05/bin/linux/amd64/kubectl.sha256
```

- Kubernetes 1.30

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.14/2026-07-05/bin/linux/amd64/kubectl.sha256
```

b. 執行下列其中一個命令，檢查您下載的二進製檔案的 SHA-256 檢查總和。

```
sha256sum -c kubectl.sha256
```

或

```
openssl sha1 -sha256 kubectl
```

c. 首先，您應該會看到 kubectl: OK，第二個是，您可以檢查輸出中產生的檢查總和是否符合下載kubectl.sha256檔案中的檢查總和。

3. 將執行許可套用至二進位檔。

```
chmod +x ./kubectl
```

4. 將二進位檔複製到 PATH 中的資料夾。如果您已安裝某一版本的 kubectl，建議您建立 $HOME/bin/kubectl 並確認您的 $PATH 中會先出現 $HOME/bin。

```
mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$HOME/bin:$PATH
```

5. (選用) 將 $HOME/bin 路徑新增到 Shell 初始化檔案，因此當您開啟 shell 時，該組態已設定完畢。

> **Note**
>
> 本步驟假設您是使用 Bash Shell；若您使用其他 Shell，請將命令更改為使用具體的 Shell 初始化檔案。

```
echo 'export PATH=$HOME/bin:$PATH' >> ~/.bashrc
```

Linux (arm64)

請遵循下列步驟以在 Linux 上安裝 `kubectl` (arm64)。步驟包括：

- 選擇並下載您想要的 Kubernetes 版本的二進位檔。
- 選擇性地檢查二進位檔的檢查總和。
- 將執行套用至二進位檔的許可。
- 將二進位檔複製到 PATH 中的資料夾。
- 選擇性地將二進位檔的目錄新增至 PATH。

程序：

1. 從 Amazon S3 下載適用於您叢集 Kubernetes 版本的 `kubectl` 二進位檔。
   - Kubernetes `1.36`

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.36.2/2026-07-05/bin/linux/arm64/kubectl
     ```

   - Kubernetes `1.35`

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.35.6/2026-07-05/bin/linux/arm64/kubectl
     ```

   - Kubernetes `1.34`

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.34.9/2026-07-05/bin/linux/arm64/kubectl
     ```

- Kubernetes 1.33

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.33.13/2026-07-05/bin/linux/arm64/kubectl
```

- Kubernetes 1.32

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.13/2026-07-05/bin/linux/arm64/kubectl
```

- Kubernetes 1.31

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.14/2026-07-05/bin/linux/arm64/kubectl
```

- Kubernetes 1.30

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.14/2026-07-05/bin/linux/arm64/kubectl
```

2. (選用) 使用您的二進位檔案 SHA-256 檢查總和，驗證下載的二進位檔案。

   a. 使用裝置硬體平台的 命令，從 Amazon S3 下載叢集 Kubernetes 版本的SHA-256檢查總和。

   - Kubernetes 1.36

   ```
   curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.36.2/2026-07-05/bin/linux/arm64/kubectl.sha256
   ```

   - Kubernetes 1.35

   ```
   curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.35.6/2026-07-05/bin/linux/arm64/kubectl.sha256
   ```

   - Kubernetes 1.34

   ```
   curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.34.9/2026-07-05/bin/linux/arm64/kubectl.sha256
   ```

   - Kubernetes 1.33

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.33.13/2026-07-05/bin/
linux/arm64/kubectl.sha256
```

- Kubernetes 1.32

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.13/2026-07-05/bin/
linux/arm64/kubectl.sha256
```

- Kubernetes 1.31

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.14/2026-07-05/bin/
linux/arm64/kubectl.sha256
```

- Kubernetes 1.30

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.14/2026-07-05/bin/
linux/arm64/kubectl.sha256
```

b. 執行下列其中一個命令，檢查您下載的二進製檔案的 SHA-256 檢查總和。

```
sha256sum -c kubectl.sha256
```

或

```
openssl sha1 -sha256 kubectl
```

c. 首先，您應該會看到 kubectl: OK，第二個是，您可以檢查輸出中產生的檢查總和是否符合下載kubectl.sha256檔案中的檢查總和。

3. 將執行許可套用至二進位檔。

```
chmod +x ./kubectl
```

4. 將二進位檔複製到 PATH 中的資料夾。如果您已安裝某一版本的 kubectl，建議您建立 $HOME/bin/kubectl 並確認您的 $PATH 中會先出現 $HOME/bin。

```
mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$HOME/bin:$PATH
```

5. (選用) 將 $HOME/bin 路徑新增到 Shell 初始化檔案，因此當您開啟 shell 時，該組態已設定完畢。

> **Note**
>
> 本步驟假設您是使用 Bash Shell；若您使用其他 Shell，請將命令更改為使用具體的 Shell 初始化檔案。

```
echo 'export PATH=$HOME/bin:$PATH' >> ~/.bashrc
```

Windows

請遵循下列步驟以在 Windows 上安裝 `kubectl`。步驟包括：

- 選擇並下載您想要的 Kubernetes 版本的二進位檔。
- 選擇性地檢查二進位檔的檢查總和。
- 將二進位檔複製到 PATH 中的資料夾。
- 選擇性地將二進位檔的目錄新增至 PATH。

程序：

1. 開啟 PowerShell 終端機。
2. 從 Amazon S3 下載適用於您叢集 Kubernetes 版本的 `kubectl` 二進位檔。
   - Kubernetes `1.36`

     ```
     curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.36.2/2026-07-05/bin/windows/amd64/kubectl.exe
     ```

   - Kubernetes `1.35`

     ```
     curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.35.6/2026-07-05/bin/windows/amd64/kubectl.exe
     ```

   - Kubernetes `1.34`

     ```
     curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.34.9/2026-07-05/bin/windows/amd64/kubectl.exe
     ```

- Kubernetes 1.33

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.33.13/2026-07-05/bin/windows/amd64/kubectl.exe
```

- Kubernetes 1.32

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.13/2026-07-05/bin/windows/amd64/kubectl.exe
```

- Kubernetes 1.31

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.14/2026-07-05/bin/windows/amd64/kubectl.exe
```

- Kubernetes 1.30

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.14/2026-07-05/bin/windows/amd64/kubectl.exe
```

3. (選用) 使用您的二進位檔案 SHA-256 檢查總和，驗證下載的二進位檔案。

   a. 對於 Windows，下載適用於您叢集 Kubernetes 版本的 SHA-256 檢查總和。

   - Kubernetes 1.36

   ```
   curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.36.2/2026-07-05/bin/windows/amd64/kubectl.exe.sha256
   ```

   - Kubernetes 1.35

   ```
   curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.35.6/2026-07-05/bin/windows/amd64/kubectl.exe.sha256
   ```

   - Kubernetes 1.34

   ```
   curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.34.9/2026-07-05/bin/windows/amd64/kubectl.exe.sha256
   ```

   - Kubernetes 1.33

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.33.13/2026-07-05/
bin/windows/amd64/kubectl.exe.sha256
```

- Kubernetes `1.32`

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.13/2026-07-05/
bin/windows/amd64/kubectl.exe.sha256
```

- Kubernetes `1.31`

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.14/2026-07-05/
bin/windows/amd64/kubectl.exe.sha256
```

- Kubernetes `1.30`

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.14/2026-07-05/
bin/windows/amd64/kubectl.exe.sha256
```

b. 檢查適用您下載之二進位檔案的 SHA-256 檢查總和。

```
Get-FileHash kubectl.exe
```

c. 確定輸出中產生的檢查總和符合下載`kubectl.exe.sha256`檔案中的檢查總和。PowerShell 輸出應該是大寫的等效字元字串。

4. 將二進位檔複製到 PATH 中的資料夾。若在 PATH 中已有命令列公用程式專用的現存目錄，請將二進位檔案複製至該目錄。否則，請完成下列步驟。

   a. 建立新目錄以供存放命令列二進位檔，例如 `C:\bin`。

   b. 將 `kubectl.exe` 二進位檔複製到該新目錄。

   c. 編輯您的使用者或系統 PATH 環境變數，將新目錄新增至 PATH。

   d. 關閉 PowerShell 終端，再重新開啟以挑選新的 PATH 變數。

5. 安裝 `kubectl` 後，您可以驗證其版本。

```
kubectl version --client
```

6. 第一次安裝 `kubectl` 時，它並未設定可與任何伺服器通訊。我們會視需要在其他程序中涵蓋此組態。如果您需要更新組態來與特定叢集進行通訊，那麼您可以執行下列命令。將 *region-code* 取代為您的叢集所在的 AWS 區域。使用您叢集的名稱取代 *my-cluster*。

```
aws eks update-kubeconfig --region region-code --name my-cluster
```

7. 請考慮設定自動完成，這樣一來，在輸入前幾個字母後，您即可使用 Tab 鍵來完成 kubectl 子命令。如需詳細資訊，請參閱 Kubernetes 文件中的 Kubectl 自動完成。

安裝 eksctl

eksctl CLI 可用於與 EKS 叢集搭配使用。它會自動化許多個別任務。如需有關安裝 eksctl 的指示，請參閱 eksctl 文件中的安裝。對於 Linux，請使用 UNIX 指示。

使用 時eksctl，您使用的 IAM 安全主體必須具有使用 Amazon EKS IAM 角色、服務連結角色、AWS CloudFormation、VPC 和相關資源的許可。如需詳細資訊，請參閱《IAM 使用者指南》中的動作和使用服務連結角色。您必須以同一位使用者的身分完成本指南中的所有步驟。若要檢查目前使用者，請執行以下命令：

```
aws sts get-caller-identity
```

後續步驟

- 快速入門：部署 Web 應用程式並存放資料

快速入門：部署 Web 應用程式並存放資料

本快速入門教學課程會逐步引導您部署 2048 遊戲範例應用程式，並使用 eksctl 將其資料保留在 Amazon EKS Auto Mode 叢集上。

Amazon EKS Auto Mode 透過自動化區塊儲存、聯網、負載平衡和運算自動擴展等例行任務，簡化叢集管理。在設定期間，它會處理使用 EC2 受管執行個體、應用程式負載平衡器和 EBS 磁碟區的建立節點。

總而言之，您將使用與服務無縫整合所需的自訂註釋來部署範例工作負載 AWS 。

於本教學課程中

您將藉助隨附的 `eksctl` 叢集範本，透過 EKS 自動模式來建置叢集，以便自動執行節點佈建。

- VPC 組態：使用後續的 eksctl 叢集範本時，eksctl 會自動為叢集建立 IPv4 Virtual Private Cloud (VPC)。依預設，除了建立公有與私有端點，eksctl 還會設定可滿足所有聯網需求的 VPC。
- 執行個體管理：EKS Auto Mode 會根據 Kubernetes 應用程式的需求，動態新增或移除 EKS 叢集中的節點。
- 資料持久性：使用 EKS Auto Mode 的區塊儲存功能來確保應用程式資料的持久性，即使在涉及 Pod 重新啟動或故障的情況下也是如此。
- 外部應用程式存取：使用 EKS Auto Mode 的負載平衡功能動態佈建 Application Load Balancer (ALB)。

先決條件

開始之前，請確定您已執行下列任務：

- 為 Amazon EKS 設定您的環境
- 安裝 eksctl 的最新版本

設定叢集

在這個部分，您將藉助 EKS 自動模式來建立叢集，以便動態佈建節點。

建立一個 cluster-config.yaml 檔案並將以下內容貼入其中。region-code 將 取代為有效的區域 （ 例如 us-east-1)。

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: web-quickstart
  region: <region-code>

autoModeConfig:
  enabled: true
```

現在，我們可隨時建立叢集。

使用 建立 EKS 叢集cluster-config.yaml：

```
eksctl create cluster -f cluster-config.yaml
```

> **Important**
>
> 若不使用 eksctl 建立叢集，您需手動標記 VPC 子網路。

建立 IngressClass

針對 EKS 自動模式建立 IngressClass Kubernetes。IngressClass 定義 EKS 自動模式處理傳入資源的方式。此步驟設定 EKS 自動模式的負載平衡功能。當您為應用程式建立輸入資源時，EKS Auto Mode 會使用此 IngressClass 來自動佈建和管理負載平衡器，並將 Kubernetes 應用程式與 AWS 負載平衡服務整合。

以 ingressclass.yaml 儲存以下 yaml 檔案：

```
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  name: alb
  annotations:
    ingressclass.kubernetes.io/is-default-class: "true"
```

```
spec:
  controller: eks.amazonaws.com/alb
```

套用 IngressClass 至您的叢集：

```
kubectl apply -f ingressclass.yaml
```

部署 2048 遊戲範例應用程式

在這個部分，我們將逐步引導您將熱門的「2048 遊戲」作為叢集內的範例應用程式進行部署。提供的資訊清單包括 Application Load Balancer (ALB) 的自訂註解。這些註釋與 整合，並指示 EKS 將傳入的 HTTP 流量視為「面向網際網路」，並使用目標類型「ip」將其路由到game-2048命名空間中的適當服務。

> **Note**
>
> 範例中的 docker-2048 映像是 x86_64 容器映像，並且不會在其他架構執行。

1. 透過 --save-config 標誌建立名稱為 game-2048 的 Kubernetes 命名空間。

   ```
   kubectl create namespace game-2048 --save-config
   ```

   您應該會看到下列回應輸出：

   ```
   namespace/game-2048 created
   ```

2. 部署 2048 遊戲範例應用程式。

   ```
   kubectl apply -n game-2048 -f https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.8.0/docs/examples/2048/2048_full.yaml
   ```

   此資訊清單針對 game-2048 命名空間，設定了 Kubernetes 部署、服務與傳入，從而建立必要的資源，以便在叢集內部署及公開 game-2048 應用程式。其包括建立一項名稱為 service-2048 的服務，該服務在連接埠 80 上公開部署；還會建立一個名稱為 ingress-2048 的傳入資源，該資源定義傳入 HTTP 流量的路由規則，以及面向網際網路的 Application Load Balancer (ALB) 的註解。您應該會看到下列回應輸出：

```
namespace/game-2048 configured
deployment.apps/deployment-2048 created
service/service-2048 created
ingress.networking.k8s.io/ingress-2048 created
```

3. 執行以下命令，獲取 game-2048 命名空間的傳入資源。

```
kubectl get ingress -n game-2048
```

您應該會看到下列回應輸出：

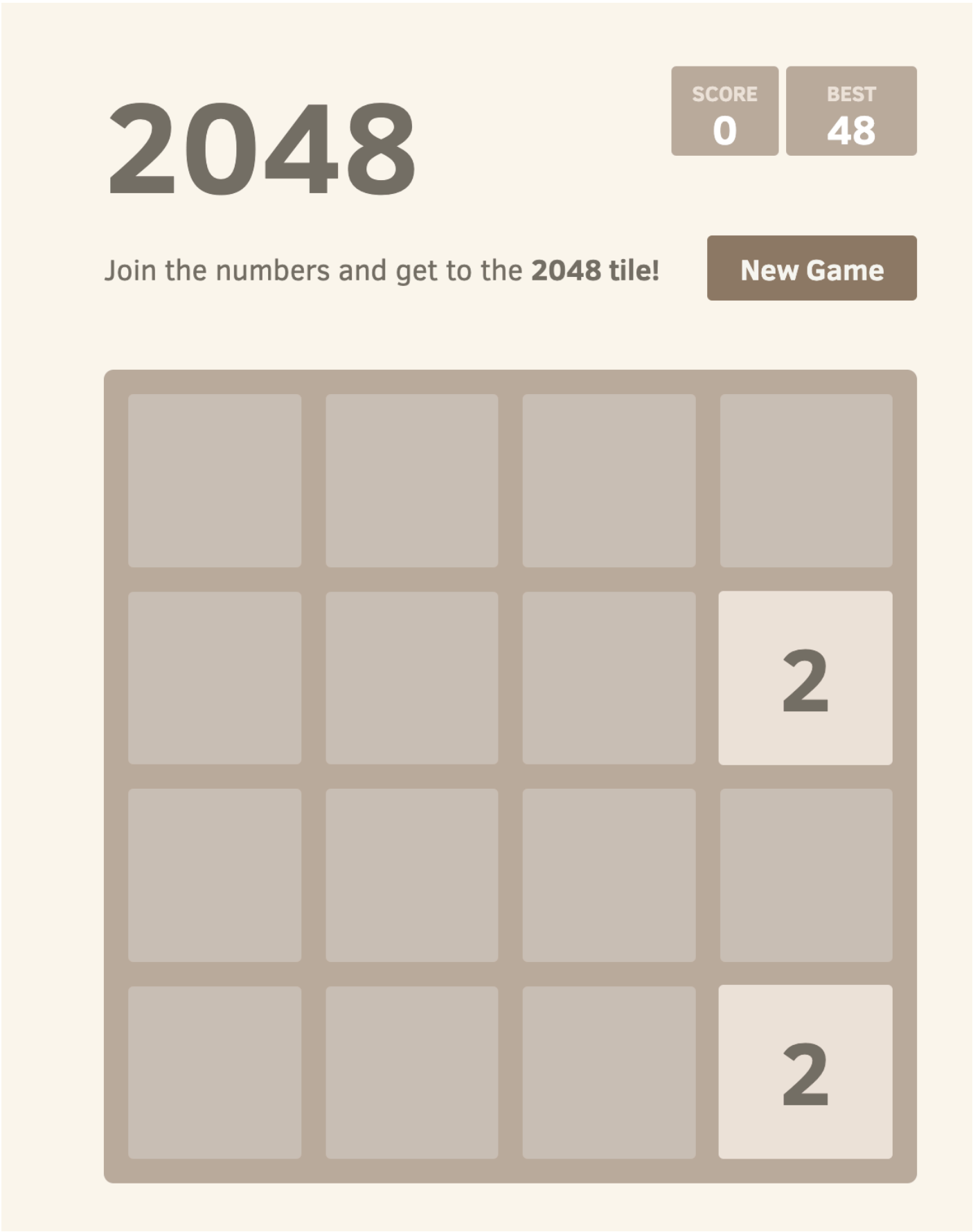
```
NAME           CLASS   HOSTS   ADDRESS
                      PORTS   AGE
ingress-2048   alb     *       k8s-game2048-ingress2-eb379a0f83-378466616.region-
code.elb.amazonaws.com   80      31s
```

開始以下步驟之前，您將需要等候數分鐘，以便 Application Load Balancer (ALB) 完成佈建。

4. 開啟 Web 瀏覽器並輸入上一步的 ADDRESS，以存取 Web 應用程式。例如：

```
k8s-game2048-ingress2-eb379a0f83-378466616.region-code.elb.amazonaws.com
```

您應在瀏覽器中看到 2048 遊戲。暢玩！


2048
SCORE
0
BEST
48
Join the numbers and get to the 2048 tile!
New Game
2
2

藉助 Amazon EKS 自動模式讓資料持續保留

2048 遊戲現在已在 Amazon EKS 叢集上啟動並執行，務必利用 Amazon EKS 自動模式的區塊儲存功能，讓遊戲資料安全、持續保留。

1. 建立名為 `storage-class.yaml` 的檔案：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: auto-ebs-sc
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: ebs.csi.eks.amazonaws.com
volumeBindingMode: WaitForFirstConsumer
parameters:
  type: gp3
  encrypted: "true"
```

2. 套用 `StorageClass`：

```
kubectl apply -f storage-class.yaml
```

3. 建立持續性磁碟區宣告 (PVC)，以請求儲存遊戲資料。建立名稱為 `ebs-pvc.yaml` 的檔案並為其新增以下內容：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: game-data-pvc
  namespace: game-2048
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  storageClassName: auto-ebs-sc
```

4. 將 PVC 套用至您的叢集：

```
kubectl apply -f ebs-pvc.yaml
```

您應該會看到下列回應輸出：

```
persistentvolumeclaim/game-data-pvc created
```

5. 您現在需更新 2048 遊戲部署，以使用此 PVC 來存放資料。設定以下部署，以使用 PVC 來存放遊戲資料。建立名稱為 ebs-deployment.yaml 的檔案並為其新增以下內容：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  namespace: game-2048
  name: deployment-2048
spec:
  replicas: 3  # Adjust the number of replicas as needed
  selector:
    matchLabels:
      app.kubernetes.io/name: app-2048
  template:
    metadata:
      labels:
        app.kubernetes.io/name: app-2048
    spec:
      containers:
        - name: app-2048
          image: public.ecr.aws/l6m2t8p7/docker-2048:latest
          imagePullPolicy: Always
          ports:
            - containerPort: 80
          volumeMounts:
            - name: game-data
              mountPath: /var/lib/2048
      volumes:
        - name: game-data
          persistentVolumeClaim:
            claimName: game-data-pvc
```

6. 套用更新的部署：

```
kubectl apply -f ebs-deployment.yaml
```

您應該會看到下列回應輸出：

```
deployment.apps/deployment-2048 configured
```

完成這些步驟後，現在即可設定叢集上的 2048 遊戲，以利用 Amazon EKS 自動模式的區塊儲存功能持續保留資料。這樣一來，即使 Pod 或節點發生故障，仍可確保遊戲進度及資料是安全的。

若您喜歡此教學課程，您可提供意見回饋告知我們，以便我們為您提供更多此類使用案例的特定快速入門教學課程。

清除

若要避免未來產生費用，您需要手動刪除相關聯的 CloudFormation 堆疊，進而刪除在本指南中建立的所有資源，包括 VPC 網路。

刪除 CloudFormation 堆疊：

```
eksctl delete cluster -f ./cluster-config.yaml
```

透過範例了解 Amazon EKS

概觀

本《Amazon EKS 使用者指南》包含從命令列或 AWS 管理主控台 建立第一個 EKS 叢集的一般用途程序，以及適用於所有主要 Amazon EKS 元件的可靠參考。不過，身為 Amazon EKS 叢集管理員或開發人員，您可以遵循本指南外部網站中存在的學習路徑，進而深入了解 Amazon EKS。這些網站可協助您：

- 設定特定的叢集類型。特定叢集類型可以根據您的工作負載類型或安全要求而定。例如，您可能想要調校叢集以執行批次、機器學習或運算密集型工作負載。
- 增強您的叢集。您可以將進階功能新增至叢集，以提供可觀測性、彈性儲存、自動擴展或專用叢集聯網等功能。
- 自動化更新。使用 GitOps 等功能，您可以根據 Git 儲存庫中這些元件出現的變更進行設定，以自動佈建叢集基礎結構和工作負載。
- 使用進階叢集設定工具。雖然 `eksctl` 提供了一種快速建立叢集的方式，但還有其他工具可以更輕鬆地設定和升級更複雜的叢集。其中包括 Terraform 和 CloudFormation 等工具。

若要開始您的 Amazon EKS 學習路徑，建議您造訪此頁面所述的一些網站。如果您在此過程中遇到問題，還會有其他資源可協助您解決這些問題。例如，Re:post 知識中心可讓您搜尋支援資料庫是否有與 Amazon EKS 相關的支援問題。此外，《Amazon EKS 最佳實務指南》也提供了有關設定生產級叢集的最佳方式的秘訣。

Amazon EKS 研討會

Amazon EKS 工作坊會從對 Kubernetes 和容器的基本了解入手，是引導叢集管理員了解 Amazon EKS 重要功能的學習平台。以下是您可以與 Amazon EKS 工作坊互動的方式：

- Amazon EKS 基本概念：觀看簡介頁面上的影片，了解 Amazon EKS 如何在 AWS 雲端實作 Kubernetes 功能。如果您需要對 Kubernetes 有更基本的了解，請觀看什麼是 Kubernetes 影片。
- Amazon EKS 設定：如果您有 AWS 帳戶，設定區段可協助您設定 CloudShell 環境，以用於建立叢集。您可以選擇使用 eksctl (簡單的叢集建立命令列) 和 Terraform (建立叢集的更為基礎結構即程式碼的方法) 建立 Amazon EKS 叢集。

- Amazon EKS 入門：從範例應用程式區段嘗試簡單的 Web 商店。您可以在其他練習中進行使用。在本節中，您還可以了解封裝容器映像以及如何使用 Kubernetes Pod、部署、服務、StatefulSets 和命名空間管理微服務。然後，使用 Kustomizee 將變更部署到 Kubernetes 資訊清單。
- Amazon EKS 基本原則：使用AWS Load Balancer控制器等 AWS 功能，研討會會示範如何向外界公開您的應用程式。對於儲存，工作坊會展示如何使用 Amazon EBS 進行區塊儲存、使用 Amazon EFS 進行檔案系統儲存，以及使用 Amazon FSx for NetApp ONTAP 在 AWS中管理 ONTAP 檔案系統。對於節點管理，工作坊可協助您設定受管節點群組。
- Amazon EKS 進階功能：Amazon EKS 工作坊提供的更多進階功能包括用於設定的實驗室：
  - 自動擴展：這包括節點自動擴展 (使用 Cluster Autoscaler 或 Karpenter) 和工作負載自動擴展 (使用 Horizontal Pod Autoscaler 和 Cluster Proportional Autoscaler)。
  - 可觀測性：在一組可觀測性實驗室中，了解記錄、OpenSearch、Amazon EKS 上的 Container Insights 以及使用 Kubecost 的成本可見性。
  - 安全性：這組安全性實驗室可讓您探索秘密管理、Amazon GuardDuty、Pod 安全標準和 Kyverno 政策管理。
  - 網路：從網路實驗室了解 Amazon EKS 的網路功能，其中包含 Amazon VPC CNI （支援網路外掛程式）和 Amazon VPC Lattice （用於跨 VPC 和使用者帳戶設定叢集）。
  - 自動化：自動化實驗室會逐步引導您完成管理叢集和專案的 GitOps 方法來，例如 Kubernetes 專用AWS 控制器和用於管理 Amazon EKS 控制平面的 Crossplane。

Amazon EKS 實作叢集設定教學

AWS 社群網站上的一組 Amazon EKS 叢集設定教學課程可協助您建立特殊用途的 Amazon EKS 叢集，並以各種方式增強這些叢集。教學分為三種不同的類型：

建置叢集

這些教學可協助您建置可用於特殊用途的叢集。這些特殊用途包括執行以下各項的能力：

- 基於 IPv6 的全球可擴展應用程式
- 非同步批次任務
- 高流量微服務
- Fargate 上的 Autoscaling with Karpenter
- 財務工作負載
- Windows 受管節點群組

增強叢集

擁有現有叢集後，您就可以各種方法擴展和增強該叢集，例如，允許它執行專門的工作負載及以其他方式增強叢集。這些教學包括以下方法：

- 使用 EFS CSI 提供儲存解決方案
- 使用 EBS CSI 提供動態資料庫儲存
- 使用 AWS Load Balancer控制器在 IPv4 叢集上公開應用程式
- 使用 AWS Load Balancer控制器在 IPv6 叢集上公開應用程式

最佳化 AWS 服務

使用這些教學課程，您可以更好地整合叢集與 AWS 服務。這些教學包括可協助您的教學：

- 使用 ExternalDNS 管理微服務的 DNS 記錄
- 使用 CloudWatch 監控應用程式
- 使用 SQS 和 EFS 儲存管理非同步任務
- 從工作負載使用 AWS Secrets Manager 秘密
- 使用 Fargate、NGINX 和 ACM PCA 設定 mTLS

Amazon EKS 範例

Amazon EKS 範例儲存庫可存放資訊清單，以便與 Amazon EKS 搭配使用。這些資訊清單讓您可以在 Amazon EKS 中嘗試不同類型的應用程式，或建立特定類型的 Amazon EKS 叢集。範例包括資訊清單，可：

- 建立 AWS Amazon EKS Fargate 叢集
- 使用現有的 IAM 角色建立叢集
- 將 Ubuntu 受管節點群組新增至叢集
- 使用磁碟區快照備份和還原 Pod 儲存
- 復原掛載為 PVC 且具有多個帳戶的 EBS 磁碟區
- 使用 Classic Load Balancer 啟用 NGINX 傳入控制器的代理通訊協定
- 將 Fargate 上的記錄設定為 AWS OpenSearch
- 使用 Web 聯合身分提供者執行 Python SDK

- 在 NFS CSI 控制器上部署範例應用程式
- 對 StatefulSets 使用磁碟區快照
- 跨不同可用區域上的節點部署 Pod

請記住，這些範例僅供學習和測試之用，且不適用於生產環境。

AWS 教學課程

AWS 教學課程網站會發佈一些 Amazon EKS 教學課程，但也提供搜尋工具來尋找 AWS 在網站（例如 AWS 社群網站）上發佈的其他教學課程。直接在此網站上發布的 Amazon EKS 教學包括：

- 在 Amazon EKS 上部署容器 Web 應用程式
- 以較低的成本執行 Kubernetes 叢集 (Amazon EKS 和 Spot 執行個體)
- 如何在 Kubernetes 上實現 Jenkins 任務成本最佳化

開發人員工作坊

如果您是軟體開發人員，想要建立或重構應用程式以在 Amazon EKS 上執行，Amazon EKS 開發人員研討會是很好的開始。工作坊不僅可協助您建置容器化應用程式，還可協助您將這些容器部署到容器登錄檔 (ECR)，以及從該處部署到 Amazon EKS 叢集。

從 Amazon EKS Python 工作坊開始，逐步完成重構 Python 應用程式的程序，然後設定您的開發環境以準備部署應用程式。逐步了解容器、Kubernetes 和 Amazon EKS 的相關章節，以準備在這些環境中執行容器化應用程式。

Terraform 工作坊

雖然 `eksctl` 是用於建立叢集的簡單工具，但對於更複雜的基礎結構即程式碼類型的 Amazon EKS 部署，Terraform 則是熱門的 Amazon EKS 叢集建立和管理工具。Terraform Amazon EKS 研討會教導如何使用 Terraform 來建置 AWS VPC、建立 Amazon EKS 叢集，以及為您的叢集新增選用增強功能。具體而言，其中有一個用於建立私有 Amazon EKS 叢集的區段

AWS Amazon EKS 訓練

AWS 提供正式訓練來了解 Amazon EKS。為期三天題為在 Amazon Elastic Kubernetes Service 上執行容器的訓練課程可教導：

- Kubernetes 和 Amazon EKS 基礎知識
- 如何建置 Amazon EKS 叢集
- 使用 IAM 和 Kubernetes RBAC AWS 授權保護 Amazon EKS
- GitOps 自動化工具
- 監控工具
- 改善成本、效率和彈性的技術

開始使用 Amazon EKS

在閱讀此入門指南之前，請確保您已完成 Amazon EKS 的設定並可開始使用。如需詳細資訊，請參閱設定。

對於在 Amazon EKS 中建立含節點的新 Kubernetes 叢集，我們提供兩種入門指南：

- Amazon EKS 入門– eksctl：本入門指南可協助您安裝將 Amazon EKS 和 `eksctl` 搭配使用所需的所有資源，後者是簡單的命令列公用程式，可用於在 Amazon EKS 上建立和管理 Kubernetes 叢集。教學結束時，您將擁有一個執行中的 Amazon EKS 叢集，而您可以在其中部署應用程式。這是 Amazon EKS 入門的最快且最簡易的方式。
- Amazon EKS 入門– AWS 管理主控台 和 AWS CLI：本入門指南可協助您使用 AWS 管理主控台 和 AWS 建立 Amazon EKS 入門所需的所有資源。教學結束時，您將擁有一個執行中的 Amazon EKS 叢集，而您可以在其中部署應用程式。在本指南中，您可以手動建立 Amazon EKS 叢集所需的各個資源。這些程序可讓您完全了解每個資源的建立方式，以及它們彼此之間的互動方式。

我們還會提供下列參考：

- 如需實作教學的集合，請參閱 AWS 社群上的 EKS 叢集設定。
- 如需程式碼範例，請參閱適用於使用 AWS SDK 的 Amazon EKS 的程式碼範例。

Amazon EKS 入門– EKS 自動模式

如同其他 EKS 入門體驗，使用 EKS 自動模式建立您的第一個叢集即會將叢集本身的管理委派給 AWS。不過，EKS 自動模式可透過處理設定工作負載基礎結構 (節點、網路和各種服務) 所需的許多基本服務，來擴展 EKS 自動化，因此您能更輕鬆地管理節點並向上擴展以滿足工作負載需求。

選擇下列其中一種方式，以使用 EKS 自動模式建立叢集：

- the section called “ AWS CLI”：使用 `aws` 命令列介面來建立叢集。
- the section called “管理主控台”：使用 AWS 管理主控台 來建立叢集。
- the section called “eksctl CLI”：使用 `eksctl` 命令列介面來建立叢集。

如果您要比較建立第一個 EKS 叢集的不同方法，您應該知道 EKS 自動模式已讓 AWS 接管額外的叢集管理責任，其中包括設定元件以：

- 隨著工作負載需求增加和減少，啟動及擴展節點。
- 定期升級叢集本身 (控制平面)、節點作業系統以及在節點上執行的服務。
- 選擇預設設定，以決定節點儲存的大小和速度以及 Pod 網路組態等事項。

如需您可使用 EKS 自動模式叢集執行之動作的詳細資訊，請參閱 EKS 自動模式。

Amazon EKS 入門 – `eksctl`

> **Note**
>
> 本主題涵蓋不使用 EKS 自動模式時的入門。
> EKS 自動模式會自動執行用於叢集運算、儲存和聯網的例行任務。了解如何開始使用 Amazon EKS Auto 模式。

本指南可協助您安裝在將 Amazon Elastic Kubernetes Service (Amazon EKS) 和 `eksctl` 搭配使用所需的所有資源，後者是簡單的命令列公用程式，可用於在 Amazon EKS 上建立和管理 Kubernetes 叢集。此教學結束時，您將擁有一個執行中的 Amazon EKS 叢集，而您可以在其中部署應用程式。

本指南中的程序會為您自動建立數個資源，而您必須在使用 AWS 管理主控台建立叢集時手動建立這些資源。如果您寧願手動建立大多數資源，以進一步了解它們如何彼此互動，請使用 AWS 管理主控台來建立叢集和運算。如需詳細資訊，請參閱the section called “建立叢集 (主控台和 CLI)”。

先決條件

開始本教學課程之前，您必須安裝和設定 AWS CLI、kubectl 和 eksctl 工具，如設定以使用 Amazon EKS 中所述。

步驟 1：建立 Amazon EKS 叢集和節點

> **Important**
>
> 為了盡可能簡單快速地開始使用，本主題包含使用預設設定建立叢集和節點的步驟。建立叢集和節點供生產使用之前，推薦您先熟悉所有設定，並使用符合您需求的設定來部署叢集和節點。如需詳細資訊，請參閱the section called “建立 叢集”及管理運算。只有在建立叢集和節點時，才能夠啟用某些設定。

您可以使用下列其中一種節點類型來建立叢集。若要進一步了解每種類型，請參閱 管理運算。叢集部署完成後，您可以新增其他節點類型。

- Fargate – Linux：如果您想要在 the section called “ AWS Fargate” 上執行 Linux 應用程式，請選取此節點類型。Fargate 是無伺服器運算引擎，可讓您部署 Kubernetes Pod，而無需管理 Amazon EC2 執行個體。
- 受管節點 – Linux：如果您想要在 Amazon EC2 執行個體上執行 Amazon Linux 應用程式，請選取此節點類型。雖然未涵蓋在本指南中，但您也可以將 Windows 自我管理和 Bottlerocket 節點新增至您的叢集。

透過以下命令建立 Amazon EKS 叢集。您可以用您自己的值取代 *my-cluster*。此名稱僅能使用英數字元 (區分大小寫) 和連字號。必須以英數字元開頭，且長度不可超過 100 個字元。名稱在您要建立叢集 AWS 的區域和 AWS 帳戶中必須是唯一的。將 *region-code* 取代為 Amazon EKS 支援的任何 AWS 區域。如需 AWS 區域清單，請參閱《 AWS 一般參考指南》中的 Amazon EKS 端點和配額。

Example

Fargate - Linux

```
eksctl create cluster --name my-cluster --region region-code --fargate
```

Managed nodes - Linux

```
eksctl create cluster --name my-cluster --region region-code
```

叢集建立需要幾分鐘的時間。在建立期間，您會看到數行輸出。輸出的最後一行類似於下面的範例行。

```
[...]
[#]  EKS cluster "my-cluster" in "region-code" region is ready
```

`eksctl` 已在 `~/.kube/config` 中建立了 `kubectl` 組態檔案，或將新叢集的組態新增至您電腦上 `~/.kube/config` 中的現有組態檔案。

叢集建立完成後，請在 AWS CloudFormation 主控台`eksctl-`*`my-cluster`*`-cluster`中檢視名為 的 AWS CloudFormation 堆疊，以查看所有已建立的資源。

步驟 2：檢視 Kubernetes 資源

1. 檢視叢集節點。

```
kubectl get nodes -o wide
```

範例輸出如下。

Example

Fargate - Linux

```
NAME                                                  STATUS   ROLES    AGE
 VERSION              INTERNAL-IP   EXTERNAL-IP   OS-IMAGE         KERNEL-VERSION
                CONTAINER-RUNTIME
fargate-ip-192-0-2-0.region-code.compute.internal   Ready    <none>
 8m3s    v1.2.3-eks-1234567   192.0.2.0     <none>        Amazon Linux 2
 1.23.456-789.012.amzn2.x86_64   containerd://1.2.3
fargate-ip-192-0-2-1.region-code.compute.internal   Ready    <none>
 7m30s   v1.2.3-eks-1234567   192.0.2.1     <none>        Amazon Linux 2
 1.23.456-789.012.amzn2.x86_64   containerd://1.2.3
```

Managed nodes - Linux

```
NAME                                       STATUS   ROLES    AGE    VERSION
        INTERNAL-IP   EXTERNAL-IP   OS-IMAGE         KERNEL-VERSION
   CONTAINER-RUNTIME
ip-192-0-2-0.region-code.compute.internal   Ready    <none>   6m7s
  v1.2.3-eks-1234567   192.0.2.0     192.0.2.2     Amazon Linux 2
 1.23.456-789.012.amzn2.x86_64   containerd://1.2.3
ip-192-0-2-1.region-code.compute.internal   Ready    <none>   6m4s
  v1.2.3-eks-1234567   192.0.2.1     192.0.2.3     Amazon Linux 2
 1.23.456-789.012.amzn2.x86_64   containerd://1.2.3
```

如需您在輸出中看到之內容的詳細資訊，請參閱 the section called “存取叢集資源”。

2. 檢視叢集上執行的工作負載。

```
kubectl get pods -A -o wide
```

範例輸出如下。

Example

Fargate - Linux

```
NAMESPACE     NAME                          READY   STATUS    RESTARTS   AGE   IP
          NODE                                                NOMINATED NODE
 READINESS GATES
kube-system   coredns-1234567890-abcde   1/1     Running   0          18m
 192.0.2.0   fargate-ip-192-0-2-0.region-code.compute.internal   <none>
 <none>
kube-system   coredns-1234567890-12345   1/1     Running   0          18m
 192.0.2.1   fargate-ip-192-0-2-1.region-code.compute.internal   <none>
 <none>
```

Managed nodes - Linux

```
NAMESPACE     NAME                          READY   STATUS    RESTARTS   AGE     IP
          NODE                                          NOMINATED NODE   READINESS
 GATES
kube-system   aws-node-12345             1/1     Running   0          7m43s
 192.0.2.1   ip-192-0-2-1.region-code.compute.internal   <none>           <none>
kube-system   aws-node-67890             1/1     Running   0          7m46s
 192.0.2.0   ip-192-0-2-0.region-code.compute.internal   <none>           <none>
kube-system   coredns-1234567890-abcde   1/1     Running   0          14m
 192.0.2.3   ip-192-0-2-3.region-code.compute.internal   <none>           <none>
kube-system   coredns-1234567890-12345   1/1     Running   0          14m
 192.0.2.4   ip-192-0-2-4.region-code.compute.internal   <none>           <none>
kube-system   kube-proxy-12345           1/1     Running   0          7m46s
 192.0.2.0   ip-192-0-2-0.region-code.compute.internal   <none>           <none>
kube-system   kube-proxy-67890           1/1     Running   0          7m43s
 192.0.2.1   ip-192-0-2-1.region-code.compute.internal   <none>           <none>
```

如需您在輸出中看到之內容的詳細資訊，請參閱 the section called “存取叢集資源”。

步驟 3：刪除您的叢集和節點

在完成為本教學建立的叢集和節點後，您應該使用以下命令刪除叢集和節點來完成清除。如果想在清理前使用此叢集執行更多作業，請參閱 the section called “後續步驟”。

```
eksctl delete cluster --name my-cluster --region region-code
```

後續步驟

以下文件主題可協助您擴展叢集的功能。

- 將範例應用程式部署至叢集。
- 建立叢集的 IAM 主體是唯一可以使用 `kubectl` 或 AWS 管理主控台對 Kubernetes API 伺服器進行呼叫的主體。如果要其他 IAM 主體能夠存取叢集，則需新增這些主體。如需詳細資訊，請參閱the section called “Kubernetes API 存取”及the section called “所需的許可”。
- 部署叢集以供生產使用之前，建議您先熟悉叢集和節點的所有設定。建立叢集時，必須進行某些設定 (例如啟用對 Amazon EC2 節點的 SSH 存取)。
- 若要提高叢集的安全性，可將 Amazon VPC 容器聯網介面外掛程式設定為使用服務帳戶的 IAM 角色。

開始使用 Amazon EKS – AWS 管理主控台 和 AWS CLI

> **Note**
>
> 本主題涵蓋不使用 EKS 自動模式時的入門。它會使用受管節點群組來部署節點。
> EKS 自動模式會自動執行用於叢集運算、儲存和聯網的例行任務。了解如何開始使用 Amazon EKS Auto 模式。EKS 自動模式是部署節點的慣用方法。

本指南可協助您建立所有必要的資源，以使用 AWS 管理主控台 和 CLI 開始使用 Amazon Elastic Kubernetes Service AWS (Amazon EKS)。在本指南中，您可以手動建立每個資源。此教學結束時，您將擁有一個執行中的 Amazon EKS 叢集，而您可以在其中部署應用程式。

本指南中的程序可讓您完全瞭解每個資源的建立方式，以及資源彼此之間的互動方式。如果您希望自動為您建立大部分資源，請使用 `eksctl` CLI 建立叢集和節點。如需詳細資訊，請參閱the section called “建立叢集 (eksctl)”。

先決條件

開始此教學之前，您必須先安裝和設定下列在建立和管理 Amazon EKS 叢集所需的工具和資源。

- AWS CLI – 用於使用 AWS 服務的命令列工具，包括 Amazon EKS。如需詳細資訊，請參閱《 AWS 命令列界面使用者指南》中的安裝。安裝 CLI AWS 之後，建議您也進行設定。如需詳細資訊，請

參閱《 AWS 命令列界面使用者指南》中的具有 aws 設定的快速組態。請注意，使用此頁面顯示的 update-kubeconfig 選項需要 AWS CLI v2。

- **kubectl** ：命令列工具，適用於使用 Kubernetes 叢集。如需詳細資訊，請參閱the section called “設定 kubectl 和 eksctl”。
- 必要的 IAM 許可 – 您使用的 IAM 安全主體必須具有使用 Amazon EKS IAM 角色、服務連結角色、AWS CloudFormation、VPC 和相關資源的許可。如需詳細資訊，請參閱《IAM 使用者指南》中的動作和使用服務連結角色。您必須以同一位使用者的身分完成本指南中的所有步驟。若要檢查目前使用者，請執行以下命令：

```
aws sts get-caller-identity
```

建議您在 Bash shell 中完成本主題中的步驟。如果您不使用 Bash shell，則某些指令碼命令 (如行接續字元以及設定和使用變數的方式) 需要針對 shell 進行調整。此外，您的 Shell 的引用及轉義規則可能會有所不同。如需詳細資訊，請參閱《 AWS 命令列界面使用者指南》中的使用引號搭配 AWS CLI 中的字串。

步驟 1：建立您的 Amazon EKS 叢集

> **Important**
>
> 為了盡可能簡單快速地開始使用，本主題包含使用預設設定建立叢集的步驟。建立叢集供生產使用之前，建議您先熟悉所有設定，再使用符合需求的設定來部署叢集。如需詳細資訊，請參閱the section called “建立 叢集”。只有在建立叢集時，才能夠啟用某些設定。

1. 使用符合 Amazon EKS 要求的公有和私有子網路建立 Amazon VPC。將 *region-code* 取代為 Amazon EKS 支援的任何 AWS 區域。如需 AWS 區域清單，請參閱《 AWS 一般參考指南》中的 Amazon EKS 端點和配額。您可以使用選擇的任何名稱取代 *my-eks-vpc-stack*。

```
aws cloudformation create-stack \
  --region region-code \
  --stack-name my-eks-vpc-stack \
  --template-url https://s3.us-west-2.amazonaws.com/amazon-eks/
cloudformation/2020-10-29/amazon-eks-vpc-private-subnets.yaml
```

> **Tip**
>
> 如需上一個命令建立的所有資源清單，請開啟位於 https：//https://console.aws.amazon.com/cloudformation/ 的 AWS CloudFormation 主控台。選擇 *my-eks-vpc-stack* 堆疊，然後選擇 Resources (資源) 索引標籤。

2. 建立叢集 IAM 角色，並將所需的 Amazon EKS IAM 受管政策連接到該角色。Amazon EKS 管理的 Kubernetes 叢集會代表您呼叫其他 AWS 服務，以管理您與服務搭配使用的資源。

   a. 將下列內容複製到名為 *eks-cluster-role-trust-policy.json* 的檔案。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

   b. 建立角色。

```
aws iam create-role \
  --role-name myAmazonEKSClusterRole \
  --assume-role-policy-document file://"eks-cluster-role-trust-policy.json"
```

   c. 將必要的 Amazon EKS 受管 IAM 政策連接到角色。

```
aws iam attach-role-policy \
  --policy-arn arn:aws: iam::aws:policy/AmazonEKSClusterPolicy \
  --role-name myAmazonEKSClusterRole
```

3. 在以下網址開啟 Amazon EKS 主控台：https://console.aws.amazon.com/eks/home#/clusters。

   請確定主控台右上角顯示的 AWS 區域是您要建立叢集 AWS 的區域。如果不是，請選擇 AWS 區域名稱旁的下拉式清單，然後選擇您要使用的 AWS 區域。

4. 選擇 Create Cluster (建立叢集)。如果沒有看到這個選項，請先在左側導覽窗格中選擇叢集。

5. 在 Configure cluster (設定叢集) 頁面上，執行下列操作：
   a. 選取自訂組態，並停用使用 EKS 自動模式。(如果您偏好 EKS 自動模式叢集，則請改為參閱 the section called “管理主控台”。)
   b. 輸入叢集的名稱，例如 *my-cluster*。此名稱僅能使用英數字元 (區分大小寫) 和連字號。必須以英數字元開頭，且長度不可超過 100 個字元。名稱在您要建立叢集 AWS 的區域和 AWS 帳戶中必須是唯一的。
   c. 對於 Cluster Service Role (叢集服務角色)，選擇 *myAmazonEKSClusterRole*。
   d. 將其餘設定維持在其預設值，然後選取 Next (下一步)。
6. 在 Specify networking (指定聯網) 頁面中，執行下列操作：
   a. 從 VPC 下拉式清單中選擇在先前步驟中建立的 VPC ID。其類似於 ** | my-eks-vpc-stack-VPC*。
   b. 從子網路下拉式清單中，選擇上一個步驟中建立的子網路。子網路將類似於 ** | my-eks-vpc-stack-**。
   c. 從額外的安全群組下拉式清單中，選擇在上一個步驟中建立的安全群組。其類似於 ** | my-eks-vpc-stack-ControlPlaneSecurityGroup-**。
   d. 將其餘設定維持在其預設值，然後選取 Next (下一步)。
7. 在設定可觀測性頁面上，選擇下一步。
8. 在選取附加元件頁面上，選擇下一步。

   如需附加元件的詳細資訊，請參閱 the section called “Amazon EKS 附加元件”。
9. 在設定選取的附加元件設定頁面中，選取下一步：
10. 在 Review and create (檢閱和建立) 頁面上，選取 Create (建立)。

    叢集名稱右側的叢集狀態會顯示建立中幾分鐘，直到叢集佈建程序完成為止。在狀態為作用中之前不要進行下一個步驟。

> **Note**
>
> 您可能會收到錯誤，表示在請求中的其中一個可用區域沒有足夠的容量可建立 Amazon EKS 叢集。如果發生這種情況，錯誤輸出包含的可用區域可支援新的叢集。使用至少兩個位於帳戶的支援可用區域子網路來建立您的叢集。如需詳細資訊，請參閱the section called “容量不足”。

步驟 2：連線至您的叢集

在本節中，您可以kubectl連線至叢集，以便管理 Kubernetes 資源。

在繼續之前，請確保在步驟 1 中，您的叢集建立已成功完成。

選項 1：使用 CLI AWS 連線

1. 為您的叢集建立或更新 kubeconfig 檔案。將 *region-code* 取代 AWS 為您建立叢集的區域。使用您叢集的名稱取代 *my-cluster*。

   ```
   aws eks update-kubeconfig --region region-code --name my-cluster
   ```

   根據預設，已在 ~/.kube 建立了 config 檔案，或者已將新叢集的組態新增至 ~/.kube 中的現有 config 檔案。
2. 測試組態。

   ```
   kubectl get svc
   ```

   > **Note**
   >
   > 如果您收到任何授權或資源類型錯誤，請參閱故障診斷主題中的the section called “未經授權或存取遭拒 (kubectl)”。

   範例輸出如下。

   ```
   NAME             TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
   svc/kubernetes   ClusterIP   10.100.0.1   <none>        443/TCP   1m
   ```

選項 2：使用 EKS 主控台連線

或者，您也可以直接從主控台連線，而不需要任何本機設定：

1. 在您在步驟 1 中建立之叢集的叢集詳細資訊頁面上，選擇連線。
2. AWS CloudShell 會開啟，並預先為您的叢集kubectl設定 。

3. 測試您的連線：

```
kubectl get svc
```

如需詳細資訊，請參閱the section called “透過 kubectl 存取叢集”。

步驟 3：建立節點

> **Important**
>
> 為了盡可能簡單快速地開始使用，本主題包含使用大多數預設設定建立節點的步驟。建立節點供生產使用之前，建議您先熟悉所有設定，再使用符合需求的設定來部署節點。如需詳細資訊，請參閱管理運算。只有在建立節點時，才能夠啟用某些設定。

此程序會將您的叢集設定為使用受管節點群組來建立節點，並指定您在先前步驟中建立的子網路和節點 IAM 角色。它可讓您在 Amazon EC2 執行個體上執行 Amazon Linux 應用程式。

若要進一步了解在 EKS 中設定節點的不同方式，請參閱 管理運算。叢集部署完成後，您可以新增其他節點類型。雖然未涵蓋在本指南中，但您也可以將 Windows 自我管理和 Bottlerocket 節點新增至您的叢集。

若要建立 EC2 Linux 受管節點群組

1. 建立節點 IAM 角色，並將所需的 Amazon EKS IAM 受管政策連接到該角色。Amazon EKS `kubelet` 節點協助程式會代表您呼叫 AWS APIs。節點透過 IAM 執行個體描述檔和關聯的政策，取得這些 API 呼叫的許可。

   a. 將下列內容複製到名為 `node-role-trust-policy.json` 的檔案。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
```

```
  ]
}
```

b. 建立節點 IAM 角色。

```
aws iam create-role \
  --role-name myAmazonEKSNodeRole \
  --assume-role-policy-document file://"node-role-trust-policy.json"
```

c. 將所需的受管 IAM 政策連接到角色。

```
aws iam attach-role-policy \
  --policy-arn arn:aws: iam::aws:policy/AmazonEKSWorkerNodePolicy \
  --role-name myAmazonEKSNodeRole
aws iam attach-role-policy \
  --policy-arn arn:aws: iam::aws:policy/AmazonEC2ContainerRegistryReadOnly \
  --role-name myAmazonEKSNodeRole
aws iam attach-role-policy \
  --policy-arn arn:aws: iam::aws:policy/AmazonEKS_CNI_Policy \
  --role-name myAmazonEKSNodeRole
```

d. 在以下網址開啟 Amazon EKS 主控台：https://console.aws.amazon.com/eks/home#/clusters。

e. 選擇您在步驟 1：建立 Amazon EKS 叢集中建立的叢集的名稱，例如 *my-cluster*。

f. 在 ***my-cluster*** 頁面上，執行下列操作：

g. 選擇 Compute (運算) 索引標籤。

h. 選擇 Add Node Group (新增節點群組)。

2. 在 Configure Node Group (設定節點群組) 頁面上，執行以下操作：

   a. 對於名稱，輸入受管節點群組的唯一名稱，例如 *my-nodegroup*。節點群組名稱不可超過 63 個字元。它必須以字母或數字開頭，但也可以包含其餘字元的連字號和底線。

   b. 對於 Node IAM role name (節點 IAM 角色名稱)，選擇您在前一個步驟建立的 *myAmazonEKSNodeRole* 角色。建議每個節點群組使用其唯一的 IAM 角色。

   c. 選擇下一步。

3. 在 Set compute and scaling configuration (設定運算和擴展組態) 頁面上，接受預設值，然後選取 Next (下一步)。

4. 在 Specify networking (指定聯網) 頁面上，接受預設值，然後選取 Next (下一步)。

5. 在 Review and create (檢閱並建立) 頁面上，檢閱您的受管節點群組組態，然後選擇 Create (建立)。

6. 數分鐘後，Node Group configuration (節點群組組態) 區段中的 Status (狀態) 將從 Creating (建立中) 變更為 Active (作用中)。在狀態為作用中之前不要進行下一個步驟。

步驟 4：檢視資源

您可以檢視節點和 Kubernetes 工作負載。

1. 在左側導覽窗格中選擇叢集。在 Clusters (叢集) 清單中，選擇您建立之叢集的名稱，例如 *my-cluster*。
2. 在 ***my-cluster*** 頁面上，選擇以下事項：
   a. Compute (運算) 標籤 – 您可以看到為叢集部署的 Nodes (節點) 清單。您可以選取節點的名稱，以查看有關節點的詳細資訊。
   b. 資源標籤：您會看到預設部署到 Amazon EKS 叢集的所有 Kubernetes 資源。在主控台中選取任何資源類型，以進一步了解其詳細資訊。

步驟 5：刪除資源

在完成為本教學建立的叢集和節點後，您應該刪除所建立的資源。如果想在刪除資源前使用此叢集執行更多作業，請參閱 the section called “後續步驟”。

1. 刪除您建立的任何節點群組或 Fargate 設定檔。
   a. 在 https://console.aws.amazon.com/eks/home#/clusters 開啟 Amazon EKS 主控台。
   b. 在左側導覽窗格中選擇叢集。在叢集清單中，選取 *my-cluster*。
   c. 選擇 Compute (運算) 索引標籤。
   d. 如果建立了節點群組，請選擇 *my-nodegroup* 節點群組，然後選擇 Delete (刪除)。輸入 *my-nodegroup*，然後選擇刪除。
   e. 請刪除節點群組設定檔後再繼續。
2. 刪除叢集。
   a. 在左側導覽窗格中選擇叢集。在叢集清單中，選取 *my-cluster*。
   b. 選擇 Delete cluster (刪除叢集)。
   c. 輸入 *my-cluster*，然後選擇刪除。刪除叢集後再繼續。
3. 刪除您建立的 VPC AWS CloudFormation 堆疊。
   a. 開啟位在 https://console.aws.amazon.com/cloudformation/ 的 CloudFormation 主控台。
   b. 選取 *my-eks-vpc-stack* 堆疊，然後選擇 Delete (刪除)。

c. 在 刪除 ***my-eks-vpc-stack*** 確認對話方塊中，選擇刪除堆疊。

4. 刪除您建立的 IAM 角色。

a. 前往 https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。

b. 在左側導覽窗格中，選擇 Roles (角色)。

c. 從清單中選取您建立的每個角色 (***myAmazonEKSClusterRole*** ，以及 *myAmazonEKSNodeRole*)。選擇 Delete (刪除)，輸入請求的確認文字，然後選擇 Delete (刪除)。

後續步驟

以下文件主題可協助您擴展叢集的功能。

- 建立叢集的 IAM 主體是唯一可以使用 `kubectl` 或 AWS 管理主控台對 Kubernetes API 伺服器進行呼叫的主體。如果要其他 IAM 主體能夠存取叢集，則需新增這些主體。如需詳細資訊，請參閱the section called “Kubernetes API 存取”及the section called “所需的許可”。
- 將範例應用程式部署至叢集。
- 部署叢集以供生產使用之前，建議您先熟悉叢集和節點的所有設定。建立叢集時，必須進行某些設定 (例如啟用對 Amazon EC2 節點的 SSH 存取)。
- 若要提高叢集的安全性，可將 Amazon VPC 容器聯網介面外掛程式設定為使用服務帳戶的 IAM 角色。

利用 EKS 自動模式自動運作叢集基礎設施

> **Tip**
>
> 註冊即將舉行的 Amazon EKS Auto Mode 研討會。

EKS Auto Mode 會將 Kubernetes 叢集的 AWS 管理延伸到叢集本身之外， AWS 以允許 也設定和管理基礎設施，讓工作負載能夠順暢地運作。您可以委派關鍵基礎設施決策，並使用 的專業知識 AWS 進行day-to-day操作。管理的叢集基礎設施 AWS 包含許多 Kubernetes 功能作為核心元件，而不是附加元件，例如運算自動擴展、Pod 和服務聯網、應用程式負載平衡、叢集 DNS、區塊儲存和 GPU 支援。

若要開始使用，您可部署新的 EKS 自動模式叢集，也可在現有叢集上啟用 EKS Auto Mode。您可以使用 eksctl、 AWS CLI、、EKS APIs 或您偏好的基礎設施即程式碼工具來部署、升級或修改 AWS 管理主控台 EKS Auto Mode 叢集。 infrastructure-as-code

在 EKS 自動模式中，您仍可繼續使用偏好的 Kubernetes 相容工具。EKS Auto Mode 使用遵循最佳實務的 AWS 雲端資源，與 Amazon EC2、Amazon EBS 和 ELB 等 AWS 服務整合。這些資源會自動擴展、最佳化成本，並定期更新，協助最大限度降低營運成本與負荷。

功能

EKS 自動模式提供以下進階功能：

簡化 Kubernetes 叢集管理：EKS 自動模式提供可直接用於生產環境的叢集，並將營運負荷降至最低，藉此簡化 EKS 管理流程。透過 EKS 自動模式，即使缺乏深厚的 EKS 專業知識，您也能安心執行高需求、動態的工作負載。

應用程式可用性：EKS 自動模式依據 Kubernetes 應用程式的需求，動態新增或移除 EKS 叢集中的節點。這能減少手動容量規劃的需求，並確保應用程式的可用性。EKS Auto Mode 也與 Amazon Application Recovery Controller (ARC) 區域轉移整合，以在受損的可用區域中自動停止佈建節點，並在區域轉移事件期間暫停自願中斷。

效率優化：EKS 自動模式的設計目標是在符合您的 NodePool 與工作負載需求所定義的彈性範圍內，最佳化運算成本。它還會終止未使用的執行個體，並將工作負載整合到其他節點，以提升成本效率。如需有關設定叢集以實現成本效益和避免封鎖合併組態的指引，請參閱 EKS Auto Mode 中的成本最佳化。

安全性：EKS Auto Mode 使用 AMIs，這些 AMI 會被視為節點不可變。這類 AMI 會強制執行鎖定的軟體設定、啟用 SELinux 強制存取控制，並提供唯讀的根檔案系統。此外，由 EKS 自動模式啟動的節點有最長 21 天的生命週期 (您可縮短此期限)，期滿後會自動被新節點取代。此機制透過定期更換節點增強安全狀態，與許多客戶已採用的最佳實務一致。

自動化升級：EKS 自動模式會確保您的 Kubernetes 叢集、節點及相關元件保持最新修補程式版本，同時尊重您設定的 Pod 中斷預算 (PDB) 和 NodePool 中斷預算 (NDB)。在最長 21 天的生命週期內，若有阻斷性的 PDB 或其他設定妨礙更新，可能需要手動介入。對於版本轉返，EKS Auto Mode 會在還原控制平面之前自動轉返工作者節點，以符合您設定的中斷控制。如需詳細資訊，請參閱the section called “回復 EKS Auto Mode 叢集”。

受管元件：EKS Auto Mode 包含 Kubernetes 和 AWS 雲端功能作為核心元件，否則必須作為附加元件進行管理。其中包括對 Pod IP 位址指派、Pod 網路政策、本機 DNS 服務、GPU 外掛程式、運作狀態檢查程式及 EBS CSI 儲存的內建支援。

可自訂的 NodePool 與 NodeClass：若您的工作負載需要變更儲存、運算或聯網設定，可透過 EKS 自動模式建立自訂的 NodePool 與 NodeClass。雖然不建議編輯預設的 NodePool 與 NodeClass，但您可在預設設定之外新增自訂的 NodePool 或 NodeClass，以符合特定需求。

自動化元件

EKS 自動模式透過自動化關鍵基礎結構元件，簡化 Amazon EKS 叢集的操作。啟用 EKS 自動模式後，管理 EKS 叢集的任務將進一步減少。

下列是已實現自動化的資料平面元件清單：

- 運算：對於多數工作負載，透過 EKS 自動模式，您無需關注 EKS 叢集運算層面的許多細節。其中包含：
  - 節點：EKS 自動模式節點的設計理念是將其視為專用設備來處理。EKS 自動模式會執行下列作業：
    - 選擇合適的 AMI，該 AMI 已預先設定執行工作負載所需的多項服務，無需介入。
    - 透過 SELinux 強制模式與唯讀根檔案系統，鎖定 AMI 上的檔案存取權；
    - 禁止 SSH 或 SSM 存取，防止直接存取節點；
    - 包含 GPU 支援，提供適用於 NVIDIA 與 Neuron GPU 的獨立核心驅動程式與外掛程式，支援高效能工作負載；
    - 自動處理 EC2 Spot 執行個體中斷通知、EC2 執行個體運作狀態/排程維護事件，以及 EC2 執行個體狀態檢查失敗

  - 自動擴展：根據 Karpenter 自動擴展，EKS Auto Mode 會監控不可排程的 Pod，並讓新節點可以部署來執行這些 Pod。當工作負載終止時，EKS 自動模式會在節點不再需要時動態中斷並終止節點，最佳化資源使用率。
  - 升級：透過掌控節點，EKS 自動模式能更順暢地提供安全性修補程式，以及必要的作業系統與元件升級。這些升級的設計旨在將工作負載的中斷降至最低。EKS 自動模式會強制執行最長 21 天的節點生命週期，確保軟體與 API 保持最新。
- 負載平衡：EKS 自動模式透過與 Amazon Elastic Load Balancing 服務整合，簡化負載平衡流程，自動為 Kubernetes Service 與 Ingress 資源佈建並設定負載平衡器。它支援 Application 與 Network Load Balancer 的進階功能、管理其生命週期，並根據叢集需求進行擴展。此整合提供符合 AWS 最佳實務的生產就緒負載平衡解決方案，可讓您專注於應用程式，而非基礎設施管理。
- 儲存：EKS 自動模式會為您設定暫時性儲存，包括設定磁碟區類型、磁碟區大小、加密政策，以及節點終止時的刪除原則。
- 聯網：EKS 自動模式會自動化 Pod 與服務連線所需的關鍵聯網作業。其中包括支援 IPv4/IPv6 協定，以及使用次要 CIDR 區塊來擴展 IP 位址空間。
- Identity and Access Management：您無需在 EKS 自動模式叢集上安裝 EKS Pod 身分識別代理程式。

有關這些元件的更多資訊，請參閱 the section called “運作方式”。

組態

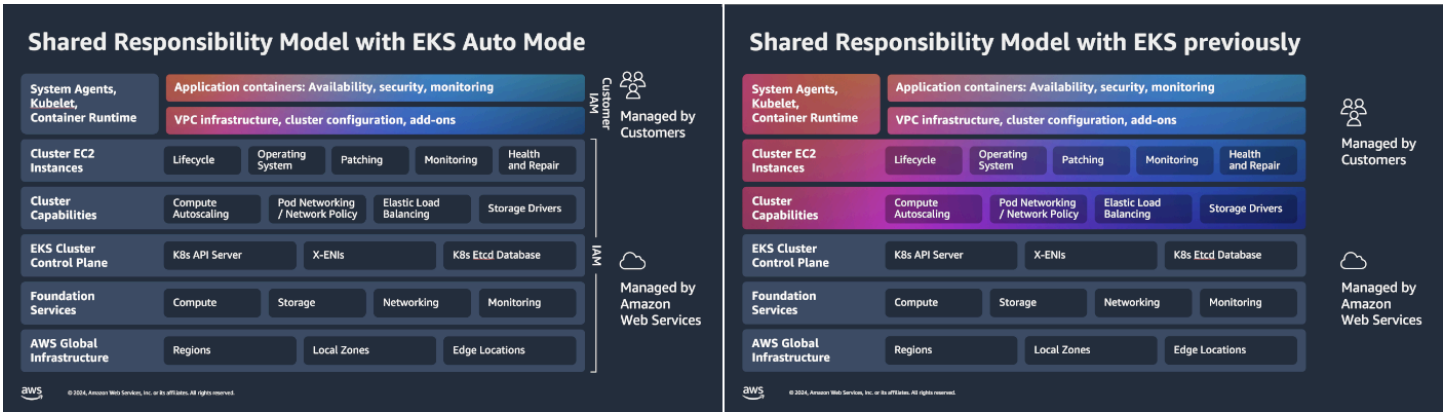
儘管 EKS 自動模式能在無需您介入的情況下，有效管理大部分資料平面服務，但有時您可能需要調整部分服務的運作行為。您可透過下列方式修改 EKS 自動模式叢集的組態：

- Kubernetes DaemonSets：您可以改為使用 Kubernetes DaemonSets，而不是修改節點上安裝的服務。Kubernetes 會管理在叢集中每個節點上執行的 DaemonSets。過這種方式，您可新增用於監控或其他節點管理場景的專用服務。
- 自訂 NodePool 與 NodeClass：預設 NodePool 和 NodeClass 由 EKS 自動模式設定，不建議自行編輯。要自訂節點行為，可針對以下使用案例建立額外的 NodePool 或 NodeClass，例如：
  - 選取特定執行個體類型 (例如，加速處理器或 EC2 Spot 執行個體)。
  - 隔離工作負載，以達成安全性或成本追蹤目的。
  - 設定暫時性儲存設定，如 IOPS、大小和輸送量。
- 負載平衡：EKS 自動模式會將部分服務 (如負載平衡) 作為 Kubernetes 物件執行，這類服務可直接在 EKS 自動模式叢集上進行設定。

如需設定 EKS 自動模式的詳細資訊，請參閱 the section called “設定”。

共同責任模型

AWS 共同責任模型定義 AWS 和 客戶之間的安全與合規責任。下列影像和文字會比較和對比 EKS Auto Mode AWS 和 EKS 標準模式之間的客戶和責任差異。



EKS 自動模式將 Kubernetes 基礎結構的大部分共用責任從客戶轉移至 AWS。透過 EKS Auto Mode，AWS 承擔更多雲端安全的責任，這曾經是客戶的責任，現在已共同承擔。客戶現在可以更專注於其應用程式，同時 AWS 管理基礎基礎設施。

客戶責任

在 EKS 自動模式下，客戶仍需負責應用程式容器的相關事務，包括容器的可用性、安全性與監控。此外，客戶也需掌控 VPC 基礎結構與 EKS 叢集組態的核心設定。此模型可讓客戶專注於應用程式特定的考量，同時委派叢集基礎設施管理給 AWS。每個節點的選用功能可以透過 AWS 附加元件包含在叢集中。

AWS 責任

使用 EKS Auto Mode 時， AWS 擴展其責任，包括管理幾個額外的關鍵元件，與已在未使用 Auto Mode 的 EKS 叢集中管理的元件相比。具體而言，EKS 自動模式會接管以下事務：由其啟動的 EC2 執行個體之設定、管理、安全性與擴展，以及叢集在負載平衡、IP 位址管理、聯網原則與區塊儲存方面的功能。下列元件由 AWS 以 EKS Auto 模式管理：

- 自動模式啟動的 EC2 執行個體：使用 Amazon EC2 受管執行個體 AWS 處理節點的完整生命週期。EC2 受管執行個體負責作業系統組態、修補、監控與運作狀態維護。在此模型中，執行個體本身及其上執行的來賓作業系統均由 AWS負責。這些節點使用專為執行容器而最佳化的 Bottlerocket AMI 變體。Bottlerocket AMI 具備軟體鎖定、不可變根檔案系統，以及安全網路存取 (防止透過 SSH

或 SSM 直接連線) 等特性。從 2026 年 4 月 22 日開始，EC2 主控台檢視和 API 清單操作預設會隱藏新的受管執行個體和相關聯的資源。您可以使用受管資源可見性設定來調整此行為。如需詳細資訊，請參閱the section called “受管執行個體”。

- Cluster Capabilities： AWS 管理運算自動擴展、具有網路政策強制執行的 Pod 網路、Elastic Load Balancing 整合和儲存驅動程式組態。
- 叢集控制平面： AWS 繼續管理 Kubernetes API 伺服器、跨帳戶 ENIs 和 等資料庫，如同標準 EKS。
- Foundation Services 和 Global Infrastructure： AWS 負責基礎運算、儲存、聯網和監控服務，以及區域、本機區域和節點的全球基礎設施。

使用 Amazon EKS 自動模式建立叢集

本章節闡釋如何使用各種工具和介面建立已啟用自動模式的 Amazon EKS 叢集。自動模式透過自動設定與管理叢集的運算、聯網及儲存基礎結構，簡化了叢集建立過程。您將了解如何使用 AWS CLI、AWS 管理主控台 或 eksctl 命令列工具來建立自動模式叢集。

> **Note**
>
> EKS 自動模式需要 Kubernetes 版本 1.29 或以上。

請根據您的需求選擇合適的工具：AWS 管理主控台 提供視覺化介面，非常適合了解 EKS 自動模式功能和建立單獨叢集。AWS CLI 最適合用於指令碼編寫和自動化任務，特別是將叢集建立整合到現有工作流程或 CI/CD 管道時。eksctl CLI 提供了 Kubernetes 原生的體驗，推薦給熟悉 Kubernetes 工具並希望使用合理的預設值來簡化命令列操作的使用者。

開始之前，請確定您已安裝並設定必要先決條件，包括建立 EKS 叢集的適當 IAM 許可。要了解如何安裝 CLI 工具，例如 `kubectl`、`aws` 和 `eksctl`，請參閱 設定。

您可以使用 AWS CLI、AWS 管理主控台 或 eksctl CLI 來建立具有 Amazon EKS 自動模式的叢集。

主題

- 使用 eksctl CLI 建立 EKS 自動模式叢集
- 使用 CLI AWS 建立 EKS 自動模式叢集
- 使用 建立 EKS 自動模式叢集 AWS 管理主控台

使用 eksctl CLI 建立 EKS 自動模式叢集

本主題展示如何使用 eksctl 命令列介面 (CLI) 建立 Amazon EKS 自動模式叢集。您可以透過執行單一 CLI 命令或套用 YAML 組態檔案來建立自動模式叢集。兩種方法提供相同的功能，其中 YAML 方法能對叢集設定提供更細緻的控制。

eksctl CLI 透過處理基礎 AWS 資源建立和組態，簡化建立和管理 EKS Auto Mode 叢集的程序。在繼續之前，請確定您已在本機電腦上設定必要的 AWS 登入資料和許可。本指南假設您熟悉基本的 Amazon EKS 概念，並且已安裝必要的 CLI 工具。

> **Note**
>
> 您必須安裝 `0.195.0` 或更高版本的 eksctl。如需詳細資訊，請參閱 GitHub 上的 eksctl 版本。

使用 CLI 命令建立 EKS 自動模式叢集

您必須已安裝 `aws` 和 `eksctl` 工具。您必須以足夠的許可登入 AWS CLI，才能管理 AWS 資源，包括：EC2 執行個體、EC2 聯網、EKS 叢集和 IAM 角色。如需詳細資訊，請參閱設定。

執行下列命令以使用 eksctl 建立新的 EKS Auto Mode 叢集：

```
eksctl create cluster --name=<cluster-name> --enable-auto-mode
```

使用 YAML 檔案建立 EKS 自動模式叢集

您必須已安裝 `aws` 和 `eksctl` 工具。您必須以足夠的許可登入 AWS CLI，才能管理 AWS 資源，包括：EC2 執行個體、EC2 聯網、EKS 叢集和 IAM 角色。如需詳細資訊，請參閱設定。

檢閱以下範例 ClusterConfig 資源中的 EKS 自動模式組態選項。有關完整的 ClusterConfig 規格，請參閱 eksctl 文件。

AWS 建議啟用 EKS 自動模式。如果這是您第一次建立 EKS 自動模式叢集，請不要指定 `nodeRoleARN`，以便為 EKS 自動模式建立節點 IAM 角色。如果您的 AWS 帳戶中已有節點 IAM 角色， AWS 建議重複使用它。

AWS 建議不要為 指定任何值`nodePools`。EKS 自動模式會建立預設節點集區。您可使用 Kubernetes API 來建立其他節點集區。

```
# cluster.yaml
```

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: <cluster-name>
  region: <aws-region>

iam:
  # ARN of the Cluster IAM Role
  # optional, eksctl creates a new role if not supplied
  # suggested to use one Cluster IAM Role per account
  serviceRoleARN: <arn-cluster-iam-role>

autoModeConfig:
  # defaults to false
  enabled: boolean
  # optional, defaults to [general-purpose, system].
  # suggested to leave unspecified
  # To disable creation of nodePools, set it to the empty array ([]).
  nodePools: []string
  # optional, eksctl creates a new role if this is not supplied
  # and nodePools are present.
  nodeRoleARN: string
```

將 `ClusterConfig` 檔案儲存為 `cluster.yaml`，並使用下列命令來建立叢集：

```
eksctl create cluster -f cluster.yaml
```

使用 CLI AWS 建立 EKS 自動模式叢集

EKS 自動模式叢集可自動執行運算、儲存和聯網的常規叢集管理任務。例如，EKS 自動模式叢集會自動檢測何時需要額外節點，並佈建新的 EC2 執行個體以滿足工作負載需求。

本主題會引導您使用 CLI 建立新的 EKS AWS 自動模式叢集，並選擇性地部署範例工作負載。

先決條件

- 在您裝置上安裝和設定的最新版本 AWS 命令列界面 (AWS CLI)。若要檢查您目前的版本，請使用 `aws --version`。若要安裝最新版本，請參閱《 AWS 命令列界面使用者指南》中的使用 aws 設定安裝 和快速組態。
  - 使用足夠的 IAM 許可登入 CLI，以建立 AWS 資源，包括 IAM 政策、IAM 角色和 EKS 叢集。

- 裝置上安裝的 kubectl 命令列工具。 AWS 建議您使用與 EKS 叢集 Kubernetes 版本相同的 kubectl 版本。若要安裝或升級 kubectl，請參閱 the section called “設定 kubectl 和 eksctl”。

指定 VPC 子網路

Amazon EKS Auto Mode 會將節點部署到 VPC 子網路。建立 EKS 叢集時，您必須指定將部署節點的 VPC 子網路。您可以使用 AWS 帳戶中的預設 VPC 子網路，或為關鍵工作負載建立專用 VPC。

- AWS 建議為您的叢集建立專用 VPC。了解如何 the section called “建立 VPC”。
- EKS 主控台可協助建立新 VPC。了解如何 the section called “管理主控台”。
- 或者，您可以使用 AWS 您帳戶的預設 VPC。使用下列說明來尋找子網路 ID。

尋找預設 VPC 的子網路 ID

使用 AWS CLI：

1. 執行以下命令以列出預設 VPC 及其子網路：

```
aws ec2 describe-subnets --filters "Name=vpc-id,Values=$(aws ec2 describe-vpcs
 --query 'Vpcs[?IsDefault==`true`].VpcId' --output text)" --query 'Subnets[*].
{ID:SubnetId,AZ:AvailabilityZone}' --output table
```

2. 儲存輸出並記下子網路 ID。

   輸出範例：

```
----------------------------------------
|             DescribeSubnets          |
----------------------------------------
|   SubnetId        |   AvailabilityZone  |
|-------------------|---------------------|
|   subnet-012345678 |   us-west-2a        |
|   subnet-234567890 |   us-west-2b        |
|   subnet-345678901 |   us-west-2c        |
----------------------------------------
```

EKS 自動模式叢集的 IAM 角色

叢集 IAM 角色

EKS Auto Mode 需要叢集 IAM 角色才能在 AWS 帳戶中執行動作，例如佈建新的 EC2 執行個體。您必須建立此角色，才能授予 EKS 必要的許可。 AWS 建議將下列 AWS 受管政策連接至叢集 IAM 角色：

- AmazonEKSComputePolicy
- AmazonEKSBlockStoragePolicyV2
- AmazonEKSLoadBalancingPolicy
- AmazonEKSNetworkingPolicy
- AmazonEKSClusterPolicy

節點 IAM 角色

當您建立 EKS 自動模式叢集時，需要指定一個節點 IAM 角色。當 EKS 自動模式建立節點以處理待處理的工作負載時，每個新的 EC2 執行個體節點都會指派節點 IAM 角色。此角色允許節點與 EKS 通訊，但通常不會由在節點上執行的工作負載存取。

如果您想要將許可授予節點上執行的工作負載，請使用 EKS Pod 身分識別。如需詳細資訊，請參閱the section called “Pod 身分識別”。

您必須建立此角色並連接下列 AWS 受管政策：

- AmazonEKSWorkerNodeMinimalPolicy
- AmazonEC2ContainerRegistryPullOnly

服務連結角色

EKS 自動模式還需要一個服務連結角色，該角色由 AWS自動建立與設定。更多資訊請參閱 AWSServiceRoleForAmazonEKS。

建立 EKS 自動模式叢集 IAM 角色

步驟 1：建立信任政策

建立信任政策，允許 Amazon EKS 服務擔任該角色。將政策儲存為 `trust-policy.json`：

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ]
    }
  ]
}
```

步驟 2：建立 IAM 角色

使用信任政策建立叢集 IAM 角色：

```
aws iam create-role \
    --role-name AmazonEKSAutoClusterRole \
    --assume-role-policy-document file://trust-policy.json
```

步驟 3：記下角色 ARN

擷取並儲存新角色的 ARN，供後續步驟使用：

```
aws iam get-role --role-name AmazonEKSAutoClusterRole --query "Role.Arn" --output text
```

步驟 4：附加所需政策

將下列 AWS 受管政策連接至叢集 IAM 角色，以授予必要的許可：

AmazonEKSClusterPolicy：

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws: iam::aws:policy/AmazonEKSClusterPolicy
```

AmazonEKSComputePolicy：

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws: iam::aws:policy/AmazonEKSComputePolicy
```

AmazonEKSBlockStoragePolicyV2：

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws: iam::aws:policy/AmazonEKSBlockStoragePolicyV2
```

AmazonEKSLoadBalancingPolicy：

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws: iam::aws:policy/AmazonEKSLoadBalancingPolicy
```

AmazonEKSNetworkingPolicy：

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws: iam::aws:policy/AmazonEKSNetworkingPolicy
```

建立 EKS 自動模式節點 IAM 角色

步驟 1：建立信任政策

建立信任政策，允許 Amazon EKS 服務擔任該角色。將政策儲存為 `node-trust-policy.json`：

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

步驟 2：建立節點 IAM 角色

使用上一步中的 node-trust-policy.json 檔案來定義哪些實體可擔任該角色。執行下列命令以建立節點 IAM 角色：

```
aws iam create-role \
    --role-name AmazonEKSAutoNodeRole \
    --assume-role-policy-document file://node-trust-policy.json
```

步驟 3：記下角色 ARN

建立角色後，擷取並儲存節點 IAM 角色的 ARN。您在後續步驟中將需要此 ARN。使用以下命令來取得 ARN：

```
aws iam get-role --role-name AmazonEKSAutoNodeRole --query "Role.Arn" --output text
```

步驟 4：附加所需政策

將下列 AWS 受管政策連接至節點 IAM 角色，以提供必要的許可：

AmazonEKSWorkerNodeMinimalPolicy：

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoNodeRole \
    --policy-arn arn:aws: iam::aws:policy/AmazonEKSWorkerNodeMinimalPolicy
```

AmazonEC2ContainerRegistryPullOnly：

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoNodeRole \
    --policy-arn arn:aws: iam::aws:policy/AmazonEC2ContainerRegistryPullOnly
```

建立 EKS 自動模式叢集

概觀

若要使用 CLI AWS 建立 EKS 自動模式叢集，您將需要下列參數：

- `cluster-name`：叢集的名稱。

- k8s-version：Kubernetes 版本 (如 1.31)。
- subnet-ids：先前步驟中識別的子網路 ID。
- cluster-role-arn：叢集 IAM 角色的 ARN。
- node-role-arn：節點 IAM 角色的 ARN。

預設叢集組態

在建立叢集之前，檢閱這些預設值與功能：

- nodePools：EKS 自動模式包含通用與系統預設節點集區。進一步了解節點集區。

注意：EKS 自動模式中的節點集區與 Amazon EKS 受管節點群組不同，但可以在同一叢集中共存。

- computeConfig.enabled：自動化常規運算任務，例如建立與刪除 EC2 執行個體。
- kubernetesNetworkConfig.elasticLoadBalancing.enabled：自動化負載平衡任務，包括建立與刪除 Elastic Load Balancer。
- storageConfig.blockStorage.enabled：自動化儲存任務，如建立與刪除 Amazon EBS 磁碟區。
- accessConfig.authenticationMode：需要 EKS 存取項目。深入了解 EKS 身分驗證模式。

設定環境變數

執行 create-cluster命令之前，請設定下列環境變數。將預留位置值取代為您自己的值：

```
export AWS_REGION=us-east-1
export CLUSTER_NAME=my-auto-cluster
export K8S_VERSION=1.35
export CLUSTER_ROLE_ARN=arn:aws:iam::111122223333:role/AmazonEKSAutoClusterRole
export NODE_ROLE_ARN=arn:aws:iam::111122223333:role/AmazonEKSAutoNodeRole
export SUBNETS_JSON='["subnet-ExampleID1","subnet-ExampleID2","subnet-ExampleID3"]'
```

> **Note**
>
> 使用上一個步驟的子網路 IDs設定 SUBNETS_JSON變數。您必須以單引號包裝值，以防止 shell 將括號和引號解譯為特殊字元。

驗證變數（選用）

確認您的變數設定正確：

```
echo "Region: ${AWS_REGION}"
echo "Cluster Name: ${CLUSTER_NAME}"
echo "K8s Version: ${K8S_VERSION}"
echo "Cluster Role ARN: ${CLUSTER_ROLE_ARN}"
echo "Node Role ARN: ${NODE_ROLE_ARN}"
echo "Subnets JSON: ${SUBNETS_JSON}"
```

執行命令

使用下列命令建立叢集：

```
aws eks create-cluster \
  --region ${AWS_REGION} \
  --cli-input-json \
  "{
      \"name\": \"${CLUSTER_NAME}\",
      \"version\": \"${K8S_VERSION}\",
      \"roleArn\": \"${CLUSTER_ROLE_ARN}\",
      \"resourcesVpcConfig\": {
        \"subnetIds\": ${SUBNETS_JSON},
        \"endpointPublicAccess\": true,
        \"endpointPrivateAccess\": true
      },
      \"computeConfig\": {
        \"enabled\": true,
        \"nodeRoleArn\":\"${NODE_ROLE_ARN}\",
        \"nodePools\": [\"general-purpose\", \"system\"]
      },
      \"kubernetesNetworkConfig\": {
        \"elasticLoadBalancing\": {
          \"enabled\": true
        }
      },
      \"storageConfig\": {
        \"blockStorage\": {
          \"enabled\": true
        }
      },
      \"accessConfig\": {
```

```
        \"authenticationMode\": \"API\"
      }
    }"
```

檢查叢集狀態

步驟 1：確認叢集建立

執行下列命令，以檢查您的叢集狀態。建立叢集通常需要約 15 分鐘的時間：

```
aws eks describe-cluster --name "${CLUSTER_NAME}" --output json
```

步驟 2：更新 kubeconfig

叢集準備就緒後，請更新本機 kubeconfig 檔案，以使 `kubectl` 與叢集通訊。此組態使用 AWS CLI 進行身分驗證。

```
aws eks update-kubeconfig --name "${CLUSTER_NAME}"
```

步驟 3：確認節點集區

使用以下命令列出叢集中的節點集區：

```
kubectl get nodepools
```

後續步驟

- 了解如何將範例工作負載部署至新的 EKS 自動模式叢集。

使用 建立 EKS 自動模式叢集 AWS 管理主控台

在 中建立 EKS Auto Mode 叢集 AWS 管理主控台 所需的組態比其他選項少。EKS 與 AWS IAM 和 VPC 網路整合，協助您建立與 EKS 叢集相關聯的資源。

您在主控台中有兩個建立叢集的選項：

- 快速組態 (使用 EKS 自動模式)
- 自訂組態

在本主題中，您將了解如何使用快速組態選項建立 EKS 自動模式叢集。

使用快速組態選項建立 EKS Auto Mode 叢集

您必須 AWS 管理主控台 以足夠的許可登入 來管理 AWS 資源，包括：EC2 執行個體、EC2 聯網、EKS 叢集和 IAM 角色。

1. 導覽至 EKS 主控台
2. 選擇建立叢集
3. 確認已選取快速組態選項
4. 確定以下值，或使用測試叢集的預設值。
   - 叢集名稱
   - Kubernetes 版本
5. 選取叢集 IAM 角色。如果這是您第一次建立 EKS 自動模式叢集，請使用建立建議的角色選項。
   - 或者，您可以為所有 EKS Auto Mode 叢集重複使用 AWS 帳戶中的單一叢集 IAM 角色。
   - 叢集 IAM 角色包含 EKS 自動模式管理資源 (包括 EC2 執行個體、EBS 磁碟區和 EC2 負載平衡器) 所需的許可。
   - 建立建議角色選項會以建議值預先填入所有欄位。選取下一步，再選取建立。角色將使用建議 `AmazonEKSAutoClusterRole` 名稱。
   - 如果您最近建立了新角色，請使用重新整理圖示，重新載入角色選取下拉式清單。
6. 選取節點 IAM 角色。如果這是您第一次建立 EKS 自動模式叢集，請使用建立建議的角色選項。
   - 或者，您可以為所有 EKS Auto Mode 叢集重複使用 AWS 帳戶中的單一節點 IAM 角色。
   - 節點 IAM 角色包含自動模式節點連線叢集所需的許可。節點 IAM 角色必須包含為容器擷取 ECR 映像的許可。
   - 建立建議角色選項會以建議值預先填入所有欄位。選取下一步，再選取建立。角色將使用建議 `AmazonEKSAutoNodeRole` 名稱。
   - 如果您最近建立了新角色，請使用重新整理圖示，重新載入角色選取下拉式清單。
7. 選取您的 EKS 自動模式叢集的 VPC。選擇建立 VPC，為 EKS 建立新的 VPC，或選擇您先前為 EKS 建立的 VPC。
   - 如果您使用 VPC 主控台建立新的 VPC， AWS 建議您為每個可用區域建立至少一個 NAT 閘道。否則，您可使用所有其他預設值。
   - 如需 IPv6 叢集需求的更多資訊和詳細資訊，請參閱 the section called “建立 VPC”。
8. (選用) EKS 自動模式會自動為您選擇的 VPC 填入私有子網路。您可移除不需要的子網路。
   - EKS 會遵循最佳實務，自動從 VPC 中選取私有子網路。您可選擇從 VPC 中選取其他子網路，例如公有子網路。

9. (選用) 選取檢視快速組態預設值，以檢閱新叢集的所有組態值。資料表指出，在建立叢集之後，某些值無法編輯。

10選取 Create cluster (建立叢集)。請注意，建立叢集可能需要 15 分鐘時間才能完成。

後續步驟

- 了解如何將範例工作負載部署到您的 EKS 自動模式叢集

在現有的 EKS 叢集上啟用 EKS 自動模式

您可以在現有的 EKS 叢集上啟用 EKS 自動模式。

AWS 支援下列移轉：

- 從 Karpenter 移轉至 EKS 自動模式節點。如需詳細資訊，請參閱the section called “從 Karpenter 移轉”。
- 從 EKS 受管節點群組移轉至 EKS 自動模式節點。如需詳細資訊，請參閱the section called “從 MNG 移轉”。
- 從 EKS Fargate 移轉至 EKS 自動模式。如需詳細資訊，請參閱the section called “從 Fargate 移轉”。

AWS 不支援下列移轉：

- 將磁碟區從 EBS CSI 控制器 (使用 Amazon EKS 附加元件) 移轉至 EKS 自動模式 EBS CSI 控制器 (由 EKS 自動模式管理)。使用其中一個製作的 PVC 無法由另一個掛載，因為它們使用的是兩個不同的 Kubernetes 磁碟區佈建程式。
  - `eks-auto-mode-ebs-migration-tool` (AWS Labs 專案） 可在標準 EBS CSI StorageClass (`ebs.csi.aws.com`) 和 EKS Auto EBS CSI StorageClass () 之間進行遷移`ebs.csi.eks.amazonaws.com`。請注意，移轉需要刪除並重新建立現有的 PersistentVolumeClaim/PersistentVolume 資源，因此在實作之前，請務必在非生產環境中進行驗證。
- 將負載平衡器從 AWS Load Balancer控制器遷移至 EKS Auto 模式

  您可以在 Amazon EKS Auto Mode 叢集上安裝 AWS Load Balancer控制器。使用 `IngressClass` 或 `loadBalancerClass` 選項，將服務和傳入資源與 Load Balancer 控制器或 EKS 自動模式建立

關聯。如需規範性指引，請參閱 Migrate NGINX Ingress Controllers when enabling Amazon EKS Auto Mode

- 使用替代 CNI 或其他不支援的網路組態移轉 EKS 叢集

移轉參考

使用下列移轉參考，將 Kubernetes 資源設定為由自我管理控制器或 EKS 自動模式擁有。

功能	資源	欄位	自我管理	EKS 自動模式
區塊儲存	StorageClass	provisioner	ebs.csi.aws.com	ebs.csi.eks.amazonaws.com
Load balancing	Service	loadBalancerClass	service.k8s.aws/nlb	eks.amazonaws.com/nlb
Load balancing	IngressClass	controller	ingress.k8s.aws/alb	eks.amazonaws.com/alb
Load balancing	IngressClassParams	apiversion	elbv2.k8s.aws/v1beta1	eks.amazonaws.com/v1
Load balancing	TargetGroupBinding	apiversion	elbv2.k8s.aws/v1beta1	eks.amazonaws.com/v1
運算	NodeClass	apiVersion	karpenter.sh/v1	eks.amazonaws.com/v1

移轉 EBS 磁碟區

將工作負載移轉至 EKS 自動模式時，由於 CSI 驅動程式佈建程式不同，您需要處理 EBS 磁碟區移轉：

- EKS 自動模式佈建程式：`ebs.csi.eks.amazonaws.com`
- 開放原始碼 EBS CSI 佈建程式：`ebs.csi.aws.com`

請遵循下列步驟，移轉您的持久性磁碟區：

1. 修改磁碟區保留政策：將現有的 PersistentVolume (PV) 變更為 `persistentVolumeReclaimPolicyRetain`，以確保不會刪除基礎 EBS 磁碟區。
2. 從 Kubernetes 移除 PV：刪除舊 PV 資源，同時保持實際 EBS 磁碟區原封不動。
3. 使用靜態佈建建立新的 PV：建立參考相同 EBS 磁碟區且可與目標 CSI 驅動程式搭配使用的新 PV。
4. 繫結至新的 PVC：使用 `volumeName` 欄位建立專門參考 PV 的新 PVC。

考量事項

- 在開始此移轉之前，請確保您的應用程式已停止。
- 在開始移轉程序之前，備份您的資料。
- 每個持久性磁碟區皆需執行此程序。
- 工作負載必須更新，才能使用新的 PVC。

移轉負載平衡器

您無法將現有的負載平衡器從自我管理 AWS 負載平衡器控制器直接轉移到 EKS Auto 模式。相反地，您必須實作藍綠部署策略。這包括在受管控制器下建立新的負載平衡器時，維護現有的負載平衡器組態。

若要最大限度地減少服務中斷，我們建議採用 DNS 型流量轉移方法。首先，使用 EKS 自動模式建立新的負載平衡器，同時保持現有組態正常運作。然後，使用 DNS 路由 (例如 Route 53)，逐步將流量從舊負載平衡器轉移至新的負載平衡器。成功移轉流量並驗證新組態後，您即可停用舊負載平衡器和自我管理控制器。

在現有叢集上啟用 EKS 自動模式

本主題描述如何在現有的 Amazon EKS 叢集上啟用 Amazon EKS Auto Mode。在現有叢集上啟用自動模式需要更新 IAM 許可，並設定 EKS 自動模式核心設定。一旦啟用，您可以開始移轉現有的運算工作負載，以利用自動模式的簡化操作和自動化基礎結構管理。

> **Important**
>
> 在啟用 EKS 自動模式之前，請確認您已安裝特定 Amazon EKS 附加元件的最低所需版本。如需詳細資訊，請參閱the section called “所需的附加元件版本”。

開始之前，請確認您擁有 Amazon EKS 叢集的管理員存取權限及修改 IAM 角色的許可。本主題中的步驟會引導您使用 AWS 管理主控台 或 CLI AWS 來啟用自動模式。

AWS 管理主控台

您必須使用管理 IAM、EKS 和 EC2 資源的許可登入 AWS 主控台。

> **Note**
>
> EKS 叢集的叢集 IAM 角色在叢集建立後無法變更。EKS 自動模式需要此角色的額外許可。您必須在目前角色上附加其他政策。

更新叢集 IAM 角色

1. 在 AWS 管理主控台中開啟您的叢集概觀頁面。
2. 在叢集 IAM 角色 ARN 下，選取在 IAM 中檢視。
3. 從新增許可下拉式清單中，選取連接政策。
4. 使用搜尋方塊，尋找並選取以下政策：
   - `AmazonEKSComputePolicy`
   - `AmazonEKSBlockStoragePolicyV2`
   - `AmazonEKSLoadBalancingPolicy`
   - `AmazonEKSNetworkingPolicy`
   - `AmazonEKSClusterPolicy`
5. 選取新增許可

6. 從信任關係索引標籤，選取編輯信任政策
7. 插入下列叢集 IAM 角色信任政策，並選取更新政策

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ]
    }
  ]
}
```

啟用 EKS 自動模式

1. 在 AWS 管理主控台中開啟您的叢集概觀頁面。
2. 在 EKS 自動模式下，選取管理
3. 將 EKS 自動模式切換為開啟。
4. 從 EKS 節點集區下拉式清單中，選取要建立的預設節點集區。
   - 進一步了解 EKS 自動模式的節點集區。如需詳細資訊，請參閱the section called “建立節點集區”。
5. 如果您先前已在此 AWS 帳戶中建立 EKS 自動模式節點 IAM 角色，請在節點 IAM 角色下拉式清單中選取該角色。如果您之前尚未建立此角色，請選取建立建議的角色，並依步驟操作。

AWS CLI

先決條件

- 現有 EKS 叢集的叢集 IAM 角色必須包含 EKS Auto Mode 的足夠許可，例如下列政策：
  - `AmazonEKSComputePolicy`
  - `AmazonEKSBlockStoragePolicyV2`

  - AmazonEKSLoadBalancingPolicy
  - AmazonEKSNetworkingPolicy
  - AmazonEKSClusterPolicy
- 叢集 IAM 角色必須具有更新的信任政策，包括 sts:TagSession 動作。如需有關建立 IAM 角色的詳細資訊，請參閱 the section called “ AWS CLI”。
- aws 已安裝、登入的 CLI 和足夠的版本。您必須具有管理 IAM、EKS 和 EC2 資源的許可。如需詳細資訊，請參閱設定。

程序

使用以下命令，在現有叢集上啟用 EKS 自動模式。

> **Note**
>
> 運算、區塊儲存和負載平衡功能必須在同一請求中全部啟用或停用。

```
aws eks update-cluster-config \
 --name $CLUSTER_NAME \
 --compute-config enabled=true \
 --kubernetes-network-config '{"elasticLoadBalancing":{"enabled": true}}' \
 --storage-config '{"blockStorage":{"enabled": true}}'
```

所需的附加元件版本

如果您計劃在現有叢集上啟用 EKS 自動模式，您可能需要更新特定附加元件。版本備註：

- 這僅適用於轉換為 EKS 自動模式的現有叢集。
- 啟用 EKS 自動模式時建立的新叢集無需這些更新。

如果您安裝了以下任何附加元件，請確保其至少為指定的最低版本：

附加元件名稱	所需的最低版本
Kubernetes 專用 Amazon VPC CNI 外掛程式	v1.19.0-eksbuild.1
Kube-proxy	• v1.26.15-eksbuild.19

附加元件名稱	所需的最低版本
	• v1.27.16-eksbuild.14 • v1.28.15-eksbuild.4 • v1.29.10-eksbuild.3 • v1.30.6-eksbuild.3 • v1.31.2-eksbuild.3
Amazon EBS CSI 驅動程式	v1.37.0-eksbuild.1
CSI 快照控制器	v8.1.0-eksbuild.2
EKS Pod 身分識別代理程式	v1.3.4-eksbuild.1

如需詳細資訊，請參閱the section called “更新附加元件”。

後續步驟

- 若要遷移受管節點群組工作負載，請參閱 the section called “從 MNG 移轉”。
- 要從自我管理 Karpenter 移轉，請參閱 the section called “從 Karpenter 移轉”。

使用 kubectl 從 Karpenter 移轉至 EKS 自動模式

本主題將逐步引導您使用 kubectl 將工作負載從 Karpenter 移轉至 Amazon EKS 自動模式。移轉可以逐步進行，讓您能夠按照自己的步調移動工作負載，同時在整個轉換期間保持叢集穩定性和應用程式可用性。

下面概述的逐步方法使您能夠在移轉期間同時執行 Karpenter 和 EKS 自動模式。這種雙重操作策略有助於確保平穩轉換，讓您可以在完全停用 Karpenter 之前，驗證工作負載在 EKS 自動模式上的行為。您可以單獨或分組移轉應用程式，提供靈活性以適應您的特定運營要求與風險承受能力。

先決條件

開始移轉之前，請確保您已具備下列項目：

- 在叢集上安裝了 Karpenter v1.1 或以上版本。如需詳細資訊，請參閱 Karpenter 文件中的升級至版本 1.1.0 及以上。
- 已安裝並將 `kubectl` 連接到您的叢集。如需詳細資訊，請參閱設定。

本主題假設您熟悉 Karpenter 及 NodePools。如需詳細資訊，請參閱 Karpenter 文件。

> **Note**
>
> 有些 CRDs (例如 `nodepools.karpenter.sh`和 `nodeclaims.karpenter.sh`) 是在 EKS Auto Mode 和 Karpenter 之間共用。請勿在遷移期間更新或修改這些 CRDs。

步驟 1：在叢集上啟用 EKS 自動模式

使用 CLI 或 管理主控台在現有叢集上啟用 AWS EKS 自動模式。如需詳細資訊，請參閱the section called “在叢集上啟用”。

> **Note**
>
> 啟用 EKS 自動模式時，切勿在轉換期間的此階段啟用 `general purpose` nodepool。此節點集區不具選擇性。
> 如需詳細資訊，請參閱the section called “檢閱內建的節點集區”。

步驟 2：建立帶有污點的 EKS 自動模式 NodePool

為 EKS 自動模式建立帶有污點的新 NodePool。這確保了現有的 Pod 不會自動排程到新的 EKS 自動模式節點上。此節點集區使用 EKS 自動模式內建的 `default` `NodeClass`。如需詳細資訊，請參閱the section called “建立節點類別”。

帶有污點的節點集區範例：

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: eks-auto-mode
spec:
  template:
    spec:
      requirements:
        - key: "eks.amazonaws.com/instance-category"
          operator: In
          values: ["c", "m", "r"]
      nodeClassRef:
        group: eks.amazonaws.com
```

```
        kind: NodeClass
        name: default
      taints:
        - key: "eks-auto-mode"
          effect: "NoSchedule"
```

更新節點集區的需求，以符合您要從中移轉的 Karpenter 組態。您至少需要一個需求。

步驟 3：更新工作負載以進行移轉

識別並更新您想要移轉至 EKS 自動模式的工作負載。向這些工作負載新增容差和節點選取器：

```
apiVersion: apps/v1
kind: Deployment
spec:
  template:
    spec:
      tolerations:
      - key: "eks-auto-mode"
        effect: "NoSchedule"
      nodeSelector:
        eks.amazonaws.com/compute-type: auto
```

此變更可讓工作負載排程至新的 EKS 自動模式節點。

EKS 自動模式使用與 Karpenter 不同的標籤。與 EC2 受管執行個體相關的標籤以 `eks.amazonaws.com` 開頭。如需詳細資訊，請參閱the section called “建立節點集區”。

步驟 4：逐步移轉工作負載

針對要移轉的每個工作負載重複步驟 3。這使您可根據需求和風險承受能力，單獨或分組移動工作負載。

步驟 5：移除原始 Karpenter NodePool

所有工作負載移轉完成後，即可移除原始的 Karpenter NodePool：

```
kubectl delete nodepool <original-nodepool-name>
```

步驟 6：從 EKS 自動模式 NodePool 移除污點 (選用)

如果您希望 EKS 自動模式成為新工作負載的預設值，可以從 EKS 自動模式 NodePool 中移除污點：

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: eks-auto-mode
spec:
  template:
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default
      # Remove the taints section
```

步驟 7：從工作負載移除節點選取器 (選用)

如果您已從 EKS 自動模式 NodePool 移除污點，可以選擇從工作負載移除節點選取器，因為 EKS 自動模式現在是預設值：

```
apiVersion: apps/v1
kind: Deployment
spec:
  template:
    spec:
      # Remove the nodeSelector section
      tolerations:
      - key: "eks-auto-mode"
        effect: "NoSchedule"
```

步驟 8：從叢集解除安裝 Karpenter

移除 Karpenter 的步驟取決於您的安裝方式。如需詳細資訊，請參閱 Karpenter 安裝說明。

從 EKS 受管節點群組移轉至 EKS 自動模式

當您將 Amazon EKS 叢集轉換為使用 EKS 自動模式時，可以使用 eksctl CLI 工具從受管節點群組順暢地移轉現有工作負載。此程序能確保應用程式持續可用，同時 EKS 自動模式會最佳化您的運算資源。移轉可以在對執行中應用程式造成最小干擾的情況下進行。

本主題將引導您完成從現有受管節點群組耗盡 Pod，並讓 EKS 自動模式在新佈建的執行個體上重新排程它們的步驟。透過遵循此程序，您可在整個移轉過程中保持應用程式可用性的同時，利用 EKS 自動模式的智慧型工作負載整合功能。

先決條件

- 已啟用 EKS 自動模式的叢集
- 已安裝並將 `eksctl` CLI 連接到您的叢集。如需詳細資訊，請參閱 設定。
- 叢集上未安裝 Karpenter。

程序

使用以下 `eksctl` CLI 命令來開始從現有受管節點群組執行個體排空 Pod。EKS 自動模式將建立新節點，以支援被取代的 Pod。

```
eksctl delete nodegroup --cluster=<clusterName> --name=<nodegroupName>
```

您需要為叢集中的每個受管節點群組執行此命令。

有關此命令的更多資訊，請參閱 eksctl 文件中的刪除和耗盡節點群組。

從 EKS Fargate 移轉至 EKS 自動模式

本主題將引導您使用 `kubectl` 將工作負載從 EKS Fargate 移轉至 Amazon EKS 自動模式的程序。移轉可以逐步進行，讓您能夠按照自己的步調移動工作負載，同時在整個轉換期間保持叢集穩定性和應用程式可用性。

下面概述的逐步方法使您能夠在移轉期間同時執行 EKS Fargate 和 EKS 自動模式。這種雙重操作策略有助於確保平穩轉換，讓您可以在完全停用 EKS Fargate 之前，驗證工作負載在 EKS 自動模式上的行為。您可以單獨或分組移轉應用程式，提供靈活性以適應您的特定運營要求與風險承受能力。

比較 Amazon EKS Auto Mode 和 EKS 與 AWS Fargate

對於想要執行 EKS 的客戶而言，Amazon EKS 搭配 AWS Fargate 仍然是選項，但 Amazon EKS Auto Mode 是建議的方法。EKS 自動模式完全符合 Kubernetes 規範，支援所有上游 Kubernetes 基本元件與 Istio 等平台工具，而這些是 Fargate 無法支援的。EKS 自動模式還完全支援所有 EC2 執行時期購買選項，包括 GPU 和 Spot 執行個體，使客戶能夠利用協商後的 EC2 折扣和其他節省機制。這些功能在使用帶有 Fargate 的 EKS 時不可用。

此外，EKS 自動模式允許客戶實現與 Fargate 相同的隔離模型，使用標準的 Kubernetes 排程功能來確保每個 EC2 執行個體運行單個應用程式容器。透過採用 Amazon EKS Auto Mode，客戶可以釋放在上執行 Kubernetes 的完整優點 AWS ，這是一個完全符合 Kubernetes 標準的平台，提供彈性以利用 EC2 和購買選項的完整廣度，同時保留 Fargate 提供的基礎設施管理的易用性和抽象性。

在 EKS Auto 模式下實現類似 Fargate 的隔離

若要複寫 Fargate 的 Pod 隔離模型，其中每個 Pod 在自己的專用執行個體上執行，您可以使用 Kubernetes 拓撲分散限制條件。這是控制跨節點 Pod 分佈的建議方法：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: isolated-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: isolated-app
  template:
    metadata:
      labels:
        app: isolated-app
      annotations:
        eks.amazonaws.com/compute-type: ec2
    spec:
      topologySpreadConstraints:
      - maxSkew: 1
        topologyKey: kubernetes.io/hostname
        whenUnsatisfiable: DoNotSchedule
        labelSelector:
          matchLabels:
            app: isolated-app
        minDomains: 1
      containers:
      - name: app
        image: nginx
        ports:
        - containerPort: 80
```

在此組態中：

- `maxSkew: 1` 確保任兩個節點之間的 Pod 計數差異最多為 1，每個節點可有效分配一個 Pod
- `topologyKey: kubernetes.io/hostname` 將節點定義為拓撲網域
- `whenUnsatisfiable: DoNotSchedule` 如果無法滿足限制條件， 會阻止排程
- `minDomains: 1` 確保在排程之前至少有一個網域（節點）

EKS Auto Mode 會視需要自動佈建新的 EC2 執行個體，以滿足此限制，提供與 Fargate 相同的隔離模型，同時讓您存取完整的 EC2 執行個體類型和購買選項。

或者，您可以使用 Pod 反親和性規則進行更嚴格的隔離：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: isolated-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: isolated-app
  template:
    metadata:
      labels:
        app: isolated-app
      annotations:
        eks.amazonaws.com/compute-type: ec2
    spec:
      affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
              - key: app
                operator: In
                values:
                - isolated-app
            topologyKey: kubernetes.io/hostname
      containers:
      - name: app
        image: nginx
        ports:
        - containerPort: 80
```

使用 的podAntiAffinity規則requiredDuringSchedulingIgnoredDuringExecution可確保在同一節點上app: isolated-app無法排程具有標籤的兩個 Pod。此方法提供與 Fargate 類似的硬隔離保證。

先決條件

開始移轉之前，請確保您已具備下列項目：

- 使用 Fargate 設定叢集。如需詳細資訊，請參閱the section called “開始使用”。
- 已安裝並將 `kubectl` 連接到您的叢集。如需詳細資訊，請參閱設定。

步驟 1：檢查 Fargate 叢集

1. 檢查帶有 Fargate 的 EKS 叢集是否正在執行：

```
kubectl get node
```

```
NAME STATUS ROLES AGE VERSION
fargate-ip-192-168-92-52.ec2.internal Ready <none> 25m v1.30.8-eks-2d5f260
fargate-ip-192-168-98-196.ec2.internal Ready <none> 24m v1.30.8-eks-2d5f260
```

2. 檢查執行中的 Pod：

```
kubectl get pod -A
```

```
NAMESPACE NAME READY STATUS RESTARTS AGE
kube-system coredns-6659cb98f6-gxpjz 1/1 Running 0 26m
kube-system coredns-6659cb98f6-gzzsx 1/1 Running 0 26m
```

3. 在名為 `deployment_fargate.yaml` 的檔案中建立部署：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
```

```
      labels:
        app: nginx
      annotations:
        eks.amazonaws.com/compute-type: fargate
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
```

4. 套用部署：

```
kubectl apply -f deployment_fargate.yaml
```

```
deployment.apps/nginx-deployment created
```

5. 檢查 Pod 與部署狀態：

```
kubectl get pod,deploy
```

```
NAME                                    READY   STATUS    RESTARTS   AGE
pod/nginx-deployment-5c7479459b-6trtm   1/1     Running   0          61s
pod/nginx-deployment-5c7479459b-g8ssb   1/1     Running   0          61s
pod/nginx-deployment-5c7479459b-mq4mf   1/1     Running   0          61s

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nginx-deployment   3/3     3            3           61s
```

6. 檢查節點：

```
kubectl get node -owide
```

```
NAME                                    STATUS  ROLES  AGE VERSION
   INTERNAL-IP     EXTERNAL-IP OS-IMAGE       KERNEL-VERSION
 CONTAINER-RUNTIME
fargate-ip-192-168-111-43.ec2.internal  Ready   <none> 31s v1.30.8-eks-2d5f260
 192.168.111.43  <none>      Amazon Linux 2 5.10.234-225.910.amzn2.x86_64
 containerd://1.7.25
```

```
fargate-ip-192-168-117-130.ec2.internal Ready   <none> 36s v1.30.8-eks-2d5f260
 192.168.117.130 <none>      Amazon Linux 2 5.10.234-225.910.amzn2.x86_64
 containerd://1.7.25
fargate-ip-192-168-74-140.ec2.internal  Ready   <none> 36s v1.30.8-eks-2d5f260
 192.168.74.140  <none>      Amazon Linux 2 5.10.234-225.910.amzn2.x86_64
 containerd://1.7.25
```

步驟 2：在叢集上啟用 EKS 自動模式

1. 使用 CLI 或 管理主控台在現有叢集上啟用 AWS EKS 自動模式。如需詳細資訊，請參閱the section called “在叢集上啟用”。
2. 檢查 nodepool：

```
kubectl get nodepool
```

```
NAME              NODECLASS   NODES   READY   AGE
general-purpose   default     1       True    6m58s
system            default     0       True    3d14h
```

步驟 3：更新工作負載以進行移轉

識別並更新您想要移轉至 EKS 自動模式的工作負載。

要將工作負載從 Fargate 移轉至 EKS 自動模式，請套用註釋 `eks.amazonaws.com/compute-type: ec2`。這能確保即使存在 Fargate 設定檔，該工作負載也不會由 Fargate 排程，而是由 EKS 自動模式 NodePool 接管。如需詳細資訊，請參閱the section called “建立節點集區”。

1. 修改您的部署 (例如 `deployment_fargate.yaml` 檔案)，將運算類型變更為 `ec2`：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
```

```
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
      annotations:
        eks.amazonaws.com/compute-type: ec2
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
```

2. 套用部署。此變更可讓工作負載排程至新的 EKS 自動模式節點：

```
kubectl apply -f deployment_fargate.yaml
```

3. 檢查部署是否在 EKS 自動模式叢集中執行：

```
kubectl get pod -o wide
```

```
NAME                               READY   STATUS    RESTARTS   AGE     IP
    NODE                   NOMINATED NODE   READINESS GATES
nginx-deployment-97967b68d-ffxxh   1/1     Running   0          3m31s
 192.168.43.240   i-0845aafcb51630ffb   <none>           <none>
nginx-deployment-97967b68d-mbcgj   1/1     Running   0          2m37s
 192.168.43.241   i-0845aafcb51630ffb   <none>           <none>
nginx-deployment-97967b68d-qpd8x   1/1     Running   0          2m35s
 192.168.43.242   i-0845aafcb51630ffb   <none>           <none>
```

4. 確認沒有 Fargate 節點在執行，且部署正在 EKS 自動模式受管節點上執行：

```
kubectl get node -owide
```

```
NAME                  STATUS ROLES  AGE   VERSION              INTERNAL-IP     EXTERNAL-
IP OS-IMAGE                                       KERNEL-VERSION CONTAINER-RUNTIME
i-0845aafcb51630ffb Ready  <none> 3m30s v1.30.8-eks-3c20087 192.168.41.125
 3.81.118.95 Bottlerocket (EKS Auto) 2025.3.14 (aws-k8s-1.30) 6.1.129
 containerd://1.7.25+bottlerocket
```

步驟 4：逐步移轉工作負載

針對要移轉的每個工作負載重複步驟 3。這使您可根據需求和風險承受能力，單獨或分組移動工作負載。

步驟 5：移除原始的 Fargate 設定檔

所有工作負載移轉完成後，即可移除原始的 `fargate` 設定檔。將 *<fargate profile name>* 取代為您的 Fargate 設定檔名稱：

```
aws eks delete-fargate-profile --cluster-name eks-fargate-demo-cluster --fargate-
profile-name <fargate profile name>
```

步驟 6：縮減 CoreDNS 規模

由於 EKS Auto Mode 處理 CoreDNS，請將`coredns`部署縮減為 0：

```
kubectl scale deployment coredns -n kube-system --replicas=0
```

在 EKS 自動模式叢集中執行範例工作負載

本章節提供如何將不同類型的工作負載部署到在自動模式下執行的 Amazon EKS 叢集的範例。這些範例展示了關鍵的工作負載模式，包括範例應用程式、負載平衡的 Web 應用程式、使用持久性儲存的具狀態工作負載，以及具有特定節點放置要求的工作負載。每個範例都包含完整的資訊清單和逐步部署說明，您可以將其用作自己應用程式的範本。

繼續進行範例之前，請確定您已在自動模式下執行 EKS 叢集，且已安裝 AWS CLI 和 kubectl。如需詳細資訊，請參閱設定。這些範例假設您基本熟悉 Kubernetes 概念與 kubectl 命令。

您可以使用這些基於使用案例的範例在 EKS 自動模式叢集中執行工作負載。

the section called “部署 inflate 工作負載”

展示如何使用 `kubectl` 命令將範例工作負載部署到 EKS 自動模式叢集。

the section called “部署負載平衡器”

展示如何在 Amazon EKS 上部署容器化版本的 2048 遊戲。

the section called “部署具狀態工作負載”

展示如何將範例具狀態應用程式部署到 EKS 自動模式叢集。

the section called “部署加速工作負載”

說明如何將硬體加速工作負載部署至 EKS Auto Mode 管理的節點。

the section called “控制部署”

展示如何使用注釋來控制工作負載是否部署到由 EKS 自動模式管理的節點。

在 Amazon EKS 自動模式叢集中部署範例 inflate 工作負載

在本教學中，您將學習如何將範例工作負載部署至 EKS 自動模式叢集，並觀察其如何自動佈建所需的運算資源。您將使用 kubectl 命令來監看叢集的行為，並親自了解自動模式如何簡化 Kubernetes 操作 AWS。完成本教學後，您將了解 EKS 自動模式如何透過自動管理底層運算資源來回應工作負載部署，無需手動設定節點群組。

先決條件

- 一個 Amazon EKS 自動模式叢集。請記下叢集的名稱和 AWS 區域。
- IAM 主體，例如使用者或角色，具有足夠的許可來管理聯網、運算和 EKS 資源。
  - 若需詳細資訊，請參閱《IAM 使用者指南》中的建立角色和附加政策。
- 已安裝 aws CLI 並以 IAM 身分進行設定。
- 已安裝 kubectl CLI 並連接至叢集。
  - 如需詳細資訊，請參閱設定。

步驟 1：檢閱現有運算資源 (選用)

第一，使用 kubectl 列出叢集上的節點集區。

```
kubectl get nodepools
```

輸出範例：

```
general-purpose
```

在本教學中，我們將部署設定為使用 general-purpose 節點集區的工作負載。此節點集區內建於 EKS Auto Mode，包含適用於一般工作負載，如微服務和 Web 應用程式的合理預設值。您也可建立自己的節點集區。如需詳細資訊，請參閱the section called “建立節點集區”。

第二，使用 kubectl 列出連接到叢集的節點。

```
kubectl get nodes
```

若您剛建立 EKS 自動模式叢集，一開始不會有任何節點。

在本教學中，您將部署一個範例工作負載。若沒有節點，或現有節點無法容納工作負載，EKS 自動模式將會佈建新節點。

步驟 2：將範例應用程式部署至叢集

請檢閱下列 Kubernetes 部署，將其儲存為 inflate.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: inflate
spec:
  replicas: 1
  selector:
    matchLabels:
      app: inflate
  template:
    metadata:
      labels:
        app: inflate
    spec:
      terminationGracePeriodSeconds: 0
      nodeSelector:
        eks.amazonaws.com/compute-type: auto
      securityContext:
        runAsUser: 1000
        runAsGroup: 3000
        fsGroup: 2000
      containers:
        - name: inflate
          image: public.ecr.aws/eks-distro/kubernetes/pause:3.7
          resources:
            requests:
              cpu: 1
          securityContext:
            allowPrivilegeEscalation: false
```

請注意，eks.amazonaws.com/compute-type: auto 選擇器要求工作負載必須部署在 Amazon EKS 自動模式節點上。

將部署套用至叢集。

```
kubectl apply -f inflate.yaml
```

步驟 3：監看 Kubernetes 事件

使用下列命令以監看 Kubernetes 事件，包括建立新節點。使用 ctrl+c 以停止監看事件。

```
kubectl get events -w --sort-by '.lastTimestamp'
```

再次使用 kubectl 以列出連接到叢集的節點。請記下新建立的節點。

```
kubectl get nodes
```

步驟 4：在 AWS 主控台中檢視節點和執行個體

您可在 EKS 主控台中檢視 EKS 自動模式節點，並且在 EC2 主控台中檢視相關聯的 EC2 執行個體。

由 EKS 自動模式部署的 EC2 執行個體有存取限制。您將無法在 EKS 自動模式節點上執行任意命令。

步驟 5：刪除部署

使用 kubectl 刪除範例部署

```
kubectl delete -f inflate.yaml
```

若您沒有部署至叢集的其他工作負載，由 EKS 自動模式建立的節點將處於空閒狀態。

在預設組態中，EKS Auto Mode 會偵測已空三十秒的節點，並終止它們。

使用 kubectl 或 EC2 主控台確認相關聯的執行個體已被刪除。

將範例負載平衡器工作負載部署至 EKS 自動模式

本指南將引導您在 Amazon EKS 上部署容器化版本的 2048 遊戲，並包含負載平衡和網際網路可存取性。

先決條件

- EKS 自動模式叢集
- kubectl 已設定為與叢集互動
- 建立 ALB 資源的適當 IAM 許可

步驟 1：建立命名空間

首先，為 2048 遊戲應用程式建立專用命名空間。

建立名為 01-namespace.yaml 的檔案：

```
apiVersion: v1
kind: Namespace
metadata:
  name: game-2048
```

套用命名空間組態：

```
kubectl apply -f 01-namespace.yaml
```

步驟 2：部署應用程式

應用程式會執行多個 2048 遊戲容器的複本。

建立名為 02-deployment.yaml 的檔案：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  namespace: game-2048
  name: deployment-2048
spec:
  selector:
    matchLabels:
      app.kubernetes.io/name: app-2048
  replicas: 5
  template:
    metadata:
      labels:
```

```
        app.kubernetes.io/name: app-2048
    spec:
      containers:
        - image: public.ecr.aws/l6m2t8p7/docker-2048:latest
          imagePullPolicy: Always
          name: app-2048
          ports:
            - containerPort: 80
          resources:
            requests:
              cpu: "0.5"
```

> **Note**
>
> 如果您收到載入映像 的錯誤public.ecr.aws/l6m2t8p7/docker-2048:latest，請確認您的節點 IAM 角色具有從 ECR 提取映像的足夠許可。如需詳細資訊，請參閱the section called “節點 IAM 角色”。此外，範例中的 docker-2048 映像是 x86_64 映像，並且不會在其他架構執行。

關鍵元件：

- 部署 5 個應用程式複本
- 使用公有 ECR 映像
- 每個 Pod 請求 0.5 個 CPU 核心
- 公開 HTTP 流量的連接埠 80

套用部署：

```
kubectl apply -f 02-deployment.yaml
```

步驟 3：建立服務

服務會將部署公開至叢集網路。

建立名為 03-service.yaml 的檔案：

```
apiVersion: v1
kind: Service
```

```
metadata:
  namespace: game-2048
  name: service-2048
spec:
  ports:
    - port: 80
      targetPort: 80
      protocol: TCP
  selector:
    app.kubernetes.io/name: app-2048
```

關鍵元件：

- 建立 ClusterIP 服務
- 將連接埠 80 映射至容器的連接埠 80
- 使用標籤選擇器來尋找 Pod

套用服務：

```
kubectl apply -f 03-service.yaml
```

步驟 4：設定負載平衡

您將設定傳入，以將應用程式公開至網際網路。

首先，建立 IngressClass。建立名為 04-ingressclass.yaml 的檔案：

```
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  labels:
    app.kubernetes.io/name: LoadBalancerController
  name: alb
spec:
  controller: eks.amazonaws.com/alb
```

> **Note**
>
> EKS 自動模式需要子網路標籤來識別公有和私有子網路。
> 若您使用 eksctl 建立叢集，則您已具備這些標籤。

了解如何 the section called “標記子網路”。

然後建立傳入資源。建立名為 05-ingress.yaml 的檔案：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  namespace: game-2048
  name: ingress-2048
  annotations:
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/target-type: ip
spec:
  ingressClassName: alb
  rules:
    - http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: service-2048
                port:
                  number: 80
```

關鍵元件：

- 建立面向網際網路的 ALB
- 使用 IP 目標類型來進行直接 Pod 路由
- 將所有流量 (/) 路由到遊戲服務

套用傳入組態：

```
kubectl apply -f 04-ingressclass.yaml
kubectl apply -f 05-ingress.yaml
```

步驟 5：確認部署

1. 檢查所有 Pod 是否處於執行狀態：

```
kubectl get pods -n game-2048
```

2. 確認服務是否已建立：

```
kubectl get svc -n game-2048
```

3. 取得 ALB 端點：

```
kubectl get ingress -n game-2048
```

傳入輸出中的 ADDRESS 欄位將顯示您的 ALB 端點。等待 2-3 分鐘，讓 ALB 完成佈建並註冊所有目標。

步驟 6：存取遊戲

開啟網頁瀏覽器，瀏覽至先前步驟中取得的 ALB 端點 URL。您應會看到 2048 遊戲介面。

步驟 7：清除

要移除在本教學課程中建立的全部資源：

```
kubectl delete namespace game-2048
```

這將刪除命名空間中的所有資源，包括部署、服務和傳入資源。

幕後情況

1. 部署會建立 5 個執行 2048 遊戲的 Pod
2. 服務為這些 Pod 提供穩定的網路存取
3. EKS 自動模式：
   - 在 中建立 Application Load Balancer AWS
   - 為 Pod 設定目標群組
   - 設定路由規則，以將流量導向服務

疑難排解

如果遊戲無法載入：

- 確保所有 Pod 均在執行：`kubectl get pods -n game-2048`
- 檢查傳入狀態：`kubectl describe ingress -n game-2048`
- 驗證 ALB 運作狀態檢查：在 AWS 主控台中檢查目標群組運作狀態

部署具狀態工作負載範例至 EKS 自動模式

這項教學課程將引導您部署具狀態範例應用程式至 EKS 自動模式叢集。應用程式可將時間戳記寫入持續性磁碟區，從而示範 EKS 自動模式的自動 EBS 磁碟區佈建與持續保留功能。

先決條件

- EKS 自動模式叢集
- AWS CLI 已設定適當許可
- `kubectl` 已安裝並已設定
  - 如需詳細資訊，請參閱 設定。

步驟 1：設定您的環境

1. 設定環境變數：

```
export CLUSTER_NAME=my-auto-cluster
export AWS_REGION="us-west-2"
```

2. 更新 kubeconfig：

```
aws eks update-kubeconfig --name "${CLUSTER_NAME}"
```

步驟 2：建立儲存類別

`StorageClass` 定義 EKS 自動模式佈建 EBS 磁碟區的方式。

EKS 自動模式不會為您建立 `StorageClass`。您必須建立 `StorageClass` 參考 `ebs.csi.eks.amazonaws.com`，以使用 EKS 自動模式的儲存功能。

1. 建立名為 `storage-class.yaml` 的檔案：

```
apiVersion: storage.k8s.io/v1
```

```
kind: StorageClass
metadata:
  name: auto-ebs-sc
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: ebs.csi.eks.amazonaws.com
volumeBindingMode: WaitForFirstConsumer
parameters:
  type: gp3
  encrypted: "true"
```

2. 套用 StorageClass：

```
kubectl apply -f storage-class.yaml
```

關鍵元件：

- provisioner: ebs.csi.eks.amazonaws.com - 使用 EKS 自動模式
- volumeBindingMode: WaitForFirstConsumer - 延遲建立磁碟區，直至 Pod 需要磁碟區
- type: gp3 - 指定 EBS 磁碟區類型
- encrypted: "true" - EBS 將使用預設 aws/ebs 金鑰，來加密使用此類別建立的磁碟區。此為選用操作，但建議您採用。
- storageclass.kubernetes.io/is-default-class: "true" - 依預設，Kubernetes 將使用此儲存類別，除非您在持續性磁碟區宣告上指定了不同的磁碟區類別。若要從其他儲存控制器移轉，則在設定該值時要謹慎對待。(選用)

步驟 3：建立持續性磁碟區宣告

PVC 可從 StorageClass 請求儲存。

1. 建立名為 pvc.yaml 的檔案：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: auto-ebs-claim
spec:
  accessModes:
    - ReadWriteOnce
```

```
  storageClassName: auto-ebs-sc
  resources:
    requests:
      storage: 8Gi
```

2. 套用 PVC：

```
kubectl apply -f pvc.yaml
```

關鍵元件：

- accessModes: ReadWriteOnce - 磁碟區可一次由一個節點掛載
- storage: 8Gi - 請求 8 GiB 磁碟區
- storageClassName: auto-ebs-sc - 參考我們建立的 StorageClass

步驟 4：部署應用程式

部署將執行一個容器，這會將時間戳記寫入持續性磁碟區。

1. 建立名為 deployment.yaml 的檔案：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: inflate-stateful
spec:
  replicas: 1
  selector:
    matchLabels:
      app: inflate-stateful
  template:
    metadata:
      labels:
        app: inflate-stateful
    spec:
      terminationGracePeriodSeconds: 0
      nodeSelector:
        eks.amazonaws.com/compute-type: auto
      containers:
        - name: bash
```

```
          image: public.ecr.aws/docker/library/bash:4.4
          command: ["/usr/local/bin/bash"]
          args: ["-c", "while true; do echo $(date -u) >> /data/out.txt; sleep 60;
 done"]
          resources:
            requests:
              cpu: "1"
          volumeMounts:
            - name: persistent-storage
              mountPath: /data
      volumes:
        - name: persistent-storage
          persistentVolumeClaim:
            claimName: auto-ebs-claim
```

2. 套用部署：

```
kubectl apply -f deployment.yaml
```

關鍵元件：

- 寫入時間戳記至檔案的簡易 bash 容器
- 在 /data 掛載 PVC
- 請求 1 個 CPU 核心
- 針對 EKS 受管節點使用節點選擇器

步驟 5：驗證設定

1. 檢查 Pod 是否在執行中：

```
kubectl get pods -l app=inflate-stateful
```

2. 確認 PVC 是否已繫結：

```
kubectl get pvc auto-ebs-claim
```

3. 檢查 EBS 磁碟區：

```
# Get the PV name
```

```
PV_NAME=$(kubectl get pvc auto-ebs-claim -o jsonpath='{.spec.volumeName}')
# Describe the EBS volume
aws ec2 describe-volumes \
  --filters Name=tag:CSIVolumeName,Values=${PV_NAME}
```

4. 驗證是否正在寫入資料：

```
kubectl exec "$(kubectl get pods -l app=inflate-stateful \
  -o=jsonpath='{.items[0].metadata.name}')" -- \
  cat /data/out.txt
```

步驟 6：清除

若要移除在本教學課程中建立的全部資源，請執行以下命令：

```
# Delete all resources in one command
kubectl delete deployment/inflate-stateful pvc/auto-ebs-claim storageclass/auto-ebs-sc
```

幕後情況

1. PVC 可從 `StorageClass` 請求儲存
2. 排程 Pod 時：
   a. EKS 自動模式可佈建 EBS 磁碟區
   b. 建立 PersistentVolume
   c. 連接磁碟區至節點
3. Pod 會掛載磁碟區及開始撰寫時間戳記

快照控制器

EKS 自動模式與 Kubernetes CSI 快照器相容，亦稱為快照控制器。然而，EKS 自動模式不包括快照控制器。您將負責安裝及設定快照控制器。如需詳細資訊，請參閱 the section called “CSI 快照器控制器”。

請檢閱下列參考 EKS 自動模式儲存功能的 `VolumeSnapshotClass`。

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
```

```
  name: auto-ebs-vsclass
driver: ebs.csi.eks.amazonaws.com
deletionPolicy: Delete
```

了解 Kubernetes CSI 快照器的相關詳細資訊。

部署加速工作負載

本教學課程示範 Amazon EKS Auto Mode 如何簡化硬體加速工作負載的啟動。Amazon EKS Auto Mode 透過自動化關鍵基礎設施元件，立即提供運算、聯網、負載平衡、儲存以及 Identity and Access Management 功能，簡化叢集本身以外的操作。

Amazon EKS Auto Mode 包含特定執行個體類型所需的驅動程式和裝置外掛程式，例如 NVIDIA 和 AWS Neuron 驅動程式。您無需安裝或更新這些元件。

EKS 自動模式會自動管理這些加速器的驅動程式：

- AWS Trainium
- AWS Inferentia
- Amazon EC2 加速執行個體上的 NVIDIA GPU

> **Note**
>
> EKS 自動模式包含 Kubernetes 專用 NVIDIA 裝置外掛程式。此外掛程式會自動執行，且在您的叢集中不會顯示為常駐程式集。

其他聯網支援：

- Elastic Fabric Adapter (EFA) — 如需在 EKS 叢集中管理 EFA 裝置的資訊，請參閱the section called “EFA 裝置”。

Amazon EKS 自動模式消除了加速器驅動程式和裝置外掛程式管理的繁瑣工作。

您還可以透過將叢集縮減至零來節省成本。您可以設定 EKS 自動模式，在沒有工作負載執行時終止執行個體。這對基於批次的推斷工作負載很有用。

以下提供一個範例，說明如何使用 Amazon EKS 自動模式啟動加速工作負載。

先決條件

- 一個已設定 Amazon EKS 自動模式的 Kubernetes 叢集。
- 當啟用 `general-purpose` 或 `system` 受管節點集區時，所建立的預設 `default` EKS 節點類別。

步驟 1：部署 GPU 工作負載

在此範例中，您將為需要 45GB GPU 記憶體的 NVIDIA 基礎工作負載建立一個 NodePool。透過 EKS 自動模式，您可使用 Kubernetes 排程限制來定義您的執行個體需求。

要部署 Amazon EKS 自動模式的 `NodePool` 和範例 `workload`，請檢閱以下的 NodePool 和 Pod 定義，並分別儲存為 `nodepool-gpu.yaml` 和 `pod.yaml`：

nodepool-gpu.yaml

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: gpu
spec:
  disruption:
    budgets:
    - nodes: 10%
    consolidateAfter: 1h
    consolidationPolicy: WhenEmpty
  template:
    metadata: {}
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default
      requirements:
        - key: "karpenter.sh/capacity-type"
          operator: In
          values: ["on-demand"]
        - key: "kubernetes.io/arch"
          operator: In
          values: ["amd64"]
        - key: "eks.amazonaws.com/instance-family"
          operator: In
          values:
```

```
          - g6e
          - g6
      taints:
        - key: nvidia.com/gpu
          effect: NoSchedule
      terminationGracePeriod: 24h0m0s
```

pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: nvidia-smi
spec:
  nodeSelector:
    eks.amazonaws.com/compute-type: auto
  restartPolicy: OnFailure
  containers:
  - name: nvidia-smi
    image: public.ecr.aws/amazonlinux/amazonlinux:2023-minimal
    args:
    - "nvidia-smi"
    resources:
      requests:
        memory: "30Gi"
        cpu: "3500m"
        nvidia.com/gpu: 1
      limits:
        memory: "30Gi"
        nvidia.com/gpu: 1
  tolerations:
  - key: nvidia.com/gpu
    effect: NoSchedule
    operator: Exists
```

請注意，`eks.amazonaws.com/compute-type: auto` 選擇器要求工作負載必須部署在 Amazon EKS 自動模式節點上。NodePool 也會設定一個污點，僅允許排程具有 NVIDIA GPUs 容錯能力的 Pod。

將 NodePool 和工作負載套用至您的叢集。

```
kubectl apply -f nodepool-gpu.yaml
```

```
kubectl apply -f pod.yaml
```

您應該會看到下列輸出：

```
nodepool.karpenter.sh/gpu created
pod/nvidia-smi created
```

等待幾秒鐘，並檢查叢集中的節點。您現在應該會看到在 Amazon EKS 自動叢集中佈建了一個新節點：

```
> kubectl get nodes

NAME         TYPE           CAPACITY    ZONE         NODE                    READY   AGE
gpu-dnknr    g6e.2xlarge    on-demand   us-west-2b   i-02315c7d7643cdee6   True    76s
```

步驟 2：驗證

根據下列 Kubernetes 排程限制，您可以看到 Amazon EKS Auto Mode 啟動了 `g6e.2xlarge`，而不是 GPU，`g6.2xlarge`因為工作負載需要具有 L40S 的執行個體：

```
...
  nodeSelector:
    eks.amazonaws.com/instance-gpu-name: l40s
...
    requests:
        memory: "30Gi"
        cpu: "3500m"
        nvidia.com/gpu: 1
      limits:
        memory: "30Gi"
        nvidia.com/gpu: 1
```

現在，執行下列命令來查看容器的日誌：

```
kubectl logs nvidia-smi
```

輸出範例：

```
+-----------------------------------------------------------------------------------------
+
```

```
| NVIDIA-SMI 535.230.02             Driver Version: 535.230.02   CUDA Version: 12.2
 |
|-----------------------------------------+----------------------
+----------------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC
 |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M.
 |
|                                         |                      |               MIG M.
 |
|=========================================+======================
+======================|
|   0  NVIDIA L40S                    On  | 00000000:30:00.0 Off |                    0
 |
| N/A   27C    P8              23W / 350W |      0MiB / 46068MiB |      0%      Default
 |
|                                         |                      |                  N/A
 |
+-----------------------------------------+----------------------
+----------------------+

+---------------------------------------------------------------------------------------
+
| Processes:
 |
|  GPU   GI   CI        PID   Type   Process name                              GPU Memory
 |
|        ID   ID                                                               Usage
 |
|
=======================================================================================|
|  No running processes found
 |
+---------------------------------------------------------------------------------------
+
```

您會看到容器已偵測到其正在具有 NVIDIA GPU 的執行個體上執行，且您無需安裝任何裝置驅動程式，因為這由 Amazon EKS 自動模式管理。

步驟 3：清理

若要移除所有建立的物件，請使用 `kubectl` 刪除範例部署和 NodePool，以便終止節點：

```
kubectl delete -f nodepool-gpu.yaml
kubectl delete -f pod.yaml
```

NodePools 範例參考

建立 NVIDIA NodePool

以下 NodePool 定義：

- 只啟動 g6e 和 g6 系列的執行個體
- 節點閒置 1 小時後進行合併
  - 的 1 小時值`consolidateAfter`支援尖峰工作負載並減少節點流失。您可根據工作負載需求調整`consolidateAfter`。

具有 GPU 執行個體系列和合併功能的範例 NodePool

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: gpu
spec:
  disruption:
    budgets:
    - nodes: 10%
    consolidateAfter: 1h
    consolidationPolicy: WhenEmpty
  template:
    metadata: {}
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default
      requirements:
        - key: "karpenter.sh/capacity-type"
          operator: In
          values: ["on-demand"]
        - key: "kubernetes.io/arch"
          operator: In
          values: ["amd64"]
        - key: "eks.amazonaws.com/instance-family"
```

```
          operator: In
          values:
          - g6e
          - g6
      terminationGracePeriod: 24h0m0s
```

而不是設定eks.amazonaws.com/instance-gpu-name您可能eks.amazonaws.com/instance-family用來指定執行個體系列的 。如需其他影響排程的知名標籤，請參閱 the section called “EKS 自動模式支援的標籤”。

若您有特定的儲存需求，可以透過建立自己的 NodeClass 並在 NodePool 中參考，來調整節點的臨時儲存 iops、size 和 throughput。深入了解可設定的 NodeClass 選項。

NodeClass 的儲存組態範例

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: gpu
spec:
  ephemeralStorage:
    iops: 3000
    size: 80Gi
    throughput: 125
```

定義 AWS Trainium 和 AWS Inferentia NodePool

以下 NodePool 設定了 eks.amazonaws.com/instance-category，表示僅啟動 Inferentia 與 Trainium 系列的執行個體：

```
        - key: "eks.amazonaws.com/instance-category"
          operator: In
          values:
            - inf
            - trn
```

進行 EKS 自動模式設定

本章內容介紹如何設定 Amazon Elastic Kubernetes Service (EKS) 自動模式叢集的特定層面。儘管 EKS 自動模式可自動管理大多數基礎結構元件，但您可自訂某些功能來滿足工作負載需求。

藉助本主題中所述的組態選項，您可修改聯網設定、運算資源及負載平衡行為，同時確保自動化管理基礎結構的優勢。做出任何組態變更之前，檢閱以下章節中所述的可用選項，確定哪種方法最能滿足您的需求。

您想要設定的功能有哪些？	組態選項
節點聯網及儲存 • 設定公有與私有子網路間的節點置放 • 定義節點存取控制的自訂安全群組 • 自訂網路位址轉譯 (SNAT) 政策 • 啟用 Kubernetes 網路政策、詳細的網路政策日誌記錄與監控 • 設定臨時性儲存參數 (大小、IOPS、輸送量) • 透過自訂 KMS 金鑰來設定加密的臨時性儲存 • 隔離單獨子網路中的 Pod 流量與節點	the section called “建立節點類別”
節點運算資源 • 選取特定 EC2 執行個體的類型及系列 • 定義 CPU 架構 (x86_64、ARM64) • 設定容量類型 (隨需、Spot) • 指定可用區域 • 設定節點污點與標籤 • 設定 CPU 與記憶體用量的資源限制	the section called “建立節點集區”
靜態容量節點集區 • 無論 Pod 需求為何，維持固定數量的節點 • 設定預留執行個體的可預測容量 • 為一致的基礎設施足跡設定特定的複本計數 • 手動擴展靜態節點集區 • 監控靜態容量節點集區狀態	the section called “建立靜態節點集區”
Application Load Balancer 設定	the section called “建立傳入類別”

您想要設定的功能有哪些？	組態選項
• 部署內部或面向網際網路的負載平衡器 • 設定跨區域負載平衡 • 設定閒置逾時期間 • 啟用 HTTP/2 與 WebSocket 支援 • 設定運作狀態檢查參數 • 指定 TLS 憑證設定 • 定義目標群組屬性 • 設定 IP 位址類型 (IPv4、雙重堆疊)	
Network Load Balancer 設定 • 設定直接 Pod IP 路由 • 啟用跨區域負載平衡 • 設定連線閒置逾時 • 設定運作狀態檢查參數 • 指定子網路置放 • 設定 IP 位址類型 (IPv4、雙重堆疊) • 設定保留用戶端來源 IP • 定義目標群組屬性	the section called “建立服務”
多叢集目標群組 • 了解負載平衡器控制器如何管理目標群組成員資格 • 啟用 MultiCluster 支援以在多個 Kubernetes 叢集之間共用目標群組 • 支援來自共用目標群組中其他來源的任意目標	the section called “多叢集目標群組”

您想要設定的功能有哪些？	組態選項
擴展受管控制器的 RBAC • 了解 EKS Auto Mode 受管控制器如何向叢集進行身分驗證 • 透過 `eks:managed` 群組將其他 Kubernetes RBAC 授予受管控制器 • 使用 / `Role` `RoleBinding` 和 `resourceNames` 為最低權限範圍授予	the section called “將 RBAC 授予受管控制器”
儲存類別設定 • 定義 EBS 磁碟區類型 (gp3、io1、io2 等) • 設定磁碟區加密與 KMS 金鑰用量 • 設定 IOPS 與輸送量參數 • 設定為預設儲存類別 • 定義佈建磁碟區的自訂標籤	the section called “建立 StorageClass”
控制 ODCR 用量 • 將工作負載部署設定為 EC2 隨需容量保留 • 針對目標容量用量，依 ID 明確選取特定 ODCR • 藉助標籤型選取項，來確定 ODCR 群組的目標 • 透過擁有跨 AWS 帳戶案例的帳戶來篩選 ODCRs • 控制工作負載是否會自動使用開放的 ODCR	the section called “控制容量保留”
節點進階安全性 • 啟用 FIPS 140-2 驗證的密碼編譯模組 • 設定核心鎖定安全模式	the section called “進階安全性”

您想要設定的功能有哪些？	組態選項
成本最佳化 • 了解整合、bin-packing 和正確調整大小如何降低運算成本 • 識別封鎖合併的組態 (do-not-disrupt 、限制性 PDBs) • 將 NodePool limits設定為成本上限 • 限制執行個體系列和大小以控制全包成本 • 套用高載 CI/CD 工作負載的建議模式	EKS Auto 模式的成本最佳化

建立 Amazon EKS 的節點類別

Amazon EKS 節點類別是範本，可讓您精細控制 EKS 自動模式受管節點的組態。節點類別定義套用至 EKS 叢集中節點群組的基礎結構層級設定，包括網路組態、儲存設定和資源標記。本主題闡釋如何建立和設定節點類別，以滿足您的特定操作需求。

當您需要自訂 EKS 自動模式在預設設定之外，佈建和設定 EC2 執行個體的方式時，建立節點類別可讓您精確控制關鍵基礎結構參數。例如，您可以指定私有子網路放置以增強安全性，為效能敏感的工作負載設定執行個體暫時性儲存，或套用自訂標記以進行成本分配。

建立節點類別

要建立 NodeClass，請執行下列步驟：

1. 使用節點類別組態建立 YAML 檔案 (例如 nodeclass.yaml)
2. 使用 kubectl 將組態套用至您的叢集
3. 在節點集區組態中參考節點類別。如需詳細資訊，請參閱the section called “建立節點集區”。

您需要已安裝並設定 kubectl。如需詳細資訊，請參閱設定。

基本節點類別範例

以下是一個節點類別範例：

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
```

```
metadata:
  name: private-compute
spec:
  subnetSelectorTerms:
    - tags:
        Name: "private-subnet"
        kubernetes.io/role/internal-elb: "1"
  securityGroupSelectorTerms:
    - tags:
        Name: "eks-cluster-sg"
  ephemeralStorage:
    size: "160Gi"
```

此 NodeClass 會增加節點上的暫時性儲存量。

透過以下方式套用此組態：

```
kubectl apply -f nodeclass.yaml
```

接下來，在節點集區組態中參考節點類別。如需詳細資訊，請參閱the section called “建立節點集區”。

建立節點類別存取項目

如果您建立了自訂節點類別，則需要建立 EKS 存取項目以允許節點加入叢集。在當您使用內建節點類別和節點集區時，EKS 會自動建立存取項目。

如需項目存取方法的相關資訊，請參閱 the section called “存取項目”。

為 EKS 自動模式節點類別建立存取項目時，您需要使用 EC2 存取項目類型。

使用 CLI 建立存取項目

要建立 EC2 節點的存取項目，並關聯 EKS Auto Node 政策：

使用您的叢集名稱和節點角色 ARN 更新以下 CLI 命令。節點角色 ARN 在節點類別 YAML 中指定。

```
# Create the access entry for EC2 nodes
aws eks create-access-entry \
  --cluster-name <cluster-name> \
  --principal-arn <node-role-arn> \
  --type EC2

# Associate the auto node policy
aws eks associate-access-policy \
```

```
  --cluster-name <cluster-name> \
  --principal-arn <node-role-arn> \
  --policy-arn arn:aws: eks::aws:cluster-access-policy/AmazonEKSAutoNodePolicy \
  --access-scope type=cluster
```

使用 CloudFormation 建立存取項目

要建立 EC2 節點的存取項目，並關聯 EKS Auto Node 政策：

使用您的叢集名稱和節點角色 ARN 更新以下 CloudFormation。節點角色 ARN 在節點類別 YAML 中指定。

```
EKSAutoNodeRoleAccessEntry:
  Type: AWS::EKS::AccessEntry
  Properties:
    ClusterName: <cluster-name>
    PrincipalArn: <node-role-arn>
    Type: "EC2"
    AccessPolicies:
      - AccessScope:
          Type: cluster
        PolicyArn: arn:aws: eks::aws:cluster-access-policy/AmazonEKSAutoNodePolicy
  DependsOn: [ <cluster-name> ] # previously defined in CloudFormation
```

有關部署 CloudFormation 堆疊的資訊，請參閱 CloudFormation 入門

節點類別規格

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: my-node-class
spec:
  # Required fields

  # role and instanceProfile are mutually exclusive fields.
  role: MyNodeRole  # IAM role for EC2 instances
  # instanceProfile: eks-MyNodeInstanceProfile  # IAM instance-profile for EC2
 instances

  subnetSelectorTerms:
    - tags:
        Name: "private-subnet"
```

```
      kubernetes.io/role/internal-elb: "1"
  # Alternative using direct subnet ID
  # - id: "subnet-0123456789abcdef0"

securityGroupSelectorTerms:
  - tags:
      Name: "eks-cluster-sg"
  # Alternative approaches:
  # - id: "sg-0123456789abcdef0"
  # - name: "eks-cluster-security-group"

# Optional: Pod subnet selector for advanced networking
podSubnetSelectorTerms:
  - tags:
      Name: "pod-subnet"
      kubernetes.io/role/pod: "1"
  # Alternative using direct subnet ID
  # - id: "subnet-0987654321fedcba0"
# must include Pod security group selector also
podSecurityGroupSelectorTerms:
  - tags:
      Name: "eks-pod-sg"
  # Alternative using direct security group ID
  # - id: "sg-0123456789abcdef0"

# Optional: Selects on-demand capacity reservations and capacity blocks
# for EKS Auto Mode to prioritize.
capacityReservationSelectorTerms:
  - id: cr-56fac701cc1951b03
  # Alternative Approaches
  - tags:
      Name: "targeted-odcr"
    # Optional owning account ID filter
    owner: "012345678901"

# Optional fields
snatPolicy: Random  # or Disabled

networkPolicy: DefaultAllow  # or DefaultDeny
networkPolicyEventLogs: Disabled  # or Enabled

ephemeralStorage:
  size: "80Gi"    # Range: 1-59000Gi or 1-64000G or 1-58Ti or 1-64T
  iops: 3000      # Range: 3000-16000
```

```
    throughput: 125 # Range: 125-1000
    # Optional KMS key for encryption
    kmsKeyID: "arn:aws: kms:region:account:key/key-id"
    # Accepted formats:
    # KMS Key ID
    # KMS Key ARN
    # Key Alias Name
    # Key Alias ARN

  advancedNetworking:
    # Optional: Controls whether public IP addresses are assigned to instances that are
 launched with the nodeclass.
    # If not set, defaults to the MapPublicIpOnLaunch setting on the subnet.
    associatePublicIPAddress: false

    # Optional: Forward proxy, commonly requires certificateBundles as well
    # for EC2, see https://repost.aws/knowledge-center/eks-http-proxy-containerd-
automation
    httpsProxy: http://192.0.2.4:3128 #commonly port 3128 (Squid) or 8080 (NGINX) #Max
 255 characters
    #httpsProxy: http://[2001:db8::4]:3128 # IPv6 address with port, use []
    noProxy: #Max 50 entries
        - localhost #Max 255 characters each
        - 127.0.0.1
        #- ::1 # IPv6 localhost
        #- 0:0:0:0:0:0:0:1 # IPv6 localhost
        - 169.254.169.254 # EC2 Instance Metadata Service
        #- [fd00:ec2::254] # IPv6 EC2 Instance Metadata Service
        # Domains to exclude, put all VPC endpoints here
        - .internal
        - .eks.amazonaws.com
    # ipv4PrefixSize is default to Auto which is prefix and fallback to secondary IP.
 "32" is the secondary IP mode.
    ipv4PrefixSize: Auto # or "32"

    # enableV4Egress is default to true. Setting it to false when using network policy
 or blocking IPv4 traffic in IPv6 clusters
    enableV4Egress: false

    # Optional: Static network interface configuration for EFA workloads
    networkInterfaces:
      # The primary ENI (networkCardIndex 0, deviceIndex 0) must use interfaceType:
 interface
```

```
      # secondaryIPv4Count and secondaryIPv4PrefixCount are only supported on
networkCardIndex 0
    - deviceIndex: 0
      interfaceType: interface
      networkCardIndex: 0
      secondaryIPv4Count: 10
    - deviceIndex: 1
      interfaceType: interface
      networkCardIndex: 0
      secondaryIPv4PrefixCount: 2 # Only one of secondaryIPv4Count or
secondaryIPv4PrefixCount per ENI
    - deviceIndex: 0
      interfaceType: efa-only
      networkCardIndex: 1

  advancedSecurity:
    # Optional, US regions only: Specifying `fips: true` will cause nodes in the
nodeclass to run FIPS compatible AMIs.
    fips: false

  # Optional: Custom certificate bundles.
  certificateBundles:
    - name: "custom-cert"
      data: "base64-encoded-cert-data"

  # Optional: EC2 Placement Group
  placementGroupSelector:
    name: "targeted-pg"
    # Alternative use direct placement group ID
    # id: "pg-02465754522cda020"

  # Optional: Additional EC2 tags (with restrictions)
  tags:
    Environment: "production"
    Team: "platform"
    # Note: Cannot use restricted tags like:
    # - kubernetes.io/cluster/*
    # - karpenter.sh/provisioner-name
    # - karpenter.sh/nodepool
    # - karpenter.sh/nodeclaim
    # - karpenter.sh/managed-by
    # - eks.amazonaws.com/nodeclass
```

考量事項

- 如果您想要確認執行個體有多少本機儲存，可以描述節點以查看暫時性儲存資源。
- 磁碟區加密 - EKS 使用設定的自訂 KMS 金鑰來加密執行個體的唯讀根磁碟區及讀取/寫入資料磁碟區。
- 取代節點 IAM 角色 - 如果您變更與 NodeClass 相關聯的節點 IAM 角色，則需要建立新的存取項目。EKS 在叢集建立期間會自動為節點 IAM 角色建立存取項目。節點 IAM 角色需要 AmazonEKSAutoNodePolicy EKS 存取政策。如需詳細資訊，請參閱the section called “存取項目”。
- 最大 Pod 密度 - EKS 會將節點上的最大 Pod 數量限制為 110。此限制是在現有的最大 Pod 計算之後套用的。如需詳細資訊，請參閱the section called “Amazon EC2 執行個體類型”。
- 標籤 - 如果您想要將標籤從 Kubernetes 傳播到 EC2，需要設定其他 IAM 許可。如需詳細資訊，請參閱the section called “身分和存取”。
- 預設節點類別 - 請勿將您的自訂節點類別命名為 default。這是因為 EKS 自動模式包含一個名為 default 的 NodePool，當您啟用至少一個內建 NodeClass 時會自動佈建該類別。如需啟用內建 NodePools 的詳細資訊，請參閱 the section called “檢閱內建的節點集區”。
- 具有多個子網路的 **subnetSelectorTerms** 行為 - 如果有多個子網路符合 subnetSelectorTerms 條件或您按 ID 提供的子網路，EKS 自動模式會建立分佈於各個子網路中的節點。
  - 如果子網路位於不同的可用區域 (AZ)，您可以使用 Kubernetes 功能，例如 Pod 拓撲分散限制條件和拓撲感知路由，分別將 Pod 和流量分散到各個區域。
  - 如果同一個 AZ 中有多個子網路符合 subnetSelectorTerms，則 EKS 自動模式會在該可用區域中分散在各個子網路的每個節點上建立 Pod。EKS 自動模式會在相同 AZ 中其他子網路中的每個節點建立次要網路介面。它根據每個子網路中可用 IP 位址的數量進行選擇，以更高效地使用子網路。但是，您無法指定 EKS 自動模式為每個 Pod 使用的子網路；如果您需要 Pod 在特定子網路中執行，請改用 the section called “Pod 的個別子網路和安全群組”。
- 置放群組策略限制 — 每個置放群組策略（叢集、分割區、分散）對於執行個體類型、AZs 和容量都有特定限制。如需詳細資訊，請參閱《Amazon EC2 使用者指南》中的置放群組策略。
- 分散置放群組 — 7 個執行個體限制 — 機架層級分散置放群組允許每個群組每個可用區域最多 7 個執行中的執行個體。這會建立下列邊緣案例：
  - 容量遭到封鎖的漂移替換 — EKS Auto Mode 會在終止舊節點之前啟動替換節點。當分散的 PG 在 AZ 中有 7 個執行個體時，取代啟動會失敗，而漂移的節點會持續執行，直到槽釋放為止。
  - 容量為 的所有 AZs — 如果分散 PG 中的每個 AZ 都在其 7 個執行個體限制內，則無法排程替換。漂移或合併候選節點會無限期持續執行。

  - 置放群組外部沒有備用 — EKS Auto Mode 不會嘗試在置放群組外部啟動替代執行個體。
  - 解決方法 — 使用`WhenEmpty`合併政策 (`consolidationPolicy: WhenEmpty`)。只有在所有非協助程式集 Pod 耗盡後，才會刪除節點，釋放 PG 插槽而不需要先替換啟動。請注意，無論合併政策為何，偏離一律使用 replace-then-delete，因此偏離會維持在容量封鎖狀態。
- 叢集置放群組 AZ 鎖定 — 第一個執行個體啟動到叢集置放群組後，PG 會鎖定到該 AZ。如果您的 NodePool 允許多個 AZs，則在初始擴展期間平行啟動可能會競爭：一個成功並釘選 AZ，其餘則失敗，並出現容量錯誤。在您的 NodePool 要求中釘選 AZ，以避免暫時性故障。
- 分割區置放群組 — 支援分割區置放群組，沒有超出標準 EC2 限制的其他限制。
- 合併可將 Pod 移出置放群組：如果 Pod 沒有置放群組排程限制（例如 `nodeSelector`上的 `eks.amazonaws.com/placement-group-id`），合併可能會將其移至 PG 外部的節點。需要置放群組成員資格的應用程式應該透過 Pod 層級限制來表達這一點。
- 不存在或刪除的置放群組 — 如果 `NodeClass`參考不存在或已刪除的置放群組，則不會啟動任何執行個體。置放群組 ID 格式會在許可時驗證，但僅在啟動時檢查是否存在。如果在節點執行時刪除置放群組，現有的節點會標記為偏離，並無限期持續執行，因為偏離取代啟動也會遭到封鎖。
- Hugepages — 如需在節點上設定巨型分頁的相關資訊，請參閱 the section called “在節點上設定巨型分頁”。

Pod 的個別子網路和安全群組

`podSubnetSelectorTerms` 和 `podSecurityGroupSelectorTerms` 欄位允許 Pod 使用與其節點不同的子網路和安全群組，以啟用進階聯網組態。兩個欄位必須一起指定。這種分離可增強對網路流量路由與安全政策的控制。

> **Note**
>
> 此功能與用於非 EKS Auto Mode 運算的 VPC CNI 的 Pod 安全群組 (SGPP) 功能不同。EKS Auto 模式不支援 SGPP。反之，請在 `podSecurityGroupSelectorTerms`中使用 `NodeClass`，將個別的安全群組套用至 Pod 流量。安全群組適用於 `NodeClass`層級，這表示使用 `NodeClass`共用相同 Pod 安全群組之節點上的所有 Pod。

運作方式

當您設定 `podSubnetSelectorTerms`和 時`podSecurityGroupSelectorTerms`：

1. 節點的主要 ENI 使用來自 和 的子網路subnetSelectorTerms和安全群組securityGroupSelectorTerms。只有節點自己的 IP 地址才會指派給此界面。
2. EKS Auto Mode 會在符合 的子網路中建立次要 ENIspodSubnetSelectorTerms，並podSecurityGroupSelectorTerms連接來自 的安全群組。根據預設，Pod IP 地址會使用 /28 字首從這些次要 ENIs 配置，並在連續字首區塊無法使用時自動回復到次要 IPs (/32)。如果在"32"中ipv4PrefixSize將 設定為 advancedNetworking，則只會使用次要 IPs。
3. 中指定的安全群組podSecurityGroupSelectorTerms適用於 VPC 內的 Pod 流量。對於目的地為 VPC 之外的流量，Pod 會使用節點的主要 ENI （及其安全群組），因為來源網路位址轉譯 (SNAT) 會將 Pod IP 轉譯為節點 IP。您可以使用 中的 snatPolicy 欄位修改此行為NodeClass。

使用案例

當您需要podSecurityGroupSelectorTerms時，請使用 podSubnetSelectorTerms和 ：

- 套用不同的安全群組，分別控制節點和 Pod 的流量。
- 將基礎設施流量 node-to-node通訊） 與應用程式流量 (Pod-to-Pod通訊） 分開。
- 對節點子網路和 Pod 子網套用不同的聯網組態。
- 專門為節點流量設定反向代理或網路篩選, 而不影響 Pod 流量。使用 advancedNetworking 和 certificateBundles 來定義您的反向代理，以及代理的任何自簽署或私有憑證。

範例組態

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: advanced-networking
spec:
  role: MyNodeRole

  # Subnets and security groups for EC2 instances (nodes)
  subnetSelectorTerms:
    - tags:
        Name: "node-subnet"
        kubernetes.io/role/internal-elb: "1"

  securityGroupSelectorTerms:
    - tags:
        Name: "eks-cluster-sg"
```

```
  # Separate subnets and security groups for Pods
  podSubnetSelectorTerms:
    - tags:
        Name: "pod-subnet"
        kubernetes.io/role/pod: "1"

  podSecurityGroupSelectorTerms:
  - tags:
      Name: "eks-pod-sg"
```

個別 Pod 子網路和安全群組的考量事項

- 安全群組範圍： 的安全群組podSecurityGroupSelectorTerms會連接至次要 ENIs，並套用至 VPC 內的 Pod 流量。啟用 SNAT 時 （ 預設 snatPolicy: Random)，離開 VPC 的流量會轉譯為節點的主要 ENI IP 地址，因此節點的安全群組securityGroupSelectorTerms會套用至該流量。如果您設定 snatPolicy: Disabled，Pod 會針對所有流量使用自己的 IP 地址，而且您必須確保相應地設定路由和安全群組。
- NodeClass 層級精細程度：Pod 安全群組適用於使用 排程在節點上的所有 PodNodeClass。若要將不同的安全群組套用至不同的工作負載，請建立個別 NodeClass 和 NodePool 資源，並使用污點、容錯或節點選擇器，將工作負載排程到適當的節點。
- 降低 Pod 密度：每個節點上執行的 Pod 較少，因為節點的主要網路介面預留給節點 IP，且無法用於 Pod。
- 子網路選擇器限制：標準subnetSelectorTerms和securityGroupSelectorTerms組態不適用於 Pod 子網路或安全群組選擇。
- 網路規劃：確保節點和 Pod 子網路中有足夠的 IP 位址空間，以支援您的工作負載需求。
- 路由組態：確認已正確設定 Pod 子網路的路由表和網路存取控制清單 (ACL)，以便在節點和 Pod 子網路之間進行通訊。
- 可用區域：確認您已在多個 AZ 中建立 Pod 子網路。如果您使用的是特定的 Pod 子網路，它必須與節點子網路 AZ 位於相同的 AZ 中。

Pod 的次要 IP 模式

ipv4PrefixSize 欄位透過僅將次要 IP 地址配置給節點來啟用進階聯網組態。此功能不會將字首 (/28) 配置給節點，而且只會維護一個次要 IP 做為 MinimalIPTarget。

使用案例

當您需要以下功能時，可使用 `ipv4PrefixSize`：

- 降低 IP 使用率：每個節點只會暖機一個 IP 地址。
- 較低的 Pod 追逐率：Pod 建立速度不是主要問題。
- 無字首分割：字首導致的分割是使用自動模式的主要考量或封鎖程式。

範例組態

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: advanced-networking
spec:
  role: MyNodeRole

  advancedNetworking:
    ipv4PrefixSize: "32"
```

次要 IP 模式的考量事項

- 降低 Pod 建立速度：由於只有一個次要 IP 已暖機，因此在建立更多 Pod 時，IPAM 服務需要更多時間來佈建 IPs。

在 IPv4IPv6 叢集中停用 IPv6 Pod 的 IPv6 輸出。

`enableV4Egress` 欄位`true`預設為 。對於自動模式 IPv6 叢集，此功能可以停用，因此自動模式不會為 IPv6 Pod 建立僅輸出 IPv4 界面。這很重要，因為 IPv4 輸出界面不受網路政策強制執行的約束。網路政策僅在 Pod 的主要界面 (eth0) 上強制執行。

使用案例

當您需要以下功能時，可使用 `enableV4Egress`：

- 使用 IPv6 叢集：預設允許 IPv4 輸出流量。
- 使用網路政策：目前 EKS 網路政策不支援雙堆疊。停用`enableV4Egress`可防止 Pod 流量意外透過 IPv4 輸出。

範例組態

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: advanced-networking
spec:
  role: MyNodeRole

  advancedNetworking:
    enableV4Egress: false
```

停用 enableV4Egress 的考量事項

- IPv6 叢集中的網路政策：IPv6 叢集預設允許 IPv4 流量。設定 會`enableV4Egress: false`封鎖 IPv4 輸出流量，提供增強的安全性，特別是在與網路政策搭配使用時。

靜態網路界面組態

下的 `networkInterfaces` 欄位`advancedNetworking`可讓您靜態定義在啟動時連接到執行個體的網路介面。這主要用於為分散式訓練和推論工作負載中的高效能節點間通訊設定 Elastic Fabric Adapter (EFA) 裝置。此功能可與靜態容量節點集區配對，以維護預先暖機且 EFA 就緒的節點。如需 EKS 中 EFA 裝置管理的詳細資訊，請參閱 the section called “EFA 裝置”。如需特定執行個體類型的建議 EFA 組態，請參閱《Amazon EC2 使用者指南》中的為啟用 EFA 的執行個體類型最大化網路頻寬。

運作方式

中的每個項目都會`networkInterfaces`定義在啟動期間連接至執行個體的網路界面。每個項目都會指定 `networkCardIndex`（網路卡，其中 `0`是主要）、 `deviceIndex`（該卡上的裝置位置） 和 `interfaceType`(`interface`適用於標準 IP 型流量，或`efa-only`適用於無 IP 地址之 RDMA 流量的 EFA 介面）。主要 ENI (`networkCardIndex: 0`、`deviceIndex: 0`) 必須使用 `interfaceType: interface` 來支援 IP 型節點通訊。這些`efa-only`介面僅支援 RDMA 的 EFA 裝置功能，且無法使用 IP 地址設定。

您可以使用 （每個單位提供 1 個 IP 地址`networkCardIndex: 0`) 或 `secondaryIPv4Count`（每個單位提供具有 16 個 IP 地址的 /28 字首），將 IP 容量指派給主要網路卡 `secondaryIPv4PrefixCount`() 上的介面。每個界面只能使用其中一個，而且僅在 上支援。`networkCardIndex: 0`它們無法在`efa-only`介面上使用。

> **Important**
>
> 設定 networkInterfaces 時，EKS Auto Mode 不會在執行個體啟動後連接其他 IPs、字首或 ENIs。只有啟動時設定的介面和 IP 地址可供 Pod 使用。您必須根據設定的 IPs 數量來規劃 Pod 密度。

考量事項

靜態定義的網路介面不支援 IPv6。定義多個網路介面時， associatePublicIPAddress不相容，因此使用多個介面的節點無法擁有公IPs （僅限公有叢集或混合公有/私有叢集）。

podSecurityGroupSelectorTerms 搭配靜態網路介面使用 podSubnetSelectorTerms和 時，請使用 0 IPs 設定主要 ENI。主要 ENI 使用節點子網路和安全群組，而 Pod 流量則使用次要 ENIs 上的 Pod 子網路組態。

範例：僅限 EFA 的 GPU 訓練界面

下列範例顯示為分散式訓練中使用的 GPU 執行個體NodeClass設定了僅限 EFA 介面的 。主要介面具有 /28 字首 (16 個 Pod IPs) 和 4 個額外的僅限 EFA 介面，可提供 RDMA 連線能力。此組態可與靜態容量節點集區配對，以維護預先暖機的 EFA 就緒節點。

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: efa-training
spec:
  role: MyNodeRole
  subnetSelectorTerms:
    - tags:
        Name: "private-subnet"
  securityGroupSelectorTerms:
    - tags:
        Name: "efa-security-group"
  placementGroupSelector:
    name: "ml-training-pg"
  advancedNetworking:
    networkInterfaces:
    - deviceIndex: 0
      interfaceType: interface
      networkCardIndex: 0
      secondaryIPv4PrefixCount: 1
```

```
    - deviceIndex: 1
      interfaceType: efa-only
      networkCardIndex: 0
    - deviceIndex: 0
      interfaceType: efa-only
      networkCardIndex: 1
    - deviceIndex: 0
      interfaceType: efa-only
      networkCardIndex: 2
    - deviceIndex: 0
      interfaceType: efa-only
      networkCardIndex: 3
```

在節點上設定巨型分頁

使用 advancedCompute.hugepages 欄位預先配置 中節點上的巨型分頁NodeClass。Hugepage 可改善延遲敏感工作負載的記憶體存取效能，例如高效能運算 (HPC)、資料庫和網路密集型應用程式。若要使用巨型分頁，請將您的 Pod 設定為明確請求巨型分頁資源。如需詳細資訊，請參閱 Kubernetes 文件中的管理 HugePages。

範例組態

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: hugepages-compute
spec:
  role: MyNodeRole

  subnetSelectorTerms:
    - tags:
        Name: "private-subnet"

  securityGroupSelectorTerms:
    - tags:
        Name: "eks-cluster-sg"

  advancedCompute:
    hugepages:
      pages:
        - size: "2Mi"
          count: 512
```

Hugepages 考量事項

在 NodeClass 上設定巨型分頁時，請考慮下列事項：

- 記憶體保留： Hugepages 會計入執行個體的總記憶體。當巨型分頁保留超過總記憶體的 80% 時，Amazon EKS 會將執行個體類型標記為與 NodeClass 不相容。
- EFA 交互排除：EFA 和巨型分頁不能在同一個 NodeClass 上共存。如果您設定 `interfaceType: efa-only`，NodeClass 驗證會失敗。Amazon EKS 不會佈建`vpc.amazonaws.com/efa`從巨型分頁 NodeClass 請求的 Pod。將單獨的 NodeClass 用於 EFA 工作負載。

為 EKS 自動模式建立節點集區

Amazon EKS 節點集區提供了彈性的方式來管理 Kubernetes 叢集中的運算資源。本主題示範如何使用 Karpenter 建立和設定節點集區，這是一種有助於最佳化叢集擴展與資源使用率的節點佈建工具。透過 Karpenter 的 NodePool 資源，您可以為運算資源定義特定要求，包括執行個體類型、可用區域、架構和容量類型。

您無法修改內建 `system` 和 `general-purpose` 節點集區。您僅可以啟用或停用它們。如需詳細資訊，請參閱the section called “檢閱內建的節點集區”。

NodePool 規格透過各種支援的標籤和要求，允許對 EKS 叢集的運算資源進行精細控制。這些選項包括指定 EC2 執行個體類別、CPU 組態、可用區域、架構 (ARM64/AMD64) 及容量類型 (Spot 或隨需)。您還可以為 CPU 和記憶體用量設定資源限制，確保您的叢集保持在所需的操作範圍內。

EKS 自動模式利用眾所周知的 Kubernetes 標籤，提供一致且標準化的方式來識別節點特性。這些標籤遵循既定的 Kubernetes 慣例，例如用於可用區域的 `topology.kubernetes.io/zone` 和用於 CPU 架構的 `kubernetes.io/arch`。此外，EKS 特定的標籤 （ 字首為 `eks.amazonaws.com/`) 使用執行個體類型、CPU 製造商、GPU 功能和聯網規格等 AWS特定屬性來擴展此功能。此標準化標籤系統可無縫整合現有的 Kubernetes 工具，同時提供深度 AWS 基礎設施整合。

建立 NodePool

請遵循以下步驟為您的 Amazon EKS 叢集建立 NodePool：

1. 使用所需的 NodePool 組態建立名為 `nodepool.yaml` 的 YAML 檔案。您可以使用以下範例組態。
2. 將 NodePool 套用至您的叢集：

```
kubectl apply -f nodepool.yaml
```

3. 確認已成功建立 NodePool：

```
kubectl get nodepools
```

4. (選用) 監控 NodePool 狀態：

```
kubectl describe nodepool default
```

確保您的 NodePool 參照了叢集中存在的有效 NodeClass。NodeClass 會為您的運算資源定義 AWS特定的組態。如需詳細資訊，請參閱the section called “建立節點類別”。

範例 NodePool

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: my-node-pool
spec:
  template:
    metadata:
      labels:
        billing-team: my-team
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default

      requirements:
        - key: "eks.amazonaws.com/instance-category"
          operator: In
          values: ["c", "m", "r"]
        - key: "eks.amazonaws.com/instance-cpu"
          operator: In
          values: ["4", "8", "16", "32"]
        - key: "topology.kubernetes.io/zone"
          operator: In
          values: ["us-west-2a", "us-west-2b"]
        - key: "kubernetes.io/arch"
          operator: In
          values: ["arm64", "amd64"]

  disruption:
```

```
    consolidationPolicy: Balanced
    consolidateAfter: 30s

  limits:
    cpu: "1000"
    memory: 1000Gi
```

EKS 自動模式支援的標籤

EKS 自動模式支援下列眾所周知的標籤。

> **Note**
>
> EKS 自動模式使用與 Karpenter 不同的標籤。與 EC2 受管執行個體相關的標籤以 `eks.amazonaws.com` 開頭。

標籤	範例	說明
topology.kubernetes.io/zone	us-east-2a	AWS 區域
node.kubernetes.io/instance-type	g4dn.8xlarge	AWS 執行個體類型
kubernetes.io/arch	amd64	架構由執行個體上的 GOARCH 值定義
karpenter.sh/capacity-type	spot	容量類型包括 `spot`、 `on-demand` 和 `reserved`。如果您未指定容量類型，Karpenter 會優先考慮 `reserved`，然後是 `spot`，然後是 `on-demand` 。
eks.amazonaws.com/instance-hypervisor	nitro	使用特定 Hypervisor 的執行個體類型
eks.amazonaws.com/compute-type	auto	識別 EKS 自動模式受管節點
eks.amazonaws.com/instance-encryption-in-transit-supported	true	支援 (或不支援) 傳輸中加密的執行個體類型

標籤	範例	說明
eks.amazonaws.com/instance-category	g	相同類別的執行個體類型，通常為產生編號之前的字串
eks.amazonaws.com/instance-generation	4	執行個體類別內的執行個體類型產生編號
eks.amazonaws.com/instance-family	g4dn	屬性類似，但資源數量不同的執行個體類型
eks.amazonaws.com/instance-size	8xlarge	資源數量類似，但屬性不同的執行個體類型
eks.amazonaws.com/instance-cpu	32	執行個體上的 CPU 數量
eks.amazonaws.com/instance-cpu-manufacturer	aws	CPU 製造商的名稱
eks.amazonaws.com/instance-memory	131072	執行個體上的記憶體 MiB 數
eks.amazonaws.com/instance-ebs-bandwidth	9500	執行個體上可用 EBS 的最大 MiB 數
eks.amazonaws.com/instance-network-bandwidth	131072	執行個體上可用的基準兆位數
eks.amazonaws.com/instance-gpu-name	t4	執行個體上的 GPU 名稱 (如有)
eks.amazonaws.com/instance-gpu-manufacturer	nvidia	GPU 製造商的名稱
eks.amazonaws.com/instance-gpu-count	1	執行個體上的 GPU 數
eks.amazonaws.com/instance-gpu-memory	16384	GPU 上的記憶體 MiB 數

標籤	範例	說明
eks.amazonaws.com/instance-local-nvme	900	執行個體上本機 nvme 儲存體的 GiB 數
eks.amazonaws.com/capacity-reservation-interruptible	true	容量保留是否可中斷。僅存在於具有 的節點上`karpenter.sh/capacity-type: reserved`。

> **Note**
>
> EKS 自動模式僅支援特定執行個體，且具有最小大小需求。如需詳細資訊，請參閱the section called “EKS 自動模式支援的執行個體參考”。

EKS 自動模式不支援的標籤

EKS 自動模式不支援以下標籤。

- EKS 自動模式僅支援 Linux
  - `node.kubernetes.io/windows-build`
  - `kubernetes.io/os`

停用內建節點集區

如果您建立了自訂節點集區，可以停用內建節點集區。如需詳細資訊，請參閱the section called “檢閱內建的節點集區”。

沒有內建節點集區的叢集

您可建立沒有內建節點集區的叢集。當您的組織已建立自訂節點集區時，這會很有幫助。

> **Note**
>
> 當您建立沒有內建節點集區的叢集時，`default` NodeClass 不會自動佈建。您需要建立自訂的 NodeClass。如需詳細資訊，請參閱the section called “建立節點類別”。

概觀：

1. 建立 EKS 叢集，其中 nodePools和 nodeRoleArn值皆為空白。
   - 範例 eksctl autoModeConfig：

     ```
     autoModeConfig:
       enabled: true
       nodePools: []
       # Do not set a nodeRoleARN
     ```

     如需詳細資訊，請參閱the section called “eksctl CLI”
2. 使用節點角色 ARN 建立自訂節點類別
   - 如需詳細資訊，請參閱the section called “建立節點類別”
3. 為自訂節點類別建立存取項目
   - 如需詳細資訊，請參閱the section called “建立節點類別存取項目”
4. 如上所述，建立自訂節點集區。

中斷

您可透過多種方式設定 EKS 自動模式，以透過您的 NodePool 來中斷節點。您可以使用 spec.disruption.consolidationPolicy、spec.disruption.consolidateAfter 或 spec.template.spec.expireAfter。您也可以透過 NodePool 的 對 EKS Auto Mode 的中斷進行評分spec.disruption.budgets。您還可以控制時間範圍和同時中斷的節點數量。有關設定此行為的說明，請參閱 Karpenter 文件中的中斷。

consolidationPolicy 欄位接受三個值：

- WhenEmpty – 僅整合空節點，因此執行中的工作負載永遠不會中斷。
- WhenEmptyOrUnderutilized – 積極整合成本，每當節點的 Pod 可以更便宜地重新封裝時，就會中斷工作負載。
- Balanced – 透過權衡中斷成本與節省成本來對每個合併動作進行評分。略過中斷超過節省量的動作。

如需每個值的詳細資訊，請參閱 Karpenter 文件中的中斷。

您可將節點集區的中斷設定為：

- 識別執行個體未得到充分利用的時間，並合併工作負載。
- 建立節點集區中斷預算，以限制因偏離、空置和合併而導致的節點終止速率。

根據預設，EKS 自動模式：

- 合併未充分利用的執行個體。
- 在 336 小時後終止執行個體。
- 設定單一中斷預算為節點的 10%。
- 當新的自動模式 AMI 發布時，允許因偏離而取代節點，這大約每週發生一次。

終止寬限期

在 `terminationGracePeriod` EKS Auto Mode NodePool 上未明確定義 時，系統會自動將預設的 24 小時終止寬限期套用至相關聯的 NodeClaim。雖然 EKS Auto Mode 客戶在其自訂 NodePool 組態中不會看到`terminationGracePeriod`預設值，但他們會在 NodeClaim 上觀察此預設值。無論寬限期是在 NodePool 上明確設定，還是在 NodeClaim 上預設，功能都保持一致，確保了叢集中可預測的節點終止行為。

EKS Auto 模式的靜態容量節點集區

Amazon EKS Auto Mode 支援靜態容量節點集區，無論 Pod 需求為何，都能維持固定數量的節點。靜態容量節點集區適用於需要可預測容量、預留執行個體或特定合規需求的工作負載，而您需要維持一致的基礎設施使用量。

與根據 Pod 排程需求擴展的動態節點集區不同，靜態容量節點集區會維持您已設定的節點數量。

設定靜態容量節點集區

若要建立靜態容量節點集區，請在 NodePool 規格中設定 `replicas` 欄位。`replicas` 欄位定義節點集區將維護的確切節點數量。如需如何設定 the section called “範例”，請參閱 `replicas`。

靜態容量節點集區考量事項

靜態容量節點集區有幾項重要的限制條件和行為：

組態限制：

- 無法切換模式：在節點集區`replicas`上設定後，就無法將其移除。節點集區無法在靜態和動態模式之間切換。

- 資源限制：限制區段僅支援 `limits.nodes` 欄位。CPU 和記憶體限制不適用。
- 無權重欄位：由於節點選擇不是根據優先順序，因此無法在靜態容量節點集區上設定 `weight` 欄位。

操作行為：

- 無合併：靜態容量集區中的節點不會考慮合併。
- 擴展操作：擴展操作會略過節點中斷預算，但仍遵守 PodDisruptionBudgets。
- 節點替換：節點仍會根據您的組態替換成偏離（例如 AMI 更新）和過期。

最佳實務

容量規劃：

- 設定`limits.nodes`高於 `replicas` 以允許在節點取代操作期間暫時擴展。
- 設定限制時，請考慮節點偏離或 AMI 更新期間所需的最大容量。

執行個體選擇：

- 當您有預留執行個體或特定硬體需求時，請使用特定執行個體類型。
- 避免在擴展期間限制執行個體可用性的過度限制需求。

中斷管理：

- 設定適當的中斷預算，以平衡可用性與維護操作。
- 設定預算百分比時，請考慮您應用程式的節點替換公差。

監控：

- 定期監控 `status.nodes` 欄位，以確保維持所需的容量。
- 設定實際節點計數偏離所需複本時的提醒。

區域分佈：

- 為了實現高可用性，請將靜態容量分散到多個可用區域。

- 當您建立跨越多個可用區域的靜態容量節點集區時，EKS Auto Mode 會將節點分散到指定區域，但不保證分佈均勻。
- 若要跨可用區域進行可預測且均勻的分佈，請建立個別的靜態容量節點集區，每個集區都會使用 `topology.kubernetes.io/zone`需求固定到特定的可用區域。
- 如果您需要將 12 個節點平均分散到三個區域，請建立三個節點集區，每個節點集區各有 4 個複本，而不是一個節點集區，在三個區域各有 12 個複本。

擴展靜態容量節點集區

您可以使用 `kubectl scale`命令變更靜態容量節點集區中的複本數量：

```
# Scale down to 5 nodes
kubectl scale nodepool static-nodepool --replicas=5
```

縮減規模時，EKS Auto Mode 會正常終止節點，遵守 PodDisruptionBudgets，並允許將執行中的 Pod 重新排程到剩餘的節點。

監控靜態容量節點集區

使用下列命令來監控靜態容量節點集區：

```
# View node pool status
kubectl get nodepool static-nodepool

# Get detailed information including current node count
kubectl describe nodepool static-nodepool

# Check the current number of nodes
kubectl get nodepool static-nodepool -o jsonpath='{.status.nodes}'
```

`status.nodes` 欄位顯示節點集區管理的目前節點數量，在正常情況下應符合您所需的`replicas`計數。

疑難排解

未到達所需複本的節點：

- 檢查`limits.nodes`值是否足夠
- 確認您的需求不會過度限制執行個體選取

- 檢閱您正在使用的執行個體類型和區域的 AWS 服務配額

節點替換耗時太長：

- 調整中斷預算以允許更多並行取代
- 檢查 PodDisruptionBudgets 是否阻止節點終止

非預期的節點終止：

- 檢閱 `expireAfter`和 `terminationGracePeriod`設定
- 檢查手動節點終止或 AWS 維護事件

範例

基本靜態容量節點集區

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: basic-static
spec:
  replicas: 5

  template:
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default

      requirements:
        - key: "eks.amazonaws.com/instance-category"
          operator: In
          values: ["m"]
        - key: "topology.kubernetes.io/zone"
          operator: In
          values: ["us-west-2a"]

  limits:
    nodes: 8  # Allow scaling up to 8 during operations
```

具有特定執行個體類型的靜態容量

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: reserved-instances
spec:
  replicas: 20

  template:
    metadata:
      labels:
        instance-type: reserved
        cost-center: production
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default

      requirements:
        - key: "node.kubernetes.io/instance-type"
          operator: In
          values: ["m5.2xlarge"]  # Specific instance type
        - key: "karpenter.sh/capacity-type"
          operator: In
          values: ["on-demand"]
        - key: "topology.kubernetes.io/zone"
          operator: In
          values: ["us-west-2a", "us-west-2b", "us-west-2c"]

  limits:
    nodes: 25

  disruption:
    # Conservative disruption for production workloads
    budgets:
      - nodes: 10%
```

多區域靜態容量節點集區

```
apiVersion: karpenter.sh/v1
kind: NodePool
```

```
metadata:
  name: multi-zone-static
spec:
  replicas: 12  # Will be distributed across specified zones

  template:
    metadata:
      labels:
        availability: high
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default

      requirements:
        - key: "eks.amazonaws.com/instance-category"
          operator: In
          values: ["c", "m"]
        - key: "eks.amazonaws.com/instance-cpu"
          operator: In
          values: ["8", "16"]
        - key: "topology.kubernetes.io/zone"
          operator: In
          values: ["us-west-2a", "us-west-2b", "us-west-2c"]
        - key: "karpenter.sh/capacity-type"
          operator: In
          values: ["on-demand"]

  limits:
    nodes: 15

  disruption:
    budgets:
      - nodes: 25%
```

具有容量保留的靜態容量

下列範例示範如何搭配 EC2 容量保留使用靜態容量節點集區。如需搭配 EKS Auto 模式使用 EC2 容量預留的詳細資訊，請參閱 the section called “控制容量保留”。

NodeClass 定義 capacityReservationSelectorTerms

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: capacity-reservation-nodeclass
spec:
  role: AmazonEKSNodeRole
  securityGroupSelectorTerms:
  - id: sg-0123456789abcdef0
  subnetSelectorTerms:
  - id: subnet-0123456789abcdef0
  capacityReservationSelectorTerms:
  - id: cr-0123456789abcdef0
```

NodePool 使用 參考上述 NodeClass和 karpenter.sh/capacity-type: reserved。

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: static-capacity-reservation-nodepool
spec:
  replicas: 5
  limits:
    nodes: 8  # Allow scaling up to 8 during operations
  template:
    metadata: {}
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: capacity-reservation-nodeclass
      requirements:
      - key: karpenter.sh/capacity-type
        operator: In
        values: ['reserved']
```

建立 IngressClass 以設定 Application Load Balancer

EKS 自動模式會自動執行負載平衡的常規任務，包括將叢集應用程式公開至網際網路。

AWS 建議使用 Application Load Balancer (ALB) 來提供 HTTP 和 HTTPS 流量。Application Load Balancer 可根據請求的內容路由請求。如需 Application Load Balancer 的詳細資訊，請參閱什麼是 Elastic Load Balancing？

EKS 自動模式會建立與設定 Application Load Balancer (ALB)。例如，EKS Auto Mode 會在您建立 `Ingress` Kubernetes 物件時建立負載平衡器，並將其設定為將流量路由至叢集工作負載。

概觀

1. 建立您想要公開到網際網路的工作負載。
2. 建立 `IngressClassParams` 資源，指定 AWS 特定組態值，例如用於 SSL/TLS 和 VPC 子網路的憑證。
3. 建立 `IngressClass` 資源，指定 EKS 自動模式將作為該資源的控制器。
4. 建立將 HTTP 路徑和連接埠與叢集工作負載建立關聯的`Ingress`資源。

EKS 自動模式將建立 Application Load Balancer，指向 `Ingress` 資源中指定的工作負載，並使用在 `IngressClassParams` 資源中指定的負載平衡器配置。

先決條件

- Amazon EKS 叢集上已啟用 EKS 自動模式
- Kubectl 已設定為連接到您的叢集
  - 您可使用 `kubectl apply -f <filename>`，將以下範例組態 YAML 檔案套用至您的叢集。

> **Note**
>
> EKS 自動模式需要子網路標籤來識別公有和私有子網路。
> 若您使用 `eksctl` 建立叢集，則您已具備這些標籤。
> 了解如何 the section called “標記子網路”。

步驟 1：建立工作負載

首先，建立您想要公開到網際網路的工作負載。此工作負載可為任何處理 HTTP 流量的 Kubernetes 資源，例如部署或服務。

此範例使用名為 `service-2048` 的簡單 HTTP 服務，其接聽連接埠 `80`。透過套用下列資訊清單 `2048-deployment-service.yaml`，建立此服務及其部署：

```
---
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deployment-2048
spec:
  selector:
    matchLabels:
      app.kubernetes.io/name: app-2048
  replicas: 2
  template:
    metadata:
      labels:
        app.kubernetes.io/name: app-2048
    spec:
      containers:
        - image: public.ecr.aws/l6m2t8p7/docker-2048:latest
          imagePullPolicy: Always
          name: app-2048
          ports:
            - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: service-2048
spec:
  ports:
    - port: 80
      targetPort: 80
      protocol: TCP
  type: NodePort
  selector:
    app.kubernetes.io/name: app-2048
```

將組態套用至您的叢集：

```
kubectl apply -f 2048-deployment-service.yaml
```

上面列出的資源將在預設命名空間中建立。您可以透過執行以下命令來進行確認：

```
kubectl get all -n default
```

步驟 2：建立 IngressClassParams

建立 IngressClassParams 物件以指定 Application Load Balancer AWS 的特定組態選項。在本範例中，我們在名為 alb-ingressclassparams.yaml 的檔案中，建立名為 alb 的 IngressClassParams 資源 (您將在下一步中使用)，並指定負載平衡器方案為 internet-facing。

```
apiVersion: eks.amazonaws.com/v1
kind: IngressClassParams
metadata:
  name: alb
spec:
  scheme: internet-facing
```

將組態套用至您的叢集：

```
kubectl apply -f alb-ingressclassparams.yaml
```

步驟 3：建立 IngressClass

建立 IngressClass，參考名為 的檔案中IngressClassParams資源中設定 AWS 的特定組態值alb-ingressclass.yaml。請記下 IngressClass 的名稱。在此範例中，IngressClass 和 IngressClassParams 都命名為 alb。

使用 is-default-class 註釋來控制 Ingress 資源是否預設使用此類別。

```
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  name: alb
  annotations:
    # Use this annotation to set an IngressClass as Default
    # If an Ingress doesn't specify a class, it will use the Default
    ingressclass.kubernetes.io/is-default-class: "true"
spec:
  # Configures the IngressClass to use EKS Auto Mode
  controller: eks.amazonaws.com/alb
  parameters:
    apiGroup: eks.amazonaws.com
    kind: IngressClassParams
```

```
    # Use the name of the IngressClassParams set in the previous step
    name: alb
```

關於設定選項的詳細資訊，請參閱 the section called “IngressClassParams 參考”。

將組態套用至您的叢集：

```
kubectl apply -f alb-ingressclass.yaml
```

步驟 4：建立傳入

在名為 alb-ingress.yaml 的檔案中建立 Ingress 資源。此資源的用途是將 Application Load Balancer 上的路徑與連接埠，與您叢集中的工作負載相關聯。在此範例中，我們建立名為 Ingress 的資源 2048-ingress，將其流量路由到連接埠 80 上名為 service-2048 的服務。

如需關於設定資源的詳細資訊，請參閱 Kubernetes 文件中的傳入。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: 2048-ingress
spec:
  # this matches the name of IngressClass.
  # this can be omitted if you have a default ingressClass in cluster: the one with
 ingressclass.kubernetes.io/is-default-class: "true"  annotation
  ingressClassName: alb
  rules:
    - http:
        paths:
          - path: /*
            pathType: ImplementationSpecific
            backend:
              service:
                name: service-2048
                port:
                  number: 80
```

將組態套用至您的叢集：

```
kubectl apply -f alb-ingress.yaml
```

步驟 5：檢查狀態

使用 kubectl 尋找 Ingress 的狀態。負載平衡器可能需要幾分鐘才能就緒。

使用您在上一步中設定的 Ingress 資源名稱。例如：

```
kubectl get ingress 2048-ingress
```

資源就緒後，擷取負載平衡器的網域名稱。

```
kubectl get ingress 2048-ingress -o
 jsonpath='{.status.loadBalancer.ingress[0].hostname}'
```

若要在 Web 瀏覽器中檢視服務，請檢閱 Ingress 資源中指定的連接埠和路徑。

步驟 6：清除

要清理負載平衡器，請使用以下命令：

```
kubectl delete ingress 2048-ingress
kubectl delete ingressclass alb
kubectl delete ingressclassparams alb
```

EKS Auto Mode 會自動刪除您 AWS 帳戶中相關聯的負載平衡器。

IngressClassParams 參考

下表屬於常用組態選項的快速參考。

欄位	說明	範例值
scheme	定義 ALB 是內部，還是可面向網際網路	internet-facing
namespaceSelector	限制哪些命名空間可使用此 IngressClass	environment: prod
group.name	將多個傳入分組，以共用單一 ALB	retail-apps
ipAddressType	設定 ALB 的 IP 位址類型	dualstack

欄位	說明	範例值
subnets.ids	用於 ALB 部署的子網路 ID 清單	subnet-xxxx, subnet-yyyy
subnets.tags	用於選擇 ALB 子網路的標籤篩選器	Environment: prod
certificateARNs	要使用的 SSL 憑證 ARN	arn:aws: acm:region:account:certificate/id
tags	AWS 資源的自訂標籤	Environment: prod, Team: platform
loadBalancerAttributes	負載平衡器特定屬性	idle_timeout.timeout_seconds: 60

考量事項

- 您無法使用 IngressClass 上的註釋來設定 EKS 自動模式的負載平衡器。IngressClass 組態應該透過 IngressClassParams 完成。不過，您可以在個別輸入資源上使用註釋來設定負載平衡器行為（例如 alb.ingress.kubernetes.io/security-group-prefix-lists或 alb.ingress.kubernetes.io/conditions.*)。
- 您無法使用 EKS 自動模式來設定 ListenerAttribute。
- 您必須更新叢集 IAM 角色，才能啟用從 Kubernetes 到 AWS Load Balancer資源的標籤傳播。如需詳細資訊，請參閱the section called “EKS Auto 資源的自訂 AWS 標籤”。
- 如需將資源與 EKS Auto Mode 或 self-managed AWS Load Balancer Controller 建立關聯的資訊，請參閱 the section called “移轉參考”。
- 有關修復負載平衡器問題的資訊，請參閱 the section called “疑難排解”。
- 有關使用 EKS 自動模式負載平衡功能的更多考量，請參閱 the section called “Load balancing”。

以下表格詳細比較了 EKS 自動模式中 IngressClassParams、傳入註釋和 TargetGroupBinding 組態的變更。這些資料表突顯了 EKS 自動模式的負載平衡功能與開放原始碼負載平衡器控制器之間的主要差異，包括 API 版本變更、已棄用的功能及更新的參數名稱。

IngressClassParams

先前的	新增	說明
elbv2.k8s.aws/v1beta1	eks.amazonaws.com/v1	API 版本變更
spec.certificateArn	spec.certificateARNs	支援多個憑證 ARN
spec.subnets.tags	spec.subnets.matchTags	變更子網路相符的結構描述
spec.listeners.listenerAttributes	不支援	EKS 自動模式尚不支援

傳入註釋

先前的	新增	說明
kubernetes.io/ingress.class	不支援	在傳入物件上使用 spec.ingressClassName
alb.ingress.kubernetes.io/group.name	不支援	僅在 IngressClass 中指定群組
alb.ingress.kubernetes.io/waf-acl-id	不支援	請改用 WAF v2
alb.ingress.kubernetes.io/web-acl-id	不支援	請改用 WAF v2
alb.ingress.kubernetes.io/wafv2-acl-name	不支援	即將推出
alb.ingress.kubernetes.io/dry-run-plan	不支援	目前不支援乾執行計劃

先前的	新增	說明
alb.ingress.kubernetes.io/create-acm-cert	不支援	目前不支援建立 ACM 憑證
alb.ingress.kubernetes.io/acm-pca-arn	不支援	目前不支援 ACM PCA ARN
alb.ingress.kubernetes.io/auth-type: oidc	不支援	目前不支援 OIDC 驗證類型

TargetGroupBinding

先前的	新增	說明
elbv2.k8s.aws/v1beta1	eks.amazonaws.com/v1	API 版本變更
spec.targetType 選用	spec.targetType 必要	明確的目標類型規格
spec.networking.ingress.from	不支援	不再支援沒有安全群組的 NLB

若要使用自訂 TargetGroupBinding 功能，您必須使用叢集名稱的 eks：eks-cluster-name 標籤來標記目標群組，以授予控制器必要的 IAM 許可。請注意，刪除 TargetGroupBinding 資源或叢集時，控制器會刪除目標群組。

使用服務注釋設定 Network Load Balancer

了解如何使用 Kubernetes 服務註釋，在 Amazon EKS 中設定 Network Load Balancer (NLB)。本主題解釋了 EKS 自動模式支援用於自訂 NLB 行為的註釋，包括網際網路可存取性、運作狀態檢查、SSL/TLS 終止及 IP 目標模式。

當您在 EKS Auto Mode LoadBalancer中建立 類型的 Kubernetes 服務時，EKS 會根據您指定的註釋自動佈建和設定 AWS Network Load Balancer。這種宣告式方法允許您直接透過 Kubernetes 資訊清單來管理負載平衡器組態，維持基礎結構即程式碼的實踐。

EKS 自動模式預設為處理所有 LoadBalancer 類型服務的 Network Load Balancer 佈建 - 無需安裝或設定額外的控制器。loadBalancerClass: eks.amazonaws.com/nlb 規格被自動設定為叢集預設值，簡化了部署程序，同時保持了與現有 Kubernetes 工作負載的相容性。

> **Note**
>
> EKS 自動模式需要子網路標籤來識別公有和私有子網路。
> 若您使用 eksctl 建立叢集，則您已具備這些標籤。
> 了解如何 the section called “標記子網路”。

範例服務

如需有關 Kubernetes Service 資源的詳細資訊，請參閱 Kubernetes 文件中。

請檢閱下面的範例 Service 資源：

```
apiVersion: v1
kind: Service
metadata:
  name: echoserver
  annotations:
    # Specify the load balancer scheme as internet-facing to create a public-facing
 Network Load Balancer (NLB)
    service.beta.kubernetes.io/aws-load-balancer-scheme: internet-facing
spec:
  selector:
    app: echoserver
  ports:
    - port: 80
      targetPort: 8080
      protocol: TCP
  type: LoadBalancer
  # Specify the new load balancer class for NLB as part of EKS Auto Mode feature
  # For clusters with Auto Mode enabled, this field can be omitted as it's the default
  loadBalancerClass: eks.amazonaws.com/nlb
```

常用註釋

下表列出 Amazon EKS Auto Mode 支援的註釋。Amazon EKS Auto Mode 不支援 AWS Load Balancer控制器中提供的所有服務註釋。

> **Tip**
>
> 下列所有註釋都需要加上字首 `service.beta.kubernetes.io/`

欄位	說明	範例
`aws-load-balancer-type`	指定負載平衡器類型。將 `external` 用於新的部署。	`external`
`aws-load-balancer-nlb-target-type`	指定是將流量路由到節點執行個體，還是直接路由到 Pod IP。將 `instance` 用於標準部署，或將 `ip` 用於直接 Pod 路由。	`instance`
`aws-load-balancer-scheme`	控制負載平衡器是內部或是面向網際網路。	`internet-facing`
`aws-load-balancer-healthcheck-protocol`	目標群組的運作狀態檢查通訊協定。常見選項是 TCP (預設) 或 HTTP。	`HTTP`
`aws-load-balancer-healthcheck-path`	使用 HTTP/HTTPS 通訊協定時，用於運作狀態檢查的 HTTP 路徑。	`/healthz`
`aws-load-balancer-healthcheck-port`	用於運作狀態檢查的連接埠。可以是特定連接埠號碼或 `traffic-port` 。	`traffic-port`

欄位	說明	範例
aws-load-balancer-subnets	指定要在哪些子網路中建立負載平衡器。可使用子網路 ID 或名稱。	subnet-xxxx, subnet-yyyy
aws-load-balancer-ssl-cert	來自 AWS Certificate Manager for HTTPS/TLS 的 SSL 憑證 ARN。	arn:aws: acm:region:account:certificate/cert-id
aws-load-balancer-ssl-ports	指定哪些連接埠應使用 SSL/TLS。	443, 8443
load-balancer-source-ranges	允許存取負載平衡器的 CIDR 範圍。	10.0.0.0/24, 192.168.1.0/24
aws-load-balancer-additional-resource-tags	要套用至負載平衡器和相關資源的其他 AWS 標籤。	Environment=prod,Team=platform
aws-load-balancer-ip-address-type	指定負載平衡器是使用 IPv4，還是雙堆疊 (IPv4 + IPv6)。	ipv4 或 dualstack
aws-load-balancer-listener-attributes.${Protocol}-${Port}	指定要套用至接聽程式的接聽程式屬性。	tcp.idle_timeout.seconds=350

考量事項

- 您必須更新叢集 IAM 角色，才能啟用從 Kubernetes 到 AWS Load Balancer資源的標籤傳播。如需詳細資訊，請參閱the section called “EKS Auto 資源的自訂 AWS 標籤”。
- 如需將資源與 EKS Auto Mode 或 self-managed AWS Load Balancer Controller 建立關聯的資訊，請參閱 the section called “移轉參考”。
- 有關修復負載平衡器問題的資訊，請參閱 the section called “疑難排解”。
- 有關使用 EKS 自動模式負載平衡功能的更多考量，請參閱 the section called “Load balancing”。

當移轉至 EKS 自動模式進行負載平衡時，服務註釋與資源組態中需要進行多項變更。以下表格概述了先前與新實作之間的關鍵差異，包括不支援的選項和推薦的替代方案。

服務註釋

先前的	新增	說明
service.beta.kubernetes.io/load-balancer-source-ranges	不支援	在服務上使用 spec.loadBalancerSourceRanges
service.beta.kubernetes.io/aws-load-balancer-type	不支援	在服務上使用 spec.loadBalancerClass
service.beta.kubernetes.io/aws-load-balancer-internal	不支援	使用 service.beta.kubernetes.io/aws-load-balancer-scheme
service.beta.kubernetes.io/aws-load-balancer-proxy-protocol	不支援	改用 service.beta.kubernetes.io/aws-load-balancer-target-group-attributes
各種負載平衡器屬性	不支援	使用 service.beta.kubernetes.io/aws-load-balancer-attributes
service.beta.kubernetes.io/aws-load-balancer-access-log-enabled	不支援	改用 service.beta.kubernetes.io/aws-load-balancer-attributes
service.beta.kubernetes.io/aws-load-balancer-access-log-s3-bucket-name	不支援	改用 service.beta.kubernetes.io/aws-load-balancer-attributes

先前的	新增	說明
service.beta.kubernetes.io/aws-load-balancer-access-log-s3-bucket-prefix	不支援	改用 service.beta.kubernetes.io/aws-load-balancer-attributes
service.beta.kubernetes.io/aws-load-balancer-cross-zone-load-balancing-enabled	不支援	改用 service.beta.kubernetes.io/aws-load-balancer-attributes

要從已棄用的負載平衡器屬性註釋移轉，請將這些設定合併至 service.beta.kubernetes.io/aws-load-balancer-attributes 註釋中。此註釋接受逗號分隔的鍵值對清單，用於各種負載平衡器屬性。例如，若要指定存取記錄和跨區域負載平衡，請使用下列格式：

```
service.beta.kubernetes.io/aws-load-balancer-attributes:
 access_logs.s3.enabled=true,access_logs.s3.bucket=my-bucket,access_logs.s3.prefix=my-prefix,load_balancing.cross_zone.enabled=true
```

這種合併格式提供了一種更一致、更靈活的方式來設定負載平衡器屬性，同時減少了所需單獨注釋的數量。請檢閱您現有的服務組態，並將其更新為使用此合併格式。

TargetGroupBinding

先前的	新增	說明
elbv2.k8s.aws/v1beta1	eks.amazonaws.com/v1	API 版本變更
spec.targetType 選用	spec.targetType 必要	明確的目標類型規格
spec.networking.ingress.from	不支援	不再支援沒有安全群組的 NLB

注意：若要使用自訂 TargetGroupBinding 功能，您必須使用叢集名稱的`eks:eks-cluster-name`標籤來標記目標群組，以授予控制器必要的 IAM 許可。請注意，刪除 TargetGroupBinding 資源或叢集時，控制器會刪除目標群組。

設定多叢集目標群組

根據預設，Amazon EKS Auto Mode 負載平衡器控制器會完全控制其設定的目標群組。當目標群組向控制器註冊時，它會取消註冊目前不在叢集中的任何目標。這可確保目標群組反映該叢集所管理的確切目標集。

啟用 MultiCluster 支援會變更此行為。啟用 MultiCluster 支援的目標群組可以與多個 Kubernetes 叢集建立關聯，或支援來自其他來源的任意目標。每個叢集只會註冊和取消註冊其擁有的目標，而不是取消註冊屬於其他叢集或來源的目標。

運作方式

啟用 MultiCluster 支援時，每個叢集會追蹤其管理的目標集，並僅協調共用目標群組中的這些目標。Amazon EKS Auto Mode 會將每個叢集追蹤的目標儲存在`ConfigMap`名為 的 Kubernetes 中`eks-lbc-targets-<targetgroupbinding-name>`，與負載平衡器資源位於相同的命名空間中。

> **Note**
>
> 上游 AWS Load Balancer控制器會將此狀態存放在`ConfigMap`具有 `aws-lbc-targets-`字首的 中。在 Amazon EKS Auto Mode 上，字首改為 `eks-lbc-targets-`，因此在遵循上游指引時，請檢查 `eks-lbc-targets-` ConfigMap。

負載平衡器會將流量平均分配到所有已註冊的目標。控制器不支援跨叢集的加權分佈。MultiCluster 支援適用於 `ip`和 `instance` 目標類型。

啟用 MultiCluster 支援

根據您的資源類型，使用下列其中一種方法啟用 MultiCluster 支援：

- TargetGroupBinding — 設定 `spec.multiClusterTargetGroup: true`。
- 輸入 (ALB) — 新增註釋 `alb.ingress.kubernetes.io/multi-cluster-target-group: "true"`。
- 服務 (NLB) — 新增註釋 `service.beta.kubernetes.io/aws-load-balancer-multi-cluster-target-group: "true"`。

Amazon EKS 自動模式的考量事項

AWS Load Balancer控制器 GitHub 網站上的上游MultiCluster目標群組指南會逐步解說在兩個叢集之間共用一個目標群組。如需多叢集架構的廣泛討論，請參閱 AWS 部落格上的使用 Amazon EKS 建置彈性多叢集應用程式。相同的流程適用於具有下列差異的 Amazon EKS Auto Mode：

- `TargetGroupBinding` 自訂資源位於 `eks.amazonaws.com/v1` API 群組中，而不是 `elbv2.k8s.aws/v1beta1`。
- `IngressClass` 控制器為 `eks.amazonaws.com/alb`。
- 請勿設定`alb.ingress.kubernetes.io/tags`註釋。`AmazonEKSLoadBalancingPolicy` 受管政策限制`aws:TagKeys`為允許清單，因此自訂標籤會導致存取遭拒錯誤，且控制器無法協調它們。這可能會導致傳入定稿器在刪除時卡住。
- 所有參與的叢集都必須位於相同的 VPC 中，因為目標群組的範圍是單一 VPC。

範例

如需在兩個叢集之間共用單一目標群組step-by-step演練，其中一個叢集透過輸入擁有負載平衡器，另一個透過 聯結`TargetGroupBinding`，請參閱 the section called “共用目標群組”。

在兩個叢集之間共用目標群組

下列範例會在兩個叢集之間共用一個目標群組。第一個叢集透過輸入擁有 Application Load Balancer，而第二個叢集透過手動建立的 加入共用目標群組`TargetGroupBinding`。開始之前，請檢閱 the section called “多叢集目標群組” 以了解 MultiCluster 支援的運作方式，以及如何啟用它。

叢集 1 — 透過輸入擁有負載平衡器

部署您的應用程式（例如，使用 `NodePort`服務的部署），然後建立`Ingress`具有多叢集註釋的`IngressClass`和 。

```
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  name: alb-mc
spec:
  controller: eks.amazonaws.com/alb
---
apiVersion: networking.k8s.io/v1
```

```
kind: Ingress
metadata:
  name: echoserver
  namespace: echoserver
  annotations:
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/target-type: instance
    alb.ingress.kubernetes.io/multi-cluster-target-group: "true"
spec:
  ingressClassName: alb-mc
  rules:
    - http:
        paths:
          - path: /
            pathType: Exact
            backend:
              service:
                name: echoserver
                port:
                  number: 80
```

控制器會佈建負載平衡器，並使用 產生 TargetGroupBinding `spec.multiClusterTargetGroup: true`。請注意所產生 TargetGroupBinding和 服務的下列值 — 第二個叢集需要這三個值：

- `targetGroupARN`。
- 服務 `nodePort`。
- 來自 的負載平衡器安全群組 ID`spec.networking.ingress[].from[].securityGroup.groupID`。

叢集 2 — 透過 TargetGroupBinding 加入共用目標群組

部署相同的應用程式，將服務`nodePort`從叢集 1 固定到 ，然後建立`TargetGroupBinding`參考叢集 1 目標群組 ARN 和安全群組 ID 的 。

```
apiVersion: eks.amazonaws.com/v1
kind: TargetGroupBinding
metadata:
  name: mc-echoserver
  namespace: echoserver
spec:
```

```
  multiClusterTargetGroup: true
  serviceRef:
    name: echoserver
    port: 80
  targetGroupARN: <cluster-1-target-group-arn>
  targetType: instance
  networking:
    ingress:
      - from:
          - securityGroup:
              groupID: <cluster-1-alb-security-group-id>
        ports:
          - port: <cluster-2-node-port>
            protocol: TCP
```

在兩個繫結上啟用 MultiCluster 支援後，每個叢集只會註冊和取消註冊自己的目標。在單一共用目標群組中，兩個叢集的節點都正常運作，負載平衡器會將流量分配到所有叢集。

將其他 Kubernetes RBAC 授予 EKS Auto Mode 受管控制器

Amazon EKS Auto Mode 會代表您管理下列控制器：負載平衡、聯網、運算、區塊儲存和叢集洞察。這些控制器會在 EKS 控制平面上執行。它們會驗證叢集做為AWSServiceRoleForAmazonEKS服務連結角色，其叢集內許可來自連接到該存取項目的 AWS 受管存取政策。

您可以自行擴充受管控制器的 Kubernetes RBAC。當受管政策尚未授予必要的許可時，這會很有用，例如在特定 get上Secret。

在 Amazon EKS Auto Mode 叢集上， 的自動建立存取項目AWSServiceRoleForAmazonEKS也會宣告 Kubernetes 群組 eks:managed。Kubernetes RBAC 是附加項目。RoleBinding （或 ClusterRoleBinding)，其subjects:名稱為 eks:managed群組在受管存取政策之上授予受管控制器額外許可。

運作方式

- EKS Auto Mode 受管控制器會以服務連結角色 的形式連接至您的叢集AWSServiceRoleForAmazonEKS。
- 該角色的自動建立存取項目宣告 kubernetesGroups: ["eks:managed"]。
- 您撰寫描述其他許可的 Role（命名空間） 或 ClusterRole（全叢集），以及 eks:managed 群組ClusterRoleBindingsubjects:名稱的 RoleBinding /。
- Kubernetes RBAC 結合了您的繫結與受管存取政策，因此受管控制器會收到兩組許可。

若要確認群組存在，請執行：

```
CLUSTER=<your-cluster>
REGION=<your-region>
ACCT=$(aws sts get-caller-identity --query Account --output text)

aws eks describe-access-entry \
  --cluster-name "$CLUSTER" --region "$REGION" \
  --principal-arn "arn:aws:iam::${ACCT}:role/aws-service-role/eks.amazonaws.com/
AWSServiceRoleForAmazonEKS" \
  --query 'accessEntry.kubernetesGroups'
```

預期的輸出結果：

```
[
    "eks:managed"
]
```

嚴格限制授予的範圍

eks:managed 群組會套用至所有 EKS Auto Mode 受管控制器。根據您的使用案例允許的範圍縮小額外授予的範圍，因此您只公開受管控制器所需的內容：

- 偏好 Role+RoleBinding （命名空間） 而不是 ClusterRole+ClusterRoleBinding （全叢集）。
- 將 resources、 resourceNames和 限制verbs為解除封鎖使用案例的最小集合。例如，get在名為 的單一 上授予 Secret，而不是在所有Secret物件list, watch上授予 。
- 刪除不再需要的 Role和 RoleBinding 。

Amazon EKS 自動模式的考量事項

- 此機制只會擴展 EKS Auto Mode 受管控制器的 Kubernetes RBAC。它不會變更受管存取政策授予的 AWS IAM 許可。
- eks:managed 群組存在於服務連結角色為 的 EKS Auto Mode 叢集上AWSServiceRoleForAmazonEKS。
- 與 的繫結eks:managed適用於每個 EKS Auto Mode 受管控制器。若要保持實際的爆量半徑較小，請使用 resourceNames和命名空間Role物件。這只會公開您命名的資源，而且只會透過您授予的動詞。

- 這是橋接模式，適用於受管存取政策尚未涵蓋您所需許可的情況。如果 AWS 受管政策已涵蓋 許可，請改用 受管政策。

範例

如需在名為 OIDC `get`上授予 EKS Auto Mode 負載平衡器控制器step-by-step演練`Secret`，請取消封鎖 ALB 上的`alb.ingress.kubernetes.io/auth-type: oidc`註釋`Ingress`，請參閱 the section called “授予對 LBC 的秘密存取權”。

授予 EKS Auto Mode 負載平衡器控制器對特定 Secret 的存取權

在本教學課程中，您會授予 Amazon EKS Auto Mode 負載平衡器控制器對特定 Kubernetes 的讀取存取權`Secret`。這會解除封鎖 ALB 上的 OIDC 身分驗證`Ingress`。您可以使用以`eks:managed`群組為`RoleBinding`目標`Role`的命名空間範圍。

本教學課程大約需要 15 分鐘才能完成。您建立的 Kubernetes RBAC 物件不會產生 AWS 費用。

Amazon EKS Auto Mode 負載平衡器控制器支援 ALB `Ingress` 物件的 OIDC 身分驗證。它使用`alb.ingress.kubernetes.io/auth-type: oidc`註釋來啟用此身分驗證。若要解決 OIDC 組態，控制器會讀取`alb.ingress.kubernetes.io/auth-idp-oidc`註釋中`Secret`名為 的 Kubernetes。其受管存取政策目前未`get`授予`Secret`物件，因此對帳程式失敗，並顯示類似如下的訊息：

```
Failed build model due to ingress: <ns>/<name>:
  secrets "<secret>" is forbidden:
  User "arn:aws:sts::<acct>:assumed-role/AWSServiceRoleForAmazonEKS/<session>"
  cannot get resource "secrets" in API group "" in the namespace "<ns>"
```

使用中所述的機制the section called “將 RBAC 授予受管控制器”，您可以在其`Secret`所需的 OIDC `get`上授予受管控制器。

先決條件

開始之前，請確定您已：

- 具有自動建立`AWSServiceRoleForAmazonEKS`存取項目的 Amazon EKS Auto Mode 叢集。
- `kubectl` 已設定叢集存取。
- 安裝並設定具有 `aws sts get-caller-identity`和 AWS 許可的 CLI`aws eks describe-access-entry`。

- 使用alb.ingress.kubernetes.io/auth-type: oidc註釋及其Secret參考的 Kubernetes Ingress的 ALB。

步驟 1：識別要新增至允許清單的秘密

ALB 上的 OIDC 組態Ingress如下所示。記錄Secret名稱和Ingress命名空間 — RBAC 物件必須位於與 相同的命名空間中Secret。

```
annotations:
  alb.ingress.kubernetes.io/auth-type: oidc
  alb.ingress.kubernetes.io/auth-idp-oidc: >
    {
      "issuer": "...",
      "authorizationEndpoint": "...",
      "tokenEndpoint": "...",
      "userInfoEndpoint": "...",
      "secretName": "oidc"
    }
```

此範例的其餘部分使用命名空間monitoring和Secret名稱 oidc。

步驟 2：套用命名空間範圍的角色和 RoleBinding

此表單會完全授予get一個命名空間Secret中名為 的 。將下列項目儲存為 eks-managed-oidc-secret-reader.yaml：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: eks-managed-oidc-secret-reader
  namespace: monitoring
rules:
- apiGroups: [""]
  resources: ["secrets"]
  resourceNames: ["oidc"]
  verbs: ["get"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: eks-managed-oidc-secret-reader
  namespace: monitoring
```

```
subjects:
- kind: Group
  name: eks:managed
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: eks-managed-oidc-secret-reader
  apiGroup: rbac.authorization.k8s.io
```

套用它：

```
kubectl apply -f eks-managed-oidc-secret-reader.yaml
```

若要稍後將更多Secret物件新增至允許清單，請將它們的名稱附加到Role's `resourceNames清單並重新套用。

步驟 3：確認傳入復原

負載平衡器控制器會在下一次通過Ingress時（通常在幾分鐘內）重新協調。授予運作時，先前的secrets "oidc" is forbidden錯誤會停止出現在 上，Ingress而且負載平衡器會繼續。如果錯誤仍然存在，請重新檢查 Role's `namespace、 resourceNames和RoleBinding主體。

移除授予

> **Important**
>
> 只要 ALB Ingress使用 ，請將 Role和 保持在RoleBinding原位alb.ingress.kubernetes.io/auth-type: oidc。移除它們會重新中斷 上的 OIDC 解析度Ingress。

如果您在 上停止使用 OIDCIngress，請移除授予：

```
kubectl delete rolebinding eks-managed-oidc-secret-reader -n monitoring
kubectl delete role        eks-managed-oidc-secret-reader -n monitoring
```

相關資源

- the section called "將 RBAC 授予受管控制器" — eks:managed群組的概念概觀，以及如何限制授予的範圍。

- the section called “存取項目” — Amazon EKS 存取項目。
- Kubernetes 文件中的使用 RBAC 授權。

建立儲存類別

在 Amazon EKS 自動模式中，StorageClass 定義了當應用程式請求持久性儲存時，如何自動佈建 Amazon EBS 磁碟區。本頁面闡釋了如何建立和設定與 Amazon EKS 自動模式協同工作的 StorageClass，以佈建 EBS 磁碟區。

透過設定 StorageClass，您可為 EBS 磁碟區指定預設設定，包括磁碟區類型、加密、IOPS 和其他儲存參數。您也可以StorageClass將 設定為使用 AWS KMS 金鑰進行加密管理。

EKS 自動模式不會為您建立 StorageClass。您必須建立 StorageClass 參考 ebs.csi.eks.amazonaws.com，以使用 EKS 自動模式的儲存功能。

首先，建立名為 storage-class.yaml 的檔案：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: auto-ebs-sc
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
allowedTopologies:
- matchLabelExpressions:
  - key: eks.amazonaws.com/compute-type
    values:
    - auto
provisioner: ebs.csi.eks.amazonaws.com
volumeBindingMode: WaitForFirstConsumer
parameters:
  type: gp3
  encrypted: "true"
```

接著，將儲存體方案套用至您的叢集。

```
kubectl apply -f storage-class.yaml
```

關鍵元件：

- `provisioner: ebs.csi.eks.amazonaws.com` - 使用 EKS 自動模式
- `allowedTopologies` - 指定 `eks.amazonaws.com/compute-type:auto` 來匹配 `matchLabelExpressions`，將確保如果您的 Pod 需要使用自動模式自動佈建磁碟區，則這些 Pod 不會被排程到非自動節點上。
- `volumeBindingMode: WaitForFirstConsumer` - 延遲建立磁碟區，直至 Pod 需要磁碟區
- `type: gp3` - 指定 EBS 磁碟區類型
- `encrypted: "true"` - EBS 將加密使用此 `StorageClass` 建立的所有磁碟區。EBS 將使用預設的 `aws/ebs` 金鑰別名。如需詳細資訊，請參閱《Amazon EBS 使用者指南》中的 Amazon EBS 加密運作方式。此值可選用的，但建議設定。
- `storageclass.kubernetes.io/is-default-class: "true"` - 依預設，Kubernetes 將使用此儲存類別，除非您在持續性磁碟區宣告上指定了不同的磁碟區類別。此值是選用的。若要從其他儲存控制器移轉，則在設定該值時要謹慎對待。

使用自我管理的 KMS 金鑰加密 EBS 磁碟區

要使用自我管理的 KMS 金鑰來加密由 EKS 自動模式自動佈建的 EBS 磁碟區，您需要：

1. 建立自我管理的 KMS 金鑰。
   - 如需詳細資訊，請參閱《KMS 使用者指南》中的建立對稱加密 KMS 金鑰或 Amazon Elastic Block Store (Amazon EBS) 如何使用 KMS。
2. 建立一個允許存取 KMS 金鑰的新政策。
   - 使用下面的範例 IAM 政策來建立該政策。插入新的自我管理 KMS 金鑰的 ARN。如需詳細資訊，請參閱《IAM 使用者指南》中的建立角色和連接政策（主控台）。 AWS
3. 將政策附加到 EKS 叢集角色。
   - 使用 AWS 主控台尋找 EKS 叢集角色的 ARN。角色資訊在概覽區段中可見。如需詳細資訊，請參閱the section called “叢集 IAM 角色”。
4. 更新 `StorageClass`，以參考 `parameters.kmsKeyId` 欄位的 KMS 金鑰 ID。

自我管理 KMS IAM 政策範例

更新下方政策中的以下值：

- `<account-id>` – AWS 您的帳戶 ID，例如 `111122223333`
- `<aws-region>` – 叢集 AWS 的區域，例如 `us-west-2`

```
{
  "Version":"2012-10-17",
  "Id": "key-auto-policy-3",
  "Statement": [
      {
          "Sid": "Enable IAM User Permissions",
          "Effect": "Allow",
          "Principal": {
              "AWS": "arn:aws:iam::123456789012:root"
          },
          "Action": "kms:*",
          "Resource": "*"
      },
      {
        "Sid": "Allow access through EBS for all principals in the account that are
 authorized to use EBS",
        "Effect": "Allow",
        "Principal": {
            "AWS": "*"
        },
        "Action": [
            "kms:Encrypt",
            "kms:Decrypt",
            "kms:ReEncrypt*",
            "kms:GenerateDataKey*",
            "kms:CreateGrant",
            "kms:DescribeKey"
        ],
        "Resource": "*",
        "Condition": {
            "StringEquals": {
                "kms:CallerAccount": "123456789012",
                "kms:ViaService": "ec2.us-east-1.amazonaws.com"
            }
        }
    }
  ]
}
```

自我管理的 KMS **StorageClass** 範例

```
parameters:
  type: gp3
```

```
  encrypted: "true"
  kmsKeyId: <custom-key-arn>
```

StorageClass 參數參考

有關 Kubernetes StorageClass 資源的一般資訊，請參閱 Kubernetes 文件中的儲存類別。

StorageClass 資源的 parameters區段是特定的 AWS。請使用下列資料表檢閱可用的選項。

Parameters	值	預設	Description
"csi.storage.k8s.io/fstype"	xfs、ext2、ext3、ext4	ext4	在磁碟區建立期間將被格式化的檔案系統類型。此參數有大小寫之分！
"type"	io1、io2、gp2、gp3、sc1、st1、Standard、sbp1、sbg1	gp3	EBS 磁碟區類型。
"iopsPerGB"			每 GiB 的每秒 I/O 操作次數。可為 IO1、IO2 和 GP3 磁碟區指定。
"allowAutoIOPSPerGBIncrease"	true、false	false	當 時"true"，當 iopsPerGB * <volume size> 太低而無法符合 支援的 IOPS 範圍時，CSI 驅動程式會增加磁碟區的 IOPS AWS。這可讓動態佈建總是成功，即使使用者指定的 PVC 容量或 iopsPerGB 值太小。另一方面，這可能會帶來額外成本，

Parameters	值	預設	Description
			因為此類磁碟區的 IOPS 比 `iopsPerGB` 中請求的要高。
"iops"			每秒 I/O 操作。可為 IO1、IO2 和 GP3 磁碟區指定。
"throughput"		125	輸送量以 MiB/秒為單位。僅在指定了 gp3 磁碟區類型時有效。
"encrypted"	true、false	false	指示是否加密磁碟區。有效值為 "true" 或 "false"。
"blockExpress"	true、false	false	啟用 io2 Block Express 磁碟區的建立。
"kmsKeyId"			加密磁碟區時要使用的金鑰的完整 ARN。如果未指定，AWS 將針對磁碟區所在的區域使用預設 KMS 金鑰。如果未變更，這將是一個自動產生的金鑰，稱為 `/aws/ebs`。
"blockSize"			格式化底層檔案系統時使用的區塊大小。僅在 Linux 節點上並使用 `ext2`、`ext3`、`ext4` 或 `xfs` 時支援。

Parameters	值	預設	Description
"inodeSize"			格式化底層檔案系統時使用的 inode 大小。僅在 Linux 節點上並使用 ext2、ext3、ext4 或 xfs 時支援。
"bytesPerInode"			格式化底層檔案系統時使用的 bytes-per-inode 。僅在 Linux 節點上並使用 ext2、ext3、ext4 時支援。
"numberOfInodes"			格式化底層檔案系統時使用的 number-of-inodes 。僅在 Linux 節點上並使用 ext2、ext3、ext4 時支援。
"ext4BigAlloc"	true、false	false	透過啟用 bigalloc 格式化選項，將 ext4 檔案系統變更為使用叢集區塊配置。警告：您的節點 Linux 核心可能不完全支援 bigalloc。
"ext4ClusterSize"			當啟用 bigalloc 功能時，格式化 ext4 檔案系統所使用的叢集大小。請注意：ext4BigAlloc 參數必須設為 true。

如需詳細資訊，請參閱 GitHub 上的 AWS EBS CSI 驅動程式。

考量事項

> **Note**
>
> 您只能在 EKS 自動模式節點上，部署依賴於 EKS 自動模式 StorageClass 的工作負載。如果您有一個混合類型節點的叢集，需要設定您的工作負載僅在 EKS 自動模式節點上執行。如需詳細資訊，請參閱the section called “控制部署”。

EKS 自動模式的區塊儲存功能與 EBS CSI 驅動程式不同。

- 靜態佈建
  - 如果您想要搭配 EKS Auto 模式使用外部建立的 EBS 磁碟區，則需要手動新增具有 金鑰`eks:eks-cluster-name`和叢集名稱值的 AWS 標籤。
- 節點啟動污點
  - 您無法使用節點啟動污點功能，以在儲存功能準備就緒之前防止 Pod 排程
- 動態佈建磁碟區上的自訂標籤
  - 您無法使用 extra-tag CLI 標誌來設定動態佈建 EBS 磁碟區上的自訂標籤
  - 您可使用 `StorageClass` 標記來新增自訂標籤。EKS Auto Mode 會將標籤新增至相關聯的 AWS 資源。您需要更新叢集 IAM 角色以用於自訂標籤。如需詳細資訊，請參閱the section called “EKS Auto 資源的自訂 AWS 標籤”。
- EBS 詳細效能指標
  - 您無法存取 EBS 詳細效能的 Prometheus 指標

安裝 CSI 快照控制器附加元件

EKS 自動模式與 CSI 快照控制器 Amazon EKS 附加元件相容。

AWS 建議您將此附加元件設定為在內建`system`節點集區上執行。

如需詳細資訊，請參閱：

- the section called “執行關鍵附加元件”
- the section called “檢閱內建的節點集區”

- the section called “CSI 快照器控制器”

在系統節點集區中安裝快照控制器

1. 在 AWS 主控台中開啟您的 EKS 叢集
2. 從附加元件索引標籤中，選取取得更多附加元件
3. 選取 CSI 快照控制器，然後選擇下一步
4. 在設定選取的附加元件設定頁面上，選取選用組態設定，以檢視附加元件組態結構描述
   a. 插入以下 yaml 以將快照控制器與 `system` 節點集區關聯。快照控制器包含對 `CriticalAddonsOnly` 污點的容錯。

```
{
        "nodeSelector": {
            "karpenter.sh/nodepool": "system"
        }
}
```

   b. 選取下一步
5. 檢閱附加元件組態，然後選取建立

停用 EKS 自動模式

您可以在現有的 EKS 叢集上停用 EKS 自動模式。這是破壞性的操作。

- EKS 將終止由 EKS 自動模式操作的所有 EC2 執行個體。
- EKS 將刪除由 EKS 自動模式操作的所有負載平衡器。
- EKS 不會刪除由 EKS 自動模式佈建的 EBS 磁碟區。

EKS 自動模式旨在完整管理其所建立的資源。手動介入可能導致 EKS 自動模式在停用時無法完全清除這些資源。例如，如果您從外部安全群組規則中參照了受管安全群組，並且在為叢集停用 EKS Auto Mode 之前忘記移除該參照，則受管安全群組將洩漏 (不會被刪除)。以下步驟說明若發生此情形，如何移除遺漏的安全群組。

停用 EKS 自動模式AWS （主控台）

1. 在 AWS 管理主控台中開啟您的叢集概觀頁面。

2. 在 EKS 自動模式下，選取管理
3. 將 EKS 自動模式切換為 off。

如果此程序結束時未刪除任何受管安全群組，您可以使用刪除安全群組中的指示手動將其刪除。

停用 EKS 自動模式 (AWS CLI)

使用以下命令在現有叢集上停用 EKS 自動模式。

您需要安裝 aws CLI，並使用足夠的許可來登入以管理 EKS 叢集。如需詳細資訊，請參閱設定。

> **Note**
> 運算、區塊儲存和負載平衡功能必須在同一請求中全部啟用或停用。

```
aws eks update-cluster-config \
 --name $CLUSTER_NAME \
 --compute-config enabled=false \
 --kubernetes-network-config '{"elasticLoadBalancing":{"enabled": false}}' \
 --storage-config '{"blockStorage":{"enabled": false}}'
```

您可按照以下方式檢查在停用 EKS 自動模式後，是否有遺漏的 EKS 自動模式安全群組未能被刪除：

```
aws ec2 describe-security-groups \
    --filters Name=tag:eks:eks-cluster-name,Values=<cluster-Name> Name=tag-
key,Values=ingress.eks.amazonaws.com/resource,service.eks.amazonaws.com/resource --
query "SecurityGroups[*].[GroupName]"
```

然後刪除安全群組：

```
aws ec2 delete-security-group --group-name=<sg-name>
```

更新 EKS 自動模式叢集的 Kubernetes 版本

本主題闡釋如何更新自動模式叢集的 Kubernetes 版本。自動模式透過處理控制平面更新與節點取代的協調來簡化版本更新程序，同時透過 Pod 中斷預算維持工作負載可用性。

升級自動模式叢集時，許多傳統上需要手動更新的元件現在作為服務的一部分進行管理。了解升級程序的自動化層面與您的責任，有助於確保叢集的版本順利轉換。

了解 EKS 自動模式的更新

在您啟動控制平面升級後，EKS 自動模式會升級叢集中的節點。當節點到期時，EKS 自動模式會用新節點取代。新節點具有相應的新 Kubernetes 版本。升級節點時，EKS 自動模式會遵守 Pod 中斷預算。

此外，您不再需要更新下列元件：

- Amazon VPC CNI
- AWS Load Balancer 控制器
- CoreDNS
- `kube-proxy`
- Karpenter
- AWS EBS CSI 驅動程式

EKS 自動模式會以服務功能取代這些元件。

您仍然負責更新：

- 部署到您的叢集的應用程式與工作負載
- 自我管理的附加元件與控制器
- Amazon EKS 附加元件
  - 了解如何 the section called “更新附加元件”

了解叢集升級的最佳實務

啟動叢集更新

要開始叢集更新，請參閱 the section called “更新 Kubernetes 版本”。

啟用或停用內建的 NodePool

EKS 自動模式具有兩個內建的 NodePool。您可以使用 AWS 主控台、CLI 或 API 啟用或停用這些 NodePools。

內建的 NodePool 參考

- `system`
  - 此 NodePool 具有 `CriticalAddonsOnly` 污點。CoreDNS 等眾多 EKS 附加元件皆可容忍此污點。藉助此系統節點集區，來分隔叢集關鍵型應用程式。
  - 支援 `amd64` 與 `arm64` 架構。
- `general-purpose`
  - 針對叢集中的一般用途工作負載，此 NodePool 提供啟動節點相關支援。
  - 只能使用 `amd64` 架構。

兩個內建的 NodePool：

- 使用預設的 EKS NodeClass
- 只能使用隨需 EC2 容量
- 使用 C、M 及 R EC2 執行個體系列
- 要求使用第 5 代或更新版本的 EC2 執行個體

> **Note**
>
> EKS 需要啟用至少一個內建的 NodePool，才能佈建「預設」NodeClass。若停用全部內建的 NodePool，需要建立自訂 NodeClass 及設定 NodePool 才能使用。若要了解 NodeClass 的相關資訊，請參閱 the section called “建立節點類別”。

> **Important**
>
> 當您從 移除內建 NodePool 名稱時`computeConfig.nodePools`，會從叢集中刪除對應的 NodePool Kubernetes 資源。由該 NodePool 管理的任何節點都會耗盡並終止。當您將內建 NodePool 名稱新增回 時`computeConfig.nodePools`，會在叢集上重新建立對應的 NodePool 資源。此行為在規劃容量變更時很重要，因為 Karpenter 透過 Kubernetes API 探索 NodePool 資源，並停止排程已刪除 NodePools 的節點。

程序

先決條件

- 在您裝置上安裝和設定的最新版本 AWS 命令列界面 (AWS CLI)。若要檢查您目前的版本，請使用 `aws --version`。若要安裝最新版本，請參閱《 AWS 命令列界面使用者指南》中的使用 aws 設定安裝 和 快速組態。
  - 使用足夠的 IAM 許可登入 CLI，以建立 AWS 資源，包括 IAM 政策、IAM 角色和 EKS 叢集。

使用 CLI AWS 啟用

使用以下命令，啟用兩個內建的 NodePool：

```
aws eks update-cluster-config \
  --name <cluster-name> \
  --compute-config '{
    "nodeRoleArn": "<node-role-arn>",
    "nodePools": ["general-purpose", "system"],
    "enabled": true
  }' \
  --kubernetes-network-config '{
  "elasticLoadBalancing":{"enabled": true}
  }' \
  --storage-config '{
  "blockStorage":{"enabled": true}
  }'
```

您可修改命令，選擇性地啟用 NodePools。

使用 AWS CLI 停用

使用以下命令，停用兩個內建的 NodePool：

```
aws eks update-cluster-config \
  --name <cluster-name> \
  --compute-config '{
  "enabled": true,
  "nodePools": []
  }' \
  --kubernetes-network-config '{
  "elasticLoadBalancing":{"enabled": true}}' \
```

```
  --storage-config '{
  "blockStorage":{"enabled": true}
  }'
```

控制工作負載是否部署在 EKS 自動模式節點上

在包含 EKS 自動模式的 EKS 叢集中執行工作負載時，您可能需要控制特定工作負載是否應在 EKS 自動模式節點，還是其他運算類型上執行。本主題描述如何使用節點選擇器與親和性，以確保您的工作負載被排程到預期的運算基礎結構上。

本主題中的範例示範如何使用 `eks.amazonaws.com/compute-type` 標籤，以要求或阻止工作負載部署到 EKS 自動模式節點上。這在混合模式叢集中特別有用，例如您同時執行了 EKS 自動模式和其他運算類型 (例如自我管理的 Karpenter 佈建程式或 EKS 受管節點群組) 的環境。

EKS 自動模式節點已將標籤 `eks.amazonaws.com/compute-type` 的值設定為 `auto`。您可使用此標籤來控制工作負載是否要部署到由 EKS 自動模式管理的節點。

需要將工作負載部署到 EKS Auto Mode 節點

> **Note**
>
> 此 `nodeSelector` 值對於 EKS 自動模式並非必要。此 `nodeSelector` 僅在您於混合模式中執行叢集時 (即叢集中包含非 EKS 自動模式管理的節點類型時) 才相關。例如，您可能已透過 EKS 受管節點群組將靜態運算容量部署到叢集中，同時也擁有由 EKS 自動模式管理的動態運算容量。

您可將此 `nodeSelector` 新增至部署或其他工作負載中，以要求 Kubernetes 將其排程到 EKS 自動模式節點上。

```
apiVersion: apps/v1
kind: Deployment
spec:
  template:
    spec:
      nodeSelector:
        eks.amazonaws.com/compute-type: auto
```

需要工作負載未部署到 EKS Auto Mode 節點

您可將此 nodeAffinity 新增至部署或其他工作負載中，以要求 Kubernetes 不要將其排程到 EKS 自動模式節點上。

```
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: eks.amazonaws.com/compute-type
            operator: NotIn
            values:
            - auto
```

以特定 NodePool 為目標

Amazon EKS Auto Mode 叢集可以有多個 NodePool。您可能希望工作負載只在來自特定 NodePool 的節點上執行。為了達成此目的，工作負載可以比對下列任一標籤：

- 您在 NodePool 上定義的自訂標籤。當您在 NodePool spec.template.metadata.labels中於下新增標籤時，Amazon EKS Auto Mode 會將這些標籤套用至 NodePool 佈建的每個節點。建議採用自訂標籤。自訂標籤會將工作負載連結至功能，例如 workload-class: gpu-inference，而不是特定 NodePool 名稱。
- 眾所周知的**karpenter.sh/nodepool**標籤。Amazon EKS Auto Mode 會使用 NodePool 的名稱做為值，將此標籤套用至其佈建的每個節點。當您尚未定義自訂標籤，並想要依名稱以 NodePool 為目標時，請使用此標籤。

將自訂 NodePool 標籤設為目標

首先，在 NodePool 上定義標籤。

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: gpu
spec:
  template:
    metadata:
      labels:
```

```
        workload-class: gpu-inference
```

如需 NodePool 組態的詳細資訊，請參閱 the section called “建立節點集區”。

接著，比對工作負載中的標籤。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: inference
spec:
  template:
    spec:
      nodeSelector:
        workload-class: gpu-inference
```

此部署排程中的 Pod 只會傳送至 gpu NodePool 佈建的節點。沒有此項目的工作負載會nodeSelector繼續在預設 NodePool 上執行。

依名稱將 NodePool 設為目標

> **NodePool 名稱變更會影響排程**
>
> 如果您重新命名或刪除目標 NodePool，依名稱固定工作負載可能會讓 Pod 處於 Pending 狀態。針對您希望超過單一 NodePool 定義的任何工作負載，使用自訂功能標籤。

```
apiVersion: apps/v1
kind: Deployment
spec:
  template:
    spec:
      nodeSelector:
        karpenter.sh/nodepool: <your-nodepool-name>
```

如需 Karpenter 如何將 Pod 與 NodePools 配對的詳細資訊，請參閱 Karpenter 文件中的排程。

在專用執行個體上執行關鍵附加元件

在本主題中，您將了解如何部署具有 CriticalAddonsOnly 容差的工作負載，以便 EKS 自動模式將其排程到 system 節點集區上。

EKS 自動模式的內建 system 節點集區專為在專用執行個體上執行關鍵附加元件而設計。這種隔離確保了關鍵元件擁有專用資源，並與一般工作負載隔離開，從而提升了叢集的整體穩定性與效能。

本指南示範如何透過使用 CriticalAddonsOnly 容差和適當的節點選擇器，將附加元件部署到 system 節點集區。遵循這些步驟，您可以確保您的關鍵應用程式被排程到專用的 system 節點上，充分利用 EKS 自動模式的專用節點集區結構所帶來的隔離與資源分配優勢。

EKS 自動模式有兩個內建節點集區：general-purpose 和 system。如需詳細資訊，請參閱 the section called “檢閱內建的節點集區”。

system 節點集區的用途是將關鍵附加元件隔離到不同的節點上。節點集區佈建的 system 節點帶有 CriticalAddonsOnly Kubernetes 污點。Kubernetes 只會將具有相應容差的 Pod 排程到這些節點上。如需詳細資訊，請參閱 Kubernetes 文件中的污點和容差。

先決條件

- 已啟用內建 system 節點集區的 EKS 自動模式叢集。如需詳細資訊，請參閱the section called “檢閱內建的節點集區”
- kubectl 已安裝並已設定。如需詳細資訊，請參閱 設定。

程序

請檢閱以下的範例 yaml。記下下列組態：

- nodeSelector：這會將工作負載與內建的 system 節點集區建立關聯。此節點集區必須透過 AWS API 啟用。如需詳細資訊，請參閱 the section called “檢閱內建的節點集區”。
- tolerations：此容差用於克服 system 節點集區中節點上的 CriticalAddonsOnly 污點。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sample-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: sample-app
  template:
    metadata:
      labels:
```

```
        app: sample-app
    spec:
      nodeSelector:
        karpenter.sh/nodepool: system
      tolerations:
      - key: "CriticalAddonsOnly"
        operator: "Exists"
      containers:
      - name: app
        image: nginx:latest
        resources:
          requests:
            cpu: "500m"
            memory: "512Mi"
```

要更新工作負載使其在 system 節點集區上執行，您需要：

1. 更新現有工作負載，以新增上述的下列組態：
   - nodeSelector
   - tolerations
2. 使用 kubectl apply 將更新後的工作負載部署到您的叢集

更新工作負載之後，它將在專用節點上執行。

在 EKS 自動模式中使用網路政策

概觀

隨著客戶使用 EKS 擴展其應用程式環境，網路流量隔離變得越來越基本，以防止未經授權存取叢集內外的資源。這在多租用戶環境中特別重要，其中多個不相關的工作負載在叢集中並排執行。Kubernetes 網路政策可讓您增強 Kubernetes 工作負載的網路安全狀態，以及其與叢集外部端點的整合。EKS Auto Mode 支援不同類型的網路政策。

第 3 層和第 4 層隔離

標準 Kubernetes 網路政策會在 OSI 網路模型的第 3 層和第 4 層運作，可讓您控制 Amazon EKS 叢集內 IP 地址或連接埠層級的流量流程。

使用案例

- 分割工作負載之間的網路流量，以確保只有相關的應用程式可以互相通訊。

- 使用 政策在命名空間層級隔離租用戶，以強制執行網路分離。

DNS 型強制執行

客戶通常會在 EKS 中部署屬於更廣泛分散式環境的工作負載，其中一些工作負載必須與叢集外部的系統和服務（北向流量）通訊。這些系統和服務可以 AWS 位於 AWS 雲端或外部。以網域名稱系統 (DNS) 為基礎的政策可讓您採用更穩定且可預測的方法來加強安全狀態，以防止未經授權的 Pod 存取叢集外部資源或端點。此機制不需要手動追蹤和允許列出特定 IP 地址。透過以 DNS 為基礎的方法保護資源，您還可以更靈活地更新外部基礎設施，而無需在上游伺服器和主機變更時放寬安全狀態或修改網路政策。您可以使用完整網域名稱 (FQDN) 或 DNS 網域名稱的相符模式，篩選外部端點的輸出流量。這可讓您靈活地擴展對與特定叢集外部端點相關聯的多個子網域的存取。

使用案例

- 標準化以 DNS 為基礎的方法，以篩選從 Kubernetes 環境到叢集外部端點的存取。
- 安全存取多租戶環境中 AWS 的服務。
- 管理混合雲端環境中從 Pod 到內部部署工作負載的網路存取。

管理員（或叢集範圍）規則

在某些情況下，如同多租戶案例，客戶可能需要強制執行適用於整個叢集的網路安全標準。您可以使用單一政策來集中管理叢集中不同工作負載的網路存取控制，而不重複定義和維護每個命名空間的不同政策，無論其命名空間為何。這些類型的政策可讓您擴展在第 3 層、第 4 層和使用 DNS 規則時套用之網路篩選規則的強制執行範圍。

使用案例

- 集中管理 EKS 叢集中所有（或一部分）工作負載的網路存取控制。
- 定義整個叢集的預設網路安全狀態。
- 以更具營運效率的方式，將組織安全標準擴展到叢集的範圍。

開始使用

先決條件

- 已啟用 EKS 自動模式的 Amazon EKS 叢集
- kubectl 已設定為連接到您的叢集

步驟 1：啟用網路政策控制器

要在 EKS 自動模式中使用網路政策，您首先需要透過將 ConfigMap 套用到您的叢集，以啟用網路政策控制器。

1. 建立名為 `enable-network-policy.yaml` 且具有下列內容的檔案：

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: amazon-vpc-cni
  namespace: kube-system
data:
  enable-network-policy-controller: "true"
```

2. 將 ConfigMap 套用至您的叢集：

```
kubectl apply -f enable-network-policy.yaml
```

步驟 2：建立和測試網路政策

您的 EKS 自動模式叢集現已設定為支援 Kubernetes 網路政策。您可利用 the section called “星政策示範” 來測試此功能。

步驟 3：調整節點類別中的網路政策代理程式組態 （選用）

您可以選擇性地建立新的節點類別，以變更節點上網路政策代理程式的預設行為，或啟用網路政策事件的記錄。若要這麼做，請依照下列步驟進行：

1. 建立或編輯節點類別 YAML 檔案 (例如 `nodeclass-network-policy.yaml`)，內容如下：

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: network-policy-config
spec:
  # Optional: Changes default network policy behavior
  networkPolicy: DefaultAllow
  # Optional: Enables logging for network policy events
  networkPolicyEventLogs: Enabled
  # Include other Node Class configurations as needed
```

2. 將節點類別組態套用至您的叢集：

```
kubectl apply -f nodeclass-network-policy.yaml
```

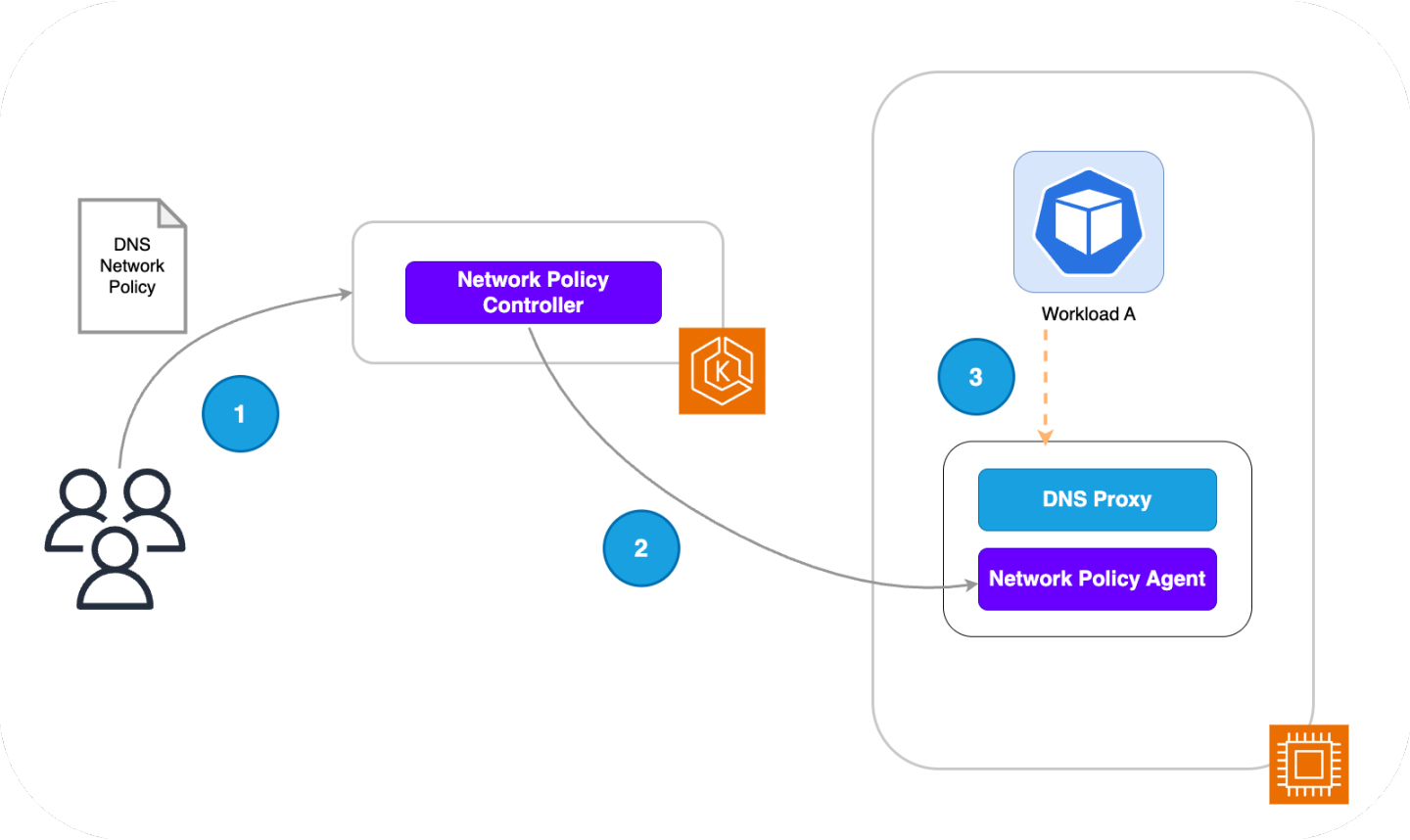
3. 確認節點類別是否已建立：

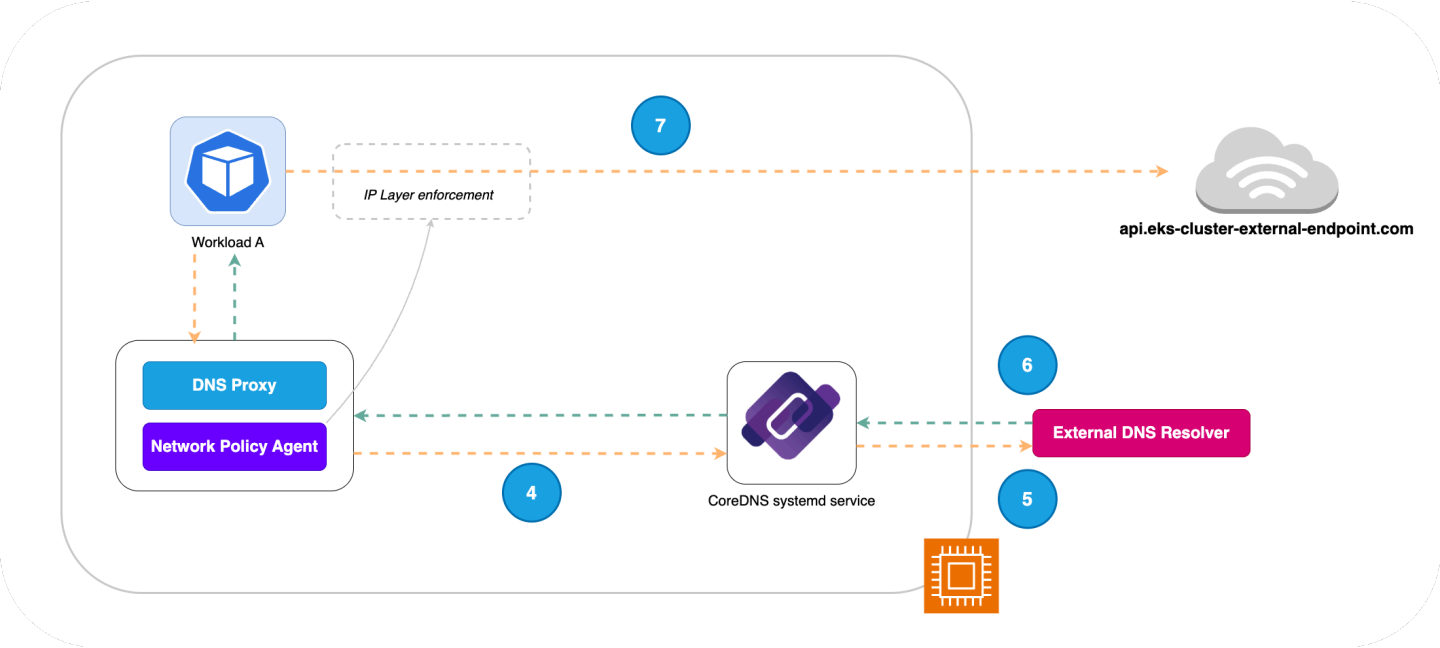
```
kubectl get nodeclass network-policy-config
```

4. 更新您的節點集區，使其使用此節點類別。如需詳細資訊，請參閱the section called “建立節點集區”。

其運作方式？

DNS 型網路政策





1. 平台團隊會將 DNS 型政策套用至 EKS 叢集。
2. 網路政策控制器負責監控叢集內政策的建立，然後協調政策端點。在此使用案例中，網路政策控制器會指示節點代理程式，根據所建立政策中允許列出的網域來篩選 DNS 請求。允許使用 FQDN 或符合 Kubernetes 資源組態中定義模式的網域名稱來列出網域名稱。
3. 工作負載 嘗試解析叢集外部端點的 IP。DNS 請求會先經過代理，根據透過網路政策套用的允許清單來篩選這類請求。
4. DNS 請求通過 DNS 篩選條件允許清單後，代理會將其轉送至 CoreDNS、
5. CoreDNS 會接著將請求傳送至外部 DNS 解析程式 (Amazon Route 53 Resolver)，以取得網域名稱後方的 IP 地址清單。
6. 回應 DNS 請求時，會傳回具有 TTL 的已解析 IPs。然後，這些 IPs會寫入 eBPF 映射中，用於執行 IP 層的下一個步驟。
7. 接著，連接至 Pod 實物界面的 eBPF 探查會根據適當的規則，篩選從工作負載 A 到叢集外部端點的輸出流量。這可確保 Pod 只能將叢集外部流量傳送至允許所列網域IPs。這些 IPs 的有效性是以從外部 DNS 解析程式 (Amazon Route 53 Resolver) 擷取的 TTL 為基礎。

使用應用程式網路政策

`ApplicationNetworkPolicy` 結合了標準 Kubernetes 網路政策的功能，以及使用單一自訂資源定義 (CRD) 在命名空間層級進行以 DNS 為基礎的篩選。因此， `ApplicationNetworkPolicy`可用於：

1. 使用 IP 區塊和連接埠號碼，定義網路堆疊第 3 層和第 4 層的限制。
2. 定義在網路堆疊第 7 層操作的規則，並讓您根據 FQDNs篩選流量。

> **Important**
>
> 使用 定義的 DNS 型規則ApplicationNetworkPolicy僅適用於在 EKS Auto Mode 啟動的 EC2 執行個體中執行的工作負載。 ApplicationNetworkPolicy支援標準 Kubernetes 的所有欄位NetworkPolicy，以及用於輸出規則的額外 FQDN 篩選條件。

> **Warning**
>
> 請勿將相同的名稱用於相同命名空間NetworkPolicy內的ApplicationNetworkPolicy和 。如果名稱衝突，產生的PolicyEndpoints物件可能無法正確反映任一政策。接受這兩個資源不會發生錯誤，使得此問題難以診斷。
> 若要解決命名衝突，請重新命名 ApplicationNetworkPolicy或 ，NetworkPolicy使其在命名空間中是唯一的，然後驗證對應的PolicyEndpoints物件是否已正確更新。

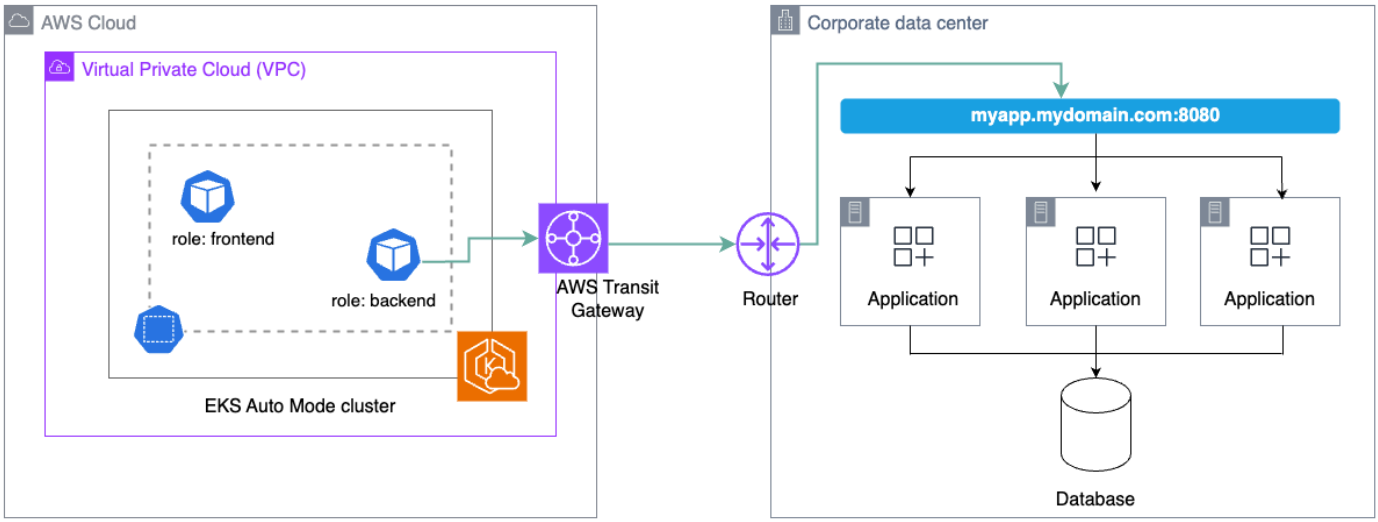
範例

EKS Auto Mode 叢集中有工作負載，需要與具有 DNS 名稱的負載平衡器後方的應用程式內部部署通訊。您可以使用下列網路政策來達成此目的：

```
apiVersion: networking.k8s.aws/v1alpha1
kind: ApplicationNetworkPolicy
metadata:
  name: my-onprem-app-egress
  namespace: galaxy
spec:
  podSelector:
    matchLabels:
      role: backend
  policyTypes:
  - Egress
  egress:
  - to:
    - ipBlock: { cidr: 10.100.0.10/32 }
    ports:
```

```
    - protocol: TCP
      port: 53
    - protocol: UDP
      port: 53
  - to:
    - domainNames:
      - "myapp.mydomain.com"
    ports:
    - protocol: TCP
      port: 8080
```

在 Kubernetes 網路層級，這將允許從標記為 的「galaxy」命名空間中的任何 Pod 輸出`role: backend`，以連接到 TCP 連接埠 8080 上的網域名稱 myapp.mydomain.com 和連接埠 53 上的 CoreDNS。此外，您需要為從 VPC 到公司資料中心的輸出流量設定網路連線。



判斷 CoreDNS IP 地址

若要讓您的應用程式解析 DNS，必須允許其與 CoreDNS 通訊，該 CoreDNS 會在每個 EKS Auto Mode 執行個體上於本機執行。CoreDNS IP 地址衍生自為叢集設定的 Service CIDR 範圍，並且在叢集的生命週期內保持固定。因此，您可以判斷一次，並直接在網路政策中參考它。

1. 擷取叢集的服務 CIDR。

   對於 IPv4 叢集：

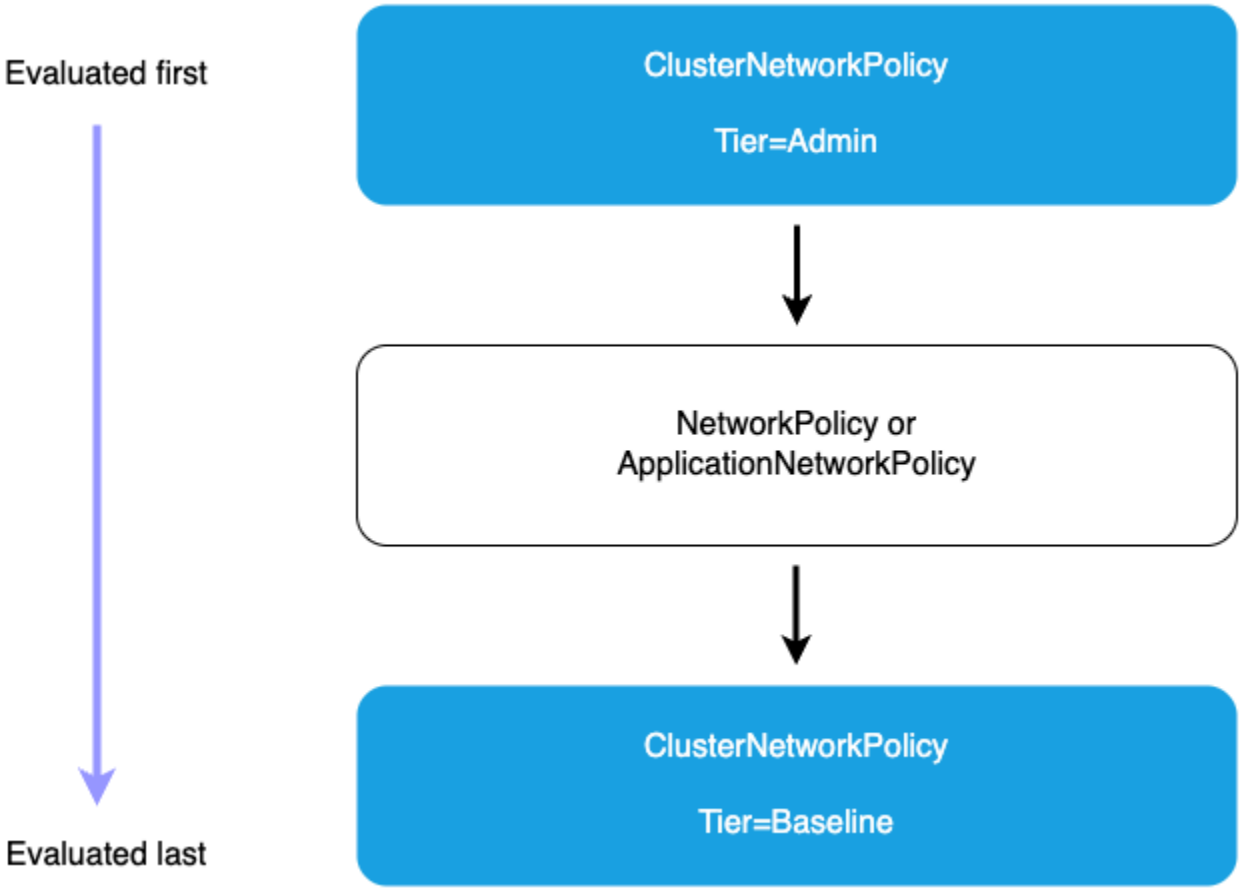
```
aws eks describe-cluster --name my-cluster --query
 'cluster.kubernetesNetworkConfig.serviceIpv4Cidr' --output text
```

對於 IPv6 叢集：

```
aws eks describe-cluster --name my-cluster --query
 'cluster.kubernetesNetworkConfig.serviceIpv6Cidr' --output text
```

2. 從服務 CIDR 衍生 CoreDNS IP 地址：
   - IPv4 — 使用 CIDR .10的地址。例如，10.100.0.0/16 會變成 10.100.0.10/32。
   - IPv6 — 附加a至網路地址。例如，fd12:3456:789a::/108 會變成 fd12:3456:789a::a/128。

管理員（或叢集）網路政策



使用叢集網路政策

使用 時ClusterNetworkPolicy，會先評估管理員層政策，且無法覆寫。評估管理員層政策後，會使用標準命名空間範圍政策來執行套用的網路分割規則。這可以透過使用

ApplicationNetworkPolicy或 來完成NetworkPolicy。最後，將強制執行定義叢集工作負載預設網路限制的基準層規則。如有需要，命名空間範圍政策可以覆寫這些基準層規則。

範例

叢集中有您要隔離其他租用戶工作負載的應用程式。您可以明確封鎖來自其他命名空間的叢集流量，以防止網路存取敏感工作負載命名空間。

```
apiVersion: networking.k8s.aws/v1alpha1
kind: ClusterNetworkPolicy
metadata:
  name: protect-sensitive-workload
spec:
  tier: Admin
  priority: 10
  subject:
    namespaces:
      matchLabels:
        kubernetes.io/metadata.name: earth
  ingress:
    - action: Deny
      from:
      - namespaces:
          matchLabels: {} # Match all namespaces.
      name: select-all-deny-all
```

考量事項

了解政策評估順序

EKS 中支援的網路政策功能會以特定順序評估，以確保可預測且安全的流量管理。因此，請務必了解評估流程，為您的環境設計有效的網路安全狀態。

1. 管理員層政策 （先評估）：所有管理員層 ClusterNetworkPolicies 都會在任何其他政策之前進行評估。在管理員層中，政策會依優先順序處理 （優先順序最低的數字優先）。動作類型會決定接下來會發生的情況。
   - 拒絕動作 （最高優先順序）：當具有拒絕動作的管理員政策符合流量時，無論任何其他政策為何，都會立即封鎖該流量。不會再處理 ClusterNetworkPolicy 或 NetworkPolicy 規則。這可確保命名空間層級政策不會覆寫整個組織的安全性控制項。

- 允許動作：評估拒絕規則之後，具有允許動作的管理員政策會依優先順序處理（優先順序最低的數字優先）。當允許動作相符時，系統會接受流量，而不會進一步評估政策。這些政策可以根據標籤選擇器授予多個命名空間的存取權，從而集中控制哪些工作負載可以存取特定資源。
- 傳遞動作：在管理層政策中傳遞動作，將決策委派給較低的層。當流量符合通過規則時，評估會略過該流量的所有剩餘管理員層規則，並直接進入 NetworkPolicy 層。這可讓管理員將特定流量模式的控制明確委派給應用程式團隊。例如，您可以使用 Pass 規則將命名空間內流量管理委派給命名空間管理員，同時對外部存取維持嚴格的控制。

2. 網路政策層：如果沒有管理員層政策與拒絕或允許相符，或者傳遞動作相符，則會接著評估命名空間範圍的 ApplicationNetworkPolicy 和傳統 NetworkPolicy 資源。這些政策可在個別命名空間內提供精細的控制，並由應用程式團隊管理。命名空間範圍政策只能比管理員政策更嚴格。他們無法覆寫管理員政策的拒絕決策，但可以進一步限制管理員政策允許或傳遞的流量。
3. 基準層管理員政策：如果沒有管理員或命名空間範圍的政策符合流量，則會評估基準層 ClusterNetworkPolicies。這些提供可由命名空間範圍政策覆寫的預設安全狀態，允許管理員設定整個組織的預設值，同時為團隊提供視需要自訂的彈性。基準政策會依優先順序（優先順序最低的數字優先）進行評估。
4. 預設拒絕（如果沒有政策相符）：此deny-by-default行為可確保只允許明確允許的連線，並維持強大的安全狀態。

套用最低權限原則

- 從限制性政策開始，並視需要逐步新增許可 - 首先在叢集層級實作deny-by-default政策，然後在驗證合法連線需求時逐步新增允許規則。這種方法會強制團隊明確地證明每個外部連線的合理性，建立更安全且可稽核的環境。
- 定期稽核並移除未使用的政策規則 - 網路政策會隨著應用程式發展而隨著時間累積，留下不必要地擴展攻擊面的過時規則。實作定期檢閱程序，以識別和移除不再需要的政策規則，確保您的安全狀態保持緊密且可維護。
- 盡可能使用特定的網域名稱，而不是廣泛的模式 - 雖然萬用字元模式`*.amazonaws.com`可提供便利性，但它們也會授予對各種服務的存取權。可行時，請指定 之類的確切網域名稱`s3.us-west-2.amazonaws.com`，以限制僅存取應用程式所需的特定服務，在工作負載洩露時降低橫向移動的風險。

在 EKS 中使用 DNS 型政策

- 使用 定義的 DNS 型規則`ApplicationNetworkPolicy`僅適用於在 EKS Auto Mode 啟動 EC2 執行個體中執行的工作負載。如果您執行混合模式叢集 （ 包含 EKS Auto 和非 EKS Auto 工作者節點 ），您的 DNS 型規則僅在 EKS Auto 模式工作者節點 (EC2 受管執行個體 ） 中有效。

驗證您的 DNS 政策

- 使用反映生產網路拓撲的預備叢集進行測試 - 您的預備環境應該複寫生產的網路架構、外部相依性和連線模式，以確保準確的政策測試。這包括相符的 VPC 組態、DNS 解析行為，以及對生產工作負載所需的相同外部服務的存取。
- 實作關鍵網路路徑的自動化測試 - 建置自動化測試，以驗證 CI/CD 管道中必要外部服務的連線能力。這些測試應驗證在封鎖未經授權的連線時是否允許合法流量流程，提供持續驗證，讓您的網路政策隨著基礎設施的演進而維持正確的安全狀態。
- 在政策變更後監控應用程式行為 - 在將新的或修改的網路政策部署至生產環境之後，請密切監控應用程式日誌、錯誤率和效能指標，以快速識別任何連線問題。建立明確的轉返程序，以便在政策變更造成非預期的應用程式行為或服務中斷時，快速還原政策變更。

與 Amazon Route 53 DNS 防火牆的互動

啟動流量時，EKS Admin 和 Network 政策會先在 Pod 層級進行評估。如果 EKS 網路政策允許輸出到特定網域，則 Pod 會執行到達 Route 53 Resolver 的 DNS 查詢。此時會評估 Route 53 DNS 防火牆規則。如果 DNS 防火牆封鎖網域查詢，DNS 解析會失敗，而且即使 EKS 網路政策允許，也無法建立連線。這會建立互補的安全層：EKS DNS 型網路政策為應用程式特定的存取要求和多租用戶安全界限提供 Pod 層級的輸出控制，而 DNS 防火牆針對已知惡意網域提供 VPC 整體保護，並強制執行整個組織的封鎖清單。

標記 EKS 自動模式的子網路

如果您使用 EKS Auto Mode 的負載平衡功能，則需要將 AWS 標籤新增至 VPC 子網路。

背景介紹

這些標籤可辨識與叢集關聯的子網路，且更重要的是辨識子網路為公有還是私有。

公有子網路可透過網際網路閘道直接存取網際網路。其適用於負載平衡器等需要公開存取的資源。

私有子網路不可直接存取網際網路，且針對傳出流量會使用 NAT 閘道。其適用於不需要公有 IP 的 EKS 節點等內部資源。

如需了解 NAT 閘道及網際網路閘道的相關詳細資訊，請參閱《Amazon Virtual Private Cloud (VPC) 使用者指南》中的連線 VPC 與其他網路。

需求

目前，EKS 自動模式用於負載平衡的子網路，需要具有以下其中一個標籤。

公有子網路

公有子網路適用於面向網際網路的負載平衡器。這些子網路必須具有以下標籤：

金鑰	值
kubernetes.io/role/elb	1 或 ``

私有子網路

私有子網路適用於內部負載平衡器。這些子網路必須具有以下標籤：

金鑰	值
kubernetes.io/role/internal-elb	1 或 ``

程序

開始操作之前，辨識哪些子網路為公有 (透過網際網路閘道存取) 及哪些為私有 (使用 NAT 閘道)。修改 VPC 資源需要許可。

AWS 管理主控台

1. 開啟 Amazon VPC 主控台並導覽至子網路。
2. 選取要標記的子網路。
3. 選擇標籤索引標籤，然後選取新增標籤。
4. 新增適當的標籤：
   - 針對公有子網路：Key=kubernetes.io/role/elb
   - 針對私有子網路：Key=kubernetes.io/role/internal-elb

5. 將值設定為 1或保持空白。
6. 儲存並重複其餘子網路。

AWS CLI

針對公有子網路：

```
aws ec2 create-tags \
    --resources subnet-ID \
    --tags Key=kubernetes.io/role/elb,Value=1
```

針對私有子網路：

```
aws ec2 create-tags \
    --resources subnet-ID \
    --tags Key=kubernetes.io/role/internal-elb,Value=1
```

使用實際的子網路 ID 取代 `subnet-ID`。

透過 kubectl debug 從 Kubernetes 節點產生 CIS 合規報告

本主題說明如何使用 `kubectl debug` 命令，為 Amazon EKS 節點產生 CIS (Center for Internet Security) 合規報告。此命令可讓您在 Kubernetes 節點上暫時建立偵錯容器，並透過 `apiclient` 工具執行 CIS 合規檢查。`apiclient` 工具是 Bottlerocket OS 的一部分，而 Bottlerocket OS 是 EKS 自動模式節點所使用的作業系統。

先決條件

開始前，請確保您具備以下條件：

- 可存取 Amazon EKS 叢集，且已設定 `kubectl` (版本必須至少為 v1.32.0；可輸入 `kubectl version` 進行檢查)。
- 具備適當的 IAM 許可以進行節點偵錯。
- 有效的設定檔，可允許執行偵錯作業 (如 `sysadmin`)。

如需有關搭配 `kubectl` 使用偵錯設定檔的詳細資訊，請參閱 Kubernetes 文件中的套用設定檔時對 Pod 或節點進行偵錯。

程序

1. 決定您要執行報告的節點 AWS 執行個體 ID。使用下列命令列出叢集中的節點。執行個體 ID 可在名稱欄位中找到，且開頭為 `i-`：

```
kubectl get nodes
```

```
NAME                  STATUS   ROLES    AGE   VERSION
i-0ea0ba0f8ef9ad609   Ready    <none>   62s   v1.30.10-eks-1a9dacd
```

2. 執行下列命令，並將 `<instance-id>` 取代為您要查詢的節點的執行個體 ID：

```
kubectl debug node/<instance-id> -it --profile=sysadmin --image=public.ecr.aws/
amazonlinux/amazonlinux:2023 -- bash -c "yum install -q -y util-linux-core; nsenter -
t 1 -m apiclient report cis --level 1 --format text"
```

   此命令的組成部分包括：

   - `kubectl debug node/<instance-id>`：在指定的 EC2 執行個體 ID 上建立偵錯工作階段。
   - `-it`：配置 TTY (命令列 Shell)，並保持 stdin 開啟以支援互動式使用。
   - `--profile=sysadmin`：使用具備適當許可的指定 `kubectl` 設定檔。
   - `--image=public.ecr.aws/amazonlinux/amazonlinux:2023`：使用 `amazonlinux:2023` 作為偵錯的容器映像。
   - `bash -c "…"` ：在 bash shell 中執行下列命令：
     - `yum install -q -y util-linux-core`：以安靜模式安裝所需的公用程式套件。
     - `nsenter -t 1 -m`：執行 `nsenter` 以輸入主機程序 (PID 1) 的命名空間。
     - `apiclient report cis --level 1 --format text`：執行等級 1 的 CIS 合規報告，並以文字格式輸出結果。

3. 檢閱報告的文字輸出。

解譯輸出

此命令會產生文字格式的報告，顯示各項 CIS 控制項的合規狀態。輸出內容包括：

- 個別的 CIS 控制項 ID
- 每個控制項的描述

- 每項檢查的通過、失敗或略過狀態
- 解釋任何合規問題的詳細資訊

以下是在 Bottlerocket 執行個體上執行報告後的輸出範例：

```
Benchmark name:  CIS Bottlerocket Benchmark
Version:         v1.0.0
Reference:       https://www.cisecurity.org/benchmark/bottlerocket
Benchmark level: 1
Start time:      2025-04-11T01:40:39.055623436Z

[SKIP] 1.2.1     Ensure software update repositories are configured (Manual)
[PASS] 1.3.1     Ensure dm-verity is configured (Automatic)[PASS] 1.4.1     Ensure
 setuid programs do not create core dumps (Automatic)
[PASS] 1.4.2     Ensure address space layout randomization (ASLR) is enabled
 (Automatic)
[PASS] 1.4.3     Ensure unprivileged eBPF is disabled (Automatic)
[PASS] 1.5.1     Ensure SELinux is configured (Automatic)
[SKIP] 1.6       Ensure updates, patches, and additional security software are
 installed (Manual)
[PASS] 2.1.1.1   Ensure chrony is configured (Automatic)
[PASS] 3.2.5     Ensure broadcast ICMP requests are ignored (Automatic)
[PASS] 3.2.6     Ensure bogus ICMP responses are ignored (Automatic)
[PASS] 3.2.7     Ensure TCP SYN Cookies is enabled (Automatic)
[SKIP] 3.4.1.3   Ensure IPv4 outbound and established connections are configured
 (Manual)
[SKIP] 3.4.2.3   Ensure IPv6 outbound and established connections are configured
 (Manual)
[PASS] 4.1.1.1   Ensure journald is configured to write logs to persistent disk
 (Automatic)
[PASS] 4.1.2     Ensure permissions on journal files are configured (Automatic)

Passed:          11
Failed:          0
Skipped:         4
Total checks:    15
```

如需關於基準的資訊，請參閱 Center for Internet Security (CIS) 的 Kubernetes 基準。

相關資源

- Bottlerocket 作業系統文件中的 Bottlerocket CIS 基準。

- Kubernetes 文件中的偵錯執行中的 Pod。
- Center for Internet Security (CIS) 的 Kubernetes 基準。

為 EKS 自動模式啟用含客戶自管 KMS 金鑰的 EBS 磁碟區加密

您可使用客戶管理的 KMS 金鑰來加密 EKS 自動模式執行個體的暫時性根磁碟區。

Amazon EKS Auto Mode 使用服務連結角色，在管理 Kubernetes 叢集的加密 EBS 磁碟區 AWS 時，將許可委派給其他服務。本主題描述在為 EKS 自動模式指定客戶管理金鑰以進行 Amazon EBS 加密時，所需設定的金鑰政策。

考量：

- EKS Auto Mode 不需要額外的授權，即可使用預設 AWS 受管金鑰來保護您帳戶中的加密磁碟區。
- 本主題涵蓋加密暫時性磁碟區，即 EC2 執行個體的根磁碟區。有關加密用於工作負載的資料磁碟區的更多資訊，請參閱 the section called “建立 StorageClass”。

概觀

當 EKS Auto Mode 啟動執行個體時，下列 AWS KMS 金鑰可用於 Amazon EBS 根磁碟區加密：

- AWS 受管金鑰 – 在您的帳戶中由 Amazon EBS 建立、擁有和管理的加密金鑰。這是新帳戶的預設加密金鑰。
- 客戶受管金鑰 – 您所建立、擁有和管理的自訂加密金鑰。

> **Note**
>
> 金鑰必須為對稱金鑰。Amazon EBS 不支援非對稱客戶受管金鑰。

步驟 1：設定金鑰政策

您的 KMS 金鑰必須具有金鑰政策，允許 EKS 自動模式使用透過客戶受管金鑰加密的 Amazon EBS 磁碟區來啟動執行個體。

使用下列結構設定您的金鑰政策：

> **Note**
>
> 此政策僅包含 EKS 自動模式所需的許可。如果其他身分需要使用金鑰或管理授予，金鑰政策可能需要額外授權。

```
{
    "Version":"2012-10-17",
    "Id": "MyKeyPolicy",
    "Statement": [
        {
            "Sid": "Allow use of the key",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::123456789012:role/ClusterServiceRole"
                ]
            },
            "Action": [
                "kms:Encrypt",
                "kms:Decrypt",
                "kms:ReEncrypt*",
                "kms:GenerateDataKey*",
                "kms:DescribeKey"
            ],
            "Resource": "*"
        },
        {
            "Sid": "Allow attachment of persistent resources",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::123456789012:role/ClusterServiceRole"
                ]
            },
            "Action": [
                "kms:CreateGrant",
                "kms:ListGrants",
                "kms:RevokeGrant"
            ],
            "Resource": "*",
            "Condition": {
```

```
                "Bool": {
                    "kms:GrantIsForAWSResource": "true"
                }
            }
        }
    ]
}
```

請務必將 取代<account-id>為您實際 AWS 的帳戶 ID。

在設定金鑰政策時：

- ClusterServiceRole 必須擁有必要的 IAM 許可，才能使用 KMS 金鑰進行加密操作
- kms:GrantIsForAWSResource 條件可確保只能為 AWS 服務建立授予

步驟 2：使用您的客戶管理金鑰設定 NodeClass

設定金鑰政策後，在 EKS 自動模式 NodeClass 組態中參考 KMS 金鑰：

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: my-node-class
spec:
  # Insert existing configuration

  ephemeralStorage:
    size: "80Gi"  # Range: 1-59000Gi or 1-64000G or 1-58Ti or 1-64T
    iops: 3000    # Range: 3000-16000
    throughput: 125  # Range: 125-1000

    # KMS key for encryption
    kmsKeyID: "arn:aws: kms:<region>:<account-id>:key/<key-id>"
```

使用您的實際值取代預留位置的值：

- <region> 您的 AWS 區域
- <account-id> 使用 AWS 您的帳戶 ID
- 將 <key-id> 取代為您的 KMS 金鑰 ID

您可使用下列任何格式指定 KMS 金鑰：

- KMS 金鑰 ID：`1a2b3c4d-5e6f-1a2b-3c4d-5e6f1a2b3c4d`
- KMS 金鑰 ARN： `arn:aws: kms:us-west-2:111122223333:key/1a2b3c4d-5e6f-1a2b-3c4d-5e6f1a2b3c4d`
- 金鑰別名名稱：`alias/eks-auto-mode-key`
- 金鑰別名 ARN： `arn:aws: kms:us-west-2:111122223333:alias/eks-auto-mode-key`

使用 kubectl 套用 NodeClass 組態：

```
kubectl apply -f nodeclass.yaml
```

相關資源

- 建立 Amazon EKS 的節點類別
- 在 AWS Key Management Service 開發人員指南中檢視詳細資訊
  - 金鑰政策中 AWS 服務的許可
  - 變更金鑰政策
  - AWS KMS 中的授予

更新 EKS 自動模式的組織控制

某些組織控制項可能會阻止 EKS 自動模式正常運作。如果是這種情況，您必須更新這些控制項，以允許 EKS 自動模式擁有代表您管理 EC2 執行個體所需的許可。

EKS 自動模式使用服務角色來啟動恢復 EKS 自動模式節點的 EC2 執行個體。服務角色是在您帳戶中建立的 IAM 角色，服務會擔任此角色以代表您執行操作。服務控制政策 (SCP) 始終適用於使用服務角色執行的操作。這能讓 SCP 限制自動模式的操作。最常見的情況是利用 SCP 來限制可以啟動的 Amazon Machine Image (AMI)。為了讓 EKS 自動模式能夠運作，請修改 SCP 以允許從 EKS 自動模式帳戶啟動 AMI。

您也可使用 EC2 允許的 AMI 功能來限制其他帳戶中 AMI 的可見性。若您使用此功能，必須擴展映像標準，以同時包含感興趣區域中的 EKS 自動模式 AMI 帳戶。

僅允許 EKS 自動模式 AMI 而封鎖所有其他 AMI 的範例 SCP

下面的 SCP 會阻止呼叫 `ec2:RunInstances`，除非 AMI 屬於 us-west-2 或 us-east-1 區域的 EKS 自動模式 AMI 帳戶。

> **Note**
>
> 請勿使用 `ec2:Owner` 內容索引鍵。Amazon 擁有 EKS 自動模式 AMI 帳戶，此索引鍵的值一律為 `amazon`。建構允許啟動 AMIs SCP，如果 `ec2:Owner amazon`將允許啟動任何 Amazon 擁有的 AMI，而不只是 EKS Auto Mode 的 AMI。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "DenyAMI",
      "Effect": "Deny",
      "Action": "ec2:RunInstances",
      "Resource": "arn:*:ec2:*::image/ami-*",
      "Condition": {
        "StringNotEquals": {
          "aws:ResourceAccount": [
            "767397842682",
            "992382739861"
          ]
        }
      }
    }
  ]
}
```

EKS 自動模式 AMI 帳戶

AWS 帳戶因區域主機 EKS Auto Mode 公有 AMIs 而異。

AWS 區域	帳戶
af-south-1	471112993317

ap-east-1	590183728416
ap-east-2	381492200852
ap-northeast-1	851725346105
ap-northeast-2	992382805010
ap-northeast-3	891377407544
ap-south-1	975049899075
ap-south-2	590183737426
ap-southeast-1	339712723301
ap-southeast-2	058264376476
ap-southeast-3	471112941769
ap-southeast-4	590183863144
ap-southeast-5	654654202513
ap-southeast-6	905418310314
ap-southeast-7	533267217478
ca-central-1	992382439851
ca-west-1	767397959864
eu-central-1	891376953411
eu-central-2	381492036002
eu-north-1	339712696471
eu-south-1	975049955519
eu-south-2	471112620929

eu-west-1	381492008532
eu-west-2	590184142468
eu-west-3	891376969258
il-central-1	590183797093
me-central-1	637423494195
me-south-1	905418070398
mx-central-1	211125506622
sa-east-1	339712709251
us-east-1	992382739861
us-east-2	975050179949
us-west-1	975050035094
us-west-2	767397842682
us-gov-east-1	446077414359
us-gov-west-1	446098668741

關聯公有 IP 地址

當呼叫 `ec2:RunInstances` 時，執行個體啟動的 `AssociatePublicIpAddress` 欄位是由啟動該執行個體的子網路類型自動決定的。SCP 可能被用來強制要求此值必須明確設定為 false，不論所啟動至的子網路類型為何。在這種情況下，可以將 NodeClass 欄位 `spec.advancedNetworking.associatePublicIPAddress` 也設定為 false，以滿足 SCP 的要求。

```
{
      "Sid": "DenyPublicEC2IPAddresses",
      "Effect": "Deny",
      "Action": "ec2:RunInstances",
      "Resource": "arn:aws:ec2:*:*:network-interface/*",
```

```
        "Condition": {
            "BoolIfExists": {
                "ec2:AssociatePublicIpAddress": "true"
            }
        }
    }
```

使用 EKS 自動模式控制工作負載部署至容量保留

您可以控制工作負載部署至容量保留的方式。EKS Auto Mode 支援 EC2 隨需容量保留 ODCRs)、適用於 ML 的 EC2 容量區塊，以及可中斷容量保留 IODCRs)。

> **Tip**
>
> 根據預設，EKS Auto Mode 可以透過開放比對在開放 ODCRs 中啟動，但不會排定它們的優先順序。透過開放比對啟動的執行個體會標記為 `karpenter.sh/capacity-type: on-demand`，而不是 `reserved`。若要排定 ODCR 用量的優先順序，並讓執行個體標記為 `karpenter.sh/capacity-type: reserved`，請在 NodeClass 定義`capacityReservationSelectorTerms`中設定 。ML 的容量區塊一律需要`capacityReservationSelectorTerms`且不會自動使用。

EC2 隨需容量保留 (ODCR)

EC2 隨需容量保留 (ODCR) 可讓您在特定可用區域中，為 Amazon EC2 執行個體保留任何期限的運算容量。使用 EKS 自動模式時，您可能希望控制您的 Kubernetes 工作負載是否部署到這些保留執行個體上，以最大化預購容量的利用率，或確保關鍵工作負載能夠存取保證的資源。

預設情況下，EKS 自動模式會自動啟動到開放的 ODCR 中。但是，透過在 NodeClass 上設定 `capacityReservationSelectorTerms`，您可以明確控制您的工作負載使用哪些 ODCR。透過已設定 ODCR 佈建的節點會帶有 `karpenter.sh/capacity-type: reserved` 標籤，且其優先順序高於隨需與 Spot 執行個體。啟用此功能後，EKS 自動模式將不再自動使用開放的 ODCR，必須透過 NodeClass 明確選取這些 ODCR，讓您能精確控制叢集中容量保留的使用情況。

> **Warning**
>
> 若您在叢集中的某個 NodeClass 上設定 `capacityReservationSelectorTerms`，則 EKS 自動模式將不再為叢集中任何 NodeClass 自動使用開放的 ODCR。

範例 NodeClass

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
spec:
  # Optional: Selects upon on-demand capacity reservations and capacity blocks
  # for EKS Auto Mode to prioritize.
  capacityReservationSelectorTerms:
    - id: cr-56fac701cc1951b03
    # Alternative Approaches
    - tags:
        app: "my-app"
      # Optional owning account ID filter
      owner: "012345678901"
```

此範例 NodeClass 示範了兩種選取 ODCR 的方式。第一種方式透過 ODCR 的 ID (`cr-56fac701cc1951b03`) 直接參考特定 ODCR。第二種方式使用基於標籤的選取邏輯，目標為帶有 `Name: "targeted-odcr"` 標籤的 ODCR。您也可以選擇性地依擁有保留 AWS 的帳戶進行篩選，這在跨帳戶案例或處理共用容量保留時特別有用。

排程容量預留的標籤

當您在 NodeClass `capacityReservationSelectorTerms`上設定 時，從容量保留佈建的節點會加上額外的排程標籤。您可以使用這些標籤做為 NodePool 要求或 Pod 排程限制 （例如節點親和性），以控制工作負載置放。

標籤	範例	說明
`eks.amazonaws.com/capacity-reservation-id`	`cr-56fac701cc1951b03`	容量保留的 ID
`eks.amazonaws.com/capacity-reservation-type`	`default` 或 `capacity-block`	容量保留的類型

標籤	範例	說明
eks.amazonaws.com/capacity-reservation-interruptible	true 或 false	容量保留是否可中斷

這些標籤僅存在於具有 的節點上karpenter.sh/capacity-type: reserved。

可中斷容量保留 (IODCRs)

可中斷的容量保留可讓您與組織內的其他工作負載共用未使用的隨需容量保留容量。透過將未使用的容量重新作為可中斷的 ODCRs，適合彈性、容錯操作的工作負載，例如批次處理和資料分析，可以受益於暫時可用的容量。保留擁有者可以隨時回收其容量，而中斷 ODCRs 的消費者將在終止之前收到中斷通知，以便在節點終止之前允許正常關閉或檢查點。

您可以使用與標準 ODCR capacityReservationSelectorTerms相同的方式，依 ID 或標籤選取可中斷的容量保留。若要控制工作負載是否排程在可中斷與不可中斷的預留容量上，請使用eks.amazonaws.com/capacity-reservation-interruptible排程標籤。

回收容量時，執行中的執行個體會透過 EventBridge 事件收到 2 分鐘的中斷警告。在通知期間之後，在回收容量中執行的執行個體會進入關閉狀態並終止。收到 2 分鐘中斷警告時，EKS Auto Mode 會自動開始耗盡從中斷容量保留啟動的節點，讓您的工作負載在回收節點之前正常終止。

ML 的 EC2 容量區塊

ML 的容量區塊可讓您為日後預留基於 GPU 的加速運算執行個體，以支援短時間的機器學習 (ML) 工作負載。容量區塊內執行的執行個體會在 Amazon EC2 UltraCluster 內自動放置於鄰近位置，以實現低延遲、Pb 級的非阻塞式聯網。

有關支援的平台和執行個體類型的詳細資訊，請參閱 EC2 使用者指南中的 ML 的容量區塊。

您可以建立使用 ML 的容量區塊的 EKS 自動模式 NodeClass，類似於 ODCR (先前說明)。

以下範例定義會建立了三個資源：

1. 一個引用您的容量區塊保留的 NodeClass
2. 一個使用該 NodeClass 並套用污點的 NodePool

3. 一個容忍該污點並請求 GPU 資源的 Pod 規格

範例 NodeClass

此 NodeClass 透過其保留 ID 引用特定 ML 的容量區塊。您可從 EC2 主控台取得此 ID。

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: gpu
spec:
  # Specify your Capacity Block reservation ID
  capacityReservationSelectorTerms:
    - id: cr-56fac701cc1951b03
```

如需詳細資訊，請參閱the section called “建立節點類別”。

NodePool 範例

此 NodePool 引用了 gpu NodeClass 並指定了重要組態：

- 它透過設定 `karpenter.sh/capacity-type: reserved` 僅使用預留容量
- 它請求適合 ML 工作負載的特定 GPU 執行個體系列
- 它會套用 `nvidia.com/gpu` 污點，確保僅 GPU 工作負載會排程至這些節點

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: gpu
spec:
  template:
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: gpu
      requirements:
        - key: eks.amazonaws.com/instance-family
          operator: In
          values:
```

```
            - g6
            - p4d
            - p4de
            - p5
            - p5e
            - p5en
            - p6
            - p6-b200
        - key: karpenter.sh/capacity-type
          operator: In
          values:
            - reserved
            # Enable other capacity types
            # - on-demand
            # - spot
      taints:
        - effect: NoSchedule
          key: nvidia.com/gpu
```

如需詳細資訊，請參閱the section called “建立節點集區”。

範例 Pod

此範例 Pod 示範了如何設定要在容量區塊節點上執行的工作負載：

- 它使用 nodeSelector，以特定 GPU 類型為目標 (此處為 H200 GPU)
- 它包含對 NodePool 所套用 `nvidia.com/gpu` 污點的容差
- 它使用 `nvidia.com/gpu` 資源類型明確請求 GPU 資源

```
apiVersion: v1
kind: Pod
metadata:
  name: nvidia-smi
spec:
  nodeSelector:
    # Select specific GPU type - uncomment as needed
    # eks.amazonaws.com/instance-gpu-name: l4
    # eks.amazonaws.com/instance-gpu-name: a100
    eks.amazonaws.com/instance-gpu-name: h200
    # eks.amazonaws.com/instance-gpu-name: b200
    eks.amazonaws.com/compute-type: auto
  restartPolicy: OnFailure
```

```
  containers:
  - name: nvidia-smi
    image: public.ecr.aws/amazonlinux/amazonlinux:2023-minimal
    args:
    - "nvidia-smi"
    resources:
      requests:
        # Uncomment if needed
        # memory: "30Gi"
        # cpu: "3500m"
        nvidia.com/gpu: 1
      limits:
        # Uncomment if needed
        # memory: "30Gi"
        nvidia.com/gpu: 1
  tolerations:
  - key: nvidia.com/gpu
    effect: NoSchedule
    operator: Exists
```

如需詳細資訊，請參閱 Kubernetes 文件中的 Pod。

相關資源

- 《Amazon EC2 使用者指南》中的 ML 的容量區塊
- 《Amazon EC2 使用者指南》中的尋找和購買容量區塊
- 《Amazon EC2 使用者指南》中的可中斷的容量保留
- 管理 Amazon EKS 上 AI/ML 工作負載的運算資源
- 《EKS 最佳實務指南》中的 GPU 資源最佳化和成本管理

將 EKS Auto Mode 節點部署至 Local Zones

EKS Auto Mode 透過自動節點佈建提供簡化的叢集管理。 AWS 本機區域可將 AWS 基礎設施擴展到更接近最終使用者的地理位置，從而減少延遲敏感應用程式的延遲。本指南會逐步解說將 EKS Auto Mode 節點部署至 AWS Local Zones 的程序，讓您針對特定地理區域的使用者執行具有較低延遲的容器化應用程式。

本指南也示範如何使用 Kubernetes 污點和容錯，以確保只有特定工作負載會在 Local Zone 節點上執行，協助您控制成本並最佳化資源用量。

先決條件

開始將 EKS Auto Mode 節點部署到 Local Zones 之前，請確定您已備妥下列先決條件：

- 現有的 EKS 自動模式叢集
- 選擇加入您 AWS 帳戶中的本機區域

步驟 1：建立本機區域子網路

將 EKS Auto Mode 節點部署到 Local Zone 的第一步是在該 Local Zone 中建立子網路。此子網路為您的節點提供網路基礎設施，並允許它們與您的其餘 VPC 通訊。遵循建立本機區域子網路指示（在 AWS 本機區域使用者指南中），在您選擇的本機區域中建立子網路。

> **Tip**
>
> 請記下本機區域子網路的名稱。

步驟 2：為本機區域子網路建立 NodeClass

建立本機區域子網路之後，您需要定義參考此子網路的 NodeClass。NodeClass 是一種 Kubernetes 自訂資源，可指定節點的基礎設施屬性，包括要使用的子網路、安全群組和儲存組態。在下面的範例中，我們建立名為 "local-zone" 的 NodeClass，根據其名稱以本機區域子網路為目標。您也可以使用子網路 ID。您需要調整此組態以鎖定您的 Local Zone 子網路。

如需詳細資訊，請參閱the section called “建立節點類別”。

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: local-zone
spec:
  subnetSelectorTerms:
    - id: <local-subnet-id>
```

步驟 3：使用 NodeClass 和 Taint 建立 NodePool

設定 NodeClass 後，您現在需要建立使用此 NodeClass 的 NodePool。NodePool 會定義節點的運算特性，包括執行個體類型。NodePool 使用 NodeClass 做為參考，以判斷在何處啟動執行個體。

在以下範例中，我們會建立參考 "local-zone" NodeClass 的 NodePool。我們也會將污點新增至節點，以確保只有具有相符公差的 Pod 才能排程在這些 Local Zone 節點上。這對 Local Zone 節點特別重要，這通常具有較高的成本，並且只能由特別受益於降低延遲的工作負載使用。

如需詳細資訊，請參閱the section called “建立節點集區”。

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: my-node-pool
spec:
  template:
    metadata:
      labels:
        node-type: local-zone
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: local-zone
      taints:
        - key: "aws.amazon.com/local-zone"
          value: "true"
          effect: NoSchedule

      requirements:
        - key: "eks.amazonaws.com/instance-category"
          operator: In
          values: ["c", "m", "r"]
        - key: "eks.amazonaws.com/instance-cpu"
          operator: In
          values: ["4", "8", "16", "32"]
```

具有金鑰`aws.amazon.com/local-zone`和效果的污點`NoSchedule`可確保在這些節點上不會排程沒有相符公差的 Pod。這可防止定期工作負載意外在 Local Zone 中執行，這可能會導致意外的成本。

步驟 4：部署具有公差和節點親和性的工作負載

若要最佳控制本機區域節點上的工作負載置放，請同時使用污點/容錯和節點親和性。此合併方法提供下列優點：

1. 成本控制：污點可確保只有具有明確公差的 Pod 可以使用可能昂貴的 Local Zone 資源。

2. 保證配置：節點親和性可確保對延遲敏感的應用程式僅在本機區域執行，而不是在一般叢集節點上執行。

以下是設定為在 Local Zone 節點上特別執行的部署範例：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: low-latency-app
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels:
      app: low-latency-app
  template:
    metadata:
      labels:
        app: low-latency-app
    spec:
      tolerations:
      - key: "aws.amazon.com/local-zone"
        operator: "Equal"
        value: "true"
        effect: "NoSchedule"
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: "node-type"
                operator: "In"
                values: ["local-zone"]
      containers:
      - name: application
        image: my-low-latency-app:latest
        resources:
          limits:
            cpu: "1"
            memory: "1Gi"
          requests:
            cpu: "500m"
```

```
            memory: "512Mi"
```

此部署有兩個金鑰排程組態：

1. 容錯允許在具有aws.amazon.com/local-zone污點的節點上排程 Pod。
2. 節點親和性要求可確保這些 Pod 只會在標籤為 的節點上執行node-type: local-zone。

這些共同確保您的延遲敏感應用程式僅在 Local Zone 節點上執行，且一般應用程式不會使用 Local Zone 資源，除非明確設定這樣做。

步驟 5：使用 AWS 主控台驗證

設定 NodeClass、NodePool 和 部署之後，您應該驗證節點是否如預期佈建在 Local Zone，以及工作負載是否在其上執行。您可以使用 AWS 管理主控台來驗證 EC2 執行個體是否在正確的本機區域子網路中啟動。

此外，您可以使用 檢查 Kubernetes 節點清單kubectl get nodes -o wide，以確認節點使用正確的標籤和污點來加入您的叢集：

```
kubectl get nodes -o wide
kubectl describe node <node-name> | grep -A 5 Taints
```

您也可以確認您的工作負載 Pod 已排程在 Local Zone 節點上：

```
kubectl get pods -o wide
```

此方法可確保只會在這些節點上排定特別容忍 Local Zone 污點的工作負載，協助您控制成本並最有效地使用 Local Zone 資源。

設定節點的進階安全設定

本主題說明如何使用節點類別中的advancedSecurity規格來設定 Amazon EKS Auto Mode 節點的進階安全設定。

先決條件

開始前，請確保您具備以下條件：

- 一個 Amazon EKS 自動模式叢集。如需詳細資訊，請參閱the section called “建立叢集”。
- kubectl 已安裝並已設定。如需詳細資訊，請參閱設定。

- 了解節點類別組態。如需詳細資訊，請參閱the section called “建立節點類別”。

設定進階安全設定

若要設定節點的進階安全設定，請在節點類別規格中設定advancedSecurity欄位：

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: security-hardened
spec:
  role: MyNodeRole

  subnetSelectorTerms:
    - tags:
        Name: "private-subnet"

  securityGroupSelectorTerms:
    - tags:
        Name: "eks-cluster-sg"

  advancedSecurity:
    # Enable FIPS-compliant AMIs (US regions only)
    fips: true

    # Configure kernel lockdown mode
    kernelLockdown: "integrity"
```

套用此組態：

```
kubectl apply -f nodeclass.yaml
```

在您的節點集區組態中參考此節點類別。如需詳細資訊，請參閱the section called “建立節點集區”。

欄位描述

- fips（布林值，選用）：設為 時true， 會使用具有 FIPS 140-2 驗證密碼編譯模組的 AMIs 佈建節點。此設定會選取符合 FIPS 標準的 AMIs；客戶負責管理其合規要求。如需詳細資訊，請參閱 AWS FIPS 合規。預設：false。
- kernelLockdown（字串，選用）：控制核心鎖定安全模組模式。接受的值：

- `integrity`：封鎖覆寫核心記憶體或修改核心程式碼的方法。防止未簽署的核心模組載入。
- `none`：停用核心鎖定保護。

  如需詳細資訊，請參閱 Linux 核心鎖定文件。

考量事項

- FIPS 相容的 AMIs 適用於 AWS 美國東部/西部、 AWS GovCloud (US) 和 AWS 加拿大 （ 中部/西部 ） 區域。如需詳細資訊，請參閱 AWS FIPS 合規。
- 使用 時`kernelLockdown: "integrity"`，請確保您的工作負載不需要載入未簽署的核心模組或修改核心記憶體。

相關資源

- the section called “建立節點類別” - 完成節點類別組態指南
- the section called “建立節點集區” - 節點集區組態

EKS Auto 模式的成本最佳化

EKS Auto Mode 透過整合、bin-packing 和正確調整大小，持續最佳化叢集的運算成本。不過，某些工作負載組態可能會阻止這些最佳化。本主題說明成本最佳化的運作方式、封鎖成本的方式，以及如何設定叢集以維持成本效益。

EKS Auto Mode 如何最佳化成本

EKS Auto Mode 透過下列機制降低運算成本：

- Bin-packing – 將 Pod 排程到節點時，EKS Auto Mode 會選取與彙總資源請求密切相符的執行個體類型，將未使用的容量降至最低。
- 合併 — EKS Auto Mode 會定期評估執行中的節點，並在工作負載可在較低或成本較低的執行個體上執行時取代或移除它們。
- 適當調整大小 — 隨著工作負載縮減，EKS Auto Mode 會將 Pod 合併到較小的節點，並終止未充分利用的執行個體。

這些最佳化會在不手動介入的情況下持續執行。不過，某些 Pod 註釋和 NodePool 組態可能會阻止合併生效。

內建節點集區和成本防護機制

內建`general-purpose`和`system`節點集區已強制執行數個成本保護預設值：

- 僅限 C、M 和 R 的執行個體系列 — 不允許加速 (P、G、Inf、Trn) 或特殊執行個體類型。
- 僅限隨需容量 — 無 Spot 執行個體，可避免中斷驅動流失，但也表示沒有 Spot 節省。
- 第 5 代或更新版本 — 不包括較舊、低成本效益的執行個體產生。

如果您只使用內建節點集區，您已受益於這些護欄。當您建立不會繼承這些限制的自訂 NodePools 時，本主題中有關排除執行個體系列和限制執行個體大小的指引最為相關。

不過，即使使用內建節點集區，下列各節仍適用於您：

- the section called “哪些項目會封鎖合併” — `do-not-disrupt`註釋和限制性 PDBs區塊合併，無論節點佈建哪個 NodePool。
- the section called “使用 NodePool 限制作為成本上限” — 內建節點集區未`limits`設定資源。如果您的工作負載可以大幅擴展，請考慮建立具有限制的自訂 NodePool，而不是依賴無限制的內建集區。
- the section called “節點生命週期和成本” — 節點取代重疊適用於所有節點，包括由內建集區佈建的節點。

護欄	內建節點集區	自訂 NodePools
加速執行個體排除	強制執行	您必須設定
執行個體大小限制	未設定	您必須設定
資源 `limits`(CPU/記憶體上限)	未設定	您必須設定
僅限隨需	強制執行	您可以選擇 (Spot/On-Demand)
合併保護 (`do-not-disrupt` /PDB)	您的責任	您的責任

哪些項目會封鎖合併

當 EKS Auto Mode 判斷中斷節點會違反工作負載的可用性需求時，會封鎖合併。下列組態可防止整合：

`do-not-disrupt` 註釋

`karpenter.sh/do-not-disrupt` 註釋會指示 EKS Auto Mode 只要註釋的 Pod 正在節點上執行，即可保留節點。這可防止節點合併、取代或終止，即使節點未充分利用也一樣。

```
metadata:
  annotations:
    karpenter.sh/do-not-disrupt: "true"
```

Important

成本隱含：當 Pod 攜帶`do-not-disrupt`註釋時，其執行的節點無需合併。這表示：

- 無論實際使用率為何，節點都會繼續以目前的執行個體大小執行。
- 即使工作負載的需求降低，該節點上的 vCPU 和記憶體用量也可以保持升高。
- 如果多個節點的多個 Pod 具有此註釋，則整個叢集的整合會大幅降低，進而持續提高成本。

`do-not-disrupt` 註釋是一種可用性機制。它不會考慮成本。僅用於執行中中斷導致資料遺失或重大重做的工作負載，例如長時間執行的批次任務或狀態程序，無需檢查點。

要考慮的替代方案：

- Pod 中斷預算 (PDBs) — 使用 PDBs控制中斷率，而不是完全封鎖中斷率。PDBs進行合併，同時確保最少數量的複本保持可用。
- 較短的工作負載：對於 CI/CD 執行器和建置代理程式，允許中斷，並依賴 CI 系統的內建重試邏輯，而不是使用 `do-not-disrupt`。
- 時間限制註釋 — `do-not-disrupt`僅適用於關鍵操作的持續時間，然後在操作完成時以程式設計方式將其移除。

Pod 中斷預算 (PDBs)

設定`maxUnavailable: 0`或`minAvailable`等於目前複本計數的 PDBs 可有效封鎖受影響 Pod 的所有合併。檢閱您的 PDBs以確保它們一次允許至少一個 Pod 中斷。

使用 NodePool 限制作為成本上限

NodePool 在 NodePool 可佈建的運算資源總數上`limits`設定硬性上限。達到限制時，EKS Auto Mode 會停止啟動該 NodePool 的新節點。即使 Pod 處於待定狀態，也會發生這種情況。

使用 `limits`作為成本防護機制，特別是對於提供非生產、測試或高載工作負載的 NodePools，這些工作負載不適合無限制擴展。

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: ci-runners
spec:
  template:
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default
      requirements:
        - key: "eks.amazonaws.com/instance-category"
          operator: In
          values: ["c", "m"]
  limits:
    cpu: "500"
    memory: 1000Gi
```

在此範例中，`ci-runners`NodePool 在佈建的所有節點中總計不得超過 500 個 vCPUs 或 1000 GiB 的記憶體。超過此限制的 Pod 會保持 `Pending` 狀態，直到容量釋放為止。

> **Tip**
>
> `limits` 根據預期的爆量大小上限加上節點替換緩衝區來設定 。定期檢閱您的 NodePool 使用率，並在工作負載模式變更時調整限制。

排除執行個體系列以進行成本控制

根據預設，EKS Auto Mode 會從廣泛的執行個體類型中選取，以最大化排程彈性。對於不需要專用硬體的工作負載，請限制執行個體系列，以防止啟動昂貴的執行個體類型。

排除加速執行個體

如果您的工作負載未請求 GPU 或加速器資源，請從 NodePool 中排除加速執行個體系列。這可防止在容量限制期間選取加速執行個體的情況。

```
spec:
  template:
    spec:
      requirements:
        - key: "eks.amazonaws.com/instance-category"
          operator: In
          values: ["c", "m", "r"]
```

透過僅指定運算最佳化、一般用途和記憶體最佳化類別，您可以從選取範圍中排除加速 (P、G、Inf、Trn) 和其他特殊執行個體系列。

執行個體選擇如何與容量限制條件互動

EKS Auto Mode 會在正常執行個體選取期間，將加速和奇特執行個體類型取消優先順序。不過，當持續啟動失敗時，EKS Auto Mode 會從任何剩餘的可用執行個體類型啟動，以排定工作負載可用性的優先順序。例如，當 EC2 服務配額暫時用盡所有偏好的執行個體類型時，就會發生這種情況。

為了防止這種備用行為， 會明確將您的 NodePool 需求限制為僅工作負載所需的執行個體類別。當偏好的類型無法使用，且 NodePool 組態不允許其他類型時，Pod 會保持 `Pending` 狀態，而不是排程在昂貴的執行個體上。

限制執行個體大小

除了限制執行個體系列之外，您還可以限制 NodePool 內的執行個體大小上限。限制執行個體大小會限制無法合併的任何單一節點的成本暴露。例如，即使`do-not-disrupt`節點的工作負載很小，註釋封鎖的節點也無法縮小。

使用 `eks.amazonaws.com/instance-cpu`標籤來限制 NodePool 需求中的執行個體大小上限：

```
requirements:
  - key: "eks.amazonaws.com/instance-cpu"
    operator: Lte
    values: ["32"]
```

此組態可防止 EKS Auto 模式在此 NodePool 中啟動大於 32 個 vCPUs執行個體。

若要識別現有叢集中的最佳化機會，請檢閱您最大的執行中執行個體。如果持續封鎖大型節點，則閒置容量的每個節點成本會按比例提高。

高載工作負載的建議模式

CI/CD 管道、批次任務和暫時性執行器會建立爆burst-and-idle模式，這些模式需要特定組態來維持成本效益。

高載工作負載的建議預設值

Configuration	建議
`do-not-disrupt`	請勿將 用於 CI/CD 執行器。請改用 CI 系統的重試和佇列機制。
NodePool `limits`	根據預期的並行上限加上節點取代重疊緩衝區，設定 CPU/記憶體上限。
執行個體類別	限制為 `c`和 `m` 系列。排除非 GPU 工作負載的加速執行個體系列 (P、G、Inf、Trn)。
執行個體大小	考慮限制為中等大小 （ 例如 4–32 vCPUs)，以限制從合併封鎖的任何單一節點的成本暴露。
合併時間	使用`consolidateAfter` 預設設定。避免設定長時間延遲，以在執行器完成後保持高載容量上線。
容量類型	針對容錯執行器使用 Spot 執行個體。針對在執行期間保留狀態的建置代理程式，結合隨需。

範例：CI Runner NodePool

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: ci-runners
spec:
  template:
    spec:
      nodeClassRef:
```

```
        group: eks.amazonaws.com
        kind: NodeClass
        name: default
      requirements:
        - key: "eks.amazonaws.com/instance-category"
          operator: In
          values: ["c", "m"]
        - key: "eks.amazonaws.com/instance-cpu"
          operator: Lte
          values: ["32"]
        - key: "karpenter.sh/capacity-type"
          operator: In
          values: ["spot", "on-demand"]
  limits:
    cpu: "500"
    memory: 1000Gi
  disruption:
    consolidationPolicy: WhenEmptyOrUnderutilized
    consolidateAfter: 30s
```

此組態：

- 限制經濟實惠的執行個體系列
- 將總 NodePool 容量限制在 500 個 vCPUs
- 允許積極整合（移除 Pod 後 30 秒）
- 允許 Spot 和隨需容量

節點生命週期和成本

EKS Auto Mode 會在節點偏離所需規格（例如，在發行新的 Auto Mode AMI 之後）或節點生命週期接近到期時，透過正常中斷取代節點。在正常取代期間：

- 新的替換節點已啟動並準備就緒。
- Pod 從舊節點耗盡，遵守 Pod 中斷預算。
- 在短時間內，舊節點和替換節點都會同時執行。

對於具有大型或多個節點的叢集，此重疊可能會定期增加成本。若要將影響降至最低：

- 檢閱中斷預算 — 確保您的中斷預算允許及時耗盡。限制性預算會延長新舊節點執行的重疊期間。

- 適當大小的執行個體 — 較小的執行個體可減少重疊時段的絕對成本。
- 縮短節點生命週期上限 — 較短的到期值（例如 7 天）會建立更頻繁但較小的替換事件。這樣可以更平均地分散成本，而不是集中成本。

如需節點生命週期的詳細資訊，請參閱了解 Amazon EKS Auto Mode 受管執行個體。

了解 EKS 自動模式的運作方式

使用本章節來了解 Amazon EKS 自動模式叢集的元件如何運作。

主題

- 了解 Amazon EKS 自動模式受管執行個體
- 了解 EKS 自動模式中的身分和存取
- 了解 EKS 自動模式中的 VPC 聯網與負載平衡

了解 Amazon EKS 自動模式受管執行個體

本主題說明 Amazon EKS 自動模式如何管理您 EKS 叢集中的 Amazon EC2 執行個體。當您啟用 EKS 自動模式後，叢集的運算資源將由 EKS 自動佈建與管理，這會改變您與作為叢集節點的 EC2 執行個體之間的互動方式。

了解 Amazon EKS 自動模式管理執行個體的機制，對於規劃工作負載部署策略與操作程序至關重要。不同於傳統 EC2 執行個體或受管節點群組，此類執行個體採用不同的生命週期模型：EKS 會負責多數操作環節，同時限制特定類型的存取與自訂功能。

Amazon EKS 自動模式會自動執行建立新 EC2 執行個體的常規工作，並將這些執行個體作為節點附加至您的 EKS 叢集。在 EKS 自動模式偵測到現有節點無法容納工作負載時，便會建立新的 EC2 執行個體。

Amazon EKS 自動模式負責 EC2 執行個體的建立、刪除與修補。您則需負責管理部署在執行個體上的容器與 Pod。

由 EKS 自動模式建立的 EC2 執行個體不同於其他 EC2 執行個體，這類執行個體屬於「受管執行個體」。此類受管執行個體歸 EKS 所有，且存取權限更為嚴格。您無法直接存取 EKS 自動模式管理的執行個體，也無法在其上安裝軟體。

AWS 建議執行 EKS Auto Mode 或自我管理 Karpenter。您可在移轉期間或進階組態中同時安裝兩者。若已同時安裝，請設定節點集區，使工作負載僅與 Karpenter 或 EKS 自動模式其一相關聯。

如需詳細資訊，請參閱《Amazon EC2 使用者指南》中的 Amazon EC2 受管執行個體。

比較表格

標準 EC2 執行個體	EKS 自動模式受管執行個體
由您負責執行個體的修補與更新作業。	AWS 會自動修補和更新執行個體。
EKS 不負責執行個體上的軟體。	EKS 負責執行個體上的特定軟體，如 `kubelet`、容器執行時期及作業系統。
您可使用 EC2 API 刪除 EC2 執行個體。	EKS 決定您帳戶中部署的執行個體數量。在刪除工作負載時，EKS 會減少帳戶中的執行個體數量。
您可透過 SSH 存取 EC2 執行個體。	您可將 Pod 與容器部署至受管執行個體。
由您決定作業系統與映像 (AMI)。	AWS 會決定作業系統和映像。
您可部署依賴 Windows 或 Ubuntu 功能的工作負載。	您可部署基於 Linux 的容器，但不能有特定作業系統相依性。
由您決定要啟動的執行個體類型與系列。	AWS 決定要啟動的執行個體類型和系列。您可透過節點集區限制 EKS 自動模式可選取的執行個體類型。

下列功能同時適用於受管執行個體與標準 EC2 執行個體：

- 您可將執行個體儲存空間作為工作負載的臨時儲存。

AMI 支援

使用 EKS Auto Mode 時， AWS 會決定運算節點所使用的映像 (AMI)。 AWS 會監控新 EKS Auto Mode AMI 版本的推出。若您遇到與 AMI 版本相關的工作負載問題，請建立支援案例。如需詳細資訊，請參閱《支援使用者指南》中的建立支援案例和案例管理。 AWS

通常情況下，EKS 每週都會發布包含 CVE 與安全性修補的新 AMI。

受管資源可見性

> **Note**
>
> 自 2026 年 4 月 22 日起，EKS Auto Mode 建立的新 Amazon EC2 受管執行個體和相關資源（例如 EC2 啟動範本、EBS 磁碟區和網路介面 (ENIs)) 預設會從 EC2 主控台檢視和 `describe` API 清單操作隱藏。在該日期之前已存在於您帳戶中的受管資源會保持可見。您可以使用受管資源可見性設定來變更此行為。

Amazon EC2 提供受管資源可見性設定，可控制受管資源是否出現在您的 EC2 主控台檢視和 API 清單操作中，例如 `DescribeInstances`。

隱藏受管資源時，EKS Auto Mode 受管執行個體及其相關聯的資源 (EBS 磁碟區、啟動範本和網路介面） 不會出現在 EC2 主控台或 `describe` API 回應中。這可以簡化控管儀表板、減少可觀測性工具中的雜訊，並防止雲端安全狀態管理 (CSPM) 掃描器中將受管資源標記為客戶錯誤組態的誤報。

可見性設定會套用至整個帳戶的所有 IAM 主體。您無法依資源類型或建立資源的服務選擇性地顯示或隱藏受管資源。隱藏的資源仍能完全運作並計費。

您可以隨時透過 Amazon EC2 主控台或 CLI AWS 變更可見性設定。如需詳細資訊，請參閱《Amazon EC2 使用者指南》中的受管資源可見性設定。

> **Note**
>
> 即使受管資源從 EC2 主控台檢視和清單 APIs 隱藏，您仍然可以透過下列方式檢視 EKS Auto Mode 執行個體：
>
> - Amazon EKS 主控台，位於叢集的運算索引標籤下。
> - Kubernetes APIs（例如，`kubectl get nodes`)。
> - 依執行個體 ID 的直接 EC2 API 查詢（例如 `describe-instances --instance-ids i-0123456789abcdef0`)。
> - 具有 `include-managed-resources` 參數的 `DescribeInstances` API。
> - EC2 主控台，變更受管資源可見性設定後。

EKS 自動模式支援的執行個體參考

EKS 自動模式僅會建立「支援類型」且「符合最小規模需求」的執行個體。

EKS 自動模式支援下列執行個體類型：

系列	執行個體類型
運算最佳化 (C)	c9g、c9gd、c8a、c8g、c8gb、c8gd、c8gn、c8i、c8i-flex、c8ib、c8id、c7a、c7g、c7gd、c7gn、c7i、c7i-flex、c6a、c6g、c6gd、c6gn、c6i、c6id、c6in、c5a、c5ad、c5d、c5n、c4
一般用途 (M)	m9gd、m8a、m8azn、m8g、m8gb、m8gd、m8gn、m8i、m8i-flex、m8id、m7a、m7g、m7gd、m7i、m7i-flex、m6a、m6g、m6gd、m6i、m6id、m6idn、m6in、m5、m5a、m5ad、m5d、m5dn、m5n、m5zn、m4
記憶體最佳化 (R)	r9g、r8a、r8g、r8gb、r8gd、r8gn、r8i、r8i-flex、r8id、r7a、r7g、r7gd、r7i、r7iz、r6a、r6g、r6gd、r6i、r6id、r6idn、r6in、r5、r5a、r5ad、r5b、r5d、r5dn、r5n、r4
爆量 (T)	t4g、t3、t3a、t2
記憶體密集型 (Z/X)	z1d、x8aedz、x8g、x8i、x2gd
儲存最佳化 (I/D)	i8g、i8ge、i7i、i7ie、i4g、is4gen、im4gn、i4i、i3、i3en、d3、d3en
加速運算 (Trn/P/Inf/G)	trn2、trn1、trn1n、p6-b200、p6-b300、p5、p5e、p5en、p4d、p4de、p3、p3dn、inf2、inf1、g7e、g6、gr6、g6e、g5、g5g、g4ad、g4dn
高效能運算 (X2/HPC)	x2idn、x2iedn、x2iezn、hpc8a

此外，EKS 自動模式僅會建立符合下列要求的 EC2 執行個體：

- CPU 數量大於 1
- 執行個體尺寸並非 nano、micro 或 small

如需詳細資訊，請參閱 Amazon EC2 執行個體類型命名慣例。

執行個體中繼資料服務

- EKS Auto Mode 預設會強制執行 IMDSv2，跳轉限制為 1，並遵循 AWS 安全最佳實務。
- 此預設組態在 Auto Mode 中無法修改。
- 對於通常需要 IMDS 存取的附加元件，請在安裝期間提供參數（例如 AWS 區域），以避免 IMDS 查詢。如需詳細資訊，請參閱the section called “您可自訂的欄位”。
- 若 Pod 在 Auto Mode 中執行時絕對需要存取 IMDS，必須將 Pod 設定為搭配 `hostNetwork: true` 執行。如此 Pod 才能直接存取執行個體中繼資料服務。
- 在授予 Pods 存取執行個體中繼資料的權限時，請務必考量安全性影響。

如需有關 Amazon EC2 執行個體中繼資料服務 (IMDS) 的詳細資訊，請參閱《Amazon EC2 使用者指南》中的設定執行個體中繼資料服務選項。

考量事項

- 若 NodeClass 中設定的臨時儲存空間小於執行個體的 NVMe 本機儲存空間，EKS 自動模式會自動執行下列動作，無需手動設定：
  - 使用較小 (20 GiB) 的 Amazon EBS 資料磁碟區以降低成本。
  - 格式化並設定 NVMe 本機儲存空間，用於儲存臨時資料。這包括在有多個 NVMe 磁碟機時設定 RAID 0 陣列。
- 若 `ephemeralStorage.size` 等於或超過本機 NVMe 容量，則會發生下列動作：
  - Auto Mode 跳過小型 EBS 磁碟區。
  - NVMe 磁碟機會直接開放給您的工作負載使用。
- Amazon EKS Auto Mode 不支援下列 Fault Injection Service AWS 動作：
  - `ec2:RebootInstances`
  - `ec2:SendSpotInstanceInterruptions`
  - `ec2:StartInstances`
  - `ec2:StopInstances`
  - `ec2:TerminateInstances`
  - `ec2:PauseVolumeIO`
- Amazon EKS Auto Mode AWS 支援 Fault Injection Service EKS Pod 動作。如需詳細資訊，請參閱 AWS 《Resilience Hub 使用者指南》中的管理故障注入服務實驗和使用 AWS FIS aws：eks：pod 動作。

- 無需在 EKS 自動模式節點上安裝 `Neuron Device Plugin`。

  若您的叢集中有其他類型的節點，需設定 Neuron 裝置外掛程式不在 Auto Mode 節點上執行。如需詳細資訊，請參閱the section called “控制部署”。

了解 EKS 自動模式中的身分和存取

本主題描述使用 EKS 自動模式所需的 Identity and Access Management (IAM) 角色與許可。EKS 自動模式使用兩個主要 IAM 角色：叢集 IAM 角色和節點 IAM 角色。這些角色與 EKS Pod 身分識別和 EKS 存取項目協同工作，為您的 EKS 叢集提供全面的存取管理。

當您設定 EKS Auto Mode 時，您將需要設定這些具有特定許可的 IAM 角色，以允許 AWS 服務與您的叢集資源互動。這包括用於管理運算資源、儲存磁碟區、負載平衡器和聯網元件的許可。了解這些角色組態對於正確的叢集操作與安全性至關重要。

在 EKS Auto 模式中， AWS IAM 角色會透過 EKS 存取項目自動映射至 Kubernetes 許可，無需手動設定 `aws-auth` ConfigMaps 或自訂繫結。當您建立新的自動模式叢集時，EKS 會使用存取項目自動建立對應的 Kubernetes 許可，確保 AWS 服務和叢集元件在 AWS 和 Kubernetes 授權系統中都具有適當的存取層級。這種自動化整合降低了設定複雜性，並有助於預防管理 EKS 叢集時常見的許可相關問題。

叢集 IAM 角色

叢集 IAM 角色是 Amazon EKS 用來管理 Kubernetes 叢集許可的 AWS Identity and Access Management (IAM) 角色。此角色會授予 Amazon EKS 必要的許可，以代表您叢集與其他 AWS 服務互動，並使用 EKS 存取項目自動設定 Kubernetes 許可。

- 您必須將 AWS IAM 政策連接至此角色。
- EKS 自動模式會使用 EKS 存取項目自動將 Kubernetes 許可附加到此角色。
- 使用 EKS Auto 模式時， AWS 建議為每個 AWS 帳戶建立單一叢集 IAM 角色。
- AWS 建議將此角色命名為 `AmazonEKSAutoClusterRole`。
- 此角色需要多個 AWS 服務的許可，才能管理 資源，包括 EBS 磁碟區、Elastic Load Balancer 和 EC2 執行個體。
- 此角色的建議組態包含多個 AWS 受管 IAM 政策，與 EKS Auto Mode 的不同功能相關。
  - `AmazonEKSComputePolicy`
  - `AmazonEKSBlockStoragePolicyV2`
  - `AmazonEKSLoadBalancingPolicy`

- AmazonEKSNetworkingPolicy
- AmazonEKSClusterPolicy

如需叢集 IAM 角色和 AWS 受管 IAM 政策的詳細資訊，請參閱：

- the section called “ AWS 受管政策”
- the section called “叢集 IAM 角色”

如需有關 Kubernetes 存取的詳細資訊，請參閱：

- the section called “檢閱存取政策”

節點 IAM 角色

節點 IAM 角色是 Amazon EKS 用來管理 Kubernetes 叢集中工作者節點許可的 AWS Identity and Access Management (IAM) 角色。此角色會授予以 Kubernetes 節點身分執行的 EC2 執行個體與 AWS 服務和資源互動的必要許可，並使用 EKS 存取項目自動設定 Kubernetes RBAC 許可。

- 您必須將 AWS IAM 政策連接至此角色。
- EKS 自動模式會使用 EKS 存取項目自動將 Kubernetes RBAC 許可附加到此角色。
- AWS 建議將此角色命名為 AmazonEKSAutoNodeRole。
- 使用 EKS Auto Mode， AWS 建議為每個 AWS 帳戶建立單一節點 IAM 角色。
- 此角色具有有限的許可。關鍵許可包括擔任 Pod 身分識別角色，以及從 ECR 提取映像。
- AWS 建議下列 AWS 受管 IAM 政策：
  - AmazonEKSWorkerNodeMinimalPolicy
  - AmazonEC2ContainerRegistryPullOnly

如需節點 IAM 角色和 AWS 受管 IAM 政策的詳細資訊，請參閱：

- the section called “ AWS 受管政策”
- the section called “節點 IAM 角色”

如需有關 Kubernetes 存取的詳細資訊，請參閱：

- the section called “檢閱存取政策”

服務連結角色

Amazon EKS 針對某些操作使用服務連結角色 (SLR)。服務連結角色是直接連結至 Amazon EKS 的一種特殊 IAM 角色類型。服務連結角色是由 Amazon EKS 預先定義，並包含該服務代表您呼叫其他 AWS 服務所需的所有許可。

AWS 會自動建立和設定 SLR。您必須先刪除 SLR 的相關資源，才能刪除角色。如此可保護您的 Amazon EKS 資源，避免您不小心移除資源的存取許可。

SLR 政策授予 Amazon EKS 觀察和刪除核心基礎結構元件的許可：EC2 資源 (執行個體、網路介面、安全群組)、ELB 資源 (負載平衡器、目標群組)、CloudWatch 功能 (記錄和指標)，以及具有「eks」字首的 IAM 角色。它還透過 VPC/託管區域關聯啟用私有端點聯網，並包含用於 EventBridge 監控和清理帶有 EKS 標籤之資源的許可。

如需詳細資訊，請參閱：

- the section called “AWS 受管政策：AmazonEKSServiceRolePolicy”
- the section called “Amazon EKS 的服務連結角色許可”

存取政策參考

如需 EKS 自動模式使用的 Kubernetes 許可的詳細資訊，請參閱 the section called “檢閱存取政策”。

了解 EKS 自動模式中的 VPC 聯網與負載平衡

本主題闡釋如何在 EKS 自動模式中設定虛擬私有雲端 (VPC) 聯網與負載平衡功能。雖然 EKS 自動模式會自動管理大多數聯網元件，但您仍然可以透過 `NodeClass` 資源和負載平衡器注釋來自訂叢集聯網組態的某些方面。

當您使用 EKS Auto Mode 時， 會 AWS 管理叢集的 VPC 容器網路界面 (CNI) 組態和負載平衡器佈建。您可透過定義 `NodeClass` 物件，以及將特定注釋套用至服務與傳入資源來影響聯網行為，同時維持 EKS 自動模式提供的自動化營運模式。

聯網功能

EKS 自動模式具有一個處理節點和 Pod 聯網的新聯網功能。您可以透過建立一個 `NodeClass` Kubernetes 物件進行設定。

先前 AWS VPC CNI 的組態選項不適用於 EKS Auto 模式。

使用 NodeClass 設定聯網

EKS 自動模式中的 NodeClass 資源讓您能自訂聯網功能的特定層面。透過 NodeClass，您可以指定安全群組選擇、控制節點在 VPC 子網路上的放置、設定 SNAT 政策、配置網路政策以及啟用網路事件記錄。這種方法在提供聯網自訂彈性的同時，保持了 EKS 自動模式的自動化操作模型。

您可以使用 NodeClass 來：

- 選取節點的安全群組
- 控制節點在 VPC 子網路上的放置方式
- 將節點 SNAT 政策設定為 random 或 disabled
- 啟用 Kubernetes 網路政策，包括：
  - 將網路政策設定為預設拒絕或預設允許
  - 啟用網路事件記錄至檔案。
- 透過將 Pod 附加至不同的子網路，將 Pod 流量與節點流量隔離。

了解如何建立 Amazon EKS NodeClass。

考量事項

EKS 自動模式支援：

- EKS 網路政策。
- Kubernetes Pod 的 HostPort 與 HostNetwork 選項。
- 位於公有或私有子網路中的節點和 Pod。
- 在節點上快取 DNS 查詢。
- EFA 工作負載的靜態網路介面組態。如需詳細資訊，請參閱the section called “靜態網路界面組態”。

EKS 自動模式不支援下列功能：

- 每個 Pod 的安全群組 (SGPP)。若要在自動模式下將個別安全群組套用至 Pod 流量，請NodeClass改為在 podSecurityGroupSelectorTerms中使用 。如需詳細資訊，請參閱the section called “Pod 的個別子網路和安全群組”。
- ENIConfig 中的自訂聯網。您可以使用 the section called “Pod 的個別子網路和安全群組” 將 Pod 放入多個子網路中，或將其與節點流量完全隔離。
- 預熱 IP、預熱前綴和預熱 ENI 組態。

- 最小 IP 目標組態。
- 開放原始碼 AWS VPC CNI 支援的其他組態。
- 網路政策組態，如連線追蹤計時器自訂 (預設為 300 秒)。
- 將網路事件日誌匯出至 CloudWatch。
- 替代 CNI 外掛程式（例如 Cilium 或 Calico) 和替代網路政策外掛程式。Amazon EKS Auto Mode 使用前述的內建聯網功能。如需詳細資訊，請參閱 Amazon EKS Auto Mode 的考量事項。

網路資源管理

EKS 自動模式透過監控聯網組態的 NodeClass 資源來處理字首、IP 定址和聯網介面管理。該服務會自動執行多項關鍵操作：

字首委派

EKS Auto Mode 預設為使用字首委派 (/28 個字首） 進行 Pod 聯網，並維護預先定義的 IP 資源暖集區，該集區會根據排程的 Pod 數量進行擴展。偵測到 Pod 子網路分段時，自動模式會佈建次要 IP 地址 (/32)。由於此預設 Pod 網路演算法，自動模式會根據每個執行個體類型支援的 ENIs 和 IPs 數目（假設片段的最壞情況），計算每個節點的最大 Pod。如需每個執行個體類型的最大 ENIs IPs 的詳細資訊，請參閱 EC2 使用者指南中的每個網路界面的最大 IP 地址。較新一代 (Nitro v6 及更高版本） 執行個體系列通常會增加每個執行個體類型的 ENIs 和 IPs，而自動模式會相應地調整最大 Pod 計算。

對於 IPv6 叢集，只會使用字首委派。自動模式會使用低於 110 的計算值，以每個節點 110 的上限套用相同的 IPv4-based計算。

冷卻時間管理

該服務為不再使用的字首或次要 IPv4 位址實作冷卻集區。在冷卻時間到期後，這些資源將被釋出回 VPC。但是，如果 Pod 在冷卻時間內重複使用這些資源，則會從冷卻集區還原這些資源。

IPv6 支援

對於 IPv6 叢集，EKS 自動模式會在主要網路介面上為每個節點佈建 `/80` IPv6 前綴。使用時`podSubnetSelectorTerms`，字首會改為配置在 Pod 子網路中的次要網路界面上。

該服務還確保所有網路介面的正確管理和垃圾回收。

Load balancing

您可以在服務和輸入資源上使用註釋來設定 EKS Auto Mode 佈建的 AWS Elastic Load Balancer。

如需詳細資訊，請參閱 the section called “建立傳入類別” 或 the section called “建立服務” 。

使用 EKS 自動模式進行負載平衡的考量事項

- 預設目標模式是 IP 模式，而非執行個體模式。
- EKS 自動模式僅支援 Network Load Balancer 的安全群組模式。
- AWS 不支援將負載平衡器從自我管理 AWS 負載平衡器控制器遷移至 EKS Auto Mode 管理。
- 不支援 `TargetGroupBinding` 規格中的 `networking.ingress.ipBlock` 欄位。
- 如果您的工作節點使用自訂安全群組 (而非 `eks-cluster-sg-` 命名模式)，則您的叢集角色需要額外的 IAM 許可。預設的 EKS 受管政策僅允許 EKS 修改名為 `eks-cluster-sg-` 的安全群組。如果沒有修改您自訂安全群組的許可，EKS 將無法新增所需的輸入規則，以允許 ALB/NLB 流量到達您的 Pod。

CoreDNS 考量事項

EKS Auto Mode 不會使用傳統 CoreDNS 部署在叢集內提供 DNS 解析。反之，自動模式節點會利用 CoreDNS，直接在每個節點上執行做為系統服務。如果將傳統叢集轉換為自動模式，您可以將工作負載移至自動模式節點後，從叢集中移除 CoreDNS 部署。

> **Important**
>
> 如果您計劃維護具有自動模式和非自動模式節點的叢集，則必須保留 CoreDNS 部署。非自動模式節點依賴傳統的 CoreDNS Pod 進行 DNS 解析，因為它們無法存取自動模式提供的節點層級 DNS 服務。

EKS Auto 模式的可觀測性

使用本章來了解 Amazon EKS Auto Mode 叢集的可觀測性選項。

主題

- EKS Auto 的存取 AWS受管元件日誌

EKS Auto 的存取 AWS受管元件日誌

您可以從 AWS EKS Auto Mode 存取受管元件日誌，以更深入地觀察叢集操作。EKS Auto Mode 支援下列來源的日誌：

- 運算自動擴展 - Karpenter
- 區塊儲存 - EBS CSI
- 負載平衡 - AWS Load Balancer控制器
- Pod 聯網 - VPC CNI IP 地址管理

日誌可以交付到您選擇的交付目的地。

建立 EKS Auto Mode 叢集時，您可以選擇啟用控制平面記錄 (API 伺服器、稽核、驗證器、控制器管理員、排程器）。EKS Auto 受管元件日誌 （例如運算、區塊儲存、負載平衡和 IPAM) 需要透過日誌交付進行個別組態。

設定日誌交付

若要設定 EKS Auto Mode 叢集的 AWS受管元件日誌交付，請使用 Amazon CloudWatch Logs API。如需詳細設定指示，請參閱《Amazon CloudWatch Logs 使用者指南》中的啟用來自 AWS 服務的日誌記錄。每個自動模式功能都可以設定為個別的 CloudWatch Vended Logs 交付來源，讓您選取要存取的日誌。

EKS Auto Mode 支援下列日誌類型：

- AUTO_MODE_COMPUTE_LOGS
- AUTO_MODE_BLOCK_STORAGE_LOGS
- AUTO_MODE_LOAD_BALANCING_LOGS
- AUTO_MODE_IPAM_LOGS

使用 Amazon CloudWatch APIs

設定記錄需要三個步驟：

1. 使用 CloudWatch PutDeliverySource API 建立 功能的交付來源
2. 使用 PutDeliveryDestination 建立交付目的地
3. 使用 CreateDelivery 建立交付以連接來源和目的地

您可以使用 CloudWatch PutDeliveryDestination API 中的 deliveryDestinationConfiguration 物件，為自動模式日誌設定目的地的詳細資訊。它接受 CloudWatch 日誌群組、S3 儲存貯體或 Amazon Data Firehose 交付串流的 ARN。

您可以設定單一自動模式功能（交付來源），透過建立多個交付將日誌傳送至多個目的地。您也可以建立多個交付，以設定多個交付來源，將日誌傳送至相同的交付目的地。

IAM 許可

根據選取的目的地，您可能需要為 CloudWatch 日誌群組、S3 儲存貯體和 Amazon Data Firehose 設定 IAM 政策或角色，以確保日誌成功交付。此外，如果您跨 AWS 帳戶傳送日誌，則需要使用 PutDeliveryDestinationPolicy API 來設定允許交付至目的地的 IAM 政策。如需其他資訊，請參閱 CloudWatch Vended Logs 許可文件。

檢視您的日誌

設定日誌交付後，日誌將交付至您指定的目的地。存取日誌的方法取決於您選擇的目的地類型：

- CloudWatch Logs - 在 CloudWatch Logs 主控台中檢視日誌、使用 AWS CLI 命令，或搭配 CloudWatch Logs Insights 進行查詢
- Amazon S3 - 透過 S3 主控台、 AWS CLI 或 Amazon Athena 等分析工具，將日誌作為 S3 儲存貯體中的物件存取
- Amazon Data Firehose - 日誌會串流到您設定的 Firehose 目標（例如 S3、OpenSearch Service、Redshift 等）

定價

根據您選擇的交付目的地，CloudWatch Vended Logs 會收取日誌交付和儲存的費用。與標準 CloudWatch Logs 相比，CloudWatch Vended Logs 能以較低的價格使用內建 AWS 身分驗證和授權來實現可靠、安全的日誌交付。如需詳細資訊，請參閱 CloudWatch 定價頁面的已修訂日誌一節。

相關資源

- Amazon EKS 控制平面記錄
- CloudWatch Logs API 參考中的 PutDeliverySource API
- CloudWatch Logs API 參考中的 PutDeliveryDestination API
- CloudWatch Logs API 參考中的 CreateDelivery API

EKS 自動模式疑難排解

使用 EKS Auto Mode， 會 AWS 對您 AWS 帳戶中的 EC2 執行個體承擔更多責任。EKS 負責節點上的容器執行時期、節點上的作業系統，以及某些控制器。這包括區塊儲存控制器、負載平衡控制器及運算控制器。

您必須使用 AWS 和 Kubernetes APIs 對節點進行故障診斷。您可以：

- 使用 Kubernetes `NodeDiagnostic` 資源，透過 the section called “節點監控代理程式” 來擷取節點日誌。如需更多步驟，請參閱 the section called “取得節點日誌”。
- 使用 Kubernetes `NodeDiagnostic` 資源擷取節點上的網路流量。如需更多步驟，請參閱 the section called “擷取網路流量”。
- 使用 AWS EC2 CLI 命令從節點`get-console-output`擷取主控台輸出。如需更多步驟，請參閱 the section called “使用 EC2 CLI 從 AWS EC2 受管執行個體取得主控台輸出”。
- 使用 Kubernetes 偵錯容器來擷取節點日誌。如需更多步驟，請參閱 the section called “使用偵錯容器和 `kubectl` CLI 取得節點日誌”。

> **Note**
>
> EKS 自動模式使用 EC2 受管執行個體。您無法直接存取 EC2 受管執行個體，包括透過 SSH。

您可能會遇到以下特定於 EKS 自動模式元件的問題及解決方案：

- Pod 卡在 `Pending` 狀態，無法排程到自動模式節點上。解決方案請參閱 the section called “對 Pod 無法排程到自動模式節點進行疑難排解”。
- EC2 受管執行個體無法以 Kubernetes 節點身分加入叢集。解決方案請參閱 the section called “對未加入叢集的節點進行疑難排解”。
- 使用 EKS 自動模式中包含的控制器的 `NodePools`、`PersistentVolumes` 和 `Services` 出現錯誤和問題。解決方案請參閱 the section called “對自動模式中包含的控制器進行疑難排解”。
- 增強的 Pod 安全性阻止了跨 Pod 共用磁碟區。解決方案請參閱 the section called “跨 Pod 共用磁碟區”。

您可使用以下方法對 EKS 自動模式元件進行疑難排解：

- the section called “使用 EC2 CLI 從 AWS EC2 受管執行個體取得主控台輸出”
- the section called “使用偵錯容器和 kubectl CLI 取得節點日誌”
- the section called “在 AWS 主控台中檢視與 EKS Auto Mode 相關聯的資源”
- the section called “檢視您 AWS 帳戶中的 IAM 錯誤”
- the section called “使用 VPC Reachability Analyzer 檢測節點連線問題”

節點監控代理程式

EKS 自動模式包含 Amazon EKS 節點監控代理程式。您可使用此代理程式來檢視有關節點的疑難排解和偵錯資訊。節點監控代理程式會發布 Kubernetes events 與節點 conditions 狀況。如需詳細資訊，請參閱the section called “節點運作狀態和修復”。

使用 EC2 CLI 從 AWS EC2 受管執行個體取得主控台輸出

此程序有助於對開機時或核心層級的問題進行疑難排解。

首先，您需要確定與您工作負載關聯的執行個體的 EC2 執行個體 ID。其次，使用 AWS CLI 擷取主控台輸出。

1. 確認您已安裝 kubectl 並連接到您的叢集
2. (選用) 使用 Kubernetes 部署的名稱來列出關聯的 Pod。

```
kubectl get pods -l app=<deployment-name>
```

3. 使用 Kubernetes Pod 的名稱來確定關聯節點的 EC2 執行個體 ID。

```
kubectl get pod <pod-name> -o wide
```

4. 使用 EC2 執行個體 ID 來擷取主控台輸出。

```
aws ec2 get-console-output --instance-id <instance id> --latest --output text
```

使用偵錯容器和 **kubectl** CLI 取得節點日誌

從 EKS Auto Mode 節點擷取日誌的建議方法是使用 NodeDiagnostic 資源。相關步驟請參閱 the section called “取得節點日誌”。

但是，您可以使用 kubectl debug node 命令從執行個體即時串流日誌。此命令會在您要偵錯的節點上啟動新的 Pod，然後您可以互動式地使用。

1. 啟動偵錯容器。以下命令使用 i-01234567890123456 作為節點的執行個體 ID，-it 分配一個 tty 並附加 stdin 以供互動式使用，並使用 kubeconfig 檔案中的 sysadmin 設定檔。

```
kubectl debug node/i-01234567890123456 -it --profile=sysadmin --image=public.ecr.aws/
amazonlinux/amazonlinux:2023
```

   範例輸出如下。

```
Creating debugging pod node-debugger-i-01234567890123456-nxb9c with container
 debugger on node i-01234567890123456.
If you don't see a command prompt, try pressing enter.
bash-5.2#
```

2. 從 shell 中，您現在可以安裝 util-linux-core，它提供了 nsenter 命令。使用 nsenter 進入主機上 PID 1 (init) 的掛載命名空間，並執行 journalctl 命令來從 kubelet 串流記錄：

```
yum install -y util-linux-core
nsenter -t 1 -m journalctl -f -u kubelet
```

為了安全起見，Amazon Linux 容器映像預設不安裝許多二進位檔。您可以使用 yum whatprovides 命令來識別必須安裝哪個套件才能提供給定的二進位檔。

```
yum whatprovides ps
```

```
Last metadata expiration check: 0:03:36 ago on Thu Jan 16 14:49:17 2025.
procps-ng-3.3.17-1.amzn2023.0.2.x86_64 : System and process monitoring utilities
Repo        : @System
Matched from:
Filename    : /usr/bin/ps
Provide    : /bin/ps

procps-ng-3.3.17-1.amzn2023.0.2.x86_64 : System and process monitoring utilities
Repo        : amazonlinux
Matched from:
Filename    : /usr/bin/ps
Provide    : /bin/ps
```

在 AWS 主控台中檢視與 EKS Auto Mode 相關聯的資源

您可以使用 AWS 主控台來檢視與 EKS Auto Mode 叢集相關聯的資源狀態。

- EBS 磁碟區
  - 透過搜尋標籤鍵 `eks:eks-cluster-name` 來檢視 EKS 自動模式磁碟區
- 負載平衡器
  - 搜尋標籤索引鍵 `eks:eks-cluster-name` 來檢視 EKS 自動模式負載平衡器
- EC2 執行個體

  EKS Auto Mode 執行個體是 EC2 受管執行個體。從 2026 年 4 月 22 日開始，新的受管執行個體和相關聯的資源預設會從 EC2 主控台檢視和 API 清單操作（例如）隱藏`DescribeInstances`。如果您在 EC2 主控台中看不到自動模式執行個體，請檢查您的受管資源可見性設定：

  1. 前往 https://console.aws.amazon.com/ec2/ 開啟 Amazon EC2 主控台。
  2. 在導覽窗格中，選擇 Dashboard (儀表板)。
  3. 在帳戶屬性卡片的設定下，選擇受管資源可見性。
  4. 選擇管理，然後選取受管執行個體的可見性切換。
  5. 選擇儲存變更。

  啟用可見性後，您可以透過搜尋標籤索引鍵 來檢視 EKS Auto Mode 執行個體`eks:eks-cluster-name`。

  或者，您可以透過下列方式檢視自動模式節點：

  - Amazon EKS 主控台，位於叢集的運算索引標籤下。
  - `kubectl get nodes` 從設定的 Kubernetes 用戶端。
  - 依執行個體 ID 或使用 `include-managed-resources` 參數直接查詢 EC2 API。

  如需詳細資訊，請參閱《Amazon EC2 使用者指南》中的受管資源可見性設定。

檢視您 AWS 帳戶中的 IAM 錯誤

1. 導覽至 CloudTrail 主控台。
2. 從左側導覽窗格中選取事件歷史記錄。
3. 選擇事件歷史記錄表右上角的設定（齒輪）圖示，並啟用錯誤碼欄。

4. 檢閱錯誤碼欄是否有許可相關的錯誤UnauthorizedOperation，例如 AccessDenied、 和 InvalidClientTokenId。若要縮小清單範圍，請依事件來源 （例如 ec2.amazonaws.com或 eks.amazonaws.com) 和時間範圍進行篩選。

如需更精確地篩選較長時段內的錯誤碼，請查詢 CloudTrail Lake 事件資料存放區或建立 Athena 資料表 （事件歷史記錄頁面頂端的 Lake 查詢和建立 Athena 資料表選項）。

尋找與您的 EKS 叢集相關的錯誤。透過錯誤訊息來更新您的 EKS 存取項目、叢集 IAM 角色或節點 IAM 角色。您可能需要為這些角色附加具有 EKS 自動模式許可的新政策。

對 Pod 無法排程到自動模式節點進行疑難排解

如果 Pod 處於 Pending 狀態，且未排程到自動模式節點，請確認您的 Pod 或部署資訊清單是否具有 nodeSelector。如果存在 nodeSelector，請確保它使用 eks.amazonaws.com/compute-type: auto 以便在由 EKS 自動模式建立的節點上進行排程。有關 EKS 自動模式使用的節點標籤的更多資訊，請參閱 the section called “控制部署”。

對未加入叢集的節點進行疑難排解

EKS 自動模式會自動設定新的 EC2 執行個體，提供正確的資訊以加入叢集，包括叢集端點和叢集憑證認證機構 (CA)。然而，這些執行個體仍可能無法作為節點加入 EKS 叢集。執行以下命令來識別未加入叢集的執行個體：

1. 執行 kubectl get nodeclaim 來檢查 NodeClaims 是否為 Ready = False。

```
kubectl get nodeclaim
```

2. 執行 kubectl describe nodeclaim <node_claim> 並在狀態下尋找阻止節點加入叢集的任何問題。

```
kubectl describe nodeclaim <node_claim>
```

常見錯誤訊息：

`Error getting launch template configs`

如果您使用預設叢集 IAM 角色許可在 NodeClass 中設定自訂標籤，可能會收到此錯誤。請參閱 the section called “身分和存取”。

`Error creating fleet`

從 EC2 API 呼叫 `RunInstances` 時可能存在某些授權問題。Check AWS CloudTrail 是否有錯誤，請參閱 the section called “自動模式叢集 IAM 角色” 以取得所需的 IAM 許可。

使用 `VPC Reachability Analyzer` 檢測節點連線問題

> **Note**
>
> 系統會針對 VPC Reachability Analyzer 執行的每個分析向您收費。如需定價的詳細資訊，請參閱 Amazon VPC 定價。

執行個體無法加入叢集的一個原因是網路連線問題阻止其連線到 API 伺服器。要診斷此問題，您可以使用 VPC Reachability Analyzer，對無法加入叢集的節點與 API 伺服器之間的連線狀態執行分析。您需要兩條資訊：

- 無法加入叢集的節點的執行個體 ID
- Kubernetes API 伺服器端點的 IP 位址

要取得執行個體 ID，您需要在叢集上建立工作負載，以促使 EKS 自動模式啟動一個 EC2 執行個體。這也會在您的叢集中建立具有執行個體 ID 的 `NodeClaim` 物件。執行 `kubectl get nodeclaim -o yaml` 以列印叢集中的全部 `NodeClaims`。每個 `NodeClaim` 都包含執行個體 ID 作為欄位，並在 providerID 中再次出現：

```
kubectl get nodeclaim -o yaml
```

範例輸出如下。

```
    nodeName: i-01234567890123456
    providerID: aws:///us-west-2a/i-01234567890123456
```

您可以透過執行 `kubectl get endpoint kubernetes -o yaml` 來確定您的 Kubernetes API 伺服器端點。位址位於位址欄位中：

```
kubectl get endpoints kubernetes -o yaml
```

範例輸出如下。

```
apiVersion: v1
kind: Endpoints
metadata:
  name: kubernetes
  namespace: default
subsets:
- addresses:
  - ip: 10.0.143.233
  - ip: 10.0.152.17
  ports:
  - name: https
    port: 443
    protocol: TCP
```

透過這兩項資訊，您可以執行分析。首先導覽至 AWS 管理主控台中的 VPC Reachability Analyzer。

1. 選擇「建立和分析路徑」
2. 為分析提供一個名稱 (例如「Node Join Failure」)
3. 針對「來源類型」，選取「執行個體」
4. 輸入失敗節點的執行個體 ID 作為「來源」
5. 對於「路徑目的地」，選取「IP 位址」
6. 輸入 API 伺服器的其中一個 IP 位址作為「目的地位址」
7. 展開「其他封包標頭組態區段」
8. 輸入 443 的「目的地連接埠」
9. 如果尚未選取，選取「通訊協定」作為 TCP
10. 選擇「建立和分析路徑」
11. 分析可能需要幾分鐘的時間才會完成。如果分析結果指出連線失敗，則會指出故障在網路路徑中的位置，讓您可以解決問題。

跨 Pod 共用磁碟區

EKS 自動模式節點設定為在強制模式下使用 SELinux，這為在同一節點上執行的 Pod 之間提供了更多隔離。啟用 SELinux 時，大多數非特殊權限 Pod 會自動套用其自己的多類別安全 (MCS) 標籤。此 MCS 標籤對每個 Pod 都是唯一的，旨在確保一個 Pod 中的程序無法操縱任何其他 Pod 或主機上的程序。即使帶有標籤的 Pod 以根目錄執行並具有對主機檔案系統的存取權，也將無法操縱檔案、在主機上進行敏感的系統呼叫、存取容器執行時期或取得 kubelet 的私密金鑰材料。

因此，當您嘗試在 Pod 之間共用資料時可能會遇到問題。例如，具有 ReadWriteOnce 存取模式的 PersistentVolumeClaim 仍然不允許多個 Pod 同時存取該磁碟區。

要啟用 Pod 之間的這種共用，您可以使用 Pod 的 seLinuxOptions 在這些 Pod 上設定相同的 MCS 標籤。在此範例中，我們為 Pod 分配了三個類別 c123,c456,c789。這不會與節點上自動分配給 Pod 的任何類別產生衝突，因為它們只會分配兩個類別。

```
securityContext:
  seLinuxOptions:
    level: "s0:c123,c456,c789"
```

在控制平面日誌中檢視 Karpenter 事件

對於啟用了控制平面日誌的 EKS 叢集，您可透過查詢記錄來深入了解 Karpenter 的操作和決策程序。這對於對與節點佈建、擴展和終止相關的 EKS 自動模式問題進行疑難排解特別有用。要檢視 Karpenter 相關事件，請使用以下 CloudWatch Logs Insights 查詢：

```
fields @timestamp, @message
| filter @logStream like /kube-apiserver-audit/
| filter @message like 'DisruptionBlocked'
or @message like 'DisruptionLaunching'
or @message like 'DisruptionTerminating'
or @message like 'DisruptionWaitingReadiness'
or @message like 'Unconsolidatable'
or @message like 'FailedScheduling'
or @message like 'NoCompatibleInstanceTypes'
or @message like 'NodeRepairBlocked'
or @message like 'Disrupted'
or @message like 'Evicted'
or @message like 'FailedDraining'
or @message like 'TerminationGracePeriodExpiring'
or @message like 'TerminationFailed'
or @message like 'FailedConsistencyCheck'
or @message like 'InsufficientCapacityError'
or @message like 'UnregisteredTaintMissing'
or @message like 'NodeClassNotReady'
| sort @timestamp desc
```

此查詢在 kube-apiserver 稽核日誌中篩選特定的 Karpenter 相關事件。這些事件包括各種中斷狀態、排程失敗、容量問題及節點相關問題。透過分析這些日誌，您可以更好地了解：

- Karpenter 採取某些動作的原因。
- 阻止正常節點佈建、擴展或終止的任何問題。
- 與執行個體類型相關的潛在容量或相容性問題。
- 節點生命週期事件，如中斷、移出或終止。

要使用此查詢：

1. 導覽至 CloudWatch 主控台
2. 從左側導覽窗格中選取「Logs Insights」
3. 選擇您的 EKS 叢集控制平面日誌的日誌群組
4. 將此查詢貼到查詢編輯器中
5. 視需要調整時間範圍
6. 執行查詢

結果會顯示 Karpenter 相關事件的時間表，協助您疑難排解問題，並了解叢集中 EKS Auto Mode 的行為。若要檢閱特定節點上的 Karpenter 動作，您可以將下列指定執行個體 ID 的行篩選條件新增至查詢：

```
|filter @message like /i-12345678910123456/
```

> **Note**
>
> 要使用此查詢，必須在您的 EKS 叢集上啟用控制平面日誌。如果您還未這麼做，請參閱 the section called “控制平面日誌”。

對自動模式中包含的控制器進行疑難排解

如果控制器出現問題，您應該研究：

- 與該控制器關聯的資源是否格式正確且有效。
- 如果已為您的叢集正確設定 AWS IAM 和 Kubernetes RBAC 資源。如需詳細資訊，請參閱the section called “身分和存取”。

相關資源

使用這些 AWS re：Post 中的文章進行進階故障診斷步驟：

- 如何在 EKS 自動模式中對常見的擴展問題進行疑難排解？
- 如何對 Amazon EKS 自動模式中的自訂 NodePool 和 NodeClass 佈建問題進行疑難排解？
- 如何對狀態不明的 EKS 自動模式內建節點集區進行故障診斷？

檢閱 EKS 自動模式版本備註

此頁面記錄 Amazon EKS 自動模式的更新。您可定期檢查本頁，以了解有關功能、錯誤修復、已知問題及已淘汰功能的公告。

如需接收有關此特定文件頁面所有來源檔案變更的通知，您可透過 RSS 閱讀器來訂閱以下 URL：

```
https://github.com/awsdocs/amazon-eks-user-guide/commits/mainline/latest/ug/automode/auto-change.adoc.atom
```

2026 年 8 月 17 日

文件：新增透過自動建立`AWSServiceRoleForAmazonEKS`存取項目上的 `eks:managed`群組授予 EKS Auto Mode 受管控制器其他 Kubernetes RBAC 的指示。這可解除封鎖使用案例，例如授予負載平衡器控制器對特定 的存取權，`Secret`以便其可以解析 ALB 上的 OIDC 組態`Ingress`。如需詳細資訊，請參閱 the section called “將 RBAC 授予受管控制器”和 的逐步解說the section called “授予對 LBC 的秘密存取權”。

2026 年 8 月 12 日

功能：Amazon EKS Auto Mode 負載平衡器控制器現在支援從 AWS Load Balancer控制器到 v3.4 的功能。請注意，尚未支援閘道 API。

- 傳入的 JWT 驗證 – 在請求到達後端之前，透過新的傳入層級動作，在 Application Load Balancer (ALB) 接聽程式規則層級驗證 JSON Web 權杖 (JWTs)。如需詳細資訊，請參閱 Application Load Balancer 現在支援 JWT 驗證的用戶端登入資料流程。
- Network Load Balancer (NLB) 加權目標群組 – 在 NLB 服務上依權重將流量分散到多個目標群組。這支援藍綠和金絲雀部署模式，包括當至少一個其他目標群組具有非零權重時的權重為 0。如需詳細資訊，請參閱 Network Load Balancer 現在支援加權目標群組。

- TargetGroupBinding 對帳事件 – Amazon EKS 現在會在 TargetGroupBinding 對帳失敗時發出 Kubernetes 事件，讓目標註冊問題可透過 顯示，`kubectl describe`而不只是控制器日誌。
- 保留子網路順序 – `aws-load-balancer-subnets`註釋現在會遵守您指定的順序，而不是在內部重新排序子網路。
- ALB 的跨區域負載平衡 – 您現在可以在 Application Load Balancer 上明確停用跨區域負載平衡。

2026 年 8 月 5 日

功能：Amazon EKS Auto Mode 負載平衡器控制器現在支援多叢集目標群組，符合 AWS Load Balancer控制器的行為。透過此功能，您可以在多個`TargetGroupBinding`資源之間共用相同的目標群組 ARN，因此單一目標群組可以為多個 Kubernetes 叢集 （在相同的 VPC 中） 提供服務，或從其他來源接受目標。如需詳細資訊，請參閱the section called “多叢集目標群組”。

2026 年 7 月 27 日

功能：Amazon Elastic Kubernetes Service (Amazon EKS) Auto Mode 負載平衡器控制器現在支援來自 AWS Load Balancer控制器 v2.13 和 v2.14 的功能。

- Application Load Balancer (ALB) URL 重寫 – 您現在可以在請求到達後端服務之前轉換請求 URLs 和主機標頭，而無需變更應用程式。如需詳細資訊，請參閱使用 AWS Application Load Balancer 來介紹 URL 和主機標頭重寫。
- PrefixListsIDs 和 LoadBalancerName IngressClassParams – 您現在可以為 Application Load Balancer 設定安全群組字首清單和自訂負載平衡器名稱。兩者都支援為輸入註釋和 IngressClassParams (prefixListsIDs 和 loadBalancerName) 中的欄位。在 IngressClassParams 中設定時，組態會套用至 IngressClass 中的所有輸入。您不再需要個別註釋每個傳入資源。
- 適用於傳入的前端 NLB – 您現在可以將 Network Load Balancer (NLB) 放在 Application Load Balancer 前面。這結合了 NLB 靜態 IP 地址和 AWS PrivateLink 與 ALB 的 layer-7 路由功能。使用 alb.ingress.kubernetes.io/enable-frontend-nlb 註釋啟用此功能。如需詳細資訊，請參閱 Network Application Load Balancer Load Balancer 類型目標群組。
- TCP_UDP 接聽程式支援 – NLB Services 現在可以使用 TCP_UDP 接聽程式，允許同一連接埠上的 TCP 和 UDP 流量。使用 service.beta.kubernetes.io/aws-load-balancer-enable-tcp-udp-listener 註釋啟用此功能。
- Per-target-group代理通訊協定 – 您現在可以使用 service.beta.kubernetes.io/aws-load-balancer-proxy-protocol-per-target-group 註釋在個別目標群組層級設定 Proxy Protocol v2 標頭，而不是統一地將組態套用至所有目標群組。

- IngressClassParams 中的 targetType 欄位 – 您現在可以直接在 IngressClassParams 中設定預設目標類型 （ 執行個體或 IP)，無需個別註釋每個輸入資源。
- 可連線性探索子網路 – 子網路選擇不再嚴格要求 kubernetes.io/role 標籤。當標籤不存在時，控制器現在會回到以route-table-based連線能力分析。對於具有雙堆疊 ip-address-type 的負載平衡器，控制器目前不支援此備用。
- ALB 的 IPv4 IP Address Manager (IPAM) 支援 – 面向網際網路的 ALB 現在可以從 Amazon Virtual Private Cloud (Amazon VPC) IPAM 集區提取其公有 IPv4 地址，而不是從 AWS受管地址範圍。這可讓您預測允許清單的 IP 地址區塊。使用 alb.ingress.kubernetes.io/ipam-ipv4-pool-id 註釋指定集區。如需詳細資訊，請參閱使用 VPC IPAM 簡化 ALB 的公有 IP 地址指派。

功能：已新增 EKS Auto Mode NodePools 的`Balanced`合併政策。設定 會權衡運算成本節省與中斷成本，藉此`spec.disruption.consolidationPolicy: Balanced`為每個整合動作評分。它會略過中斷超過節省量的動作。如果您`WhenEmpty`今天使用 ，您可以切換到 `Balanced`以取得整合的成本節省。如果您`WhenEmptyOrUnderutilized`今天使用 ，您可以切換到 `Balanced`，以消除 Pod 中斷帶來邊際利益。 `WhenEmpty` 和 `WhenEmptyOrUnderutilized` 保持不變，而現有的 NodePools 會保留其目前的行為。如需詳細資訊，請參閱 Karpenter 文件中的 the section called “建立節點集區”和中斷。 https://karpenter.sh/docs/concepts/disruption/

2026 年 7 月 21 日

功能：新增對 NodeClass 上靜態網路介面組態的支援。您現在可以`advancedNetworking.networkInterfaces`使用 為動態和靜態容量佈建設定 Elastic Fabric Adapter (EFA) 網路介面，為分散式訓練和推論工作負載啟用 EFA 就緒節點。如需詳細資訊，請參閱the section called “靜態網路界面組態”。

2026 年 6 月 30 日

功能：EKS Auto Mode 負載平衡器控制器現在支援來自 AWS Load Balancer控制器 v2.10、v2.11 和 v2.12 的功能。

從上游 v2.12.0：

- 接聽程式規則優先順序管理 — 控制器現在可以明確設定接聽程式規則優先順序，並在多個傳入規則以相同的接聽程式為目標時解決排序衝突

從上游 v2.11.0：

- Load Balancer容量單位 (LCU) 保留 — 您現在可以在 Application Load Balancer (ALBs) 和 Network Load Balancer (NLBs) 上保留容量單位，確保具有已知流量模式的工作負載的可預測效能

從上游 v2.10.0：

- ALB Shield Advanced 保護 — ALB 資源現在可以透過 alb.ingress.kubernetes.io/shield-advanced-protection 註釋使用 AWS Shield Advanced 進行保護
- 使用您自己的自訂 TargetGroupBinding — 您現在可以參考控制器未建立的現有目標群組，以便與外部受管基礎設施整合
- UDP 支援 IPv6 叢集上的雙堆疊 NLB — IPv6 叢集上的 NLB 服務現在支援 UDP 通訊協定接聽程式
- ALB HTTP 和 HTTPS 接聽程式屬性 — 透過註釋精細控制接聽程式層級屬性 （例如路由行為、標頭修改）

受管政策的更新

AWS 已更新 AmazonEKSServiceRolePolicy 和 AmazonEKSLoadBalancingPolicy，以支援這些新功能。

使用自訂 IAM 政策的客戶所需的動作

如果您要為 EKS Auto Mode 叢集角色提供自己的自訂 IAM 政策，而不是使用受管 AWS AmazonEKSLoadBalancingPolicy，您必須確保您的政策包含上述許可。若未更新自訂政策，使用新功能時將導致存取遭拒錯誤。

若要驗證同位，請將您的自訂政策與最新版本的 AmazonEKSLoadBalancingPolicy 進行比較。

具體而言，確保您的政策包含：

- elasticloadbalancing：ModifyCapacityReservation、 elasticloadbalancing：ModifyIpPools、elasticloadbalancing：ModifyListenerAttributes 和 elasticloadbalancing：SetRulePriorities
- ec2：DescribeIpamPools 和 ec2：DescribeRouteTables
- shield：CreateProtection、Shield：DeleteProtection 和 shield：TagResource

2026 年 6 月 9 日

功能：將執行個體狀態檢查監控新增至 EKS Auto 模式。運算控制器現在會輪詢 `EC2DescribeInstanceStatus`，以偵測排定的維護事件和執行個體或系統狀態檢查失敗，並自動取代運作狀態不佳的節點。

2026 年 6 月 4 日

文件：新增在 EKS Auto 模式下控制運算成本的指引，包括整合的運作方式、封鎖的區塊，以及爆量工作負載的建議模式。如需詳細資訊，請參閱 EKS Auto 模式的成本最佳化。

2026 年 6 月 3 日

功能：新增對 EKS Auto Mode 中可中斷容量保留的支援。如需詳細資訊，請參閱the section called “控制容量保留”。

2026 年 5 月 5 日

功能：新增對 EKS Auto 模式 EC2 置放群組的支援。如需詳細資訊，請參閱the section called “節點類別規格”。

2026 年 4 月 10 日

新支援的執行個體類型：p6-b200、p6-b300、p5e、p5en、trn2、hpc8a、x8aedz、x8i。如需支援執行個體的完整清單，請參閱 the section called “受管執行個體”。

2026 年 4 月 2 日

Chore：NodeClass 試轉驗證現在會根據連結的 NodePools 使用動態選取的執行個體類型。

2026 年 2 月 2 日

功能：新增在 EKS Auto Mode IPv6 叢集中停用來自 IPv6 Pod 的 v4Egress 流量的支援。如需詳細資訊，請參閱the section called “在 IPv4IPv6 叢集中停用 IPv6 Pod 的 IPv6 輸出。”。

2025 年 12 月 19 日

功能：新增對次要 IP 模式的支援，這些模式會將次要 IP 地址而非字首佈建至 Auto 節點。模式會將一個次要 IP 維護為 MinimalIPTarget，並為不需要暖機更多次要 IP 或字首的客戶儲存 IPs 資源。如需詳細資訊，請參閱the section called “節點類別規格”及the section called “Pod 的次要 IP 模式”。

2025 年 11 月 19 日

功能：啟用了具有本機 NVMe 儲存體的 G、P 和 Trn 系列執行個體的 Seekable OCI (SOCI) 平行提取和解壓縮。SOCI 平行提取和解壓縮一律用於具有 EKS Auto 模式的這些執行個體系列，而且不需要變更組態即可啟用。如需 SOCI 的詳細資訊，請參閱啟動部落格。

2025 年 11 月 19 日

功能：新增對維護固定數量節點的靜態容量節點集區的支援。如需詳細資訊，請參閱the section called “建立靜態節點集區”。

2025 年 10 月 23 日

功能：在美國區域中具有叢集的使用者現在可以在 NodeClass 定義`spec.advancedSecurity.fips`中指定 來請求使用 FIPS 相容 AMIs。

2025 年 10 月 1 日

功能：EKS Auto Mode 現在支援將節點部署至 AWS Local Zones。如需詳細資訊，請參閱the section called “使用本機區域”。

2025 年 9 月 30 日

功能：將對 instanceProfile 的支援新增至從 `spec.role` 欄位`spec.instanceProfile`互斥的 NodeClass。

2025 年 9 月 29 日

EKS Auto Mode 目前不支援 DRA。

2025 年 9 月 10 日

例行維護：自動模式運算控制器觸發的事件，現將使用名稱 `eks-auto-mode/compute`，而非 `karpenter`。

2025 年 8 月 24 日

錯誤修復：若 VPC 使用的 DHCP 選項集包含含大寫字母的自訂網域名稱，會因產生無效主機名稱導致節點無法加入叢集。此問題已解決，含大寫字母的網域名稱現可正常運作。

2025 年 8 月 15 日

錯誤修復：在 IPv4 EKS 叢集中，Pod 身分識別代理程式現將僅在 IPv4 連結本機位址上接聽，以避免 Pod 無法連接 IPv6 位址的問題。

2025 年 8 月 6 日

功能：在 NodeClass `spec.advancedNetworking.associatePublicIPAddress` 上新增組態，可透過此設定防止 Amazon EKS 自動模式節點被指派給公有 IP 位址。

2025 年 6 月 30 日

功能：除可讀寫資料磁碟區外，自動模式 NodeClass 現將使用已設定的自訂 KMS 金鑰，對執行個體的唯讀根磁碟區進行加密。此前，自訂 KMS 金鑰僅用於加密資料磁碟區。

2025 年 6 月 20 日

功能：支援控制工作負載部署至 EC2 隨需容量保留 (ODCR)。此功能在 NodeClass 中新增選用金鑰 `capacityReservationSelectorTerms`，可讓您明確控制工作負載所使用的 ODCR。如需詳細資訊，請參閱the section called “控制容量保留”。

2025 年 6 月 13 日

功能：在 `NodeClass` 中支援獨立的 Pod 子網路。此功能新增了選用金鑰 `podSubnetSelectorTerms` 和 `podSecurityGroupSelectorTerms`，可用於設定 Pod 的子網路與安全群組。如需詳細資訊，請參閱the section called “Pod 的個別子網路和安全群組”。

2025 年 4 月 30 日

功能：在 `NodeClass` 中支援轉送網路代理。此功能新增選用金鑰 `advancedNetworking`，可用於設定您的 HTTPS 代理。如需詳細資訊，請參閱the section called “節點類別規格”。

2025 年 4 月 18 日

功能：支援透過單播 DNS 解析 .local 網域 (通常保留給多播 DNS 使用)。

2025 年 4 月 11 日

功能：在 `NodeClass` 中新增 `certificateBundles` 與 `ephemeralStorage.kmsKeyID`。如需詳細資訊，請參閱the section called “節點類別規格”。

功能：提升映像提取速度，尤其對於具本機執行個體儲存空間、可運用更快映像檔解壓縮速度的執行個體類型更為明顯。

錯誤修復：解決了競爭條件問題，該問題導致 Pod 在啟動時立即排程到節點時，有時會發生 FailedCreatePodSandBox 錯誤，以及撥號錯誤：dial tcp 127.0.0.1:50051: connect: connection refused。

2025 年 4 月 4 日

功能：將 `registryPullQPS` 從 5 增加到 25，並將 `registryBurst` 從 10 增加到 50，以減少用戶端強制執行的映像提取限流 (`Failed to pull image xyz: pull QPS exceeded`)。

2025 年 3 月 31 日

錯誤修正：修復了一個問題，即當 Core DNS Pod 在自動模式節點上運行時，來自該節點上 Pod 的 DNS 查詢會指向該 Core DNS Pod，而不是節點本機 DNS 伺服器。自動模式節點上 Pod 的 DNS 查詢現將一律指向節點本機 DNS。

2025 年 3 月 21 日

錯誤修復：即使叢集中未安裝 `kube-dns` 服務，自動模式節點現也能正確解析 `kube-dns.kube-system.svc.cluster.local`。解決 GitHub 問題 #2546。

2025 年 3 月 14 日

功能：在 IPv4 叢集中啟用 IPv6 輸出。從 IPv6 自動模式叢流出的 IPv4 流量，現在將自動轉譯為節點主要 ENI 的 v4 位址。

Amazon EKS 叢集生命週期與組態

本節提供關於 Kubernetes 叢集完整生命週期管理的深入指南，以及不同的設定方式。內容涵蓋叢集建立、監控叢集運作狀態、更新 Kubernetes 版本，以及刪除叢集。同時也包含進階主題，例如如何設定端點存取、啟用/停用 Windows 支援、設定私有叢集、實作自動擴展，以及如何在多可用區域設定中使用區域轉移來增強流量重新導向的彈性。

主題

- 建立 Amazon EKS 自動模式叢集
- 建立 Amazon EKS 叢集
- Amazon EKS 佈建控制平面
- 進階 Kubernetes 控制平面組態
- 利用叢集洞見為 Kubernetes 版本升級做好準備，並為錯誤組態進行故障診斷
- 將現有叢集更新至全新 Kubernetes 版本
- 將叢集復原至先前的 Kubernetes 版本
- 刪除叢集
- 叢集 API 伺服器端點
- 在 EKS 叢集上部署 Windows 節點
- 停用 Windows 支援
- 部署網際網路存取受到限制的私有叢集
- 設定控制平面輸出路由
- 透過 Karpenter 與 Cluster Autoscaler 擴展叢集運算資源
- 了解 Amazon EKS 中的 Amazon 應用程式復原控制器 (ARC) 區域轉移
- 啟用 EKS 區域轉移來避免可用區域受損

建立 Amazon EKS 自動模式叢集

本主題提供使用進階組態選項建立 Amazon EKS 自動模式叢集的詳細說明。內容涵蓋先決條件、聯網選項和附加元件組態。程序包括設定 IAM 角色、設定叢集設定、指定網路參數以及選取附加元件。使用者可以使用 AWS 管理主控台 或 AWS CLI 建立叢集，並提供兩種方法的step-by-step指引。

對於尋求較簡單設定程序的使用者，請參閱以下簡化叢集建立步驟：

- the section called “eksctl CLI”
- the section called “ AWS CLI”
- the section called “管理主控台”

本進階組態指指南適用於需要對其 EKS 自動模式叢集設定進行更精細控制，並熟悉 Amazon EKS 概念和要求的進階使用者。在繼續進行進階組態之前，請確保您已滿足所有先決條件，並徹底了解 EKS 自動模式叢集的聯網和 IAM 要求。

EKS 自動模式需要額外的 IAM 許可。如需詳細資訊，請參閱：

- the section called “EKS 自動模式叢集的 IAM 角色”
- the section called “身分和存取”

> **Note**
>
> 如果您想建立沒有 EKS 自動模式的叢集，請參閱 the section called “建立 叢集”。本主題涵蓋進階組態。如果您剛開始使用 EKS 自動模式，請參閱 the section called “建立叢集”。

先決條件

- 現有 VPC 和子網符合 Amazon EKS 要求。部署叢集以供生產使用之前，建議您先徹底了解 VPC 和子網要求。如果您沒有 VPC 和子網路，您可以使用 Amazon EKS provided AWS CloudFormation 範本建立它們。
- `kubectl` 命令列工具安裝在您的裝置或 AWS CloudShell 上。版本可以與您的叢集 Kubernetes 版本相同，或是為最多比該版本更舊一版或更新一版的次要版本。例如，如果您的叢集版本為 `1.29`，則可以搭配使用 `kubectl` `1.28`、`1.29` 或 `1.30` 版。若要安裝或升級 `kubectl`，請參閱 the section called “設定 kubectl 和 eksctl”。
- 在您的裝置或 AWS CloudShell 上安裝和設定的 AWS 命令列界面 (AWS CLI) 版本 `1.27.160` `2.12.3`或更新版本。若要檢查您目前的版本，請使用 `aws --version`。若要安裝最新版本，請參閱《 AWS 命令列界面使用者指南》中的使用 aws 設定安裝 和快速組態。 https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-quickstart.html#cli-configure-quickstart-config
- 具有建立修改 EKS 與 IAM 資源許可的 IAM 主體。

建立叢集 - AWS 主控台

1. 開啟 Amazon EKS 主控台。
2. 選取 Add cluster (新增叢集)，然後選取 Create (建立)。
3. 在組態選項下，請選取自訂組態。
   - 本主題涵蓋自訂組態。如需快速組態的資訊，請參閱 the section called “管理主控台”。
4. 確認使用 EKS 自動模式已啟用。
   - 本主題涵蓋使用 EKS 自動模式建立叢集。有關建立無 EKS 自動模式的叢集的更多資訊，請參閱 the section called “建立 叢集”。
5. 在 Configure cluster (設定叢集) 頁面上，輸入下列欄位：
   - Name (名稱)：叢集的名稱。名稱僅包含英數字元 (區分大小寫)、連字號和底線。必須以英數字元開頭，且長度不可超過 100 個字元。名稱在您要建立叢集 AWS 的區域和 AWS 帳戶中必須是唯一的。
   - 叢集 IAM 角色 – 選擇您建立的 Amazon EKS 叢集 IAM 角色，以允許 Kubernetes 控制平面代表您管理 AWS 資源。如果您先前尚未為 EKS 自動模式建立叢集 IAM 角色，請選取建立建議的角色按鈕，以在 IAM 主控台中建立具有所需許可的角色。
   - Kubernetes version (Kubernetes 版本) – 您的叢集使用的 Kubernetes 版本。我們建議選取最新版本，除非您需要較早版本。
   - 升級政策：您要為叢集設定的 Kubernetes 版本政策。如果您希望叢集僅在標準支援版本上執行，可以選擇標準。如果您希望叢集在某個版本的標準支援結束時進入延伸支援，可以選擇延伸。如果您選擇的 Kubernetes 版本目前處於延伸支援狀態，則無法選取標準支援作為選項。
6. 在設定叢集頁面的自動模式運算區段，輸入以下欄位：
   - 節點集 區 — 判斷您是否要使用內建節點集區。如需詳細資訊，請參閱the section called “檢閱內建的節點集區”。
   - 節點 IAM 角色：如果您啟用任何內建節點集區，則需要選取一個節點 IAM 角色。EKS 自動模式會將此角色分配給新節點。建立叢集之後，便無法變更此值。如果您先前尚未為 EKS 自動模式建立節點 IAM 角色，請選取建立建議的角色按鈕，以建立具有所需許可的角色。如需有關此角色的詳細資訊，請參閱 the section called “身分和存取”。
7. 在設定叢集頁面的叢集存取部分，輸入以下欄位：
   - 引導叢集管理員存取：叢集建立者自動成為 Kubernetes 管理員。如果您要停用此功能，請選擇不允許叢集管理員存取。
   - 叢集身分驗證模式：EKS 自動模式需要 EKS 存取項目，也就是 EKS API 身分驗證模式。您可以選擇性地透過選擇 EKS API 和 ConfigMap 來啟用 `ConfigMap` 身分驗證模式。

8. 在設定叢集頁面上，輸入以下欄位：

   - Secrets encryption (秘密加密)：(選用) 選擇使用 KMS 金鑰啟用 Kubernetes 秘密的秘密加密。您也可以在建立叢集後啟用此功能。啟用此功能之前，請確定您已熟悉 the section called “啟用秘密加密” 中的資訊。
   - ARC 區域轉移 — EKS Auto 模式不支援 ARC 區域轉移。
   - Tags (標籤) – (選用) 將任何標籤新增到您的叢集。如需詳細資訊，請參閱the section called “標記您的 資源”。

     完成此頁面後，請選擇下一步。

9. 在 Specify networking (指定網路) 頁面上，選取下列欄位的值：

   - VPC：選擇符合 Amazon EKS VPC 要求的現有 VPC 來建立您的叢集。選擇 VPC 之前，建議您先熟悉 the section called “VPC 和子網要求” 中的所有要求和考量事項。建立叢集後，您無法變更要使用的 VPC。如果沒有列出 VPC，則您需要先建立一個。如需詳細資訊，請參閱 the section called “建立 VPC”。
   - Subnets (子網路)：根據預設，前一個欄位指定的 VPC 中的所有可用子網路會預先選取。您必須選取至少兩個。

     您選擇的子網必須符合 Amazon EKS 子網要求。選擇子網路之前，建議您先熟悉 Amazon EKS VPC 和子網路要求和考量事項。

     安全群組:(選用) 指定一或多個您希望 Amazon EKS 與其建立的網路介面相關聯的安全群組。

     無論您是否選擇任何安全群組，Amazon EKS 都會建立一個安全群組，以支援您的叢集和 VPC 之間的通訊。Amazon EKS 將此安全群組及您選擇的任何群組與其建立的網路介面相關聯。如需有關 Amazon EKS 建立的叢集安全群組的詳細資訊，請參閱 the section called “安全群組要求”。您可以修改 Amazon EKS 建立的叢集安全群組中的規則。
   - 選擇叢集 IP 地址系列 – 您可以選擇 IPv4 或 IPv6。

     預設情況下，Kubernetes 會為 Pod 和服務指派 `IPv4` 位址。在決定使用 `IPv6` 系列之前，請確認您已熟悉 VPC 要求和考量事項、the section called “子網需求和注意事項”、the section called “安全群組要求” 和 the section called “IPv6” 主題中的所有考量事項和要求。如果您選擇 `IPv6` 系列，則無法像對 `IPv4` 系列那樣，指定 Kubernetes 從中分配 `IPv6` 服務位址的位址範圍。Kubernetes 從唯一的本地位址範圍指派服務位址 (`fc00::/7`)。
   - (選用) 選擇設定 Kubernetes 服務 IP 位址範圍，並指定服務 **`IPv4`** 範圍。

     指定您自己的範圍有助於防止 Kubernetes 服務與對等或連接到您的 VPC 的其他網路之間發生衝突。在 CIDR 標記法中輸入範圍。例如：`10.2.0.0/16`。

CIDR 區塊必須符合下列需求：

- 處於以下範圍之一：10.0.0.0/8、172.16.0.0/12 或 192.168.0.0/16。
- 具有最小尺寸的 /24 和最大尺寸的 /12。
- 與您的 Amazon EKS 資源的 VPC 範圍不重疊。

您只能在使用 IPv4 地址系列時指定此選項，並且只能在建立叢集時指定。如果您未指定此項，則 Kubernetes 會從 10.100.0.0/16 或 172.20.0.0/16 CIDR 區塊指派服務 IP 位址。

- 針對 Cluster endpoint access (叢集端點存取)，選取一個選項。建立叢集後，您可以變更此選項。在選取非預設選項之前，請務必熟悉這些選項及其含義。如需詳細資訊，請參閱the section called “叢集端點存取”。

完成此頁面後，請選擇下一步。

10.(選用) 在設定可觀測性頁面上，選擇要開啟的指標和控制平面日誌記錄選項。根據預設，系統會關閉每個日誌類型。

- 如需有關 Prometheus 指標選項的詳細資訊，請參閱 the section called “步驟 1：開啟 Prometheus 指標”。
- 如需控制平面日誌記錄選項的詳細資訊，請參閱 the section called “控制平面日誌”。
- 完成此頁面後，請選擇下一步。

11.在 Select add-ons (選取附加元件) 頁面上，選擇您要新增至叢集的附加元件。您可以視需要選擇任意數量的 Amazon EKS 附加元件和 AWS Marketplace 附加元件。如果您想要安裝的 AWS Marketplace 附加元件未列出，您可以選擇頁面編號以檢視其他頁面結果，或在搜尋方塊中輸入文字來搜尋可用的 AWS Marketplace 附加元件。您也可以依類別、廠商或定價模式搜尋，然後從搜尋結果中選擇附加元件。建立叢集時，您可以檢視、選取和安裝支援 EKS Pod 身分識別的任何附加元件，詳見 the section called “Pod 身分識別”。

- EKS 自動模式會自動化某些附加元件的功能。如果您計劃將 EKS 受管節點群組部署到 EKS 自動模式叢集，選取其他 Amazon EKS 附加元件並檢閱選項。您可能需要安裝 CoreDNS 和 kube-proxy 等附加元件。EKS 僅會在自我管理節點與節點群組上安裝本節中的附加元件。
- 完成此頁面後，請選擇下一步。

12.在設定選取的附加元件設定頁面上，選取您要安裝的版本。建立叢集後，您隨時皆可更新至更新版本。

對於支援 EKS Pod 身分的附加元件，您可以使用 主控台自動產生角色，其中包含專為附加元件預先填入的名稱、 AWS 受管政策和信任政策。您可以重複使用現有角色或為支援的附加元件建立新角色。有關使用主控台為支援 EKS Pod 身分識別的附加元件建立角色的步驟，請參閱 the section

called “建立附加元件AWS （主控台）”。如果附加元件不支援 EKS Pod 身分識別，則會顯示一則訊息，指示在叢集建立後使用精靈建立服務帳戶的 IAM 角色 (IRSA)。

您可以在建立叢集後更新每個附加元件的組態。如需有關設定附加元件的詳細資訊，請參閱the section called “更新附加元件”。完成此頁面後，請選擇下一步。

13.在 Review and create (檢閱並建立) 頁面上，檢閱您在先前頁面上輸入或選取的資訊。如需變更，請選擇 Edit (編輯)。當您感到滿意時，請選擇建立。在佈建叢集時，Status (狀態) 欄位顯示 CREATING (正在建立)。

> **Note**
>
> 您可能會收到錯誤，表示在請求中的其中一個可用區域沒有足夠的容量可建立 Amazon EKS 叢集。如果發生這種情況，錯誤輸出包含的可用區域可支援新的叢集。使用至少兩個位於帳戶的支援可用區域子網路來建立您的叢集。如需詳細資訊，請參閱the section called “容量不足”。

叢集佈建需要幾分鐘的時間。

建立叢集 - AWS CLI

以下 CLI 說明涵蓋建立 IAM 資源和建立叢集。

建立 EKS 自動模式叢集 IAM 角色

步驟 1：建立信任政策

建立信任政策，允許 Amazon EKS 服務擔任該角色。將政策儲存為 `trust-policy.json`：

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
```

```
      ]
    }
  ]
}
```

步驟 2：建立 IAM 角色

使用信任政策建立叢集 IAM 角色：

```
aws iam create-role \
    --role-name AmazonEKSAutoClusterRole \
    --assume-role-policy-document file://trust-policy.json
```

步驟 3：記下角色 ARN

擷取並儲存新角色的 ARN，供後續步驟使用：

```
aws iam get-role --role-name AmazonEKSAutoClusterRole --query "Role.Arn" --output text
```

步驟 4：附加所需政策

將下列 AWS 受管政策連接至叢集 IAM 角色，以授予必要的許可：

AmazonEKSClusterPolicy：

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws: iam::aws:policy/AmazonEKSClusterPolicy
```

AmazonEKSComputePolicy：

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws: iam::aws:policy/AmazonEKSComputePolicy
```

AmazonEKSBlockStoragePolicyV2：

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws: iam::aws:policy/AmazonEKSBlockStoragePolicyV2
```

AmazonEKSLoadBalancingPolicy：

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws: iam::aws:policy/AmazonEKSLoadBalancingPolicy
```

AmazonEKSNetworkingPolicy：

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws: iam::aws:policy/AmazonEKSNetworkingPolicy
```

建立 EKS 自動模式節點 IAM 角色

步驟 1：建立信任政策

建立信任政策，允許 Amazon EKS 服務擔任該角色。將政策儲存為 `node-trust-policy.json`：

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

步驟 2：建立節點 IAM 角色

使用上一步中的 node-trust-policy.json 檔案來定義哪些實體可擔任該角色。執行下列命令以建立節點 IAM 角色：

```
aws iam create-role \
    --role-name AmazonEKSAutoNodeRole \
    --assume-role-policy-document file://node-trust-policy.json
```

步驟 3：記下角色 ARN

建立角色後，擷取並儲存節點 IAM 角色的 ARN。您在後續步驟中將需要此 ARN。使用以下命令來取得 ARN：

```
aws iam get-role --role-name AmazonEKSAutoNodeRole --query "Role.Arn" --output text
```

步驟 4：附加所需政策

將下列 AWS 受管政策連接至節點 IAM 角色，以提供必要的許可：

AmazonEKSWorkerNodeMinimalPolicy：

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoNodeRole \
    --policy-arn arn:aws: iam::aws:policy/AmazonEKSWorkerNodeMinimalPolicy
```

AmazonEC2ContainerRegistryPullOnly：

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoNodeRole \
    --policy-arn arn:aws: iam::aws:policy/AmazonEC2ContainerRegistryPullOnly
```

建立叢集

1. 使用下列命令建立您的叢集。執行命令之前，請執行下列替換：
   - 將 *region-code* 取代為您要在其中建立叢集 AWS 的區域。
   - 使用您的叢集名稱取代 *my-cluster*。名稱僅包含英數字元 (區分大小寫)、連字號和底線。必須以英數字元開頭，且長度不可超過 100 個字元。名稱在您要建立叢集 AWS 的區域和 AWS 帳戶中必須是唯一的。
   - 將 *1.30* 取代為任何 Amazon EKS 支援的版本。
   - 使用您的帳戶 ID 取代 *111122223333*
   - 如果您為叢集和節點角色建立了不同名稱的 IAM 角色，請取代 ARN。
   - 以您自己的值取代 `subnetIds` 值。您也可以新增其他 ID。您必須指定至少兩個子網路 ID。

     您選擇的子網必須符合 Amazon EKS 子網要求。選擇子網路之前，建議您先熟悉 Amazon EKS VPC 和子網路要求和考量事項。
   - 如果您不想要指定安全群組 ID，請從命令中移除 `,securityGroupIds=sg-<ExampleID1>`。如果要指定一或多個安全群組 ID，請將 `securityGroupIds` 取代為您自己的值。您也可以新增其他 ID。

     無論您是否選擇任何安全群組，Amazon EKS 都會建立一個安全群組，以支援您的叢集和 VPC 之間的通訊。Amazon EKS 將此安全群組及您選擇的任何群組與其建立的網路介面相關聯。如需

有關 Amazon EKS 建立的叢集安全群組的詳細資訊，請參閱 the section called “安全群組要求”。您可以修改 Amazon EKS 建立的叢集安全群組中的規則。

```
aws eks create-cluster \
  --region region-code \
  --name my-cluster \
  --kubernetes-version 1.30 \
  --role-arn arn:aws: iam::111122223333:role/AmazonEKSAutoClusterRole \
  --resources-vpc-config '{"subnetIds": ["subnet-ExampleID1","subnet-ExampleID2"],
 "securityGroupIds": ["sg-ExampleID1"], "endpointPublicAccess": true,
 "endpointPrivateAccess": true}' \
  --compute-config '{"enabled": true, "nodeRoleArn": "arn:aws:
 iam::111122223333:role/AmazonEKSAutoNodeRole", "nodePools": ["general-purpose",
 "system"]}' \
  --kubernetes-network-config '{"elasticLoadBalancing": {"enabled": true}}' \
  --storage-config '{"blockStorage": {"enabled": true}}' \
  --access-config '{"authenticationMode": "API"}'
```

> **Note**
>
> 您可能會收到錯誤，表示在請求中的其中一個可用區域沒有足夠的容量可建立 Amazon EKS 叢集。如果發生這種情況，錯誤輸出包含的可用區域可支援新的叢集。使用至少兩個位於帳戶的支援可用區域子網路來建立您的叢集。如需詳細資訊，請參閱the section called “容量不足”。

以下是選用設定，如有需要，必須將其新增至上一個命令中。您只能在建立叢集時啟用這些選項，而不能在建立叢集之後啟用。

- 如果您想指定 Kubernetes 要從哪個 IPv4 無類別域間路由 (CIDR) 區塊中指派服務 IP 地址，您必須將 `--kubernetes-network-config serviceIpv4Cidr=<cidr-block>` 新增至下列命令來指定。

  指定您自己的範圍有助於防止 Kubernetes 服務與對等或連接到您的 VPC 的其他網路之間發生衝突。在 CIDR 標記法中輸入範圍。例如：`10.2.0.0/16`。

  CIDR 區塊必須符合下列需求：

  - 處於以下範圍之一：`10.0.0.0/8`、`172.16.0.0/12` 或 `192.168.0.0/16`。
  - 具有最小尺寸的 `/24` 和最大尺寸的 `/12`。

- 與您的 Amazon EKS 資源的 VPC 範圍不重疊。

  您只能在使用 IPv4 地址系列時指定此選項，並且只能在建立叢集時指定。如果您未指定此項，則 Kubernetes 會從 10.100.0.0/16 或 172.20.0.0/16 CIDR 區塊指派服務 IP 位址。

- 如果您正在建立叢集，並希望叢集將 IPv6 位址 (而不是 IPv4 位址) 指派給 Pod 和服務，請將 --kubernetes-network-config ipFamily=ipv6 新增至以下命令。

  預設情況下，Kubernetes 會為 Pod 和服務指派 IPv4 位址。在決定使用 IPv6 系列之前，請確認您已熟悉 VPC 要求和考量事項、the section called “子網需求和注意事項”、the section called “安全群組要求” 和 the section called “IPv6” 主題中的所有考量事項和要求。如果您選擇 IPv6 系列，則無法像對 IPv4 系列那樣，指定 Kubernetes 從中分配 IPv6 服務位址的位址範圍。Kubernetes 從唯一的本地位址範圍指派服務位址 (fc00::/7)。

2. 佈建叢集需要幾分鐘才能完成。您可以使用下列命令來查詢叢集的狀態。

```
aws eks describe-cluster --region region-code --name my-cluster --query
 "cluster.status"
```

後續步驟

- the section called “透過 kubectl 存取叢集”
- the section called “存取項目”
- 為叢集啟用密鑰加密。
- 設定叢集的日誌。
- 將節點新增至叢集。

建立 Amazon EKS 叢集

> **Note**
>
> 本主題涵蓋在沒有 EKS Auto 模式的情況下建立 Amazon EKS 叢集。
> 如需建立 EKS 自動模式叢集的詳細資訊，請參閱 the section called “建立自動叢集”。
> 要開始使用 EKS 自動模式，請參閱 the section called “建立叢集 (EKS 自動模式)”。

本主題提供可用選項的概觀，並介紹您建立 Amazon EKS 叢集時需要考慮的問題。如果您需要使用內部部署基礎結構建立叢集，作為節點的運算，請參閱 the section called “建立叢集”。如果這是您第一次建立 Amazon EKS 叢集，建議您按照 開始使用 中一個指南的步驟進行。這些指南可協助您建立一個簡單的預設叢集，而無需擴展到所有可用選項。

先決條件

- 現有 VPC 和子網符合 Amazon EKS 要求。部署叢集以供生產使用之前，建議您先徹底了解 VPC 和子網要求。如果您沒有 VPC 和子網路，您可以使用 Amazon EKS provided AWS CloudFormation 範本建立它們。
- `kubectl` 命令列工具安裝在您的裝置或 AWS CloudShell 上。版本可以與您的叢集 Kubernetes 版本相同，或是為最多比該版本更舊一版或更新一版的次要版本。若要安裝或升級 `kubectl`，請參閱 the section called “設定 kubectl 和 eksctl”。
- 在您的裝置或 AWS CloudShell 上安裝和設定的 AWS 命令列界面 (AWS CLI) 版本 `1.27.160` `2.12.3`或更新版本。若要檢查您目前的版本，請使用 `aws --version | cut -d / -f2 | cut -d ' ' -f1`。適用於 macOS 的 `yum`、 `apt-get`或 Homebrew 等套件管理員通常是最新版本 CLI AWS 後面的幾個版本。若要安裝最新版本，請參閱《 AWS 命令列界面使用者指南》中的使用 aws 設定安裝 和快速組態。 https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-quickstart.html#cli-configure-quickstart-config在 AWS CloudShell 中安裝的 AWS CLI 版本也可能是最新版本後面的幾個版本。若要更新它，請參閱《CloudShell AWS 使用者指南》中的將 CLI 安裝到您的主目錄。 AWS CloudShell
- 具有對 Amazon EKS 叢集 `create` 和 `describe` 的許可的 IAM 主體。如需詳細資訊，請參閱the section called “在 Outpost 上建立本機 Kubernetes 叢集”及the section called “所有叢集的清單或描述”。

步驟 1：建立叢集 IAM 角色

1. 如果您已擁有叢集 IAM 角色，或者您要使用 `eksctl` 來建立叢集，則可以略過此步驟。根據預設，`eksctl` 會為您建立角色。
2. 執行下列命令以建立 IAM 信任政策 JSON 檔案。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
```

```
        "Service": "eks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

3. 建立 Amazon EKS 叢集 IAM 角色。如有必要，請在 *eks-cluster-role-trust-policy.json* 前面加上您在上一步中寫入檔案的電腦的路徑。命令會將您在上一步驟中建立的信任策略與角色相關聯。若要建立 IAM 角色，必須為建立角色的 IAM 主體指派以下 `iam:CreateRole` 動作 (許可)。

```
aws iam create-role --role-name myAmazonEKSClusterRole --assume-role-policy-document
 file://"eks-cluster-role-trust-policy.json"
```

4. 您可以指派 Amazon EKS 受管政策或建立自己的自訂政策。如需了解在自訂政策中必須使用的最低許可，請參閱 the section called “叢集 IAM 角色”。

   將名為 AmazonEKSClusterPolicy 的 Amazon EKS 受管 IAM 政策連接到角色。若要將 IAM 政策連接至 IAM 主體，必須為連接政策的 IAM 實體指派以下 IAM 動作之一 (許可)：`iam:AttachUserPolicy` 或 `iam:AttachRolePolicy`。

```
aws iam attach-role-policy --policy-arn arn:aws: iam::aws:policy/
AmazonEKSClusterPolicy --role-name myAmazonEKSClusterRole
```

服務連結角色

Amazon EKS 會自動建立名為 的服務連結角色`AWSServiceRoleForAmazonEKS`。

這是叢集 IAM 角色之外的額外角色。服務連結角色是直接連結至 Amazon EKS 的一種特殊 IAM 角色類型。該角色可讓 Amazon EKS 管理您帳戶中的叢集。如需詳細資訊，請參閱the section called “叢集角色”。

您用來建立 EKS 叢集的 IAM 主體必須具有建立服務連結角色的許可。這包括 `iam:CreateServiceLinkedRole` 許可。

如果服務連結角色尚不存在，且您目前的 IAM 角色沒有足夠的許可來建立，叢集建立操作將會失敗。

步驟 2：建立叢集

您可以透過以下方式建立叢集：

- eksctl
- 的 AWS 管理主控台
- AWS CLI

建立叢集 - eksctl

1. 您需要在裝置0.215.0或 AWS CloudShell 上安裝版本 或更新版本的eksctl命令列工具。如需有關安裝或更新 eksctl 的指示，請參閱 eksctl 文件中的安裝一節。
2. 在預設 AWS 區域中使用 Amazon EKS 預設 Kubernetes 版本建立 Amazon EKS IPv4叢集。執行命令之前，請執行下列替換：
3. 將 *region-code* 取代為您要在其中建立叢集 AWS 的區域。
4. 使用您的叢集名稱取代 *my-cluster*。此名稱僅能使用英數字元 (區分大小寫) 和連字號。必須以英數字元開頭，且長度不可超過 100 個字元。名稱在您要建立叢集 AWS 的區域和 AWS 帳戶中必須是唯一的。
5. 將 *1.36* 取代為任何 Amazon EKS 支援的版本。
6. 變更 vpc-private-subnets 的值以符合您的要求。您也可以新增其他 ID。您必須指定至少兩個子網路 ID。如果您希望指定公有子網路，則可以將 --vpc-private-subnets 變更為 --vpc-public-subnets。公有子網路具有與網際網路閘道路由相關的路由表，但私有子網路沒有關聯的路由表。我們建議盡可能使用私有子網。

   您選擇的子網必須符合 Amazon EKS 子網要求。選擇子網路之前，建議您先熟悉 Amazon EKS VPC 和子網路要求和考量事項。
7. 執行以下命令：

   ```
   eksctl create cluster --name my-cluster --region region-code --version 1.36 --vpc-private-subnets subnet-ExampleID1,subnet-ExampleID2 --without-nodegroup
   ```

   叢集佈建需要幾分鐘的時間。建立叢集時，會出現幾行輸出。輸出的最後一行類似於下面的範例行。

   ```
   [#]  EKS cluster "my-cluster" in "region-code" region is ready
   ```
8. 繼續進行 the section called “步驟 3：更新 kubeconfig”

選項設定

若要查看使用 eksctl 建立叢集時可指定的大部分選項，請使用 eksctl create cluster --help 命令。若要查看所有可用的選項，您可使用 config 檔案。如需詳細資訊，請參閱 eksctl 文件中的使用組態檔與組態檔結構描述。您可以在 GitHub 上找到組態檔範例。

以下是選用設定，如有需要，必須將其新增至上一個命令中。您只能在建立叢集時啟用這些選項，而不能在建立叢集之後啟用。如果需要指定這些選項，您必須使用 eksctl 組態檔案建立叢集並指定設定，而不是使用上一個命令。

- 如果您要指定一或多個 Amazon EKS 指派給其建立的網路介面的安全群組，請指定 securityGroup 選項。

  無論您是否選擇任何安全群組，Amazon EKS 都會建立一個安全群組，以支援您的叢集和 VPC 之間的通訊。Amazon EKS 將此安全群組及您選擇的任何群組與其建立的網路介面相關聯。如需有關 Amazon EKS 建立的叢集安全群組的詳細資訊，請參閱 the section called “安全群組要求”。您可以修改 Amazon EKS 建立的叢集安全群組中的規則。
- 如果您想指定 Kubernetes 要從哪個 IPv4 無類別域間路由 (CIDR) 區塊中指派服務 IP 位址，請指定 serviceIPv4CIDR 選項。

  指定您自己的範圍有助於防止 Kubernetes 服務與對等或連接到您的 VPC 的其他網路之間發生衝突。在 CIDR 標記法中輸入範圍。例如：10.2.0.0/16。

  CIDR 區塊必須符合下列需求：

  - 處於以下範圍之一：10.0.0.0/8、172.16.0.0/12 或 192.168.0.0/16。
  - 具有最小尺寸的 /24 和最大尺寸的 /12。
  - 與您的 Amazon EKS 資源的 VPC 範圍不重疊。

    您只能在使用 IPv4 地址系列時指定此選項，並且只能在建立叢集時指定。如果您未指定此項，則 Kubernetes 會從 10.100.0.0/16 或 172.20.0.0/16 CIDR 區塊指派服務 IP 位址。對於啟用混合節點的叢集，Amazon EKS 可能會選取不同的 CIDR，以避免與設定的遠端節點和 Pod 網路發生衝突。
- 如果您要建立叢集，並希望叢集將IPv6地址指派給 Pod 和服務，而不是IPv4地址，請指定 ipFamily 選項。

  預設情況下，Kubernetes 會為 Pod 和服務指派 IPv4 位址。在決定使用 IPv6 系列之前，請確認您已熟悉 the section called “VPC 要求和注意事項”、the section called “子網需求和注意事項”、the section called “安全群組要求” 和 the section called “IPv6” 主題中的所有考量事項和要求。如果您選

擇 IPv6 系列，則無法像對 IPv4 系列那樣，指定 Kubernetes 從中分配 IPv6 服務位址的位址範圍。Kubernetes 從唯一的本地位址範圍指派服務位址 (fc00::/7)。

建立叢集 - AWS 主控台

1. 開啟 Amazon EKS 主控台。
2. 選取 Add cluster (新增叢集)，然後選取 Create (建立)。
3. 在組態選項下，請選取自訂組態
   - 如需使用 EKS Auto 模式快速建立叢集的資訊，請參閱 the section called “管理主控台”。
4. 在 EKS 自動模式下，關閉使用 EKS 自動模式。
   - 有關使用自訂組態建立 EKS 自動模式叢集的詳細資訊，請參閱 the section called “建立自動叢集”。
5. 在 Configure cluster (設定叢集) 頁面上，輸入下列欄位：
   - Name (名稱)：叢集的名稱。名稱僅包含英數字元 (區分大小寫)、連字號和底線。必須以英數字元開頭，且長度不可超過 100 個字元。名稱在您要建立叢集 AWS 的區域和 AWS 帳戶中必須是唯一的。
   - 叢集 IAM 角色 – 選擇您建立的 Amazon EKS 叢集 IAM 角色，以允許 Kubernetes 控制平面代表您管理 AWS 資源。
   - Kubernetes version (Kubernetes 版本) – 您的叢集使用的 Kubernetes 版本。我們建議選取最新版本，除非您需要較早版本。
   - 支援類型：您要為叢集設定的 Kubernetes 版本政策。如果您希望叢集僅在標準支援版本上執行，可以選擇標準支援。如果您希望叢集在某個版本的標準支援結束時進入延伸支援，可以選擇延伸支援。如果您選擇的 Kubernetes 版本目前處於延伸支援狀態，則無法選取標準支援作為選項。
   - Secrets encryption (秘密加密)：(選用) 選擇使用 KMS 金鑰啟用 Kubernetes 秘密的秘密加密。您也可以在建立叢集後啟用此功能。啟用此功能之前，請確定您已熟悉 the section called “啟用秘密加密” 中的資訊。
   - Tags (標籤) – (選用) 將任何標籤新增到您的叢集。如需詳細資訊，請參閱the section called “標記您的 資源”。
   - ARC 區域轉移 — （ 選用 ） 您可以使用 Amazon Application Recovery Controller 來緩解受損的可用區域。如需詳細資訊，請參閱the section called “了解區域轉移”。
6. 在設定叢集頁面的叢集存取部分，輸入以下欄位：
   - 引導叢集管理員存取：叢集建立者自動成為 Kubernetes 管理員。如果您要停用此功能，請選擇不允許叢集管理員存取。

- 叢集身分驗證模式：決定您想要如何授予 IAM 使用者和角色對 Kubernetes API 的存取權。如需詳細資訊，請參閱the section called “設定叢集驗證模式”。

  完成此頁面後，請選擇下一步。

7. 在 Specify networking (指定網路) 頁面上，選取下列欄位的值：

- VPC：選擇符合 Amazon EKS VPC 要求的現有 VPC 來建立您的叢集。選擇 VPC 之前，建議您先熟悉 the section called “VPC 和子網要求” 中的所有要求和考量事項。建立叢集後，您無法變更要使用的 VPC。如果沒有列出 VPC，則您需要先建立一個。如需詳細資訊，請參閱 the section called “建立 VPC”。
- Subnets (子網路)：根據預設，前一個欄位指定的 VPC 中的所有可用子網路會預先選取。您必須選取至少兩個。

  您選擇的子網必須符合 Amazon EKS 子網要求。選擇子網路之前，建議您先熟悉 Amazon EKS VPC 和子網路要求和考量事項。

  安全群組:(選用) 指定一或多個您希望 Amazon EKS 與其建立的網路介面相關聯的安全群組。

  無論您是否選擇任何安全群組，Amazon EKS 都會建立一個安全群組，以支援您的叢集和 VPC 之間的通訊。Amazon EKS 將此安全群組及您選擇的任何群組與其建立的網路介面相關聯。如需有關 Amazon EKS 建立的叢集安全群組的詳細資訊，請參閱 the section called “安全群組要求”。您可以修改 Amazon EKS 建立的叢集安全群組中的規則。
- 選擇叢集 IP 地址系列 – 您可以選擇 IPv4 或 IPv6。

  預設情況下，Kubernetes 會為 Pod 和服務指派 `IPv4` 位址。在決定使用 `IPv6` 系列之前，請確認您已熟悉 VPC 要求和考量事項、the section called “子網需求和注意事項”、the section called “安全群組要求” 和 the section called “IPv6” 主題中的所有考量事項和要求。如果您選擇 `IPv6` 系列，則無法像對 `IPv4` 系列那樣，指定 Kubernetes 從中分配 `IPv6` 服務位址的位址範圍。Kubernetes 從唯一的本地位址範圍指派服務位址 (`fc00::/7`)。
- (選用) 選擇設定 Kubernetes 服務 IP 位址範圍，並指定服務 **`IPv4`** 範圍。

  指定您自己的範圍有助於防止 Kubernetes 服務與對等或連接到您的 VPC 的其他網路之間發生衝突。在 CIDR 標記法中輸入範圍。例如：`10.2.0.0/16`。

  CIDR 區塊必須符合下列需求：

  - 處於以下範圍之一：`10.0.0.0/8`、`172.16.0.0/12` 或 `192.168.0.0/16`。
  - 具有最小尺寸的 `/24` 和最大尺寸的 `/12`。
  - 與您的 Amazon EKS 資源的 VPC 範圍不重疊。

您只能在使用 IPv4 地址系列時指定此選項，並且只能在建立叢集時指定。如果您未指定此項，則 Kubernetes 會從 10.100.0.0/16 或 172.20.0.0/16 CIDR 區塊指派服務 IP 位址。對於啟用混合節點的叢集，Amazon EKS 可能會選取不同的 CIDR，以避免與設定的遠端節點和 Pod 網路發生衝突。

- 針對 Cluster endpoint access (叢集端點存取)，選取一個選項。建立叢集後，您可以變更此選項。在選取非預設選項之前，請務必熟悉這些選項及其含義。如需詳細資訊，請參閱the section called “叢集端點存取”。
- （選用）針對控制平面輸出，選擇如何將輸出流量從控制平面路由到 VPC。預設值為 Amazon EKS 受管。如果您選擇客戶路由，您將負責控制平面到 VPC 資源的網路路徑。如需詳細資訊，請參閱the section called “控制平面輸出”。

  完成此頁面後，請選擇下一步。

8. (選用) 在設定可觀測性頁面上，選擇要開啟的指標和控制平面日誌記錄選項。根據預設，系統會關閉每個日誌類型。
   - 如需有關 Prometheus 指標選項的詳細資訊，請參閱 the section called “步驟 1：開啟 Prometheus 指標”。
   - 如需控制平面日誌記錄選項的詳細資訊，請參閱 the section called “控制平面日誌”。

   完成此頁面後，請選擇下一步。
9. 在 Select add-ons (選取附加元件) 頁面上，選擇您要新增至叢集的附加元件。某些附加元件是預先選取的。您可以視需要選擇任意數量的 Amazon EKS 附加元件和 AWS Marketplace 附加元件。如果您想要安裝的 AWS Marketplace 附加元件未列出，您可以選擇頁面編號以檢視其他頁面結果，或在搜尋方塊中輸入文字來搜尋可用的 AWS Marketplace 附加元件。您也可以依類別、廠商或定價模式搜尋，然後從搜尋結果中選擇附加元件。建立叢集時，您可以檢視、選取和安裝支援 EKS Pod 身分識別的任何附加元件，詳見 the section called “Pod 身分識別”。

   完成此頁面後，請選擇下一步。

   一些附加元件，例如 Amazon VPC CNI、CoreDNS 和 kube-proxy，是預設安裝的。如果停用任何預設附加元件，可能會影響您執行 Kubernetes 應用程式的能力。
10. 在設定選取的附加元件設定頁面上，選取您要安裝的版本。建立叢集後，您隨時皆可更新至更新版本。

    對於支援 EKS Pod 身分的附加元件，您可以使用 主控台自動產生角色，其中包含專為附加元件預先填入的名稱、 AWS 受管政策和信任政策。您可以重複使用現有角色或為支援的附加元件建立新角色。有關使用主控台為支援 EKS Pod 身分識別的附加元件建立角色的步驟，請參閱 the section

called “建立附加元件AWS （主控台）”。如果附加元件不支援 EKS Pod 身分識別，則會顯示一則訊息，指示在叢集建立後使用精靈建立服務帳戶的 IAM 角色 (IRSA)。

您可以在建立叢集後更新每個附加元件的組態。如需有關設定附加元件的詳細資訊，請參閱the section called “更新附加元件”。完成此頁面後，請選擇下一步。

11. 在 Review and create (檢閱並建立) 頁面上，檢閱您在先前頁面上輸入或選取的資訊。如需變更，請選擇 Edit (編輯)。當您感到滿意時，請選擇建立。在佈建叢集時，Status (狀態) 欄位顯示 CREATING (正在建立)。

> **Note**
>
> 您可能會收到錯誤，表示在請求中的其中一個可用區域沒有足夠的容量可建立 Amazon EKS 叢集。如果發生這種情況，錯誤輸出包含的可用區域可支援新的叢集。使用至少兩個位於帳戶的支援可用區域子網路來建立您的叢集。如需詳細資訊，請參閱the section called “容量不足”。

叢集佈建需要幾分鐘的時間。

12. 繼續進行 the section called “步驟 3：更新 kubeconfig”

建立叢集 - AWS CLI

1. 使用下列命令建立您的叢集。執行命令之前，請執行下列替換：
   - 將 *region-code* 取代為您要在其中建立叢集 AWS 的區域。
   - 使用您的叢集名稱取代 *my-cluster*。名稱僅包含英數字元 (區分大小寫)、連字號和底線。必須以英數字元開頭，且長度不可超過 100 個字元。名稱在您要建立叢集 AWS 的區域和 AWS 帳戶中必須是唯一的。
   - 將 *1.36* 取代為任何 Amazon EKS 支援的版本。
   - 使用帳戶 ID 取代 *111122223333*，並使用叢集 IAM 角色名稱取代 *myAmazonEKSClusterRole*。
   - 以您自己的值取代 `subnetIds` 值。您也可以新增其他 ID。您必須指定至少兩個子網路 ID。

     您選擇的子網必須符合 Amazon EKS 子網要求。選擇子網路之前，建議您先熟悉 Amazon EKS VPC 和子網路要求和考量事項。

- 如果您不想要指定安全群組 ID，請從命令中移除 `,securityGroupIds=sg-<ExampleID1>`。如果要指定一或多個安全群組 ID，請將 `securityGroupIds` 取代為您自己的值。您也可以新增其他 ID。

  無論您是否選擇任何安全群組，Amazon EKS 都會建立一個安全群組，以支援您的叢集和 VPC 之間的通訊。Amazon EKS 將此安全群組及您選擇的任何群組與其建立的網路介面相關聯。如需有關 Amazon EKS 建立的叢集安全群組的詳細資訊，請參閱 the section called “安全群組要求”。您可以修改 Amazon EKS 建立的叢集安全群組中的規則。

- （選用）若要自行管理從控制平面到 VPC 的輸出路徑，請將 `controlPlaneEgressMode=CUSTOMER_ROUTED`新增至 `--resources-vpc-config`。預設值為 `AWS_MANAGED`。您也可以在叢集建立後變更此設定。如需詳細資訊，請參閱the section called “控制平面輸出”。

```
aws eks create-cluster --region region-code --name my-cluster --kubernetes-version
 1.36 \
   --role-arn arn:aws: iam::111122223333:role/myAmazonEKSClusterRole \
   --resources-vpc-config subnetIds=subnet-ExampleID1,subnet-
ExampleID2,securityGroupIds=sg-ExampleID1
```

> **Note**
>
> 您可能會收到錯誤，表示在請求中的其中一個可用區域沒有足夠的容量可建立 Amazon EKS 叢集。如果發生這種情況，錯誤輸出包含的可用區域可支援新的叢集。使用至少兩個位於帳戶的支援可用區域子網路來建立您的叢集。如需詳細資訊，請參閱the section called “容量不足”。

以下是選用設定，如有需要，必須將其新增至上一個命令中。您只能在建立叢集時啟用這些選項，而不能在建立叢集之後啟用。

- 預設情況下，EKS 在叢集建立期間會安裝多個聯網附加元件。這包括 Amazon VPC CNI、CoreDNS 及 kube-proxy。

  如果您想要停用這些預設聯網附加元件的安裝，請使用以下參數。這可能用於替代 CNI，例如 Cilium。如需詳細資訊，請參閱 EKS API 參考。

  `aws eks create-cluster --no-bootstrap-self-managed-addons …`

- 如果您想指定 Kubernetes 要從哪個 IPv4 無類別域間路由 (CIDR) 區塊中指派服務 IP 地址，您必須將 `--kubernetes-network-config serviceIpv4Cidr=<cidr-block>` 新增至下列命令來指定。

  指定您自己的範圍有助於防止 Kubernetes 服務與對等或連接到您的 VPC 的其他網路之間發生衝突。在 CIDR 標記法中輸入範圍。例如：`10.2.0.0/16`。

  CIDR 區塊必須符合下列需求：

  - 處於以下範圍之一：`10.0.0.0/8`、`172.16.0.0/12` 或 `192.168.0.0/16`。
  - 具有最小尺寸的 `/24` 和最大尺寸的 `/12`。
  - 與您的 Amazon EKS 資源的 VPC 範圍不重疊。

  您只能在使用 IPv4 地址系列時指定此選項，並且只能在建立叢集時指定。如果您未指定此項，則 Kubernetes 會從 `10.100.0.0/16` 或 `172.20.0.0/16` CIDR 區塊指派服務 IP 位址。對於啟用混合節點的叢集，Amazon EKS 可能會選取不同的 CIDR，以避免與設定的遠端節點和 Pod 網路發生衝突。

- 如果您正在建立叢集，並希望叢集將 IPv6 位址 (而不是 IPv4 位址) 指派給 Pod 和服務，請將 `--kubernetes-network-config ipFamily=ipv6` 新增至以下命令。

  預設情況下，Kubernetes 會為 Pod 和服務指派 IPv4 位址。在決定使用 IPv6 系列之前，請確認您已熟悉 VPC 要求和考量事項、the section called “子網需求和注意事項”、the section called “安全群組要求” 和 the section called “IPv6” 主題中的所有考量事項和要求。如果您選擇 IPv6 系列，則無法像對 IPv4 系列那樣，指定 Kubernetes 從中分配 IPv6 服務位址的位址範圍。Kubernetes 從唯一的本地位址範圍指派服務位址 (`fc00::/7`)。

2. 佈建叢集需要幾分鐘才能完成。您可以使用下列命令來查詢叢集的狀態。

```
aws eks describe-cluster --region region-code --name my-cluster --query
 "cluster.status"
```

   在傳回的輸出為 `ACTIVE` 之前，請勿進行下一個步驟。

3. 繼續進行 the section called “步驟 3：更新 kubeconfig”

步驟 3：更新 kubeconfig

1. 如果您使用 `eksctl` 建立叢集，則可以略過此步驟。這是因為 `eksctl` 已為您完成此步驟。透過向 `kubectl config` 檔案新增內容，使 `kubectl` 能夠與您的叢集通訊。如需建立和更新檔案的詳細資訊，請參閱 the section called “透過 kubectl 存取叢集”。

   ```
   aws eks update-kubeconfig --region region-code --name my-cluster
   ```

   範例輸出如下。

   ```
   Added new context arn:aws: eks:region-code:111122223333:cluster/my-cluster to /home/
   username/.kube/config
   ```

2. 透過執行以下命令確認與叢集的通訊。

   ```
   kubectl get svc
   ```

   範例輸出如下。

   ```
   NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
   kubernetes   ClusterIP   10.100.0.1   <none>        443/TCP   28h
   ```

步驟 4：叢集設定

1. （建議）若要使用一些 Amazon EKS 附加元件，或讓個別 Kubernetes 工作負載具有特定的 AWS Identity and Access Management (IAM) 許可，請為您的叢集建立 IAM OpenID Connect (OIDC) 供應商。您只需要為叢集建立 IAM OIDC 提供商一次。若要進一步了解 Amazon EKS 附加元件，請參閱 the section called “Amazon EKS 附加元件”。若要進一步了解如何將特定 IAM 許可指派給您的工作負載，請參閱 the section called “具有 IRSA 的憑證”。
2. (建議) 將叢集設定為 Kubernetes 專用 Amazon VPC CNI 外掛程式，然後再將 Amazon EC2 節點部署到叢集。預設情況下，外掛程式已隨叢集一起安裝。將 Amazon EC2 節點新增到叢集時，外掛程式會自動部署至您新增的每個 Amazon EC2 節點。外掛程式要求您將以下 IAM 政策之一連接至 IAM 角色。如果您的叢集使用 `IPv4` 系列，可使用 AmazonEKS_CNI_Policy 受管 IAM 政策。如果您的叢集使用 `IPv6` 系列，可使用您建立的 IAM 政策。

您連接政策的 IAM 角色可以是節點 IAM 角色，也可以是僅用於外掛程式的專用角色。我們建議將政策連接至此角色。有關建立角色的更多資訊，請參閱 the section called “為 IRSA 設定” 或 Amazon EKS 節點 IAM 角色。

3. 如果您使用 部署叢集 AWS 管理主控台，則可以略過此步驟。根據預設， AWS 管理主控台 部署適用於 Kubernetes、CoreDNS 和 Amazon EKS 附加元件`kube-proxy`的 Amazon VPC CNI 外掛程式。

   如果您使用 `eksctl`或 AWS CLI 部署叢集，則會部署適用於 Kubernetes、CoreDNS 和`kube-proxy`自我管理附加元件的 Amazon VPC CNI 外掛程式。您可以將隨叢集一起部署的 Kubernetes 專用 Amazon VPC CNI 外掛程式、CoreDNS 和 `kube-proxy` 自我管理附加元件移轉到 Amazon EKS 附加元件。如需詳細資訊，請參閱the section called “Amazon EKS 附加元件”。

4. (選用) 如果您尚未執行此操作，則可啟用叢集的 Prometheus 指標。如需詳細資訊，請參閱《Amazon Managed Service for Prometheus 使用者指南》中的建立湊集器。

5. 如果您計劃將使用 Amazon EBS 磁碟區的工作負載部署到叢集，則必須在部署工作負載之前將 Amazon EBS CSI 安裝到叢集。

後續步驟

- 建立叢集的 IAM 主體是唯一可以存取叢集的 IAM 主體。將許可授予其他 IAM 主體，這樣他們就可以存取您的叢集。
- 如果建立叢集的 IAM 主體僅具有先決條件中所參考的最低 IAM 許可，那麼您可能需要為該主體新增其他 Amazon EKS 許可。如需將 Amazon EKS 許可授予 IAM 主體的詳細資訊，請參閱 the section called “IAM 參考”。
- 如果您希望建立叢集的 IAM 主體或任何其他主體在 Amazon EKS 主控台中檢視 Kubernetes 資源，請對實體授予必要許可。
- 如果您希望節點和 IAM 主體從 VPC 內存取您的叢集，請啟用叢集的私有端點。根據預設，公有端點為啟用狀態。如有需要，您可以在啟用私有端點後停用公有端點。如需詳細資訊，請參閱the section called “叢集端點存取”。
- 為叢集啟用密鑰加密。
- 設定叢集的日誌。
- 將節點新增至叢集。

Amazon EKS 佈建控制平面

概觀

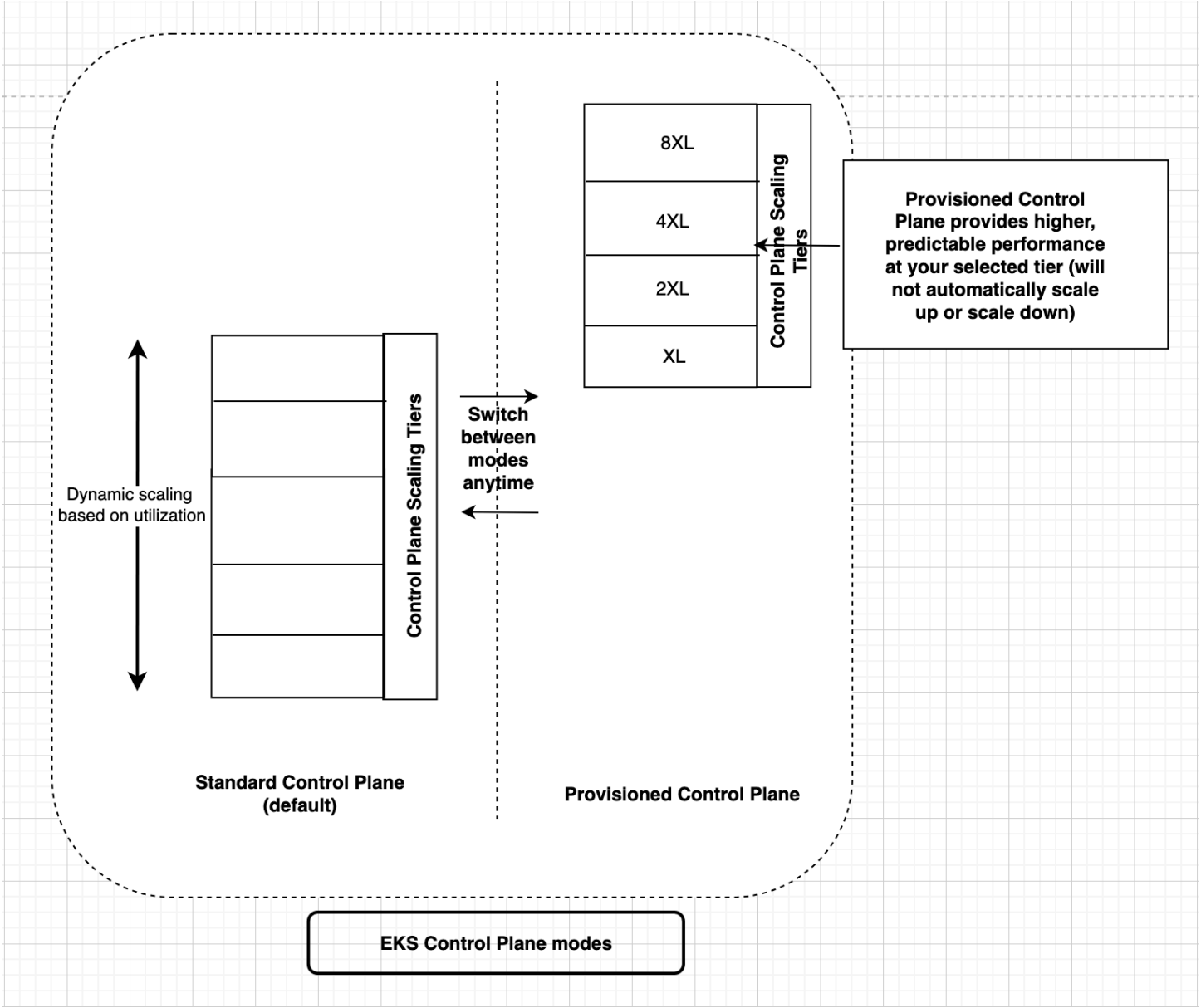
Amazon EKS 佈建控制平面是一項功能，可讓叢集管理員從一組擴展層中選取，並從叢集的控制平面指定其所選層，以獲得極高且可預測的效能。這可讓叢集管理員確保控制平面一律以指定的容量佈建。

Amazon EKS 為叢集的控制平面提供兩種操作模式。根據預設，Amazon EKS 叢集會使用標準模式，其中控制平面會根據工作負載需求自動擴展和縮減規模。標準模式可動態配置足夠的控制平面容量，以滿足工作負載需求，是大多數使用案例的建議解決方案。不過，對於因為控制平面擴展或需要非常大量的控制平面容量而導致無法容忍任何效能變化的特殊工作負載，您可以選擇使用佈建模式。佈建模式可讓您預先配置控制平面容量，隨時準備好處理嚴苛的工作負載需求。

> **Note**
>
> 佈建模式是額外的控制平面操作模式，以及預設的標準模式。佈建模式的引入不會變更標準模式行為。

使用 EKS 佈建控制平面，叢集管理員可以預先佈建所需的控制平面容量，從叢集的控制平面提供可預測且高效能，且永遠可用。EKS 佈建控制平面也可讓叢集管理員跨環境佈建相同的控制平面容量，從預備到生產和災難復原站點。這對於確保跨環境取得的控制平面效能一致且可預測非常重要。最後，EKS 佈建控制平面可讓您存取非常高層級的控制平面效能，以便在 Kubernetes 上執行大規模可擴展的 AI 工作負載、高效能運算和大規模資料處理工作負載。

根據預設，所有現有和新的 Amazon EKS 叢集都會以標準模式運作。對於需要控制平面高、可預測效能的叢集，您可以選擇使用 EKS 佈建控制平面功能。除了標準或延長支援 EKS 的每小時費用之外，還會按特定控制平面擴展方案的每小時費率向您收費。如需定價的詳細資訊，請參閱 Amazon EKS 定價。



使用案例

EKS 佈建控制平面旨在解決特定案例，其中高且可預測的控制平面效能對您的操作至關重要。了解這些使用案例可協助您判斷 EKS 佈建控制平面是否為適合您工作負載的解決方案。

效能關鍵工作負載 – 對於需要 Kubernetes 控制平面最低延遲和最高效能的工作負載，EKS 佈建控制平面提供容量，可消除控制平面擴展的效能變化。

大規模可擴展的工作負載 – 如果您執行高度可擴展的工作負載，例如 AI 訓練和推論、高效能運算，或需要大量節點在叢集中執行的大規模資料處理，佈建控制平面會提供必要的控制平面容量，以支援這些嚴苛的工作負載。

預期的高需求事件 – 當您預期由於電子商務銷售或促銷、產品推出、假日購物季節或主要運動或娛樂活動等近期事件，控制平面請求突然激增時，佈建控制平面可讓您事先擴展控制平面容量。這種主動方法可確保您的控制平面已準備好處理增加的負載，而無需等待自動擴展來回應需求。

關鍵任務高可用性 - 對於需要最高水準可用性的工作負載，佈建控制平面提供 99.99% 的服務水準協議 (SLA)，以 1 分鐘為間隔衡量。相較於標準控制平面模式的 99.95% SLA，此增強型 SLA 提供更嚴格的可用性承諾，每隔 5 分鐘測量一次。

環境一致性 – 佈建的控制平面可讓您在預備環境和生產環境中比對控制平面容量和效能，協助您在部署至生產環境之前及早識別潛在問題。透過跨環境維護相同的控制平面層，您可以確保測試結果準確反映生產行為，從而降低在推出期間與效能相關的意外風險。

災難復原和業務持續性 – 對於災難復原案例，佈建的控制平面可讓您佈建與主要環境具有相同容量層級的容錯移轉環境。這可確保在容錯移轉事件期間盡可能減少中斷和快速復原，因為您的災難復原叢集在啟動時將具有與生產叢集相同的控制平面效能特性。

控制平面擴展層

EKS 佈建控制平面提供使用 T-shirt 大小 (XL、2XL、4XL 和 8XL) 命名的擴展層。每個層會透過四個關鍵屬性來定義其功能，以決定叢集控制平面的效能特性。了解這些屬性可協助您為工作負載需求選取適當的層。

API 請求並行測量 Kubernetes 控制平面 API 伺服器可以同時處理的請求數量，這對高輸送量工作負載至關重要。

Pod 排程率指出預設 Kubernetes 排程器在節點上排程 Pod 的速度，以每秒 Pod 為單位。

叢集資料庫大小表示配置給 etcd 的儲存空間，即保留叢集狀態/中繼資料的資料庫。

SLA （服務水準協議） 指出 EKS 叢集 Kubernetes 端點的每月運作時間承諾

當您使用佈建控制平面在特定擴展層上佈建叢集的控制平面時，EKS 可確保叢集的控制平面維持與該層對應的限制。控制平面擴展層的限制因 Kubernetes 版本而異，如下表所示。

EKS 1.30 版到 1.33 版

佈建的控制平面擴展層	API 請求並行（座位）	Pod 排程速率 (Pod/秒）	叢集資料庫大小 (GB)	SLA （以 1 分鐘間隔測量）
XL	1700	167	16	99.99%

佈建的控制平面擴展層	API 請求並行（座位）	Pod 排程速率 (Pod/秒）	叢集資料庫大小 (GB)	SLA（以 1 分鐘間隔測量）
2XL	3400	283	16	99.99%
4XL	6800	400	16	99.99%
8XL	13600	400	16	99.99%

EKS 1.34 版及更新版本

佈建的控制平面擴展層	API 請求並行（座位）	Pod 排程速率 (Pod/秒）	叢集資料庫大小 (GB)	SLA（以 1 分鐘間隔測量）
XL	2000	167	16	99.99%
2XL	4000	283	16	99.99%
4XL	8000	400	16	99.99%
8XL	16000	400	16	99.99%

以程式設計方式擷取擴展層值

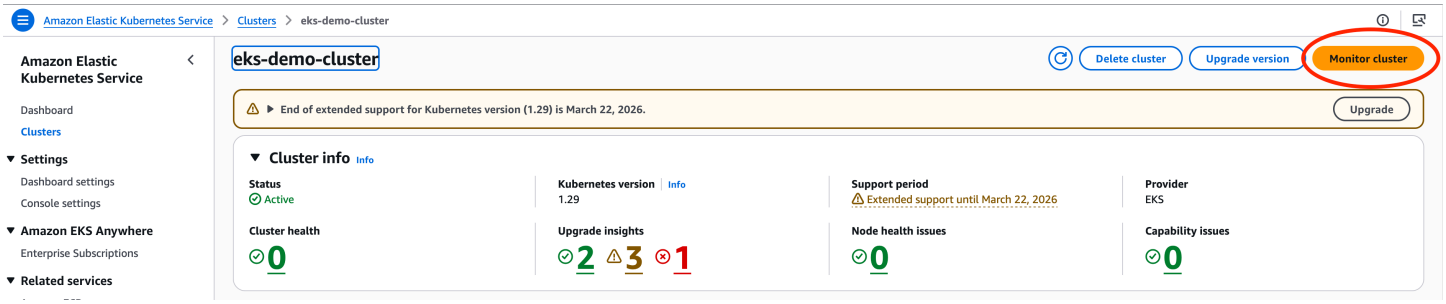
您可以使用 `DescribeClusterVersions` API 來擷取每個 Kubernetes 版本的擴展層屬性及其值。當您自動化層選取或需要以程式設計方式跨版本比較層功能時，這非常有用。如需詳細資訊，請參閱《Amazon EKS API 參考》中的 DescribeClusterVersions。

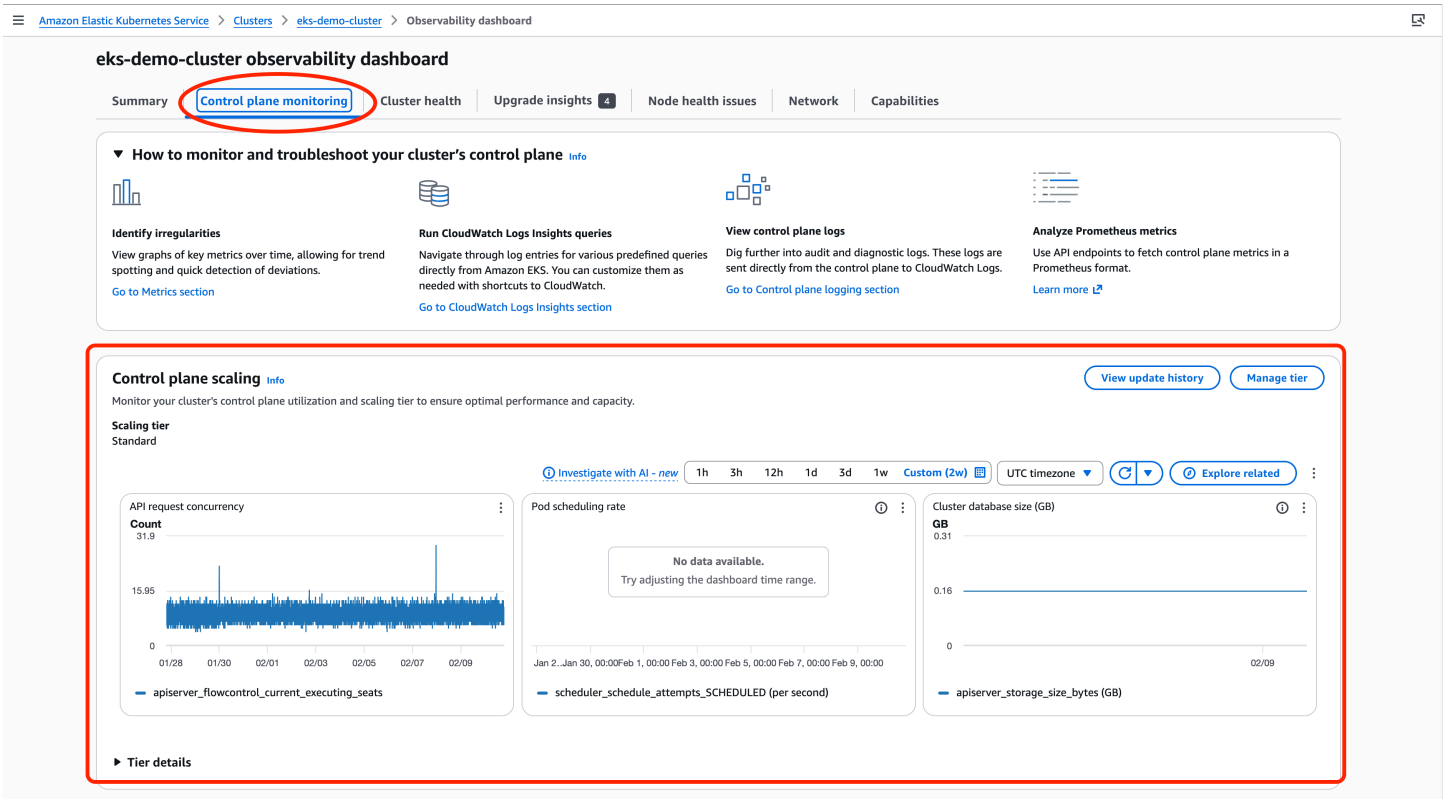
監控控制平面擴展層使用率

Amazon EKS 提供多種指標，協助您監控控制平面的層使用率。這些指標會發佈為 Amazon CloudWatch 指標，並可透過 CloudWatch 和 EKS 主控台存取。此外，這些指標可從 EKS 叢集的 Prometheus 端點抓取（請參閱此處）。

	Prometheus 指標	CloudWatch 指標
API 請求並行	apiserver_flowcontrol_current_executing_seats	apiserver_flowcontrol_current_executing_seats
Pod 排程率	scheduler_schedule_attempts_total	scheduler_schedule_attempts_total、scheduler_schedule_attempts_SCHEDULED、scheduler_schedule_attempts_UNSCHEDULABLE
叢集資料庫大小	apiserver_storage_size_bytes （注意：etd_mvcc_db_total_size_in_use_in_bytes 將在 2026 年下半年作為 prometheus 指標推出至所有 EKS 叢集）	etcd_mvcc_db_total_size_in_use_in_bytes

您可以在 Amazon EKS 主控台中檢視控制平面使用率。在叢集的概觀頁面中，選擇監控叢集以存取可觀測性儀表板，然後選取控制平面監控索引標籤以檢視控制平面擴展區段下的控制平面使用率。





了解層容量與實際效能

當您選取佈建的控制平面擴展層時，層屬性代表 Amazon EKS 套用至控制平面的基礎組態。不過，您達到的實際效能取決於您的特定工作負載模式、組態，以及對 Kubernetes 最佳實務的遵循。例如，當 4XL 層設定具有 6，800 個並行請求座位的 API Priority and Fairness (APF) 時，您從控制平面取得的實際請求輸送量取決於正在執行的操作類型。例如，Kubernetes 會對清單請求進行超過 get 的懲罰，因此控制平面同時處理的清單請求有效數量低於 get 請求。如需詳細資訊，請參閱《EKS 最佳實務指南》中的 API 優先順序和公平性。同樣地，雖然 4XL 層的預設排程器 QPS 設定為 400，但您的實際 Pod 排程速率取決於節點準備好且排程正常等因素。為了獲得最佳效能，請確保您的應用程式遵循 Kubernetes 最佳實務，並針對工作負載特性正確設定。如需詳細資訊，請參閱《EKS 最佳實務指南》中的可擴展性最佳實務。

水平 Pod Autoscaler 同步並行

除了控制平面擴展層屬性之外，EKS 佈建控制平面叢集還會使用更高的水平 Pod Autoscaler (HPA) 同步並行進行調校。HPA 同步並行是 Kubernetes 控制器管理員平行處理的 HPA 物件數量。上游 Kubernetes 預設為 5。較高的並行允許控制平面同時協調更多 HPA 物件，從而減少負載變更與對應擴展動作之間的時間。這有利於執行大量 HPA 物件的叢集，其中依序處理它們可能會延遲自動擴展。

每個佈建的控制平面擴展層都會設定下列 HPA 同步並行。

佈建的控制平面擴展層	HPA 同步並行
XL	50
2XL	100
4XL	200
8XL	200

您可以透過 HPA 控制器的 Kubernetes 控制器管理員工作佇列指標，觀察較高 HPA 同步並行的效果：

- `workqueue_depth{name="horizontalpodautoscaler"}` – HPA 控制器正在等待處理的 HPA 物件數量。隨著並行性提高，更多工作者會平行耗盡此佇列，因此深度保持低且穩定（最好在每個調校期間內達到零）。保持為零或接近零的工作佇列深度表示控制平面與叢集中的 HPA 物件保持同步，且自動調整規模決策不會因待處理項目而延遲。

您觀察到的實際自動擴展回應速度也取決於控制平面以外的因素。每個 HPA 對帳都會擷取驅動其擴展決策的指標，通常來自 CPU 和記憶體使用率的 Kubernetes 指標伺服器，或從自訂或外部指標轉接器。叢集中的 HPA 物件數量和這些指標查詢的延遲都會影響控制器處理佇列的速度。如果您的指標來源無法以足夠快的速度提供請求，無論平行處理多少次，對帳都會變慢。執行足夠的 Metrics Server 複本來有效率地提供請求磁碟區和範圍指標查詢，可協助您獲得更高 HPA 同步並行的完整優勢。如需 Horizontal Pod Autoscaler 運作方式及其擷取指標位置的詳細資訊，請參閱 Kubernetes 網站上的 Horizontal Pod Autoscaling。

考量事項

- 標準控制平面容量 – EKS 標準控制平面模式提供最佳的價格與效能比，是絕大多數使用案例的建議選項。不過，對於因為控制平面擴展或需要非常大量控制平面容量而導致無法容忍任何效能變化的特殊工作負載，您可以選擇性地考慮使用佈建模式。
- 需要選擇加入 – 現有的叢集不會自動從標準控制平面擴展到價格較高的 EKS 佈建控制平面層。您必須明確選擇加入其中一個新的 EKS 佈建控制平面擴展層。
- 結束限制 – 標準控制平面模式支援高達 8 GB 的叢集資料庫（等）大小。如果叢集的資料庫大小在使用佈建模式時超過 8 GB，則在將資料庫大小降至低於 8 GB 之前，您無法切換回標準模式。例如，如果您在佈建模式下使用 14 GB 的資料庫儲存，您必須先將資料庫使用率降低到小於 8GB，才能返回標準模式。

- 無自動層擴展 – EKS 佈建控制平面不會在層之間自動擴展。選取擴展層後，叢集的控制平面仍會固定到該層，以確保一致且可預測的效能。不過，您可以透過監控層使用率指標和使用 EKS 佈建控制平面 APIs 來彈性實作自己的自動擴展解決方案，以便在這些指標超過您定義的閾值時擴展或縮減規模，讓您完全控制擴展策略和成本最佳化。
- 檢視目前層 – 您可以使用 Amazon EKS 主控台、Amazon Web Services CLI 或 API 來檢視目前的控制平面擴展層。在 CLI 中，您可以執行 `describe-cluster`命令： `aws eks describe-cluster --name cluster-name`
- 層轉換時間 – 您可以使用 Amazon EKS 主控台、Amazon EKS APIs 或 CLI 在擴展層之間結束或移動。Amazon EKS 已推出名為 的新叢集更新類型`ScalingTierConfigUpdate`，您可以檢查以監控轉換的進度。執行層變更命令後，您可以列出叢集上的更新，以查看狀態`ScalingTierConfigUpdate`為 的 類型新更新`Updating`。狀態會在更新完成`Successful`時變更為 ，或在發生錯誤`Failed`時變更為 。更新中的錯誤欄位指出失敗的原因。您在層之間切換的頻率沒有限制。變更控制平面層需要幾分鐘的時間才能完成。在此過程中不會停機 API 伺服器，因為 EKS 會在終止舊的 API 伺服器之前啟動新的 API 伺服器。
- 選取最佳方案 – 若要判斷叢集的最佳佈建控制平面擴展方案，您可以透過在最高方案 (8XL) 上佈建叢集來執行負載測試。然後執行負載測試來模擬叢集控制平面上的尖峰需求。在尖峰負載時觀察控制平面層使用率指標，並使用這些觀察做為引導因素，為佈建模式選取適當的層。
- 佈建控制平面定價 – 將按您叢集所在的佈建控制平面擴展層的每小時費率向您收費。這是標準或延長支援每小時費用以外的費用。如需詳細資訊，請參閱 Amazon EKS 定價頁面。
- 較大的擴展層 – 如果您想要在大於 8XL 的擴展層上執行叢集，請聯絡您的 Amazon Web Services 客戶團隊以取得其他定價資訊。
- Kubernetes 版本和區域支援 – 所有 Amazon Web Services 商業、 GovCloud 和中國區域都支援 EKS 佈建控制平面。佈建的控制平面適用於 EKS v1.28 及更高版本。
- 較高的 SLA - EKS 佈建控制平面提供的 EKS SLA 高於標準控制平面。在 EKS SLA 頁面中進一步了解。

Amazon EKS 佈建控制平面入門

本指南會逐步解說使用 AWS CLI 和 設定和使用 EKS 佈建控制平面的步驟 AWS 管理主控台。

先決條件

開始前，請確保您具備以下條件：

- AWS CLI – 用於使用 AWS 服務的命令列工具，包括 Amazon EKS。如需詳細資訊，請參閱《AWS 命令列界面使用者指南》中的安裝。安裝 CLI AWS 之後，建議您也進行設定。如需詳細資

訊，請參閱《 AWS 命令列界面使用者指南》中的具有 aws 設定的快速組態。請注意，使用此頁面中顯示的 update-kubeconfig 選項需要 AWS CLI v2。

- 所需的 IAM 許可：您使用的 IAM 安全主體必須具有使用 Amazon EKS IAM 角色、服務連結角色、AWS CloudFormation、VPC 和相關資源的許可。如需詳細資訊，請參閱《IAM 使用者指南》中的動作和使用服務連結角色。您必須以同一位使用者的身分完成本指南中的所有步驟。若要檢查目前使用者，請執行以下命令：

```
aws sts get-caller-identity
```

> **Note**
>
> 建議您在 Bash shell 中完成本主題中的步驟。如果您不使用 Bash shell，則某些指令碼命令 (如行接續字元以及設定和使用變數的方式) 需要針對 shell 進行調整。此外，您的 Shell 的引用及轉義規則可能會有所不同。如需詳細資訊，請參閱《 AWS 命令列界面使用者指南》中的使用引號搭配 AWS CLI 中的字串。

EKS 佈建控制平面 - AWS CLI

使用 EKS 佈建控制平面擴展層建立叢集

```
aws eks create-cluster --name prod-cluster \
--role-arn arn:aws:iam::012345678910:role/eks-service-role-AWSServiceRoleForAmazonEKS-
J7ONKE3BQ4PI \
--resources-vpc-config subnetIds=subnet-6782e71e,subnet-
e7e761ac,securityGroupIds=sg-6979fe18 \
--control-plane-scaling-config tier=tier-xl
```

回應：

```
{
    "cluster": {
        "name": "my-eks-cluster",
        "arn": "arn:aws:eks:us-east-2:111122223333:cluster/my-eks-cluster",
        "createdAt": "2024-03-14T11:31:44.348000-04:00",
        "version": "1.26",
        "endpoint": "https://JSA79429HJDASKJDJ8223829MNDNASW.yl4.us-
east-2.eks.amazonaws.com",
```

```
        "roleArn": "arn:aws:iam::111122223333:role/eksctl-my-eks-cluster-cluster-
ServiceRole-zMF6CBakwwbW",
        "resourcesVpcConfig": {
            "subnetIds": [
                "subnet-0fb75d2d8401716e7",
                "subnet-02184492f67a3d0f9",
                "subnet-04098063527aab776",
                "subnet-0e2907431c9988b72",
                "subnet-04ad87f71c6e5ab4d",
                "subnet-09d912bb63ef21b9a"
            ],
            "securityGroupIds": [
                "sg-0c1327f6270afbb36"
            ],
            "clusterSecurityGroupId": "sg-01c84d09d70f39a7f",
            "vpcId": "vpc-0012b8e1cc0abb17d",
            "endpointPublicAccess": true,
            "endpointPrivateAccess": true,
            "publicAccessCidrs": [
                "22.19.18.2/32"
            ]
        },
        "controlPlaneScalingConfig": {
            "tier": "tier-xl"
        },
        "kubernetesNetworkConfig": {
            "serviceIpv4Cidr": "10.100.0.0/16",
            "ipFamily": "ipv4"
        },
        "logging": {
            "clusterLogging": [
                {
                    "types": [
                        "api",
                        "audit",
                        "authenticator",
                        "controllerManager",
                        "scheduler"
                    ],
                    "enabled": true
                }
            ]
        },
        "identity": {
```

```
            "oidc": {
                "issuer": "https://oidc.eks.us-east-2.amazonaws.com/id/
JSA79429HJDASKJDJ8223829MNDNASW"
            }
        },
        "status": "CREATING",
        "certificateAuthority": {
            "data": "CA_DATA_STRING..."
        },
        "platformVersion": "eks.14",
        "tags": {
            "aws:cloudformation:stack-name": "eksctl-my-eks-cluster-cluster",
            "alpha.eksctl.io/cluster-name": "my-eks-cluster",
            "karpenter.sh/discovery": "my-eks-cluster",
            "aws:cloudformation:stack-id": "arn:aws:cloudformation:us-
east-2:111122223333:stack/eksctl-my-eks-cluster-cluster/e752ea00-e217-11ee-
beae-0a9599c8c7ed",
            "auto-delete": "no",
            "eksctl.cluster.k8s.io/v1alpha1/cluster-name": "my-eks-cluster",
            "EKS-Cluster-Name": "my-eks-cluster",
            "alpha.eksctl.io/cluster-oidc-enabled": "true",
            "aws:cloudformation:logical-id": "ControlPlane",
            "alpha.eksctl.io/eksctl-version": "0.173.0-dev
+a7ee89342.2024-03-01T03:40:57Z",
            "Name": "eksctl-my-eks-cluster-cluster/ControlPlane"
        },
        "health": {
            "issues": []
        },
        "accessConfig": {
            "authenticationMode": "API_AND_CONFIG_MAP"
        }
    }
}
```

檢視叢集的控制平面擴展層

```
aws eks describe-cluster --name prod-cluster
```

回應：

```
{
    "cluster": {
```

```
        "name": "my-eks-cluster",
        "arn": "arn:aws:eks:us-east-2:111122223333:cluster/my-eks-cluster",
        "createdAt": "2024-03-14T11:31:44.348000-04:00",
        "version": "1.26",
        "endpoint": "https://JSA79429HJDASKJDJ8223829MNDNASW.yl4.us-
east-2.eks.amazonaws.com",
        "roleArn": "arn:aws:iam::111122223333:role/eksctl-my-eks-cluster-cluster-
ServiceRole-zMF6CBakwwbW",
        "resourcesVpcConfig": {
            "subnetIds": [
                "subnet-0fb75d2d8401716e7",
                "subnet-02184492f67a3d0f9",
                "subnet-04098063527aab776",
                "subnet-0e2907431c9988b72",
                "subnet-04ad87f71c6e5ab4d",
                "subnet-09d912bb63ef21b9a"
            ],
            "securityGroupIds": [
                "sg-0c1327f6270afbb36"
            ],
            "clusterSecurityGroupId": "sg-01c84d09d70f39a7f",
            "vpcId": "vpc-0012b8e1cc0abb17d",
            "endpointPublicAccess": true,
            "endpointPrivateAccess": true,
            "publicAccessCidrs": [
                "22.19.18.2/32"
            ]
        },
        "controlPlaneScalingConfig": {
            "tier": "tier-xl"
        },
        "kubernetesNetworkConfig": {
            "serviceIpv4Cidr": "10.100.0.0/16",
            "ipFamily": "ipv4"
        },
        "logging": {
            "clusterLogging": [
                {
                    "types": [
                        "api",
                        "audit",
                        "authenticator",
                        "controllerManager",
                        "scheduler"
```

```
                    ],
                    "enabled": true
                }
            ]
        },
        "identity": {
            "oidc": {
                "issuer": "https://oidc.eks.us-east-2.amazonaws.com/id/
JSA79429HJDASKJDJ8223829MNDNASW"
            }
        },
        "status": "ACTIVE",
        "certificateAuthority": {
            "data": "CA_DATA_STRING..."
        },
        "platformVersion": "eks.14",
        "tags": {
            "aws:cloudformation:stack-name": "eksctl-my-eks-cluster-cluster",
            "alpha.eksctl.io/cluster-name": "my-eks-cluster",
            "karpenter.sh/discovery": "my-eks-cluster",
            "aws:cloudformation:stack-id": "arn:aws:cloudformation:us-
east-2:111122223333:stack/eksctl-my-eks-cluster-cluster/e752ea00-e217-11ee-
beae-0a9599c8c7ed",
            "auto-delete": "no",
            "eksctl.cluster.k8s.io/v1alpha1/cluster-name": "my-eks-cluster",
            "EKS-Cluster-Name": "my-eks-cluster",
            "alpha.eksctl.io/cluster-oidc-enabled": "true",
            "aws:cloudformation:logical-id": "ControlPlane",
            "alpha.eksctl.io/eksctl-version": "0.173.0-dev
+a7ee89342.2024-03-01T03:40:57Z",
            "Name": "eksctl-my-eks-cluster-cluster/ControlPlane"
        },
        "health": {
            "issues": []
        },
        "accessConfig": {
            "authenticationMode": "API_AND_CONFIG_MAP"
        }
    }
}
```

更新叢集以使用 EKS 佈建控制平面

```
aws eks update-cluster-config --name prod-cluster \
--control-plane-scaling-config tier=tier-2xl
```

回應：

```
{
    "update": {
        "id": "7551c64b-1d27-4b1e-9f8e-c45f056eb6fd",
        "status": "InProgress",
        "type": "ScalingTierConfigUpdate",
        "params": [
            {
                "type": "UpdatedTier",
                "value": "tier-2xl"
            },
            {
                "type": "PreviousTier",
                "value": "tier-xl"
            }
        ],
        "createdAt": 1565807210.37,
        "errors": []
    }
}
```

檢視控制平面擴展更新

```
aws eks list-updates --name example
```

回應：

```
{
    "updateIds": [
        "7551c64b-1d27-4b1e-9f8e-c45f056eb6fd1"
    ]
}
```

將佈建的控制平面退出至標準控制平面

```
aws eks update-cluster-config --name prod-cluster \
```

```
--control-plane-scaling-config tier=standard
```

回應：

```
{
    "update": {
        "id": "7551c64b-1d27-4b1e-9f8e-c45f056eb6fd",
        "status": "InProgress",
        "type": "ScalingTierConfigUpdate",
        "params": [
            {
                "type": "UpdatedTier",
                "value": "standard"
            },
            {
                "type": "PreviousTier",
                "value": "tier-2xl"
            }
        ],
        "createdAt": 1565807210.37,
        "errors": []
    }
}
```

EKS 佈建控制平面 - AWS 管理主控台

1. 開啟 Amazon EKS 主控台。
2. 選擇 Create Cluster (建立叢集)。
3. 在組態選項下，請選取自訂組態。
4. 向下捲動至控制平面擴展層。選取使用擴展層以啟用佈建控制平面。
5. 從 XL、2XL、4XL 和 8XL 等各種擴展層選項中，選取要為叢集佈建的控制平面擴展層。
6. 視需要選取其他叢集組態選項。在最後一個步驟中，選取建立叢集。請注意，建立叢集可能需要幾分鐘的時間才能完成。

進階 Kubernetes 控制平面組態

概觀

Amazon EKS 會管理叢集的 Kubernetes 控制平面，包括 API 伺服器、排程器和控制器管理員。EKS 使用適用於大多數工作負載的預設上游 Kubernetes 設定來執行這些元件，而且您不需要對大多數叢集進行變更。但是，某些工作負載受益於不同的控制平面設定。您可能希望排程器將 Pod 封裝在較少的節點上，以降低運算成本、將 Kubernetes 事件保留較短的時間，以限制叢集資料庫 (etcd) 成長，或更頻繁地評估自動擴展決策。

使用進階 Kubernetes 控制平面組態，您可以直接在叢集上設定這些參數。EKS 會將它們套用至控制平面，而您的叢集會繼續以相同的可用性和效能特性運作。

這些是進階組態。每個組態參數都會變更核心 Kubernetes 控制平面元件在叢集上執行之工作負載的行為，而正確的值取決於您的工作負載。變更參數之前，請閱讀下列各節的考量事項，並測試非生產叢集中的變更。

您可以在建立叢集時設定進階控制平面組態參數，或隨時在現有叢集上更新參數。此功能使用現有 `CreateCluster`和 `UpdateClusterConfig`操作搭配新參數，因此您可以透過 AWS 管理主控台、AWS CLI、 AWS SDKs或 AWS CloudFormation 進行設定。EKS 會在套用之前驗證每個組態，並記錄 AWS CloudTrail 中的變更。

進階控制平面參數會套用至整個叢集，以及其上執行的所有工作負載。您無法將它們範圍限定為個別命名空間或工作負載。EKS 會將每個參數限制為經過驗證的範圍。每個參數的支援值都會在下一節中與該參數一起列出。

支援的 Kubernetes 控制平面參數

Amazon EKS 支援下列參數。每個參數都屬於控制平面元件，並透過該元件的組態欄位設定：`kubeSchedulerConfig`、 `kubeControllerManagerConfig`或 `kubeApiServerConfig`。

元件	參數	支援的值	預設	需要佈建的控制平面
kube-scheduler	`nodeResourcesFit.scoringStrategy`	`LeastAllocated`，`MostAllocated`	`LeastAllocated` ，使用 `cpu: 1`和 `memory: 1`	否

元件	參數	支援的值	預設	需要佈建的控制平面
kube-controller-manager	horizontalPodAutoscalerControllerConfig.horizontalPodAutoscalerSyncPeriod	10s 至 15s	15s（秒）	是
kube-controller-manager	podGcControllerConfig.terminatedPodGcThreshold	10000 至 12500	12500	是
kube-apiserver	eventTtl	10m 至 60m	60m（分鐘）	否
kube-apiserver	serviceNodePortRange	minPort 以及介於 10260 和 maxPort之間的 32767	minPort: 30000, maxPort: 32767	否

本主題中的預設值和支援值適用於發佈時可用的 Kubernetes 版本 (EKS v1.31 及更高版本），並且可能會在更新版本中變更。DescribeClusterVersions 操作會報告每個參數和 Kubernetes 版本的目前預設和支援值，因此如果您管理多個版本的叢集或自動化叢集組態，請使用它做為事實來源。如需詳細資訊，請參閱the section called “設定控制平面參數”。

下列各節說明每個參數、變更時間，以及在執行之前要考量的內容。

排程器：節點資源適合

排程器會以兩個階段將 Pod 指派給節點。它會先篩選可執行 Pod 的節點，然後對剩餘的候選項目進行評分，並將 Pod 放置在最高評分的節點上。nodeResourcesFit 外掛程式會檢查節點是否具有 Pod 請求的資源，並根據評分策略對節點進行評分。

欄位	說明	支援的值	預設
nodeResourcesFit.s	用於依資源配置對節點進行評分的策略。	LeastAllocated，	LeastAllocated

欄位	說明	支援的值	預設
coringStrategy.type		MostAllocated	
nodeResourcesFit.scoringStrategy.resources	評分時考慮的資源，每個資源都有相對權重。	cpu、memory、nvm/gpu、aws.amazon.com/neuron 、aws.amazon.com/neuroncore 。 從1到 的權重100。	cpu: 1, memory: 1

LeastAllocated 偏好資源配置較低的節點，這會將 Pod 分散到叢集中的節點，並在每個節點上留下空間。這是預設的 Kubernetes 行為，當您希望每個節點上的可用容量能夠吸收現有 Pod 的成長時，這是不錯的選擇。

MostAllocated 偏好已有較高資源配置的節點，這會將 Pod 封裝到較少的節點上。由於工作負載佔用的總容量較少，因此您可以在較少數量的節點上執行它們，並減少運算支出。隨著時間的推移，此封裝行為可讓少量使用的節點免於新工作負載，因此支援整合的節點集區可以移除它們。

EKS 支援 LeastAllocated和 MostAllocated策略。不支援上游 Kubernetes RequestedToCapacityRatio策略。

資源權重

您可以選擇性地指定具有自訂權重的resources陣列，以影響哪些資源在評分決策中最重要。當特定資源是叢集中的限制條件時，這會很有用。例如，在加速器 (GPU) 是稀少資源的叢集上，在 CPU 和記憶體nvidia.com/gpu上方的權重會將加速器請求 Pod 集中在已部分佔用的節點上。

權重是相對的，而不是絕對的。設定 cpu: 100和 memory: 1不會導致排程器忽略記憶體。它比評分公式中的記憶體重 100 倍的 CPU。如果每個候選節點都有相同的 CPU 可用性，CPU 就不會再區分它們，而且分數會有效地落入記憶體。

省略資源與提供低權重不同。當您指定resources陣列時，只會對您列出的資源進行評分。您離開的資源會完全從計算中排除。例如，只有 CPU 上cpu: 100沒有memory項目分數節點，且記憶體可用性不會影響結果。若要在計算中保留資源，同時減少其影響，請列出它，而非省略它。

加權加速器資源，例如 nvidia.com/gpu 只會影響實際為該資源宣告之 Pod resources.requests 的評分。未請求加速器的 Pod 不會受到加速器權重的影響。

這三個加速器資源是 Kubernetes 延伸資源，這表示節點層級的資源是由外掛程式公告至 Kubernetes，而不是內建資源。只有在裝置外掛程式透過裝置外掛程式 API 將它們公告至 kubelet 時，才會對它們進行評分。nodeResourcesFit 外掛程式無法單獨看到裝置驅動程式在節點上提供的資源，也不會評分。透過動態資源分配 (DRA) 管理的資源是由個別的外掛程式排程，且不屬於nodeResourcesFit評分的一部分，因此啟用 DRA 不會變更此參數的行為。如需設定 NVIDIA 裝置外掛程式的詳細資訊，請參閱 NVIDIA DRA 和裝置外掛程式。如需設定 Neuron 裝置的詳細資訊，請參閱 Neuron 裝置管理。

評分策略是排程器用來計算每個節點分數的數個輸入之一。如需排程器如何篩選和評分節點的詳細資訊，請參閱 Kubernetes 文件中的排程架構。

評分策略的考量事項

- 執行中的 Pod 不會移動。Kubernetes 排程器永遠不會重新定位已在執行的 Pod。變更評分策略只會影響未來的排程決策，而現有的 Pod 放置是永久的。若要重新平衡已在執行中的 Pod，請移出或重新啟動它們。
- 篩選行為不會變更。評分策略只會影響評分階段，其中排程器會依偏好設定排名節點。決定 Pod 是否可以在節點上執行的篩選階段保持不變。不符合節點的 Pod 仍未在任一策略下排程。
- **MostAllocated** 集中爆量半徑。將工作負載封裝到較少的節點，表示如果節點運作狀態不佳、執行個體淘汰或可用區域中斷，會一次影響更多 Pod。在高 Pod 流失下，密集封裝的節點也會更快填滿，這可能會在佈建新容量時讓 Pod 處於 Pending 狀態。
- 排程器和節點管理在不同層操作。評分策略會影響 Pod 放置在已執行它們的節點之間的位置。它不會變更 EKS Auto Mode 或 Karpenter 佈建或移除節點的方式。變更組態之前，請先驗證工作負載的合併行為。

控制器管理員：Horizontal Pod Autoscaler 同步期間

控制器管理員會執行 Kubernetes 控制器，將叢集狀態推向所需的狀態，包括 Horizontal Pod Autoscaler (HPA) 控制器。在每個週期中，HPA 控制器會擷取每個HorizontalPodAutoscaler物件的指標、計算所需的複本計數，並在計數變更時更新目標工作負載。

欄位	說明	支援的值	預設
horizontalPodAutoscalerControllerCon	HPA 控制器評估擴展決策的頻率。	10s 至 15s	15s

欄位	說明	支援的值	預設
fig.horizontalPodA utoscalerSyncPeriod			

縮短同步期間表示您的工作負載在負載增加後更快擴展，而不是在新增容量之前等待完整週期。

若要設定此參數，您的叢集必須位於 Amazon EKS 佈建控制平面上。縮短間隔會增加 HPA 控制器協調叢集中每個HorizontalPodAutoscaler物件的速率，進而產生更多 API 請求。每個對帳會耗用至少一個 API 請求，而變更複本計數的對帳會再耗用兩個。佈建的控制平面叢集會預先配置控制平面容量，隨時準備好應付繁重的工作負載，因此它們的大小足以吸收額外的負載。如需詳細資訊，請參閱the section called “建立佈建的控制平面”。

對叢集支援的 HPA 物件數量的影響

- 縮短同步期間可減少您的控制平面可以按排程協調的**HorizontalPodAutoscaler**物件數量，因為控制器在相同佇列中的作業時間較少。將期間從 縮短15s為 ，可將支援的物件計數10s降低大約三分之一。在縮短同步期間之前，請計算叢集中的HorizontalPodAutoscaler物件，並確認較短的期間仍支援擴展層上的該計數：

  ```
  kubectl get hpa --all-namespaces --no-headers | wc -l
  ```

- EKS 不會針對您的 HPA 物件計數驗證同步期間。即使您的叢集已有超過較短期間支援的HorizontalPodAutoscaler物件，組態變更也會成功。在進行變更之前，請先自行驗證計數。
- 超過支援的計數會無提示地降低自動擴展。如果控制器無法在期間內處理每個物件，則某些物件不會按排程進行協調。EKS 不會發出此條件的警示或 Kubernetes 事件，而且症狀正在自動調整規模，回應速度比預期慢，與預期效果相反。如果您在縮短同步期間後觀察到延遲擴展，請將 參數傳回預設值 15s。
- 同步期間適用於叢集中的每個 HPA 物件。您無法為不同的物件或命名空間設定不同的同步期間。

控制器管理員：已終止的 Pod 垃圾回收閾值

控制器管理員會執行已終止的 Pod 垃圾收集器 (Pod GC 控制器) 。此控制器會在叢集中已終止的 Pod 數目超過閾值後，刪除已終止的 Pod，即 Succeeded或 Failed階段中的 Pod。terminatedPodGcThreshold 參數會設定該閾值。

欄位	說明	支援的值	預設
podGcControllerConfig.terminatedPodGcThreshold	在終止的 Pod 垃圾收集器開始刪除終止的 Pod 之前，可以存在的已終止 Pod 數量。	10000 至 12500	12500

垃圾收集器會以固定的 20 秒週期執行。當您降低閾值時，收集器會在下一個週期開始強制刪除最舊的終止 Pod，直到計數達到新的閾值。

若要設定此參數，您的叢集必須位於 Amazon EKS 佈建控制平面上。降低閾值會增加控制器針對叢集資料庫（等）執行的垃圾回收工作。每個集合傳遞都有資格查詢、處理和刪除更多終止的 Pod。佈建的控制平面叢集會預先配置控制平面容量，隨時準備好應付繁重的工作負載，因此可以吸收額外的負載。如需詳細資訊，請參閱the section called “建立佈建的控制平面”。

終止 Pod 垃圾回收閾值的考量

- 降低閾值會立即刪除多餘的終止 Pod。如果您降低閾值（例如，從 12500到 10000)，垃圾回收控制器會在下一個週期開始強制刪除最舊的終止 Pod。控制器會繼續執行，直到終止的 Pod 計數達到新的閾值為止。減少不是漸進的。
- 閾值適用於所有終止的 Pod，無論擁有者為何。它會影響 Succeeded或 Failed階段中的 Pod，無論它們是任務、CronJob 或部署所擁有，還是獨立。實際上，Job 和 CronJob Pod 是終止 Pod 計數的最常見貢獻者。
- 降低閾值可減少偵錯時段。已完成和失敗的 Pod 從 消失kubectl get pods並kubectl logs更快。檢查已完成任務 Pod 的結束代碼或日誌的自動化具有較小的操作時段。
- 閾值是全域叢集設定。您無法為每個命名空間或每個任務進行設定。若要控制個別任務 Pod 的生命週期，請在該任務ttlSecondsAfterFinished上使用 。

API 伺服器：事件保留

API 伺服器是 Kubernetes 控制平面的前端。它為 Kubernetes API 提供服務，並將叢集狀態持續存在到叢集資料庫 (etcd)。Kubernetes 會記錄事件來描述叢集中發生的情況，例如 Pod 排程決策、映像提取、運作狀態檢查失敗和擴展動作。

欄位	說明	支援的值	預設
eventTtl	API 伺服器在刪除 Kubernetes 事件之前會保留多長時間。	10m 至 60m	60m

執行高流失工作負載的叢集，例如大規模批次工作、AI 工作負載、CI/CD 管道和頻繁CronJobs，可快速累積數千個事件。每個保留的事件都會使用與叢集需要執行的物件競爭的叢集資料庫空間，而大型事件集合會使 API 伺服器清單操作更昂貴。

縮短事件保留會更快地清除此短期診斷資料，進而降低叢集資料庫儲存壓力，並改善事件繁重查詢的 API 伺服器回應時間。

在以下情況下，較短的保留期非常適合：

- 您的叢集會執行批次、CI/CD、AI 或 CronJob 工作負載，以產生大量事件。
- 您觀察到叢集資料庫儲存體成長至其限制。
- 您依賴外部系統持久地擷取事件，而且不依賴 kubectl get events進行歷史偵錯。

事件保留的考量事項

- 變更僅適用於新事件。建立事件時，Kubernetes 會設定事件的到期時間。已存在的事件會保留其建立時生效的保留期間，並根據該排程過期。縮短eventTtl不會縮短叢集資料庫中已存在事件的生命週期，因此儲存體的減少會隨著現有事件的過時而逐漸生效。
- 已刪除的事件無法復原。Kubernetes 移除事件後，該事件會永久消失。如果您將保留時間縮短超過預期並遺失事件歷史記錄，則無法還原。縮短此值之前，請確認您在叢集外部擷取出任何需要進行故障診斷的物件。
- 事件可能會持續稍微超過設定的期間。在某些情況下，事件的到期時間可以延長到超過您設定的值，因為在控制平面領導者選擇期間可能發生的推定租賃續約。
- 減少偵錯時段。較短的保留期間會縮小 kubectl get events和 顯示的視窗kubectl describe。從叢集抓取事件的工具可用資料較少。選擇在儲存效率與除錯工作流程之間取得平衡的值。
- 設定適用於整個叢集。保留適用於所有事件，包括每個命名空間的 Pod 排程、節點條件和擴展事件。您無法為每個命名空間設定不同的保留期間。

API 伺服器：服務節點連接埠範圍

Kubernetes 會為每個需要的 服務在每個節點上配置此範圍的連接埠。這包括 類型的服務，NodePort以及預設 類型的服務LoadBalancer。

欄位	說明	支援的值	預設
serviceNodePortRange.minPort	範圍內的最低連接埠。	10260 至 32767	30000
serviceNodePortRange.maxPort	範圍中最高的連接埠。	10260 至 32767	32767

minPort 必須小於或等於 maxPort。Amazon EKS 會拒絕 大於 minPort的組態maxPort。

透過變更範圍，您可以將節點連接埠配置與組織已強制執行的網路和防火牆政策保持一致。擴大範圍也會增加單一叢集可支援的服務數量。此參數在遷移期間特別有用。移至 Amazon EKS 的應用程式，以及呼叫它們的用戶端，通常會預期特定固定連接埠上的服務。當這些連接埠超出預設範圍時，通常的選項是修改應用程式或將代理放在其前面。將範圍與應用程式已使用的連接埠對齊會移除該運作狀態，因此您可以將工作負載移至 EKS，而無需重寫或新增要維護的網路元件。

為什麼範圍限制為 10260 和 32767

的下限10260可避免NodePort配置節點上 Kubernetes 系統元件已使用的連接埠，包括 kubelet 運作狀態連接埠 (10248) 和 kube-proxy 運作狀態檢查連接埠 ()10256。

的上限會32767保持範圍沒有 Linux 暫時性連接埠範圍，通常從 開始32768。如果 NodePort 落在暫時性範圍內，核心可以為節點的傳出連線選取該連接埠，並與 服務衝突。

服務節點連接埠範圍的考量事項

- 現有的 服務會保留其指派的連接埠。如果您縮小範圍，已經將連接埠保持在新範圍之外的服務會繼續運作，而 kube-proxy 會繼續將流量路由到它們。當範圍變更時，Amazon EKS 不會重新指派現有服務的連接埠。
- 重新建立服務會重新配置其連接埠。如果已刪除並重新建立保留out-of-range連接埠的服務，則無法再指派該連接埠。在縮小現有服務所依賴的範圍之前，請為此進行規劃，特別是當您的部署程序重新建立服務而非更新服務時。

- 範圍外的新配置會遭到拒絕。建立或更新需要設定範圍外連接埠的服務會失敗，並顯示來自 API 伺服器的驗證錯誤。
- 也會驗證明確指定的連接埠。如果服務直接指定nodePort值，而不是讓 Kubernetes 指派值，則該連接埠必須位於設定的範圍內。範圍外的靜態連接埠請求會遭到拒絕，即使在您先前設定的較寬範圍內，相同的連接埠仍然有效。
- 範圍是整個叢集。您無法為不同的命名空間設定不同的範圍。

變更此參數之前，請確認您的安全群組和網路 ACLs新範圍的流量，且範圍不會與節點上其他軟體使用的連接埠衝突。

考量事項

設定進階控制平面參數之前，請先檢閱下列項目。

- 整個叢集範圍 – 控制平面參數適用於整個叢集，以及其上執行的所有工作負載。您無法將它們範圍限定為個別命名空間或工作負載。在非生產叢集中測試參數變更，然後再將其套用至生產。
- Horizontal Pod Autoscaler 同步期間和終止的 Pod 垃圾收集閾值所需的佈建控制平面 – `horizontalPodAutoscalerSyncPeriod`和 `terminatedPodGcThreshold` 參數僅適用於使用 Amazon EKS 佈建控制平面的叢集。Amazon EKS 會將大幅增加控制平面資源消耗的參數限制在具有預先配置控制平面容量的叢集。在標準控制平面模式中設定叢集上的任一參數都會失敗。若要使用它們，請先將叢集移至佈建的控制平面擴展層。如需詳細資訊，請參閱the section called “建立佈建的控制平面”。
- Horizontal Pod Autoscaler 同步期間和終止 Pod 垃圾回收閾值的結束限制 – 如果 `horizontalPodAutoscalerSyncPeriod`或 `terminatedPodGcThreshold` 設定為預設值以外的值，則您無法將叢集的控制平面從佈建模式移回標準模式。若要返回標準模式，請先將兩個參數設回其預設值 (`15s` 和 `12500`)，然後將控制平面擴展層變更為 `standard`。
- 返回預設值 – Amazon EKS 不提供專用重設操作，而從更新省略欄位會保留其目前值，而不是將其清除。若要將參數傳回其預設值，請將其明確設定為預設值。針對叢集執行`DescribeClusterVersions`的 Kubernetes 版本，使用 擷取預設值。如需詳細資訊，請參閱the section called “設定控制平面參數”。
- 更新語意 – 更新會與您現有的組態合併。只有您指定的欄位會變更，而您省略的欄位會保留其目前值。這同時適用於元件和單一元件。例如，僅指定排程器組態的更新會讓您的控制器管理員和 API 伺服器組態保持不變。
- 檢視目前組態 – `describe-cluster`操作會傳回在控制平面上執行的完整組態，包括您尚未自訂的參數及其預設值。

- Kubernetes 版本之間的預設值和支援值可能會變更 – 本主題中記錄的值適用於發佈時可用的 Kubernetes 版本。使用 `DescribeClusterVersions` 擷取每個參數和 Kubernetes 版本的目前預設和支援值。請參閱 the section called “設定控制平面參數”。
- 現有叢集保持不變 – Amazon EKS 不會變更現有叢集的行為。所有叢集都會繼續執行預設參數值，直到您明確設定參數為止。
- 變更不會立即套用 – `UpdateClusterConfig`傳回時，組態變更不會生效。Amazon EKS 透過控制平面的滾動更新來套用新組態，因此在變更生效前，預期會有幾分鐘的時間。更新完成時，叢集會返回 `ACTIVE` 狀態。您可以使用 DescribeUpdate 操作或 區塊追蹤進度，直到使用 完成變更為止`aws eks wait cluster-active`。
- 可稽核性 – Amazon EKS 會在套用之前驗證每個組態，並記錄 AWS CloudTrail 中的組態變更。
- 工具支援 – 進階 Kubernetes 控制平面組態可在啟動時透過 AWS 管理主控台、eksctl、 AWS CLI、Amazon EKS API、 AWS CloudFormation 和 AWS CDK 提供。即將支援 Kubernetes (ACK) 和 Terraform 的 AWS 控制器。
- Kubernetes 版本支援 – 執行 Kubernetes 1.31 版或更新版本的新叢集和現有叢集支援進階 Kubernetes 控制平面組態。
- AWS 區域支援 – 進階 Kubernetes 控制平面組態適用於所有 AWS 商業區域、 AWS GovCloud (US) 區域，以及提供 Amazon EKS 的 AWS 中國區域。
- 定價 – 設定控制平面參數無需額外費用。使用 `horizontalPodAutoscalerSyncPeriod`需要佈建的控制平面，以您擴展層的每小時費率計費。如需詳細資訊，請參閱 Amazon EKS 定價。

後續步驟

- the section called “設定控制平面參數” – 使用 CLI 和 AWS 設定和檢視控制平面參數 AWS 管理主控台。
- the section called “建立佈建的控制平面” – 預先配置控制平面容量，以實現可預測的高效能。

設定進階 Kubernetes 控制平面參數

本指南將逐步引導您使用 CLI 和 來設定和檢視進階 Kubernetes AWS 控制平面參數 AWS 管理主控台。如需每個參數的說明及其對叢集的影響，請參閱 the section called “進階控制平面組態”。

先決條件

開始之前，請確定您具有下列先決條件：

- 執行 Kubernetes 1.31 版或更新版本的 Amazon EKS 叢集 – 執行 Kubernetes 1.31 版或更新版本的新叢集和現有叢集支援進階 Kubernetes 控制平面組態。本主題中的所有參數都會共用此最低版本。
- AWS CLI – 用於使用 AWS 服務的命令列工具，包括 Amazon EKS。如需詳細資訊，請參閱《 AWS 命令列界面使用者指南》中的安裝。安裝 CLI AWS 之後，建議您也進行設定。如需詳細資訊，請參閱《 AWS 命令列界面使用者指南》中的具有 aws 設定的快速組態。
- 必要的 IAM 許可 – 您使用的 IAM 主體必須具有描述和更新 Amazon EKS 叢集的許可。若要檢查目前的委託人，請執行下列命令：

```
aws sts get-caller-identity
```

> **Note**
>
> 建議您在 Bash shell 中完成本主題中的步驟。如果您不使用 Bash shell，則某些指令碼命令 (如行接續字元以及設定和使用變數的方式) 需要針對 shell 進行調整。此外，您的 Shell 的引用及轉義規則可能會有所不同。如需詳細資訊，請參閱《 AWS 命令列界面使用者指南》中的使用引號搭配 AWS CLI 中的字串。

執行本主題中的命令之前，請先設定您偏好的叢集名稱和區域：

```
export CLUSTER=my-eks-cluster
export AWS_REGION=us-west-2
```

支援的值

Amazon EKS 會根據下列範圍驗證每個參數，並拒絕其外部的請求。

元件參數	支援的值	預設
kubeSchedulerConfig.nodeResourcesFit.scoringStrategy.type	LeastAllocated，MostAllocated	LeastAllocated
kubeSchedulerConfig.nodeResourcesFit	1 至 100	cpu: 1, memory: 1

元件參數	支援的值	預設
.scoringStrategy.resources[].weight		
kubeControllerManagerConfig.horizontalPodAutoscalerControllerConfig.horizontalPodAutoscalerSyncPeriod	10s 至 15s	15s
kubeControllerManagerConfig.podGcControllerConfig.terminatedPodGcThreshold	10000 至 12500	12500
kubeApiServerConfig.eventTtl	10m 至 60m	60m
kubeApiServerConfig.serviceNodePortRange.minPort	10260 至 32767	30000
kubeApiServerConfig.serviceNodePortRange.maxPort	10260 至 32767	32767

這些值適用於發佈時可用的 Kubernetes 版本，並且可能會在更新版本中變更。若要擷取特定版本的目前值，請參閱下一節。

尋找 Kubernetes 版本的預設值和支援值

`DescribeClusterVersions` 操作會針對每個 Kubernetes 版本，報告每個控制平面參數的預設值和支援的值。讀取此操作中的值，而不是硬式編碼，尤其是在自動化叢集組態或管理多個 Kubernetes 版本的叢集時。

```
aws eks describe-cluster-versions --cluster-versions 1.35
```

回應包含controlPlaneComponentConfig物件，由控制平面元件組織。每個參數都會報告 defaultValue和 constraints 物件：

```
{
    "clusterVersions": [
        {
            "clusterVersion": "1.35",
            "clusterType": "eks",
            "defaultVersion": false,
            "releaseDate": "2026-01-23T00:00:00+00:00",
            "endOfStandardSupportDate": "2027-03-23T00:00:00+00:00",
            "endOfExtendedSupportDate": "2028-03-23T00:00:00+00:00",
            "status": "STANDARD_SUPPORT",
            "controlPlaneScalingTiers": [
                {
                    "tierName": "tier-xl",
                    "apiRequestConcurrency": 2000,
                    "podSchedulingRatePerSecond": 167,
                    "clusterDatabaseSizeGb": 16
                },
                {
                    "tierName": "tier-2xl",
                    "apiRequestConcurrency": 4000,
                    "podSchedulingRatePerSecond": 283,
                    "clusterDatabaseSizeGb": 16
                },
                {
                    "tierName": "tier-4xl",
                    "apiRequestConcurrency": 8000,
                    "podSchedulingRatePerSecond": 400,
                    "clusterDatabaseSizeGb": 16
                },
                {
                    "tierName": "tier-8xl",
                    "apiRequestConcurrency": 16000,
                    "podSchedulingRatePerSecond": 400,
                    "clusterDatabaseSizeGb": 16
                }
            ],
            "controlPlaneComponentConfig": {
                "kubeApiServerConfig": {
```

```
                    "eventTtl": {
                        "defaultValue": "60m",
                        "constraints": {
                            "min": "10m",
                            "max": "60m"
                        }
                    },
                    "serviceNodePortRange": {
                        "defaultValue": {
                            "minPort": 30000,
                            "maxPort": 32767
                        },
                        "constraints": {
                            "minPort": {"min": 10260, "max": 32767},
                            "maxPort": {"min": 10260, "max": 32767}
                        }
                    }
                },
                "kubeSchedulerConfig": {
                    "nodeResourcesFit": {
                        "scoringStrategy": {
                            "defaultValue": {
                                "type": "LeastAllocated",
                                "resources": [
                                    {"name": "cpu", "weight": 1},
                                    {"name": "memory", "weight": 1}
                                ]
                            },
                            "constraints": {
                                "scoringStrategy": {
                                    "allowedValues": ["LeastAllocated",
"MostAllocated"]
                                },
                                "resources": {
                                    "name": {
                                        "allowedValues": [
                                            "cpu",
                                            "memory",
                                            "nvidia.com/gpu",
                                            "aws.amazon.com/neuron",
                                            "aws.amazon.com/neuroncore"
                                        ]
                                    },
                                    "weight": {"min": 1, "max": 100}
```

```
                                }
                            }
                        }
                    }
                },
                "kubeControllerManagerConfig": {
                    "horizontalPodAutoscalerControllerConfig": {
                        "horizontalPodAutoscalerSyncPeriod": {
                            "defaultValue": "15s",
                            "constraints": {
                                "min": "10s",
                                "max": "15s"
                            }
                        }
                    },
                    "podGcControllerConfig": {
                        "terminatedPodGcThreshold": {
                            "defaultValue": 12500,
                            "constraints": {
                                "min": 10000,
                                "max": 12500
                            }
                        }
                    }
                }
            }
        ]
}
```

上述回應顯示版本支援日期、佈建的控制平面擴展層，以及控制平面組態欄位。如需詳細資訊，請參閱《Amazon EKS API 參考》中的 DescribeClusterVersions。

若要僅傳回組態欄位，請使用 `--query`選項：

```
aws eks describe-cluster-versions --cluster-versions 1.35 \
--query 'clusterVersions[0].controlPlaneComponentConfig'
```

控制平面組態 - AWS CLI

每個控制平面元件都有自己的參數：--kube-scheduler-config、 --kube-controller-manager-config和 --kube-api-server-config。您只能指定要自訂的元件。每個參數接受內嵌 JSON。

使用控制平面參數建立叢集

下列命令會建立叢集，將新的 Kubernetes 事件保留 10 分鐘，而非預設的 60 分鐘。您未指定的參數會使用其預設值。

```
aws eks create-cluster \
--name "$CLUSTER" \
--kubernetes-version 1.35 \
--role-arn arn:aws:iam::111122223333:role/eks-service-role \
--resources-vpc-config subnetIds=subnet-abc123,subnet-def456,securityGroupIds=sg-xyz789
 \
--kube-api-server-config '{
  "eventTtl": "10m"
}'
```

回應：

```
{
    "cluster": {
        "name": "my-eks-cluster",
        "arn": "arn:aws:eks:us-west-2:111122223333:cluster/my-eks-cluster",
        "createdAt": 1709640000.0,
        "version": "1.35",
        "roleArn": "arn:aws:iam::111122223333:role/eks-service-role",
        "status": "CREATING",
        "kubeSchedulerConfig": {
            "nodeResourcesFit": {
                "scoringStrategy": {
                    "type": "LeastAllocated"
                }
            }
        },
        "kubeControllerManagerConfig": {
            "horizontalPodAutoscalerControllerConfig": {
                "horizontalPodAutoscalerSyncPeriod": "15s"
            }
```

```
        },
        "kubeApiServerConfig": {
            "eventTtl": "10m",
            "serviceNodePortRange": {
                "minPort": 30000,
                "maxPort": 32767
            }
        }
    }
}
```

檢視叢集的控制平面組態

```
aws eks describe-cluster --name "$CLUSTER"
```

回應包含 kubeSchedulerConfig、 kubeControllerManagerConfig和 kubeApiServerConfig作為最上層欄位。這些欄位一律存在，並顯示在控制平面上執行的完整組態，包括您尚未自訂的參數及其預設值。

若要傳回單一元件的組態，請使用 --query選項：

```
aws eks describe-cluster --name "$CLUSTER" \
--query 'cluster.kubeSchedulerConfig'
```

回應：

```
{
    "nodeResourcesFit": {
        "scoringStrategy": {
            "type": "LeastAllocated"
        }
    }
}
```

設定排程器評分策略

根據預設，排程器會使用 LeastAllocated策略，將 Pod 分散到各個節點。下列命令會將其變更為 MostAllocated，這偏好具有較高資源配置的節點，並將 Pod 封裝到較少節點。

```
aws eks update-cluster-config \
```

```
--name "$CLUSTER" \
--kube-scheduler-config '{
  "nodeResourcesFit": {
    "scoringStrategy": {
      "type": "MostAllocated"
    }
  }
}'
```

回應：

```
{
    "update": {
        "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "status": "InProgress",
        "type": "ControlPlaneComponentConfigUpdate",
        "params": [
            {
                "type": "kubeSchedulerConfig",
                "value": "{\"nodeResourcesFit\":{\"scoringStrategy\":{\"type\":
\"MostAllocated\"}}}"
            }
        ],
        "createdAt": 1709643600.0,
        "errors": []
    }
}
```

您也可以在評分計算中指定`resources`陣列來權重個別資源，當特定資源是叢集中的限制條件時，這會很有用。如需詳細資訊，請參閱the section called “進階控制平面組態”。

其他控制平面參數遵循相同的模式，每個都具有自己的元件旗標：`--kube-api-server-config`適用於 `eventTtl`和 `serviceNodePortRange`，`--kube-controller-manager-config`適用於 Horizontal Pod Autoscaler 同步期間。如需了解每個參數的功能，以及在變更參數之前要考慮的內容，請參閱 the section called “進階控制平面組態”。

一次更新多個元件

您可以在單一呼叫中指定多個元件參數：

```
aws eks update-cluster-config \
--name "$CLUSTER" \
```

```
--kube-scheduler-config '{"nodeResourcesFit":{"scoringStrategy":
{"type":"MostAllocated"}}}' \
--kube-api-server-config '{"eventTtl":"30m"}'
```

更新會與您現有的組態合併。只有您指定的欄位會變更，而您省略的欄位會保留其目前值。上述命令會讓您的控制器管理員組態保持不變。

監控控制平面組態更新

update-cluster-config 傳回時，組態變更不會生效。Amazon EKS 透過控制平面的滾動更新來套用新組態，這需要幾分鐘的時間。若要封鎖 ，直到變更完成：

```
aws eks wait cluster-active --name "$CLUSTER"
```

此命令會在叢集達到ACTIVE狀態時傳回，通常在幾分鐘內。

若要列出叢集的更新：

```
aws eks list-updates --name "$CLUSTER"
```

回應：

```
{
    "updateIds": [
        "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
    ]
}
```

若要檢視特定更新的狀態：

```
aws eks describe-update --name "$CLUSTER" \
--update-id a1b2c3d4-5678-90ab-cdef-EXAMPLE11111
```

當更新完成Successful時，狀態會從 InProgress 變更為 ，或在發生錯誤Failed時變更為 。errors 欄位指出失敗的原因。

控制平面組態 — AWS 管理主控台

若要在建立叢集時設定進階 Kubernetes 控制平面參數：

1. 開啟 Amazon EKS 主控台。

2. 選擇 Create Cluster (建立叢集)。
3. 在組態選項下，請選取自訂組態。
4. 向下捲動至控制平面組態。
5. 選擇啟用控制平面組態。
6. 設定您要設定的參數。您保持不變的參數會使用其預設值。
7. 視需要選取其他叢集組態選項。在最後一個步驟中，選擇建立叢集。叢集建立可能需要幾分鐘的時間才能完成。

若要在現有叢集上設定進階 Kubernetes 控制平面參數：

1. 開啟 Amazon EKS 主控台。
2. 選擇您要更新的叢集。
3. 選擇概觀索引標籤，然後向下捲動至控制平面組態。
4. 選擇管理，然後選取啟用控制平面組態以設定您要變更的參數。接著選擇 Save changes (儲存變更)。

利用叢集洞見為 Kubernetes 版本升級做好準備，並為錯誤組態進行故障診斷

Amazon EKS 叢集洞察功能提供問題檢測和解決建議，以協助您管理叢集。每個 Amazon EKS 叢集都會根據 Amazon EKS 精選洞察清單定期進行自動檢查。這些洞察檢查完全由 Amazon EKS 管理，並提供有關如何處理調查結果的建議。

叢集洞見類型

- 組態洞見：找出 EKS 混合節點設定中的錯誤組態，這些設定可能會使叢集或工作負載的功能受損。
- 升級洞見：找出可能會影響升級至 Kubernetes 新版本的能力的問題。
- 回復整備洞察：識別可能影響您在升級後回復至先前 Kubernetes 版本的問題。

考量事項

- 頻率：Amazon EKS 每 24 小時重新整理一次叢集洞察，或者您可以手動重新整理以查看最新狀態。例如，您可以在解決問題後手動重新整理叢集洞察，以確認問題是否已解決。

- 許可：Amazon EKS 會自動為每個 EKS 叢集中的叢集洞察建立叢集存取項目。此項目授予 EKS 檢視叢集相關資訊的許可。Amazon EKS 使用此資訊來產生洞見。如需詳細資訊，請參閱the section called “AmazonEKSClusterInsightsPolicy”。
- 回復整備可用性：回復整備洞察僅適用於在過去 7 天內升級的叢集。在 7 天復原資格時段到期後，不再為叢集產生這些洞見。

使用案例

Amazon EKS 中的叢集洞察提供自動化檢查，以協助維護 Kubernetes 叢集的運作狀態、可靠性和最佳組態。以下是叢集洞察的主要使用案例，包括升級準備與組態疑難排解。

升級洞見

升級洞見是叢集洞察中的一種特定洞察檢查類型。這些檢查會傳回與 Kubernetes 版本升級準備相關的洞見。Amazon EKS 會在每個 EKS 叢集上執行升級洞見檢查。

> **Important**
>
> Amazon EKS 已臨時復原一項功能，若發生特定叢集洞察問題，會要求您使用 `--force` 標誌來升級叢集。如需詳細資訊，請參閱 GitHub 上對更新叢集版本強制執行升級洞察的臨時復原。
> 如需更新叢集的詳細資訊，請參閱 the section called “步驟 3：更新叢集控制平面”。

在更新叢集 Kubernetes 版本之前，您可使用 Amazon EKS 主控台中，可觀測性儀表板的升級洞見索引標籤。如果您的叢集已識別出問題，請檢視這些問題並進行適當的修復。這些問題包含指向 Amazon EKS 與 Kubernetes 文件的連結。修復問題後，請按需重新整理叢集洞察，以擷取最新的洞察。如果所有問題都已解決，請更新您的叢集。

Amazon EKS 會傳回與 Kubernetes 版本升級準備度相關的洞見。升級洞察提供有關可能影響 Kubernetes 叢集升級問題的資訊。這可將管理員準備升級所需的工作量降至最低，並提高較新 Kubernetes 版本上應用程式的可靠性。Amazon EKS 會根據可能影響 Kubernetes 版本升級的問題清單自動掃描叢集。Amazon EKS 會經常透過檢閱每個 Kubernetes 版本發布中所做的變更來更新洞察檢查清單。

Amazon EKS 升級洞察有助於加快新版本的測試和驗證流程。這些洞察還透過突出問題並提供補救建議，讓叢集管理員和應用程式開發人員能夠來利用 Kubernetes 最新的功能。

組態洞見

EKS 叢集洞察會自動掃描具有混合節點的 Amazon EKS 叢集，以識別損害 Kubernetes 控制平面與 Webhook 通訊、kubectl 命令 (如 exec 和 logs) 等的組態問題。組態洞見會發現問題並提供修復建議，從而加速實現功能完整的混合節點設定。

回復整備洞察

回復整備洞見是叢集洞見內特定類型的洞見檢查。這些檢查會傳回與 Kubernetes 版本復原準備相關的洞見。Amazon EKS 會對過去 7 天內已升級的叢集執行復原整備洞見檢查。回復整備洞察是point-in-time檢查 - 它們反映評估時的叢集狀態，而不是持續。

升級叢集 Kubernetes 版本後，您可以使用 Amazon EKS 主控台中可觀測性儀表板的升級洞察索引標籤來檢閱復原準備洞察。如果您的叢集已識別出問題，請檢視這些問題並進行適當的修復。這些問題包含指向 Amazon EKS 與 Kubernetes 文件的連結。修復問題後，請按需重新整理叢集洞察，以擷取最新的洞察。如果所有問題都已解決，您可以繼續轉返。

Amazon EKS 會傳回與 `ROLLBACK_READINESS`類別下 Kubernetes 版本復原準備度相關的洞見。回復整備洞察可識別可能影響 Kubernetes 叢集版本回復的可能問題。這可最大限度地減少管理員為復原做準備所花費的精力，並提高還原至先前版本的可靠性。Amazon EKS 會根據可能的復原影響問題清單自動掃描叢集，包括 API 使用相容性（涵蓋移除先前 API 版本的 API 版本畢業期間的不相容檢查、不存在於所需版本的新資源，以及新的欄位或列舉變更）、叢集運作狀態、kubelet 和 kube-proxy 版本扭曲、EKS 受管附加元件相容性，以及自動模式叢集、中斷預算和註釋檢查。

> **Note**
>
> 回復整備洞見只會檢查 EKS 受管附加元件版本。對於自我管理附加元件，或者如果您在 EKS 附加元件生命週期之外覆寫了受管附加元件的版本，則洞見不會偵測版本不相容。在復原之前，您必須負責驗證這些附加元件與目標版本的相容性。

具有 ERROR 或 UNKNOWN 狀態的洞見會封鎖復原，直到問題解決為止。如果您選擇自負風險繼續，您可以使用 `--force`旗標略過洞見檢查。具有 WARNING 狀態的洞見是諮詢，不會阻止轉返。如需詳細資訊，請參閱the section called “復原 Kubernetes 版本”。

開始使用

若要查看已執行的洞見檢查清單，以及 Amazon EKS 已識別的任何相關問題，您可以使用 AWS 管理主控台、 AWS CLI、 AWS SDKs和 Amazon EKS `ListInsights` API 操作。若要開始使用，請參閱the section called “檢視叢集洞察”。

檢視叢集洞察

Amazon EKS 提供三種類型的洞見：組態洞見、升級洞見和復原準備度洞見。組態洞察可辨識出 EKS 混合節點設定中的錯誤組態，這些設定可能會使叢集或工作負載的功能受損。升級洞察可辨識出可能會影響升級至 Kubernetes 新版本的能力的問題。回復整備洞察會識別可能影響您在升級後回復至先前 Kubernetes 版本的問題。

若要查看已執行的洞見檢查清單，以及 Amazon EKS 已識別的任何相關問題，您可以使用 AWS 管理主控台、 AWS CLI、 AWS SDKs和 Amazon EKS `ListInsights` API 操作。

檢視組態洞察 (主控台)

1. 開啟 Amazon EKS 主控台。
2. 從叢集清單中，選擇您想要查看其相關洞察之 Amazon EKS 叢集的名稱。
3. 選擇監控叢集。
4. 選擇叢集運作狀態索引標籤。
5. 在組態洞察資料表中，您將看到以下各欄：
   - 名稱：Amazon EKS 對叢集執行的檢查。
   - 洞察狀態 – 洞察狀態為 `Error`，意味著組態錯誤很可能會影響叢集功能。洞察狀態為 `Warning`，意味著組態與記錄的方法不相符，但若您有意設定，則叢集功能可能會運作。洞察狀態為 `Passing`，意味著 Amazon EKS 在此洞察檢查中未發現叢集中存在任何問題。
   - 版本 – 適用的版本。
   - 上次重新整理時間 – 上次重新整理此叢集的洞察狀態的時間。
   - 描述：來自洞察檢查的資訊，其中包括提醒和建議的補救措施。

檢視升級洞察 (主控台)

1. 開啟 Amazon EKS 主控台。
2. 從叢集清單中，選擇您想要查看其相關洞察之 Amazon EKS 叢集的名稱。
3. 選擇監控叢集。
4. 選擇升級洞察索引標籤。
5. 如需檢視最新資料，選擇重新整理洞察按鈕，然後等候完成重新整理操作。
6. 在升級洞察資料表中，您將看到以下各欄：
   - 名稱：Amazon EKS 對叢集執行的檢查。

- 洞察狀態 – 洞察狀態為「錯誤」，通常意味著受影響的 Kubernetes 版本是目前叢集版本的 N+1，而「警告」狀態意味著該洞察適用於未來 Kubernetes 版本 N+2 或更高。「通過」狀態意味著 Amazon EKS 在此洞察檢查中未發現叢集中存在任何問題。「不明」狀態意味著 Amazon EKS 無法確定叢集是否受到此洞察檢查的影響。
- 版本 – 洞察檢查對之進行可能問題檢查的 Kubernetes 版本。
- 上次重新整理時間 – 上次重新整理此叢集的洞察狀態的時間。
- 上次轉換時間 – 此洞察的狀態上次變更的時間。
- 描述：來自洞察檢查的資訊，其中包括提醒和建議的補救措施。

> **Note**
>
> 在您執行升級後，復原準備洞察會顯示在相同的索引標籤中，並在 7 天復原資格時段期間保持可見。

檢視叢集洞見 (AWS CLI)

1. 如需檢視最新資料，重新整理指定叢集的洞察。視需要對命令進行下列修改，然後執行修改後的命令。
   - 使用您區域的程式碼取代 AWS *region-code*。
   - 使用您叢集的名稱取代 *my-cluster*。

   ```
   aws eks start-insights-refresh --region region-code --cluster-name my-cluster
   ```

2. 如需追蹤洞察重新整理的狀態，執行以下命令。使用您叢集的名稱取代 *my-cluster*。

   ```
   aws eks describe-insights-refresh --cluster-name my-cluster
   ```

   範例輸出如下。

   ```
   {
       "message": "Insights refresh is in progress",
       "status": "IN_PROGRESS",
       "startedAt": "2025-07-30T13:36:09-07:00"
   ```

```
}
```

3. 列示指定叢集的洞察。視需要對命令進行下列修改，然後執行修改後的命令。
   - 使用您區域的程式碼取代 AWS *region-code*。
   - 使用您叢集的名稱取代 *my-cluster*。

```
aws eks list-insights --region region-code --cluster-name my-cluster
```

範例輸出如下。

```
{
"insights":
    [
        {
            "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
            "name": "Deprecated APIs removed in Kubernetes vX.XX",
            "category": "UPGRADE_READINESS",
            "kubernetesVersion": "X.XX",
            "lastRefreshTime": 1734557315.000,
            "lastTransitionTime": 1734557309.000,
            "description": "Checks for usage of deprecated APIs that are scheduled
 for removal in Kubernetes vX.XX. Upgrading your cluster before migrating to the
 updated APIs supported by vX.XX could cause application impact.",
            "insightStatus":
            {
                "status": "PASSING",
                "reason": "No deprecated API usage detected within the last 30
 days.",
            },
        },
        {
            "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
            "name": "Kubelet version skew",
            "category": "UPGRADE_READINESS",
            "kubernetesVersion": "X.XX",
            "lastRefreshTime": 1734557309.000,
            "lastTransitionTime": 1734557309.000,
            "description": "Checks for kubelet versions of worker nodes in the
 cluster to see if upgrade would cause non compliance with supported Kubernetes
 kubelet version skew policy.",
            "insightStatus":
```

```
            {
                "status": "UNKNOWN",
                "reason": "Unable to determine status of node kubelet versions.",
            },
        },
        {
            "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE33333",
            "name": "Deprecated APIs removed in Kubernetes vX.XX",
            "category": "UPGRADE_READINESS",
            "kubernetesVersion": "X.XX",
            "lastRefreshTime": 1734557315.000,
            "lastTransitionTime": 1734557309.000,
            "description": "Checks for usage of deprecated APIs that are scheduled
for removal in Kubernetes vX.XX. Upgrading your cluster before migrating to the
updated APIs supported by vX.XX could cause application impact.",
            "insightStatus":
            {
                "status": "PASSING",
                "reason": "No deprecated API usage detected within the last 30
days.",
            },
        },
        {
            "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
            "name": "Cluster health issues",
            "category": "UPGRADE_READINESS",
            "kubernetesVersion": "X.XX",
            "lastRefreshTime": 1734557314.000,
            "lastTransitionTime": 1734557309.000,
            "description": "Checks for any cluster health issues that prevent
successful upgrade to the next Kubernetes version on EKS.",
            "insightStatus":
            {
                "status": "PASSING",
                "reason": "No cluster health issues detected.",
            },
        },
        {
            "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
            "name": "EKS add-on version compatibility",
            "category": "UPGRADE_READINESS",
            "kubernetesVersion": "X.XX",
            "lastRefreshTime": 1734557314.000,
            "lastTransitionTime": 1734557309.000,
```

```
            "description": "Checks version of installed EKS add-ons to ensure they
are compatible with the next version of Kubernetes. ",
            "insightStatus": { "status": "PASSING", "reason": "All installed EKS
add-on versions are compatible with next Kubernetes version."},
        },
        {
            "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLEccccc",
            "name": "kube-proxy version skew",
            "category": "UPGRADE_READINESS",
            "kubernetesVersion": "X.XX",
            "lastRefreshTime": 1734557314.000,
            "lastTransitionTime": 1734557309.000,
            "description": "Checks version of kube-proxy in cluster to see if
upgrade would cause non compliance with supported Kubernetes kube-proxy version
skew policy.",
            "insightStatus":
            {
                "status": "PASSING",
                "reason": "kube-proxy versions match the cluster control plane
version.",
            },
        },
        {
            "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLEddddd",
            "name": "Deprecated APIs removed in Kubernetes vX.XX",
            "category": "UPGRADE_READINESS",
            "kubernetesVersion": "X.XX",
            "lastRefreshTime": 1734557315.000,
            "lastTransitionTime": 1734557309.000,
            "description": "Checks for usage of deprecated APIs that are scheduled
for removal in Kubernetes vX.XX. Upgrading your cluster before migrating to the
updated APIs supported by vX.XX could cause application impact.",
            "insightStatus":
            {
                "status": "PASSING",
                "reason": "No deprecated API usage detected within the last 30
days.",
            },
        },
        {
            "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE44444",
            "name": "Incompatible API usage",
            "category": "ROLLBACK_READINESS",
            "kubernetesVersion": "X.XX",
```

```
            "lastRefreshTime": 1734557315.000,
            "lastTransitionTime": 1734557309.000,
            "description": "Checks for usage of APIs that are not compatible
with the previous Kubernetes version. Rolling back your cluster before removing
incompatible API usage could cause data loss or application impact.",
            "insightStatus":
            {
                "status": "ERROR",
                "reason": "Incompatible API usage detected.",
            },
        },
        {
            "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE55555",
            "name": "Kubelet version rollback compatibility",
            "category": "ROLLBACK_READINESS",
            "kubernetesVersion": "X.XX",
            "lastRefreshTime": 1734557309.000,
            "lastTransitionTime": 1734557309.000,
            "description": "Checks for kubelet versions of worker nodes in the
cluster to see if rollback would cause non compliance with supported Kubernetes
kubelet version skew policy.",
            "insightStatus":
            {
                "status": "ERROR",
                "reason": "At least one node kubelet version matches the cluster
control plane version.",
            },
        },
        {
            "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE66666",
            "name": "EKS add-on version rollback compatibility",
            "category": "ROLLBACK_READINESS",
            "kubernetesVersion": "X.XX",
            "lastRefreshTime": 1734557314.000,
            "lastTransitionTime": 1734557309.000,
            "description": "Checks version of installed EKS add-ons to ensure they
are compatible with the previous version of Kubernetes.",
            "insightStatus":
            {
                "status": "PASSING",
                "reason": "All installed EKS add-on versions are compatible with
previous Kubernetes version.",
            },
        },
```

```
    ],
"nextToken": null,
}
```

4. 如需洞察的描述性資訊，執行以下命令。視需要對命令進行下列修改，然後執行修改後的命令。
   - 使用您區域的程式碼取代 AWS *region-code*。
   - 使用從列示叢集洞察的清單中擷取的洞察 ID 來取代 *a1b2c3d4-5678-90ab-cdef-EXAMPLE22222*。
   - 使用您叢集的名稱取代 *my-cluster*。

```
aws eks describe-insight --region region-code --id a1b2c3d4-5678-90ab-cdef-
EXAMPLE22222 --cluster-name my-cluster
```

   範例輸出如下。

```
{
  "insight":
    {
      "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
      "name": "Kubelet version skew",
      "category": "UPGRADE_READINESS",
      "kubernetesVersion": "1.27",
      "lastRefreshTime": 1734557309.000,
      "lastTransitionTime": 1734557309.000,
      "description": "Checks for kubelet versions of worker nodes in the cluster
 to see if upgrade would cause non compliance with supported Kubernetes kubelet
 version skew policy.",
      "insightStatus":
        {
          "status": "UNKNOWN",
          "reason": "Unable to determine status of node kubelet versions.",
        },
      "recommendation": "Upgrade your worker nodes to match the Kubernetes version
 of your cluster control plane.",
      "additionalInfo":
        {
          "Kubelet version skew policy": "https://kubernetes.io/releases/version-
skew-policy/#kubelet",
          "Updating a managed node group": "https://docs.aws.amazon.com/eks/latest/
userguide/update-managed-node-group.html",
        },
```

```
      "resources": [],
      "categorySpecificSummary":
        { "deprecationDetails": [], "addonCompatibilityDetails": [] },
    },
}
```

上述輸出同時顯示 UPGRADE_READINESS和 ROLLBACK_READINESS 洞察。回復整備洞見僅適用於過去 7 天內已升級的叢集。

依類別篩選洞見

您可以依類別篩選洞見，以僅檢視特定類型：

```
aws eks list-insights \
  --cluster-name my-cluster \
  --region region-code \
  --filter '{"categories": ["ROLLBACK_READINESS"]}'
```

您也可以依狀態篩選，只查看封鎖的問題：

```
aws eks list-insights \
  --cluster-name my-cluster \
  --region region-code \
  --filter '{"categories": ["ROLLBACK_READINESS"], "statuses": ["ERROR"]}'
```

檢視轉返整備洞見詳細資訊

對於復原整備洞察，describe-insight命令會傳回有關受影響資源和修復步驟的類似資訊。例如：

```
{
    "clusterId": "73a0e91f-f016-4555-bb7c-177496c47c9d",
    "insight": {
        "category": "ROLLBACK_READINESS",
        "name": "API usage rollback compatibility",
        "kubernetesVersion": "1.32",
        "insightStatus": {
            "status": "ERROR",
            "reason": "Detected incompatible API objects with version rollback."
        },
        "resources": [
```

```
            {
                "kubernetesResourceUri": "/apis/networking.k8s.io/v1/servicecidrs/
kubernetes",
                "status": {
                    "status": "ERROR",
                    "reason": "networking.k8s.io/v1 is not compatible with Kubernetes
 version 1.32"
                }
            }
        ]
    }
}
```

如果您使用 找到復原準備洞見`"status": "ERROR"`，您必須在執行復原之前解決問題，或使用 `--force`旗標略過洞見檢查。如需轉返程序的詳細資訊，請參閱 the section called “復原 Kubernetes 版本”。

如果洞見顯示UNKNOWN狀態，EKS 就無法評估洞見。系統會封鎖轉返，直到成功評估洞見，或者您使用 `--force`旗標略過洞見檢查。

將現有叢集更新至全新 Kubernetes 版本

> **Tip**
>
> 註冊即將舉行的 Amazon EKS 研討會。

Amazon EKS 中提供新的 Kubernetes 版本時，您可以將 Amazon EKS 叢集更新至最新版本。

> **Important**
>
> 升級之後，如果發生問題，您可以在 7 天內轉返至先前的版本。如需詳細資訊，請參閱the section called “復原 Kubernetes 版本”。建議您先檢閱了解 EKS 上的 Kubernetes 版本生命週期中的資訊，以及本主題中的更新步驟，再更新至全新 Kubernetes 版本。

新的 Kubernetes 版本有時會加入重大變更。新的 Kubernetes 版本加入了重大變更，因此推薦您在生產叢集執行更新之前，先針對新的 Kubernetes 版本測試您應用程式的行為。若要達成此目標，您可以在移至新的 Kubernetes 版本之前，先建置持續的整合工作流程來測試應用程式行為。

更新程序包括 Amazon EKS 啟動新的 API 伺服器節點，並使用更新的 Kubernetes 版本取代現有的節點。Amazon EKS 會針對這些新節點的網路流量執行標準的基礎結構和整備運作狀態檢查，以確認新節點如同預期順利運作。然而，一旦開始叢集升級，您就無法暫停或停止升級。如果檢查有失敗，Amazon EKS 會還原基礎設施部署，之前的 Kubernetes 版本仍然保留您的叢集。執行中的應用程式不會受到影響，您的叢集絕對不會處於非確定性或無法恢復狀態。Amazon EKS 會定期備份所有受管叢集，並且具備在必要時復原叢集的機制。我們會持續評估和改善 Kubernetes 基礎結構管理程序。

若要更新叢集，Amazon EKS 最多需要五個來自子網的可用 IP 地址，子網是在您建立叢集時指定的。Amazon EKS 會在您指定的任何子網中建立新的叢集彈性網路介面（網路介面）。網路介面可能會在與您現有網路介面所在的子網不同的子網中建立，因此請確認您的安全群組規則針對您建立叢集時指定的任何子網，允許所需的叢集通訊 如果您在建立叢集時指定的任何子網路不存在、沒有足夠的可用 IP 地址，或沒有允許必要叢集通訊的安全群組規則，則更新可能會失敗。

為了確保叢集的 API 伺服器端點一律可供存取，Amazon EKS 提供了高可用性的 Kubernetes 控制平面，並在更新作業期間執行 API 伺服器執行個體的滾動式更新。為了說明更改支援您的 Kubernetes API 伺服器端點的 API 伺服器執行個體之 IP 位址，您必須確保您的 API 伺服器用戶端會有效管理重新連線。最新版本的 `kubectl` 和 Kubernetes 用戶端程式庫已正式獲得支援，可透明地執行此重新連線程序。

> **Note**
>
> 如需了解叢集更新的相關內容，請參閱《EKS 最佳實務指南》中的叢集升級最佳實務。此資源可協助您規劃升級，以及了解升級叢集的策略。

Amazon EKS 自動模式的考量事項

- Amazon EKS 自動模式的運算功能可控制節點的 Kubernetes 版本。在升級控制平面之後，EKS 自動模式將開始逐步更新受管節點。EKS 自動模式遵守 Pod 中斷預算。
- 您無須手動升級 Amazon EKS 自動模式的功能，包括運算自動擴展、區塊儲存，以及負載平衡功能。

> **Note**
>
> 如果您在升級後遇到問題，可以啟動轉返。對於執行 EKS Auto Mode 的叢集，EKS 會在還原控制平面之前自動復原工作者節點。如需詳細資訊，請參閱the section called “復原 Kubernetes 版本”。

摘要

請參閱如下所示 Amazon EKS 叢集升級程序的高階摘要：

1. 確保叢集處於支援升級的狀態。這包括檢查叢集內部署資源所用的 Kubernetes API，從而確認叢集無任何運作狀態問題。在評估叢集的升級準備度時，您應使用 Amazon EKS 升級洞察。
2. 將控制平面升級至下一個次要版本（例如，從 1.35 升級至 1.36)。
3. 升級資料平面中的節點，以便與控制平面的節點相符。
4. 升級叢集上執行的任何其他應用程式 (例如 `cluster-autoscaler`)。
5. 升級 Amazon EKS 提供的附加元件，例如，預設包含的附加元件：
   - Amazon VPC CNI 建議的版本
   - CoreDNS 建議的版本
   - kube-proxy 建議的版本
6. 升級與叢集通訊的任何用戶端 (例如 `kubectl`)。

步驟 1：準備升級

將叢集控制平面的 Kubernetes 版本與節點的 Kubernetes 版本進行比較。

- 取得叢集控制平面的 Kubernetes 版本。

```
kubectl version
```

- 取得節點的 Kubernetes 版本。此命令會傳回所有自我管理和受管 Amazon EC2、Fargate，以及混合式節點。每個 Fargate Pod 會被列為其自己的節點。

```
kubectl get nodes
```

在將控制平面更新為新的 Kubernetes 版本之前，您叢集中的受管節點和 Fargate 節點的 Kubernetes 次要版本，必須要與控制平面目前版本的版本相同。例如，如果您的控制平面正在執行版本 `1.29`，且其中一個節點正在執行版本 `1.28`，則您必須先將節點更新為版本 `1.29`，才能將控制平面更新為 1.30。我們也推薦您在更新控制平面之前，先將自我管理節點與混合式節點更新為與控制平面相同的版本。如需詳細資訊，請參閱the section called “更新”、the section called “更新方法”及the section called “升級混合節點”。若 Fargate 節點的次要版本低於控制平面版本，請先刪除該節點表示的 Pod。接著更新您的控制平面。重新部署後，任何剩餘的 Pod 都將更新為新版本。

步驟 2：檢閱升級考量

Amazon EKS 叢集洞察會針對影響問題的潛在 Kubernetes 版本升級清單來自動掃描叢集，例如棄用 Kubernetes API。Amazon EKS 會依據 Kubernetes 專案中的變更評估，來定期更新要執行的洞察檢查清單。隨著 Amazon EKS 服務及新版本推出變更，Amazon EKS 亦會更新洞察檢查清單。如需詳細資訊，請參閱the section called “叢集洞察”。

請檢閱 Kubernetes 文件中的棄用 API 移轉指南。

請檢閱升級洞察

使用 Amazon EKS 升級洞察來辨識問題。如需詳細資訊，請參閱the section called “檢視升級洞察 (主控台)”。

詳細考量

- 由於 Amazon EKS 執行高可用性的控制平面，因此您一次只能更新一個次要版本。如需此要求的詳細資訊，請參閱 Kubernetes Version and Version Skew Support Policy (Kubernetes 版本和 版本差異支援政策)。假設您目前的叢集版本是版本 `1.28`，並且您想要將其更新為版本 `1.30`。您必須先將您的版本 `1.28` 叢集更新為版本 `1.29`，再將版本 `1.29` 叢集更新為版本 `1.30`。
- 請檢閱節點上 Kubernetes `kube-apiserver` 和 `kubelet` 之間的版本差異。
  - 從 Kubernetes `1.28` 版開始，`kubelet` 最多可能比 `kube-apiserver` 低三個次要版本。請參閱 Kubernetes upstream version skew policy 一節。
  - 如果受管節點和 Fargate 節點上的 `kubelet` 為 Kubernetes `1.25` 版或更高版本，則您可以將叢集的版本更新到最多超前三個版本，而無需更新 `kubelet` 版。例如，如果 `kubelet` 的版本是 `1.25`，在保持 `kubelet` 的版本為 `1.25` 的情況下，您可以將您的 Amazon EKS 叢集版本從 `1.25` 更新為 `1.26`、`1.27` 或 `1.28`
- 作為最佳實務，在開始更新之前，請確認節點上的 `kubelet` 與控制平面的 Kubernetes 版本相同。
- 如果您的叢集設定的適用於 Kubernetes 的 Amazon VPC CNI 外掛程式版本比 `1.8.0` 還舊，則建議您將外掛程式更新至最新版本，然後再更新您的叢集。若要更新外掛程式，請參閱 the section called “Amazon VPC CNI”。
- 您可以備份 Amazon EKS 叢集，以便在升級程序失敗時還原 Amazon EKS 叢集狀態和持久性儲存。請參閱 the section called “ AWS 備份”

步驟 3：更新叢集控制平面

> **Important**
>
> Amazon EKS 已臨時復原一項功能，若發生特定叢集洞察問題，會要求您使用 `--force` 標誌來升級叢集。如需詳細資訊，請參閱 GitHub 上對更新叢集版本強制執行升級洞察的臨時復原。
> Amazon EKS 在「上次重新整理時間」後 24 小時內，將會重新整理叢集洞察。您可將解決問題的時間與叢集洞察的「上次重新整理時間」作比較。
> 另外，解決啟用 API 之後，更新洞察狀態可能需要最多 30 天。在滾動的 30 天時段內，升級洞察始終會尋找棄用的 API。

您可透過以下方式來提交升級 EKS 控制平面版本的請求：

- eksctl
- AWS 主控台
- AWS CLI

更新叢集 - eksctl

此程序需要 `eksctl` 版本 `0.215.0` 或更新版本。您可使用以下命令檢查您的版本：

```
eksctl version
```

如需有關安裝和更新 `eksctl` 的指示，請參閱 `eksctl` 文件中的 Installation 一節。

更新 Amazon EKS 控制平面的 Kubernetes 版本。使用您的叢集名稱取代 `<cluster-name>`。使用您要更新叢集的 Amazon EKS 支援版本號碼取代 `<version-number>`。若要獲取支援的版本編號清單，請參閱 Amazon EKS 支援的版本。

```
eksctl upgrade cluster --name <cluster-name> --version <version-number> --approve
```

此更新需要幾分鐘的時間來完成。

繼續進行the section called “步驟 4：更新叢集元件”。

更新叢集 - AWS 主控台

1. 開啟 Amazon EKS 主控台。
2. 針對您想要升級的叢集，選擇立即升級。
3. 選取叢集要更新的版本，然後選擇更新。
4. 此更新需要幾分鐘的時間來完成。繼續進行the section called “步驟 4：更新叢集元件”。

更新叢集 - AWS CLI

1. 確認已安裝 AWS CLI 且您已登入。如需詳細資訊，請參閱安裝或更新至最新版本的 AWS CLI。
2. 使用以下 CLI 命令更新您的 Amazon EKS AWS 叢集。取代您想要升級的叢集的 <cluster-name> 及 <region-code>。使用您要更新叢集的 Amazon EKS 支援版本號碼取代 <version-number>。若要獲取支援的版本編號清單，請參閱 Amazon EKS 支援的版本。

```
aws eks update-cluster-version --name <cluster-name> \
  --kubernetes-version <version-number> --region <region-code>
```

範例輸出如下。

```
{
    "update": {
        "id": "<update-id>",
        "status": "InProgress",
        "type": "VersionUpdate",
        "params": [
            {
                "type": "Version",
                "value": "<version-number>"
            },
            {
                "type": "PlatformVersion",
                "value": "eks.1"
            }
        ],
[...]
        "errors": []
    }
```

3. 此更新需要幾分鐘的時間來完成。使用下列命令來監控叢集更新的狀態。除了使用相同的 <cluster-name> 及 <region-code>，還需使用上一個命令傳回的 <update-id>。

```
aws eks describe-update --name <cluster-name> \
   --region <region-code> --update-id <update-id>
```

   顯示 Successful 狀態時，即表示更新已完成。

4. 繼續進行the section called “步驟 4：更新叢集元件”。

步驟 4：更新叢集元件

1. 叢集更新完成之後，請將您的節點更新至與已更新叢集相同的 Kubernetes 最低版本。如需詳細資訊，請參閱the section called “更新方法”、the section called “更新”及the section called “升級混合節點”。在 Fargate 上啟動的任何新 Pod 都會有符合您叢集版本的 kubelet 版本。現有的 Fargate Pod 不會變更。
2. (選用) 如果您在更新叢集之前已將 Kubernetes Cluster Autoscaler 部署至叢集，則請將 Cluster Autoscaler 更新為符合您最後更新之 Kubernetes 主要和次要版本的最新版本。
   a. 在網頁瀏覽器中開啟 Cluster Autoscaler 版本頁面，並尋找符合叢集 Kubernetes 主要和次要版本的最新 Cluster Autoscaler 版本。例如，如果叢集的 Kubernetes 版本是 1.30，請尋找開頭為 1.30 的 Cluster Autoscaler 最新版本。記錄該版本的語意版本編號 (例如，1.30.n)，以在下一步中使用。
   b. 使用下列命令，將 Cluster Autoscaler 映像標籤設定為您在前一個步驟中記錄的版本。如有必要，請使用您自己的值取代 X.XX.X。

```
kubectl -n kube-system set image deployment.apps/cluster-autoscaler cluster-
autoscaler=registry.k8s.io/autoscaling/cluster-autoscaler:vX.XX.X
```

3. （僅限具有 GPU 節點的叢集）如果您的叢集具有具有 GPU 支援的節點群組（例如，p3.2xlarge)，您必須在叢集上更新適用於 Kubernetes DaemonSet 的 NVIDIA 裝置外掛程式。 DaemonSet 請先以您想要的 NVIDIA/k8s-device-plugin 版本來取代 <vX.X.X>，再執行下列命令。

```
kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/<vX.X.X>/
deployments/static/nvidia-device-plugin.yml
```

4. 更新適用於 Kubernetes、CoreDNS 及 kube-proxy 附加元件的 Amazon VPC CNI 外掛程式。建議您將附加元件更新為服務帳戶字符中列出的最低版本。

- 如果您正在使用 Amazon EKS 附加元件，請在 Amazon EKS 主控台中選取 Clusters (叢集)，然後在左導覽窗格中選取您已更新的叢集名稱。通知會顯示在主控台。它們會通知您每個有可用更新的附加元件有新的可用版本。若要更新附加元件，請選擇 Add-ons (附加元件) 標籤。在具有可用更新之附加元件的其中一個方塊中，選取 Update now (立即更新)，選取可用的版本，然後選取 Update (更新)。
- 或者，您可以使用 AWS CLI 或 `eksctl` 來更新附加元件。如需詳細資訊，請參閱the section called “更新附加元件”。

5. 如有必要，請更新您的 `kubectl` 版本。您所使用的 `kubectl` 版本，必須與 Amazon EKS 叢集控制平面的版本差距在一個版本以內。

升級適用於 Amazon EKS 叢集的 Kubernetes 版本

您可以在升級後 7 天內將 Amazon EKS 叢集的 Kubernetes 版本復原至先前的次要版本。如需詳細資訊，請參閱the section called “復原 Kubernetes 版本”。在 7 天時段之後，您無法降級。取而代之的是，在之前的 Amazon EKS 版本上建立新的叢集，然後移轉工作負載。

將叢集復原至先前的 Kubernetes 版本

透過 Amazon EKS 版本復原，您可以在執行就地升級之後，將叢集的 Kubernetes 控制平面還原為先前的次要版本。如果您在升級後遇到問題，例如應用程式不相容、已棄用 API 使用或意外行為，您可以轉返以將叢集還原至已知的良好狀態。

在轉返期間，Amazon EKS 會將 Kubernetes API 伺服器和控制平面元件還原至先前的版本，同時保留所有刻印資料、客戶工作負載和持久性磁碟區。

復原的內容

下列元件會復原：

- Kubernetes API 伺服器版本
- 控制平面元件及其組態
- 平台版本 （還原至舊版 Kubernetes 的最新平台版本）
- EKS Auto Mode 工作者節點。對於執行 EKS Auto Mode 的叢集，Amazon EKS 會在還原控制平面之前自動管理 Auto Mode 工作者節點的轉返。如需詳細資訊，請參閱the section called “回復 EKS Auto Mode 叢集”。

不會復原的內容

下列元件不會復原：

- 刻印的資料。所有叢集狀態、資源和組態都會保留。
- 客戶工作負載。您的 Pod、部署和服務會繼續執行。
- EKS 附加元件。附加元件版本保持不變。您可以分別管理這些項目。
- 持久性磁碟區和資料。所有客戶資料保持不變。
- 自我管理節點和混合節點。您要負責轉返這些項目。
- 受管節點群組。您必須使用 `UpdateNodegroupVersion` API 分別復原這些項目。

先決條件

在您可以復原叢集之前，必須符合下列所有條件：

需求	詳細資訊
7 天時段	您必須在升級完成後的 7 天內啟動轉返。7 天後，將無法再轉返。
升級的叢集	叢集必須透過就地升級升級至其目前版本。在目前版本建立的叢集無法復原。
僅限單一版本	您只能回復一個次要版本 (N 至 N-1)。如果您從 1.31 升級到 1.32，然後升級到 1.33，則只能回復到 1.32，而不是 1.31。
支援的版本	目前支援的 Amazon EKS 版本可使用版本轉返。
延伸支援政策	若要轉返至延伸支援的版本，您必須先將叢集的升級政策變更為 `EXTENDED`。
沒有end-of-extended-support 自動升級	如果您的叢集在延伸支援結束時自動升級，則無法復原至先前的版本。如果您的叢集在標準支援結束時自動升級，您可以復原，但必須先將升級政策變更為 `EXTENDED`。
叢集狀態	叢集必須處於 `ACTIVE` 狀態。當另一個更新正在進行時，您無法啟動轉返。

需求	詳細資訊
EKS 功能相容性	如果舊版不支援在叢集上啟用的 EKS 功能，則轉返請求會失敗。無法使用 略過此檢查`--force`。

除了上述要求之外，某些條件即使有 `--force`旗標，也無法復原。這些條件包括：叢集是在目前版本建立的、升級後已超過 7 天、叢集已再次升級至較新的版本，或在目前版本界限中啟用回溯不相容的 EKS 功能。

摘要

以下是 Amazon EKS 叢集復原程序的高階摘要：

1. 檢閱復原整備洞察，以識別可能影響復原的任何問題。
2. 解決任何封鎖問題 (ERROR 狀態洞見），或使用 `--force`繞過洞見檢查。
3. 確認您的應用程式、自訂控制器和第三方工具與先前的 Kubernetes 版本相容。
4. 如果您的工作者節點執行與控制平面相同的 Kubernetes 版本，請先復原工作者節點。
5. 如果您有執行版本與先前 Kubernetes 版本不相容的附加元件，請將它們降級為相容的版本。
6. 啟動控制平面轉返。
7. 監控轉返進度。

> **Important**
>
> 對於執行 EKS Auto Mode 的叢集，會自動處理步驟 4。當您啟動轉返時，Amazon EKS 會在控制平面之前轉返自動模式節點。如需詳細資訊，請參閱the section called “回復 EKS Auto Mode 叢集”。

步驟 1：檢閱轉返整備洞見

Amazon EKS 會根據一組point-in-time復原準備度檢查自動評估您的叢集，並透過 `ROLLBACK_READINESS`類別下的叢集洞見顯示任何問題。這些洞見會在您執行升級後顯示，並在 7 天復原資格時段期間保持可用。

檢視轉返整備洞見

AWS 主控台：

1. 開啟 Amazon EKS 主控台。
2. 選取您的叢集。
3. 選擇升級洞察索引標籤。升級後，復原整備洞見會顯示在此處。
4. 檢閱任何具有 ERROR 或 WARNING 狀態的洞見。

AWS CLI：

```
aws eks list-insights \
  --cluster-name my-cluster \
  --region us-west-2 \
  --filter '{"categories": ["ROLLBACK_READINESS"]}'
```

若要取得特定洞見的詳細資訊：

```
aws eks describe-insight \
  --cluster-name my-cluster \
  --region us-west-2 \
  --id <insight-id>
```

重新整理洞見

Amazon EKS 每 24 小時重新整理一次洞見。您可以在 Amazon EKS 主控台中選擇重新整理按鈕或使用 CLI，在解決問題後手動觸發重新整理：

```
aws eks start-insights-refresh \
  --cluster-name my-cluster \
  --region us-west-2
```

> **Note**
>
> 當您啟動復原時，Amazon EKS 會自動重新整理洞見，以確保針對最新的叢集狀態執行檢查。

Insight 狀態行為

下表說明每個洞見狀態的意義及其對復原的影響：

狀態	意義	對轉返的影響
傳遞	此檢查未偵測到問題	允許轉返
WARNING	偵測到潛在問題，未封鎖	允許轉返（僅限諮詢）
ERROR (錯誤)	偵測到封鎖問題	已封鎖轉返直到解決，或使用 `--force`繞過
UNKNOWN (不明)	無法判斷狀態	已封鎖轉返直到解決，或使用 `--force`繞過

具有 ERROR 或 UNKNOWN 狀態的 Insights 會封鎖復原。具有 PASSING 或 WARNING 狀態的洞見不會阻止您復原。

轉返準備度檢查

Amazon EKS 會執行一組檢查，做為復原整備洞察的一部分。這些檢查會評估 API 用量相容性（包括欄位層級變更偵測）、叢集運作狀態、kubelet 版本偏斜、kube-proxy 版本偏斜，以及附加版本相容性。對於執行 EKS Auto Mode 的叢集，其他檢查會評估 NodePool 中斷預算、do-not-disrupt註釋和 PodDisruptionBudget 組態。

使用 --force 旗標

如果復原整備洞察顯示錯誤狀態，而且您想要繼續而不解決問題，您可以使用 `--force`旗標略過所有洞察檢查：

```
aws eks update-cluster-version \
  --name my-cluster \
  --kubernetes-version 1.30 \
  --force \
  --region us-west-2
```

> **Warning**
>
> 使用 會`--force`略過所有洞見檢查 (ERROR、WARNING、UNKNOWN)，並直接進行復原。略過洞見檢查時，Amazon EKS 無法保證復原的安全性。對於出現的任何問題，您承擔全部責任。

`--force` 旗標只會略過洞見檢查。它不會略過先決條件驗證，例如 7 天時段、建立版本檢查或循序轉返檢查。對於自動模式叢集， `--force`不會覆寫中斷控制。NodePool 中斷預算、PDBs和do-not-disrupt註釋仍然被接受。

步驟 2：準備工作者節點

在復原控制平面之前，請確定您的工作者節點與目標版本相容。Kubernetes 版本扭曲政策要求工作者節點無法執行比控制平面更新的版本。

EKS 自動模式

不需要採取行動。當您啟動轉返時，Amazon EKS 會在控制平面之前自動轉返自動模式節點。如需詳細資訊，請參閱the section called “回復 EKS Auto Mode 叢集”。

受管節點群組 (MNG)

您必須先將受管節點群組復原至先前的版本，才能復原控制平面。使用 `UpdateNodegroupVersion` API：

```
aws eks update-nodegroup-version \
  --cluster-name my-cluster \
  --nodegroup-name my-nodegroup \
  --kubernetes-version 1.30 \
  --region us-west-2
```

節點群組更新遵守您設定的更新設定 (`maxUnavailable` 或 `maxUnavailablePercentage`) 和更新策略 （滾動或強制）。

自我管理節點和混合節點

您負責復原自我管理節點和混合節點。在復原控制平面之前，更新您的節點 AMIs 或組態以使用先前的 Kubernetes 版本。

Fargate

Fargate 工作者節點不支援版本轉返。您可以復原使用 Fargate 之叢集的控制平面，但執行與控制平面相同 Kubernetes 版本的 Fargate Pod 會觸發具有 ERROR 狀態的 kubelet 版本扭曲洞見。

Amazon EKS 無法自動將 Fargate Pod 復原至較舊的 kubelet 版本。

解決方法：如果您的 Fargate Pod 執行與控制平面相同的 Kubernetes 版本，請在啟動復原之前刪除這些 Pod。然後復原您的控制平面。當您重新部署時，任何剩餘的 Pod 都會使用復原版本啟動。

或者，使用 `--force` 略過洞見檢查。不過，繼續進行 kubelet 版本扭曲違規可能會導致 Fargate 工作負載發生意外行為，直到這些 Pod 被取代為止。

步驟 3：復原叢集控制平面

您可以使用 AWS 主控台、 AWS CLI 或 EKS API 來啟動轉返。

使用 AWS 主控台復原叢集

1. 開啟 Amazon EKS 主控台。
2. 選取您的叢集。
3. 選擇動作下拉式清單。
4. 選擇轉返叢集版本。
5. 檢閱復原摘要，包括任何洞見警告。
6. 選擇轉返版本。

復原需要幾分鐘的時間才能完成。對於自動模式叢集，節點復原階段可能需要更長的時間。如需詳細資訊，請參閱the section called “回復 EKS Auto Mode 叢集”。

使用 CLI AWS 復原叢集

使用現有`update-cluster-version`命令搭配先前的 (N-1) Kubernetes 版本：

```
aws eks update-cluster-version \
  --name my-cluster \
  --kubernetes-version 1.30 \
  --region us-west-2
```

回應範例：

```
{
    "update": {
        "id": "e4091a28-ea14-48fd-a8c7-975aeb469e8a",
        "status": "InProgress",
        "type": "VersionRollback",
        "params": [
            {
                "type": "Version",
                "value": "1.30"
            },
            {
                "type": "PlatformVersion",
                "value": "eks.16"
            }
        ],
        "createdAt": "2026-05-12T16:56:01.082000-04:00",
        "errors": []
    }
}
```

> **Note**
>
> 如果洞見資料過時，Amazon EKS 會在執行轉返之前執行洞見重新整理。

步驟 4：監控轉返進度

您可以使用 Amazon EKS 主控台或 AWS CLI 來監控叢集轉返的狀態。

AWS CLI：

```
aws eks describe-update \
  --name my-cluster \
  --region us-west-2 \
  --update-id e4091a28-ea14-48fd-a8c7-975aeb469e8a
```

AWS 主控台：

1. 開啟 Amazon EKS 主控台。
2. 選取您的叢集。

3. 選擇更新歷史記錄索引標籤。
4. 找到與復原相關聯的更新 ID，以檢視其目前狀態。

狀態轉換

對於標準叢集（無自動模式）：

```
InProgress # Successful
InProgress # Failed
```

對於自動模式叢集，叢集狀態會在節點復原ACTIVE時保持，並且UPDATING只有在控制平面復原開始時才會變更為 。使用 `describe-update` 追蹤整體轉返進度。如需詳細資訊，請參閱the section called “回復 EKS Auto Mode 叢集”。

顯示Successful狀態時，復原即完成。

考量事項和警告

洞見是最佳努力和point-in-time

叢集洞察會在觸發轉返時進行評估。如果您在檢查洞見後但在復原完成之前對叢集進行變更（例如，使用新 APIs建立資源），這些變更不會由初始洞見檢查擷取，而且可能會在復原完成之後造成問題。

等式資料保留

Amazon EKS 會在轉返期間保留已刪除的資料。使用 `--force`旗標略過的不相容資源會保留，不會收集垃圾。

延長支援費用

如果您從標準支援下的版本轉返至延伸支援下的版本，您的叢集會開始產生延伸支援費用。例如，如果您從 1.30（延伸支援）升級至 1.31（標準支援），然後復原至 1.30，則延伸支援費用會繼續。

復原的共同責任模型

Amazon EKS 會將 Kubernetes 控制平面復原為所需的版本。作為共同責任模型的一部分，您有責任驗證應用程式與先前版本的相容性：

- Amazon EKS 負責安全地還原控制平面元件。
- 您有責任確保您的應用程式、組態和相依性與舊版相容。

- 您必須檢閱版本之間的任何不相容、評估叢集是否公開，以及緩解任何問題。

CloudFormation 堆疊復原行為

如果 AWS CloudFormation 堆疊更新失敗並觸發堆疊復原，則還原至指定較低 Kubernetes 版本的先前範本版本不會觸發叢集版本復原。版本轉返必須透過 `UpdateClusterVersion` API、CLI 或主控台明確啟動。

轉返和附加元件

在叢集版本復原期間，Amazon EKS 不會自動復原附加元件版本。您必須分別管理附加元件版本。

在轉返控制平面之前：

1. 使用復原整備洞察，檢查附加元件與目標版本的相容性。
2. 如果附加元件版本與先前的 Kubernetes 版本不相容，請先降級：

```
aws eks update-addon \
  --cluster-name my-cluster \
  --addon-name vpc-cni \
  --addon-version v1.22.4-eksbuild.3 \
  --region us-west-2
```

3. 控制平面復原完成後，請確認所有附加元件都正常運作。

> **Note**
>
> 回復整備洞見只會檢查 EKS 受管附加元件版本。對於自我管理的附加元件，您有責任在復原之前驗證與目標版本的相容性。

相關資源

- the section called “回復 EKS Auto Mode 叢集”
- the section called “更新 Kubernetes 版本”
- the section called “叢集洞察”
- 了解 Amazon EKS 上的 Kubernetes 版本生命週期

- 更新受管節點群組
- 叢集升級的最佳實務

回復 EKS Auto Mode 叢集

當您在執行 EKS Auto Mode 的叢集上啟動版本轉返時，Amazon EKS 會在還原控制平面之前自動管理自動模式工作者節點的轉返。本頁說明自動模式節點復原的運作方式、加速方式，以及視需要取消的方式。

如需版本轉返的一般資訊，包括先決條件、洞見檢查和整體轉返程序，請參閱 the section called “復原 Kubernetes 版本”。

自動模式轉返的運作方式

EKS Auto Mode 使用 Karpenter 型系統來管理工作者節點基礎設施，包括 Kubernetes 版本升級和轉返。當您`UpdateClusterVersion`在啟用自動模式的叢集上使用先前的 (N-1) 版本呼叫 時，EKS 會執行下列順序：

1. 驗證先決條件並重新整理回復整備洞察。
2. 使用 Karpenter 型系統將節點漂向所需的轉返版本，遵守設定的中斷控制。
3. 在所有節點都位於所需轉返版本的 Kubernetes 版本扭曲政策中之後，EKS 會重新檢查洞見並繼續控制平面轉返。

控制平面會保留在目前（較新）版本上，並在節點轉返時繼續正常提供流量。Kubernetes 版本扭曲政策允許節點執行比 kube-apiserver 最多三個較舊的次要版本，因此此中繼狀態有效。

> **Note**
>
> 您可以使用 中所述的相同 API 和程序來觸發轉返the section called “復原 Kubernetes 版本”。自動模式節點復原沒有單獨的 API。

> **Note**
>
> 節點復原階段（步驟 2) 可能需要幾分鐘到 7 天，視您的中斷控制而定。如果節點復原未在設定的逾時內完成，則更新會標記為失敗。

> **Note**
>
> 當復原正在進行時，會封鎖其他客戶觸發的控制平面更新。若要執行不同的更新，請先使用 CancelUpdate API 取消轉返。

復原期間的叢集狀態

階段	叢集狀態	發生了什麼事
節點復原進行中	ACTIVE	Karpenter 正在將節點取代為舊版 AMI。控制平面運作狀態良好，可為目前版本的流量提供服務。
控制平面轉返	UPDATING	API 伺服器和控制平面元件正在還原至先前的版本。
復原完成	ACTIVE	叢集完全位於舊版上。

叢集狀態會在節點復原階段`ACTIVE`保持。使用 `ListUpdates`或 `DescribeUpdate` 來判斷是否正在進行轉返。在 Amazon EKS 主控台中，導覽至您的叢集並開啟更新歷史記錄索引標籤，以檢視與復原相關聯的更新 ID 狀態。

若要在復原期間追蹤個別節點進度，請檢查自動模式節點的 Kubernetes 版本：

```
kubectl get nodes -l karpenter.sh/nodepool=<nodepool-name> -o wide
```

中斷控制

自動模式轉返會遵守所有現有的中斷控制。這些控制項會決定節點的取代速度，並可能大幅影響復原持續時間。

NodePool 中斷預算

NodePool 中斷預算可控制可同時中斷的節點數量。在復原期間，Karpenter 在將節點漂移至先前版本時遵守這些預算。

- 漂移區塊無限期轉返nodes： 0的預算。這會觸發 ERROR 洞見。
- 限制性預算 （ 例如 nodes： 1) 會減慢轉返速度，但允許轉返進度。

具有中斷預算的範例 NodePool，允許一次取代 10% 的節點：

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: default
spec:
  disruption:
    budgets:
      - nodes: "10%"
        reasons:
          - Drifted
```

如需設定 NodePool 中斷預算的詳細資訊，請參閱為 EKS Auto Mode 建立節點集區和 Karpenter 中斷預算。

PodDisruptionBudgets

Kubernetes PodDisruptionBudgets 會在節點取代期間接受。如果 PDB 防止 Pod 移出，節點中斷會延遲至 TerminationGracePeriod。

- 具有maxUnavailable： 0延遲節點中斷PDBs。這會觸發警告洞察。
- PDBs不會永久封鎖轉返，但可能會大幅降低轉返速度。

如需詳細資訊，請參閱使用 PDB 保護關鍵工作負載和 Kubernetes PDB 文件。

Do-not-disrupt註釋

karpenter.sh/do-not-disrupt 註釋可以在節點或 Pod 上設定：

在節點上：無限期地封鎖節點中斷。這會觸發 ERROR 洞見，且必須先移除，才能在該節點上繼續轉返。

在 Pod 上：將節點中斷延遲至 TerminationGracePeriod。這會觸發警告洞察，但不會永久封鎖轉返。

如需 Karpenter 中斷行為的詳細資訊，請參閱 Karpenter 中斷文件。

加速轉返

如果回復時間超過預期，您可以在回復進行時調整中斷控制。

增加 NodePool 中斷預算

編輯 NodePool 資源以允許更多並行節點替換：

```
kubectl edit nodepool default
```

將預算變更為較高的值：

```
spec:
  disruption:
    budgets:
      - nodes: "50%"
        reasons:
          - Drifted
```

從節點移除do-not-disrupt註釋

列出具有註釋的節點並將其移除：

```
# List nodes with the annotation
kubectl get nodes -o json | jq '.items[] | select(.metadata.annotations["karpenter.sh/
do-not-disrupt"] == "true") | .metadata.name'

# Remove from a specific node
kubectl annotate node <node-name> karpenter.sh/do-not-disrupt-
```

調整 PodDisruptionBudgets

如果 PDBs 正在減緩 Pod 移出速度，請暫時調整它們：

```
kubectl edit pdb <pdb-name> -n <namespace>
```

> **Warning**
>
> 調整中斷控制會影響您的應用程式可用性保證。在生產環境中進行變更之前，請務必了解影響。

如需管理中斷控制的詳細資訊，請參閱在 Amazon EKS Auto 模式下防止 Pod 和節點中斷。

取消轉返

只有在節點復原時，復原才會處於可取消狀態。您可以在此階段使用 CancelUpdate 來停止操作。

```
aws eks cancel-update \
  --name my-cluster \
  --update-id <update-id> \
  --region us-west-2
```

取消行為

面向	Behavior (行為)
當 可取消時	只有在自動模式節點正在復原時 （ 在控制平面復原開始之前 ）
語意學	盡最大努力停止。停止節點復原操作。
中斷中節點	如果節點在取消時間中斷中，則會完成其目前的操作。
取消後狀態	更新從 Cancelling 到 的轉換Cancelled 。
叢集狀態	會保留ACTIVE整個 。

取消之後

成功取消後，節點會照常漂向目前的叢集版本。更新會從 轉換為 Cancelling Cancelled。

取消之後，您可以立即：

- 重試轉返 （ 只要您仍在 7 天的資格時段內 ）。
- 執行不同的叢集更新。
- 將叢集維持在目前版本上。

當無法取消時

如果節點轉返已完成且控制平面轉返已開始，或者更新已完成，或者Failed狀態為 Successful或 ，則取消會失敗。

> **Note**
>
> CloudFormation 和 Terraform 不支援 CancelUpdate API。如果您需要取消透過 IaC 啟動的轉返，您必須直接呼叫 API。

轉返逾時

自動模式節點復原具有由 中的 `timeoutMinutes` 參數控制的可設定逾時`rollbackConfig`。預設逾時為 720 分鐘 (12 小時）。您可以設定介於 120 分鐘 (2 小時） 到 10080 分鐘 (7 天） 之間的值。逾時是最小限制屬性，表示它不會比您指定的時間更早發生，但會在之後不久發生。

```
aws eks update-cluster-version \
  --name my-cluster \
  --kubernetes-version 1.30 \
  --rollback-config timeoutMinutes=1440 \
  --region us-west-2
```

如果所有節點未在指定的逾時內完成轉返：

1. 轉返逾時。
2. 節點開始漂移回目前的叢集版本。
3. 控制平面會保留在目前版本 （從未復原）。
4. 更新狀態會轉換為 `Failed`。

在逾時之後，如果您仍在原始升級的 7 天復原資格時段內，您可以重試復原。實際上，如果節點在第 7 天轉返逾時，則轉返資格時段可能也已過期，因為兩者都是 7 天。

若要避免逾時，請在啟動轉返之前檢閱轉返整備洞察。這些洞察會警告可能會減慢節點復原程序的中斷預算或註釋。

--force 旗標和 Auto 模式

上的 `--force`旗標`UpdateClusterVersion`只會略過叢集洞見檢查。它不會影響自動模式節點中斷行為。

即使使用 `--force`：

- 仍會遵守 NodePool 中斷預算。
- 仍會遵守 PodDisruptionBudgets。
- 仍會遵守Do-not-disrupt註釋。
- 7 天的節點復原逾時仍然適用。

加速節點復原的唯一方法是自行調整中斷控制。如需詳細資訊，請參閱 the section called “加速轉返”。

IaC 逾時衝突

Infrastructure-as-Code工具有逾時限制，可能與自動模式轉返持續時間衝突。CloudFormation 每個資源最多允許 36 小時。如果操作逾時，CloudFormation 會將其視為無操作，這會讓叢集處於偏離狀態，其中範本不會反映實際的叢集版本。必須明確啟動版本轉返。Terraform Enterprise/Cloud 有大約 24 小時的逾時，但用戶端逾時可能會因憑證過期和其他因素而有所不同。

若要將轉返持續時間與您的 IaC 工具保持一致，請使用 中的 `timeoutMinutes` 參數`rollbackConfig`來設定適當的逾時。如果您的 IaC 工具逾時，請直接使用 CancelUpdate API 重新取得控制權。如果您有限制性的中斷預算，請考慮直接透過 CLI 或 API 而非 IaC 啟動轉返。

節點復原期間的系統更新

當自動模式節點轉返時，EKS 會繼續保持控制平面的安全和可用性。

節點復原進行中時，會封鎖客戶觸發的更新（例如 UpdateClusterVersion 或 UpdateClusterConfig)。如果您需要執行高優先順序更新，請先使用 取消轉返`CancelUpdate`，然後執行更新，如果仍在資格時段內，則重新啟動轉返。

相關資源

- the section called “復原 Kubernetes 版本”
- EKS 自動模式概觀
- 為 EKS 自動模式建立節點集區
- 更新 EKS Auto Mode 叢集的 Kubernetes 版本
- 在 Amazon EKS Auto 模式中防止 Pod 和節點中斷
- EKS Auto 模式故障診斷
- Karpenter 中斷文件
- Kubernetes PodDisruptionBudgets

刪除叢集

當您完成使用 Amazon EKS 叢集後，應刪除與之相關聯的資源，才不會產生任何不必要的成本。

您可以使用 `eksctl`、 AWS 管理主控台或 CLI AWS 刪除叢集。

考量事項

- 如果由於已移除叢集建立者而收到錯誤，請參閱此篇文章來解決問題。
- 叢集生命週期不包括適用於 Prometheus 的 Amazon Managed Service 資源，需要獨立於叢集進行維護。刪除叢集後，確認亦同時刪除任何適用的湊集器，以免產生適用的費用。如需詳細資訊，請參閱 Amazon Managed Service for Prometheus 使用者指南中的尋找及刪除湊集器。
- 若要移除連接的叢集，請參閱 the section called “取消註冊叢集”
- 在您可以刪除叢集之前，請確定已停用叢集的刪除保護。
- 如果您的叢集具有任何 EKS 功能 (ACK、Argo CD 或 kro)，您必須先刪除它們，才能刪除叢集。如需詳細資訊，請參閱the section called “刪除功能”。

EKS 自動模式的考量事項

- 將刪除任何 EKS 自動模式節點，包括 EC2 受管執行個體
- 將刪除所有負載平衡器

如需詳細資訊，請參閱the section called “停用 EKS 自動模式”。

先決條件步驟

以下是您必須先執行的步驟，才能刪除叢集。無論您用來刪除叢集的方法為何，這些步驟都適用。

1. 列出所有在叢集中執行的服務。

```
kubectl get svc --all-namespaces
```

2. 刪除任何與 `EXTERNAL-IP` 值相關的服務。這些服務都是由 Elastic Load Balancing 負載平衡器所朝向的，且您必須在 Kubernetes 中刪除它們，以允許負載平衡器與相關資源可正確釋出。將 *`service-name`* 取代為如上所述列出的每個服務的名稱。

```
kubectl delete svc service-name
```

3. 同時刪除任何輸入資源。如果您不刪除傳入資源，即使您刪除叢集，應用程式負載平衡器仍會保留。將 *inress-name* 取代為您的輸入資源名稱。

```
kubectl get ingress --all-namespaces
```

```
kubectl delete ing ingress-name
```

刪除叢集 (eksctl)

此程序需要 `eksctl` 版本 `0.215.0` 或更新版本。您可使用以下命令檢查您的版本：

```
eksctl version
```

如需有關安裝或更新 `eksctl` 的指示，請參閱 `eksctl` 文件中的 Installation 一節。

1. 完成先決條件步驟。執行此操作後，請使用下列命令刪除叢集及其相關聯的節點，將 *prod* 取代為您的叢集名稱。

```
eksctl delete cluster --name prod
```

輸出：

```
[#]  using region region-code
[#]  deleting EKS cluster "prod"
[#]  will delete stack "eksctl-prod-nodegroup-standard-nodes"
[#]  waiting for stack "eksctl-prod-nodegroup-standard-nodes" to get deleted
[#]  will delete stack "eksctl-prod-cluster"
[#]  the following EKS cluster resource(s) for "prod" will be deleted: cluster. If in
 doubt, check CloudFormation console
```

刪除叢集AWS （主控台）

1. 完成先決條件步驟。執行此操作後，請刪除所有節點群組、Fargate 設定檔和功能。
   a. 開啟 Amazon EKS 主控台。
   b. 在左側導覽窗格中，選擇 Amazon EKS Clusters (叢集)，然後在叢集的標籤式清單中，選擇您要刪除的叢集名稱。

c. 選擇 Compute (運算) 索引標籤，然後選取要刪除的節點群組。選擇 Delete (刪除)，輸入節點群組的名稱，然後選擇 Delete (刪除)。刪除叢集中的所有節點群組。

> **Note**
>
> 列出的節點群組僅為受管節點群組。

d. 選擇要刪除的 Fargate Profile (Fargate 描述檔)，選取 Delete (刪除)，輸入描述檔的名稱，然後選擇 Delete (刪除)。刪除叢集中的所有 Fargate 描述檔。

e. 選擇功能索引標籤，然後選擇要刪除的功能。選擇刪除功能，輸入功能的名稱，然後選擇刪除。刪除叢集中的所有功能。如需詳細資訊，請參閱the section called “刪除功能”。

2. 刪除所有自我管理的 node AWS CloudFormation 堆疊。

a. 開啟 AWS CloudFormation 主控台。

b. 選擇要刪除的節點堆疊，然後選擇 Delete (刪除)。

c. 在 Delete stack (刪除堆疊) 確認對話方塊中，選擇 Delete stack (刪除堆疊)。刪除叢集中的所有自我管理節點堆疊。

3. 刪除叢集。

a. 開啟 Amazon EKS 主控台。

b. 選擇要刪除的叢集，然後選擇刪除。

c. 在刪除叢集確認畫面上，選擇 Delete (刪除)。

4. （選用）刪除 VPC AWS CloudFormation 堆疊。

a. 開啟 AWS CloudFormation 主控台。

b. 選取要刪除的 VPC 堆疊，然後選擇 Delete (刪除)。

c. 在 Delete stack (s刪除堆疊) 確認對話方塊中，選擇 Delete stack (刪除堆疊)。

刪除叢集 (AWS CLI)

1. 完成先決條件步驟。執行此操作後，請刪除所有節點群組、Fargate 設定檔和功能。

a. 使用下列命令列出叢集中的節點群組。

```
aws eks list-nodegroups --cluster-name my-cluster
```

> **Note**
>
> 列出的節點群組僅為受管節點群組。

b. 使用下列命令來刪除每個節點群組。刪除叢集中的所有節點群組。

```
aws eks delete-nodegroup --nodegroup-name my-nodegroup --cluster-name my-cluster
```

c. 使用下列命令列出您叢集中的 Fargate 描述檔。

```
aws eks list-fargate-profiles --cluster-name my-cluster
```

d. 使用下列命令來刪除每個 Fargate 描述檔。刪除叢集中的所有 Fargate 描述檔。

```
aws eks delete-fargate-profile --fargate-profile-name my-fargate-profile --
cluster-name my-cluster
```

e. 使用下列命令列出叢集上的功能。

```
aws eks list-capabilities --cluster-name my-cluster
```

f. 使用下列命令刪除每個功能。刪除叢集中的所有功能。如需詳細資訊，請參閱the section called "刪除功能"。

```
aws eks delete-capability --cluster-name my-cluster --capability-name my-
capability-name
```

2. 刪除所有自我管理的 node AWS CloudFormation 堆疊。

a. 使用下列命令列出您的可用 AWS CloudFormation 堆疊。在產生輸出中尋找節點範本名稱。

```
aws cloudformation list-stacks --query "StackSummaries[].StackName"
```

b. 使用以下命令，將 *node-stack* 取代為您節點堆疊的名稱，來刪除工作節點堆疊。刪除叢集中的所有自我管理節點堆疊。

```
aws cloudformation delete-stack --stack-name node-stack
```

3. 使用以下命令，將 *my-cluster* 替換為您的叢集名稱以刪除叢集。

```
aws eks delete-cluster --name my-cluster
```

4. （選用）刪除 VPC AWS CloudFormation 堆疊。
   a. 使用下列命令列出您的可用 AWS CloudFormation 堆疊。在產生輸出中尋找 VPC 範本名稱。

   ```
   aws cloudformation list-stacks --query "StackSummaries[].StackName"
   ```

   b. 使用以下命令，將 *my-vpc-stack* 替換為您的 VPC 堆疊名稱以刪除 VPC 堆疊。

   ```
   aws cloudformation delete-stack --stack-name my-vpc-stack
   ```

避免 EKS 叢集遭到意外刪除

意外刪除 EKS 叢集可能會影響 Kubernetes 叢集運作。

現在可以避免 EKS 叢集遭到意外刪除。如果您在叢集上啟用了刪除保護，必須先停用刪除保護，才能刪除叢集。

刪除保護的目的是防止意外發生。您應謹慎限制授權刪除叢集的人員。

如果您嘗試刪除已啟用刪除保護的作用中叢集，您將會收到 `InvalidRequestException`。

> **Important**
>
> 如果您在叢集上啟用刪除保護，您必須同時擁有 UpdateClusterConfig 和 DeleteCluster IAM 許可，方能先移除刪除保護，最後刪除叢集。

> **Note**
>
> 如果叢集狀態為建立中、失敗或刪除中，即使啟用了刪除保護，您也可以刪除叢集。

對現有叢集啟用刪除保護

您只能在狀態為作用中的叢集上執行此操作。

```
aws eks update-cluster-config --name <cluster-name> --region <aws-region> --deletion-
protection
```

停用現有叢集的刪除保護

```
aws eks update-cluster-config --name <cluster-name> --region <aws-region> --no-
deletion-protection
```

叢集 API 伺服器端點

本主題可協助您啟用 Amazon EKS 叢集 Kubernetes API 伺服器端點和限制的私有存取，或完全停用從網際網路的公有存取。

建立新的叢集時，Amazon EKS 會為您用來與叢集通訊的受管 Kubernetes API 伺服器建立端點 (使用 Kubernetes 管理工具，例如 `kubectl`)。根據預設，此 API 伺服器端點對網際網路是公有的，而且會使用 AWS Identity and Access Management (IAM) 和原生 Kubernetes 角色型存取控制 (RBAC) 的組合來保護對 API 伺服器的存取。此端點稱為叢集公有端點。此外還有一個叢集私有端點。有關叢集私有端點的更多資訊，請參閱以下章節 the section called “叢集私有端點”。

IPv6 叢集端點格式

對於 2024 年 10 月之後建立的新 IPv6 叢集，EKS 會以下列格式建立唯一的雙堆疊端點。IPv6 叢集是指您在叢集的 IP 系列 (`ipFamily`) 設定中選取了 IPv6 的叢集。

Example

AWS

EKS 叢集公有/私有端點：`eks-cluster.region.api.aws`

AWS GovCloud (US)

EKS 叢集公有/私有端點：`eks-cluster.region.api.aws`

Amazon Web Services in China

EKS 叢集公有/私有端點：`eks-cluster.region.api.amazonwebservices.com.cn`

> **Note**
>
> 雙堆疊叢集端點於 2024 年 10 月引入。如需 IPv6 叢集的詳細資訊，請參閱 the section called "IPv6"。2024 年 10 月之前建立的叢集改用下列端點格式。

IPv4 叢集端點格式

EKS 會針對叢集的 IP 系列 (ipFamily) 設定IPv4中選取的每個叢集，以下列格式建立唯一的端點：

Example

AWS

EKS 叢集公有/私有端點 `eks-cluster.region.eks.amazonaws.com`

AWS GovCloud (US)

EKS 叢集公有/私有端點 `eks-cluster.region.eks.amazonaws.com`

Amazon Web Services in China

EKS 叢集公有/私有端點 `eks-cluster.region.amazonwebservices.com.cn`

> **Note**
>
> 在 2024 年 10 月之前，IPv6 叢集也會使用此端點格式。對於這些叢集，公有端點和私有端點都只能從此端點解析出 IPv4 位址。

叢集私有端點

您可啟用 Kubernetes API 伺服器的私有存取，讓節點和 API 伺服器間的所有通訊都不會離開 VPC。您可以限制可以從網際網路存取 API 伺服器的 IP 地址，或完全停用對 API 伺服器的網際網路存取。

> **Note**
>
> 由於此端點適用於 Kubernetes API 伺服器，而不是用於與 AWS API 通訊的傳統 AWS PrivateLink 端點，因此不會在 Amazon VPC 主控台中顯示為端點。

啟用叢集的端點私有存取時，Amazon EKS 會代表您建立 Route 53 私有託管區域，並將其與您叢集的 VPC 建立關聯。此私有託管區域由 Amazon EKS 管理，不會顯示在帳戶的 Route 53 資源中。若要讓私有託管區域正確將流量路由到 API 伺服器，您的 VPC 必須將 `enableDnsHostnames` 和 `enableDnsSupport` 設為 `true`，且 VPC 設定的 DHCP 選項必須在網域名稱伺服器清單中包含 `AmazonProvidedDNS`。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的更新 VPC 的 DNS 支援。

建立新的叢集時，可定義您 API 伺服器端點的存取要求，您也可隨時更新叢集的 API 伺服器端點存取。

> **Note**
>
> 端點存取控制可存取 Kubernetes API 伺服器的人員。如果您也想要管理來自控制平面的輸出流量如何到達 VPC 中的資源 （ 例如 Webhook 伺服器和 OIDC 供應商 ），請參閱 the section called “控制平面輸出”。

修改叢集端點存取

使用本節所述程序來修改現有叢集的端點存取。下表說明支援的 API 伺服器端點存取組合及其相關的行為。

端點公有存取	端點私有存取	Behavior (行為)
已啟用	Disabled	• 這是新 Amazon EKS 叢集的預設行為。 • 來自您叢集 VPC 內 (如節點與控制平面間的通訊) 的 Kubernetes API 請求會離開 VPC，但不會離開 Amazon 的網路。 • 您的叢集 API 伺服器可從網際網路上存取。您可以選擇性地使用公有存取 CIDRs 清單，透過 CIDR 區塊清單控制對公有端點的存取。如果您限制對特定 CIDR 區塊的存取，我們建議您也啟用私有端點，或確保您指定的 CIDR 區塊包含節點和 Fargate Pod （ 如果您使用它們 ） 從中存取公有端點的地址。 • 標準 AWS 資料傳輸費用適用於工作者節點和控制平面之間的流量。Amazon EKS 會在連線的控制平面端吸

端點公有存取	端點私有存取	Behavior (行為)
		收資料傳輸成本。如需詳細資訊，請參閱the section called “Amazon EKS 定價”。
已啟用	已啟用	• 來自您叢集 VPC 內 (如節點與控制平面間的通訊) 的 Kubernetes API 請求會使用私有 VPC 端點。您可以使用叢集安全群組上的規則來控制對私有端點的存取。 • 您的叢集 API 伺服器可從網際網路上存取。您可以選擇性地使用公有存取 CIDRs 清單，透過 CIDR 區塊清單控制對公有端點的存取。 • 如果您搭配 Amazon EKS 叢集使用混合節點，則不建議同時啟用公有和私有端點存取。由於您的混合節點在 VPC 外部執行，它們會將叢集端點解析為公有 IP 位址。對於具有混合節點的叢集，建議使用公有或私有端點存取。
Disabled	已啟用	• 前往您叢集 API 伺服器的所有流量必須來自叢集的 VPC 或連接的網路。 • 您的 API 伺服器無法從網際網路上公有存取。任何 `kubectl` 命令都必須來自 VPC 或連接的網路內。如需連接選項，請參閱 the section called “存取僅限私有 API 伺服器”。 • 您可以使用叢集安全群組上的規則來控制對私有端點的存取。請注意，公有存取 CIDRs不會影響私有端點。 • 公有 DNS 伺服器會將叢集的 API 伺服器端點解析為 VPC 的私有 IP 位址。在過去，端點只能從 VPC 內解析。 如果您的端點未針對現有叢集解析為 VPC 內的私有 IP 地址，您可以： • 啟用公有存取，然後再次停用它。您只需對叢集執行一次，從此以後，端點就會解析為私有 IP 地址。 • 更新您的叢集。

端點存取控制

請注意，下列每個控制端點存取的方法只會影響個別的端點。

叢集安全群組

叢集安全群組控制兩種類型的連線：連線至 kubelet API 和私有端點。API 的連線kubelet用於 `kubectl attach`、`kubectl cp`、`kubectl logs`、 `kubectl exec`和 `kubectl port-forward`命令。叢集安全群組不會影響公有端點。

公開存取 CIDRs

公有存取 CIDRs 會依 CIDR 區塊清單控制對公有端點的存取。請注意，公有存取 CIDRs不會影響私有端點。公有存取 CIDRs 在IPv6叢集和IPv4叢集上的行為，取決於其建立日期，如下所述：

公有端點中的 CIDR 區塊 (**IPv6** 叢集)

您可以將 IPv6 和 IPv4 CIDR 區塊新增至 IPv6 叢集的公有端點，因為該公有端點是雙重堆疊的。這僅適用於 2024 年 10 月或之後建立的、`ipFamily` 設定為 IPv6 的新叢集。您可以透過新的端點網域名稱 `api.aws` 來識別這些叢集。

公有端點中的 CIDR 區塊 (**IPv4** 叢集)

您可以將 IPv4 CIDR 區塊新增至 IPv4 叢集的公有端點。您無法將 IPv6 CIDR 區塊新增至 IPv4 叢集的公有端點。如果您嘗試這樣做，EKS 會傳回以下錯誤訊息：`The following CIDRs are invalid in publicAccessCidrs`

公有端點中的 CIDR 區塊 (2024 年 10 月之前建立的 **IPv6** 叢集)

您可以將 IPv4 CIDR 區塊新增到您在 2024 年 10 月之前建立的舊 IPv6 叢集的公有端點。您可以透過 `eks.amazonaws.com` 端點來識別這些叢集。您無法將 IPv6 CIDR 區塊新增到您在 2024 年 10 月之前建立的這些舊 IPv6 叢集的公有端點。如果您嘗試這樣做，EKS 會傳回以下錯誤訊息：`The following CIDRs are invalid in publicAccessCidrs`

存取僅限私有 API 伺服器

若您已停用叢集 Kubernetes API 伺服器端點的公有存取，將只能在 VPC 內或連接的網路存取 API 伺服器。以下是存取 Kubernetes API 伺服器端點的幾種可能方法：

連線的網路

使用 AWS 傳輸閘道或其他連線選項，將您的網路連線到 VPC，然後使用連線網路中的電腦。您必須確認 Amazon EKS 控制平面安全群組包含的規則允許連接埠 443 上來自連接網路的傳入流量。

Amazon EC2 堡壘主機

您可以在叢集 VPC 中的公有子網路啟動 Amazon EC2 執行個體，然後透過 SSH 登入該執行個體，以執行 kubectl 命令。如需詳細資訊，請參閱 AWS上的 Linux 堡壘主機。您必須確認 Amazon EKS 控制平面安全群組包含的規則允許連接埠 443 上來自堡壘主機的傳入流量。如需詳細資訊，請參閱the section called “安全群組要求”。

當您kubectl為堡壘主機設定 時，請務必使用已映射至叢集 RBAC 組態的 AWS 登入資料，或在移除端點公開存取之前，將堡壘將使用的 IAM 主體新增至 RBAC 組態。如需詳細資訊，請參閱the section called “Kubernetes API 存取”及the section called “未經授權或存取遭拒 (kubectl)”。

AWS Cloud9 IDE

AWS Cloud9 是以雲端為基礎的整合式開發環境 (IDE)，可讓您使用瀏覽器撰寫、執行和偵錯程式碼。您可以在叢集的 VPC 中建立 an AWS Cloud9 IDE，並使用 IDE 與叢集通訊。如需詳細資訊，請參閱在 AWS Cloud9 中建立環境。您必須確認 Amazon EKS 控制平面安全群組包含的規則允許連接埠 443 上來自 IDE 安全群組的傳入流量。如需詳細資訊，請參閱the section called “安全群組要求”。

當您kubectl為 AWS Cloud9 IDE 設定 時，請務必使用已映射至叢集 RBAC 組態的 AWS 登入資料，或在移除端點公開存取之前，將 IDE 將使用的 IAM 主體新增至 RBAC 組態。如需詳細資訊，請參閱the section called “Kubernetes API 存取”及the section called “未經授權或存取遭拒 (kubectl)”。

AWS CloudShell

在 Amazon EKS 主控台的叢集詳細資訊頁面上選擇連線。對於私有叢集，CloudShell 會啟動可到達叢集私有 API 伺服器端點的 VPC 環境。如需詳細資訊，請參閱the section called “透過 kubectl 存取叢集”。

在 GitHub 上編輯此頁面

設定叢集 API 伺服器端點的網路存取

您可以在下列各節中使用 AWS 管理主控台 或 CLI AWS 修改叢集 API 伺服器端點存取。

設定端點存取 - AWS 主控台

1. 開啟 Amazon EKS 主控台。
2. 選擇叢集名稱以顯示您叢集的資訊。

3. 選擇聯網索引標籤，然後選擇管理端點存取。
4. 對於 Private access (私有存取)，選擇是否啟用或停用您叢集的 Kubernetes API 伺服器端點的私有存取。如果您啟用私有存取，源自於您叢集的 VPC 的 Kubernetes API 將使用私有 VPC 端點。您必須啟用私有存取，才能停用公有存取。
5. 對於 Public access (公有存取)，選擇是否啟用或停用您叢集的 Kubernetes API 伺服器端點的公有存取。若您停用公有存取，您叢集的 Kubernetes API 伺服器僅可接收來自叢集 VPC 內的請求。
6. (選用) 如果您已啟用 Public access (公開存取)，則可指定網際網路可用來與公有端點通訊的位址。選取 Advanced Settings (進階設定)。輸入 CIDR 區塊，例如 *203.0.113.5/32*。區塊不能包含預留地址。您可以透過選取 Add source (新增來源) 來輸入其他區塊。您可以指定的 CIDR 區塊有數量上限。如需詳細資訊，請參閱the section called “Service Quotas”。如果不指定區塊，則公有 API 伺服器端點會接收來自所有 IPv4 (`0.0.0.0/0`) IP 位址的要求，此外對於雙重堆疊 IPv6 叢集，還會接受來自 IPv6 (`::/0`) 的要求。如果您使用 CIDR 區塊限制對公有端點的存取，則建議您也啟用私有端點存取，以便節點和 Fargate Pods (如果您使用它們) 可與叢集通訊。若不啟用私有端點，您的公有存取端點 CIDR 來源必須包含來自 VPC 的輸出來源。例如，如果您的私有子網路中有節點，透過 NAT 閘道與網際網路通訊，則您必須將 NAT 閘道的對外 IP 地址新增至公有端點上的白名單 CIDR 區塊。
7. 選擇 Update (更新) 以完成操作。

設定端點存取 - AWS CLI

使用 CLI AWS 版本 `1.27.160` 或更新版本完成下列步驟。您可以使用 `aws --version` 來檢查您的目前版本。若要安裝或升級 AWS CLI，請參閱安裝 AWS CLI。

1. 使用以下 CLI AWS 命令更新您的叢集 API 伺服器端點存取。替換叢集名稱和所需的端點存取值。如果您設定 `endpointPublicAccess=true`，則可以 （選擇性） 為 輸入單一 CIDR 區塊，或以逗號分隔的 CIDR 區塊清單`publicAccessCidrs`。區塊不能包含預留地址。如果您指定 CIDR 區塊，則公有 API 伺服器端點只會接收來自所列出區塊的要求。您可以指定的 CIDR 區塊有數量上限。如需詳細資訊，請參閱the section called “Service Quotas”。如果您使用 CIDR 區塊限制對公有端點的存取，則建議您也啟用私有端點存取，以便節點和 Fargate Pods (如果您使用它們) 可以與叢集通訊。若不啟用私有端點，您的公有存取端點 CIDR 來源必須包含來自 VPC 的輸出來源。例如，如果您的私有子網路中有節點，透過 NAT 閘道與網際網路通訊，則您必須將 NAT 閘道的對外 IP 地址新增至公有端點上的白名單 CIDR 區塊。如果不指定 CIDR 區塊，則公有 API 伺服器端點會接收來自所有 (0.0.0.0/0) IP 位址的要求，此外對於雙重堆疊 IPv6 叢集，還會接受來自 IPv6 (`::/0`) 的請求。

> **Note**
>
> 下列命令會啟用 API 伺服器端點的私有存取以及來自單一 IP 地址的公有存取。使用單一 CIDR 區塊或您想要限制網路存取的 CIDR 區塊清單取代 *203.0.113.5/32*。

```
aws eks update-cluster-config \
    --region region-code \
    --name my-cluster \
    --resources-vpc-config
 endpointPublicAccess=true,publicAccessCidrs="203.0.113.5/32",endpointPrivateAccess=true
```

範例輸出如下。

```
{
    "update": {
        "id": "e6f0905f-a5d4-4a2a-8c49-EXAMPLE00000",
        "status": "InProgress",
        "type": "EndpointAccessUpdate",
        "params": [
            {
                "type": "EndpointPublicAccess",
                "value": "true"
            },
            {
                "type": "EndpointPrivateAccess",
                "value": "true"
            },
            {
                "type": "publicAccessCidrs",
                "value": "[\"203.0.113.5/32\"]"
            }
        ],
        "createdAt": 1576874258.137,
        "errors": []
    }
}
```

2. 使用叢集名稱和前述命令傳回的更新 ID，藉由以下命令監控端點存取的更新狀態，當狀態顯示為 `Successful`，您的更新就完成了。

```
aws eks describe-update \
    --region region-code \
    --name my-cluster \
    --update-id e6f0905f-a5d4-4a2a-8c49-EXAMPLE00000
```

範例輸出如下。

```
{
    "update": {
        "id": "e6f0905f-a5d4-4a2a-8c49-EXAMPLE00000",
        "status": "Successful",
        "type": "EndpointAccessUpdate",
        "params": [
            {
                "type": "EndpointPublicAccess",
                "value": "true"
            },
            {
                "type": "EndpointPrivateAccess",
                "value": "true"
            },
            {
                "type": "publicAccessCidrs",
                "value": "[\"203.0.113.5/32\"]"
            }
        ],
        "createdAt": 1576874258.137,
        "errors": []
    }
}
```

在 GitHub 上編輯此頁面

在 EKS 叢集上部署 Windows 節點

了解如何啟用及管理對 Amazon EKS 叢集的 Windows 支援，以便同時執行 Linux 容器與 Windows 容器。

考量事項

部署 Windows 工作節點之前，請注意下列考量事項。

- EKS 自動模式不支援 Windows 節點
- 您可以透過 `HostProcess` Pod 在 Windows 節點上使用主機聯網。如需詳細資訊，請參閱 Kubernetes 文件中的建立 Windows HostProcess Pod。
- Amazon EKS 叢集必須包含一或多個 Linux 或 Fargate 節點，才能執行只在 Linux 上執行的核心系統 Pod，例如 CoreDNS。
- `kubelet` 和 `kube-proxy` 事件日誌會重新導向至 `EKS Windows` 事件日誌，且其限制會設定為 200 MB。
- 您不能對在 Windows 節點上執行 Pod 使用指派安全群組給個別 Pod。
- 您不能將自訂聯網與 Windows 節點搭配使用。
- 您不能將 `IPv6` 與 Windows 節點搭配使用。
- Windows 節點支援每個節點一個彈性網路介面。依預設，每個 Windows 節點可以執行的 Pod 數量，等於節點執行個體類型的每個彈性網路介面可用的 IP 位址數量減去一。如需詳細資訊，請參閱 Amazon EC2 使用者指南中的每個執行個體類型每個網路介面的 IP 位址。
- 在 Amazon EKS 叢集中，具有負載平衡器的單一服務最多可支援 1024 個後端 Pod。每個 Pod 都有自己的唯一 IP 地址。自 OS Build 17763.2746 起的 Windows Server 更新後，過去的 64 個 Pod 限制已不存在。
- Windows 容器不支援 Fargate 上的 Amazon EKS Pod。
- Windows 作為主機的作業系統時，您不能使用 Amazon EKS 混合節點。
- 無法從 `vpc-resource-controller` Pod 擷取日誌。您先前在將控制器部署到資料平面時可以這麼做。
- 在將 `IPv4` 地址指派給新的 Pod 之前，有一段冷卻期間。這可以防止流量因 `kube-proxy` 規則過時而流動到具有相同 `IPv4` 地址的較舊 Pod。
- 在 GitHub 上管理控制器的來源。若要對控制器做出貢獻或提出問題，請造訪 GitHub 上的專案。
- 為 Windows 受管節點群組指定自訂 AMI ID 時，請將 `eks:kube-proxy-windows`新增至您的 AWS IAM Authenticator 組態映射。如需詳細資訊，請參閱the section called “指定 AMI ID 時的限制和條件”。
- 若保留您的可用 IPv4 位址對於子網路非常重要，請參閱 EKS 最佳實務指南 - Windows 網路 IP 位址管理來獲取指引。
- EKS 存取項目考量

- 若搭配 Windows 節點使用存取項目，需要使用類型 EC2_WINDOWS。如需詳細資訊，請參閱the section called “建立存取項目”。

  如需建立適用於 Windows 節點的存取項目：

  ```
  aws eks create-access-entry --cluster-name my-cluster --principal-arn arn:aws:
   iam::111122223333:role/<role-name> --type EC2_Windows
  ```

先決條件

- 現有的叢集。
- 叢集必須至少有一個 (建議至少有兩個) Linux 節點或 Fargate Pod，才能執行 CoreDNS。如果啟用舊式 Windows 支援，則必須使用 Linux 節點 (不能使用 Fargate Pod) 來執行 CoreDNS。
- 現有的 Amazon EKS 叢集 IAM 角色。

啟用 Windows 支援

1. 如果叢集中沒有 Amazon Linux 節點，並且使用 Pod 的安全群組，請跳至下一個步驟。否則，請確認 AmazonEKSVPCResourceController 受管政策會連接至叢集角色。使用叢集角色名稱取代 *eksClusterRole*。

   ```
   aws iam list-attached-role-policies --role-name eksClusterRole
   ```

   範例輸出如下。

   ```
   {
       "AttachedPolicies": [
           {
               "PolicyName": "AmazonEKSClusterPolicy",
               "PolicyArn": "arn:aws: iam::aws:policy/AmazonEKSClusterPolicy"
           },
           {
               "PolicyName": "AmazonEKSVPCResourceController",
               "PolicyArn": "arn:aws: iam::aws:policy/AmazonEKSVPCResourceController"
           }
       ]
   }
   ```

如果已連接政策 (如上一個輸出所示)，請略過下一個步驟。

2. 連接 AmazonEKSVPCResourceController 受管政策至您的 Amazon EKS 叢集 IAM 角色。使用叢集角色名稱取代 *eksClusterRole*。

```
aws iam attach-role-policy \
  --role-name eksClusterRole \
  --policy-arn arn:aws: iam::aws:policy/AmazonEKSVPCResourceController
```

3. 更新 VPC CNI ConfigMap 以啟用 Windows IPAM。請注意，若使用 Helm Chart 在叢集上安裝 VPC CNI，或作為 Amazon EKS 附加元件，可能無法直接修改 ConfigMap。若要了解設定 Amazon EKS 附加元件的相關資訊，請參閱 the section called “您可自訂的欄位”。

   a. 使用下列內容，建立名為 *vpc-resource-controller-configmap.yaml* 的檔案。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: amazon-vpc-cni
  namespace: kube-system
data:
  enable-windows-ipam: "true"
```

   b. 將 ConfigMap 套用至您的叢集。

```
kubectl apply -f vpc-resource-controller-configmap.yaml
```

4. 若叢集的驗證模式已設定為啟用 aws-auth configmap：

   - 請確認您的 aws-auth ConfigMap 包含 Windows 節點執行個體角色的映射，以包含 eks:kube-proxy-windows RBAC 許可群組。您可以透過執行以下命令來驗證。

```
kubectl get configmap aws-auth -n kube-system -o yaml
```

   範例輸出如下。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: aws-auth
  namespace: kube-system
data:
```

```
  mapRoles: |
    - groups:
      - system:bootstrappers
      - system:nodes
      - eks:kube-proxy-windows # This group is required for Windows DNS resolution
 to work
      rolearn: arn:aws: iam::111122223333:role/eksNodeRole
      username: system:node:{{EC2PrivateDNSName}}
[...]
```

您應該會在群組下看到 eks:kube-proxy-windows 列出。若未指定群組，您需要更新 ConfigMap 或建立群組以包含所需的群組。如需 aws-auth ConfigMap 的詳細資訊，請參閱 the section called “將 aws-auth ConfigMap 套用至您的叢集”。

5. 若叢集的驗證模式已設定為停用 aws-auth configmap，則您可使用 EKS 存取項目。建立新的節點角色，以便與 Windows 執行個體配合使用，並且 EKS 將自動建立類型為 EC2_WINDOWS 的存取項目。

部署 Windows Pod

在將 Pod 部署到叢集時，您需要指定在執行混合節點類型時所使用的作業系統。

對於 Linux Pod，請在清單檔案中使用下列節點選取器文字。

```
nodeSelector:
        kubernetes.io/os: linux
        kubernetes.io/arch: amd64
```

對於 Windows Pod，請在清單檔案中使用下列節點選取器文字。

```
nodeSelector:
        kubernetes.io/os: windows
        kubernetes.io/arch: amd64
```

您可以部署範例應用程式來查看正在使用的節點選擇器。

在 Windows 節點上支援更多數量的 Pod

在 Amazon EKS 中，每個 Pod 都會從您的 VPC 配置一個 IPv4 地址。因此，即使有足夠的資源可以在節點上執行更多 Pod，您能部署到節點的 Pod 數量也會受到可用 IP 位址的限制。由於 Windows 節點僅支援一個彈性網路介面，因此依預設，Windows 節點上可用 IP 地址的數量上限為：

```
Number of private IPv4 addresses for each interface on the node - 1
```

一個 IP 位址用作網路介面的主要 IP 位址，因此無法將其配置給 Pod。

您可以啟用 IP 字首委派，在 Windows 節點上啟用更多數量的 Pod。此功能可讓您將 /28 IPv4 字首指派給主要網路介面，而不是指派次要 IPv4 位址。而指派 IP 字首會將節點上可用 IPv4 地址的數量上限增加到：

```
(Number of private IPv4 addresses assigned to the interface attached to the node - 1) *
 16
```

由於可用 IP 位址的數量大幅增加，可用的 IP 位址不應限制您擴展節點上 Pod 數量的能力。如需詳細資訊，請參閱the section called “增加 IP 位址”。

停用 Windows 支援

1. 如果叢集包含 Amazon Linux 節點，並且使用 Pod 的安全群組，則請跳過這個步驟。

   從叢集角色中移除 AmazonVPCResourceController 受管 IAM 政策。使用叢集角色名稱取代 *eksClusterRole*。

   ```
   aws iam detach-role-policy \
       --role-name eksClusterRole \
       --policy-arn arn:aws:iam::aws:policy/AmazonEKSVPCResourceController
   ```

2. 停用 amazon-vpc-cni ConfigMap 中的 Windows IPAM。

   ```
   kubectl patch configmap/amazon-vpc-cni \
                       -n kube-system \
                       --type merge \
                       -p '{"data":{"enable-windows-ipam":"false"}}'
   ```

部署網際網路存取受到限制的私有叢集

本主題說明如何部署部署在 AWS 雲端上，但沒有傳出網際網路存取權的 Amazon EKS 叢集。如果您在 AWS Outposts 上有本機叢集，請參閱 the section called “節點”，而非本主題。

> **Note**
>
> 本主題涵蓋資料平面輸出（節點和沒有傳出網際網路存取的工作負載）。若要控制來自叢集控制平面的輸出流量如何到達您的 VPC，請參閱 the section called “控制平面輸出”。

如果您不熟悉 Amazon EKS 聯網，請參閱探究 Amazon EKS 工作節點之叢集網路的奧秘。如果您的叢集沒有對外網際網路存取，則其必須符合下列需求：

叢集架構要求

- 叢集必須從 VPC 中的容器登錄檔中提取映像。您可以在 VPC 中建立 Amazon Elastic Container Registry，並將容器映像複製到其中，以供節點提取。如需詳細資訊，請參閱the section called “將映像複製到儲存庫”。
- 您的叢集必須啟用端點私有存取。這對節點向叢集端點註冊而言是必要的。端點公有存取權限並非必要。如需詳細資訊，請參閱the section called “叢集端點存取”。

節點要求

- 自我管理 Linux 與 Windows 節點在啟動之前必須包含下列引導引數。這些引數會略過 Amazon EKS 自我檢查，並且不需要從 VPC 內存取 Amazon EKS API。
  a. 使用下列命令判斷叢集端點的值。使用您叢集的名稱取代 *my-cluster*。

  ```
  aws eks describe-cluster --name my-cluster --query cluster.endpoint --output text
  ```

  範例輸出如下。

  ```
  https://EXAMPLE108C897D9B2F1B21D5EXAMPLE.sk1.region-code.eks.amazonaws.com
  ```

  b. 使用下列命令判斷叢集憑證授權單位的值。使用您叢集的名稱取代 *my-cluster*。

  ```
  aws eks describe-cluster --name my-cluster --query cluster.certificateAuthority --
  output text
  ```

  傳回的輸出是長字串。

c. 將 NodeConfig 物件`certificateAuthority`中 `apiServerEndpoint`和 的值取代為先前命令輸出中傳回的值。如需在啟動自我管理 Amazon Linux 2023 節點時指定引導引數的詳細資訊，請參閱 the section called “Amazon Linux”和 the section called “Windows”。

- 對於 Linux 節點：

```
---
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="BOUNDARY"

--BOUNDARY
Content-Type: application/node.eks.aws

---
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name: my-cluster
    apiServerEndpoint: https://EXAMPLE108C897D9B2F1B21D5EXAMPLE.sk1.region-
code.eks.amazonaws.com
    certificateAuthority: Y2VydGlmaWNhdGVBdXRob3JpdHk=
    ...
```

如需其他引數，請參閱 GitHub 上的 Bootstrap 指令碼。

- 對於 Windows 節點：

> **Note**
>
> 如果您使用的是自訂服務 CIDR，則需要使用 `-ServiceCIDR` 參數來對其進行指定。否則，叢集中 Pod 的 DNS 解析將會失敗。

```
-APIServerEndpoint cluster-endpoint -Base64ClusterCA certificate-authority
```

如需其他引數，請參閱 the section called “引導指令碼組態參數”。

- 您的叢集 `aws-auth ConfigMap` 必須從 VPC 建立。若要進一步了解如何建立項目並將項目新增至 `aws-auth ConfigMap`，請在終端機中輸入 `eksctl create iamidentitymapping --`

help。如果您的伺服器上不存在 ConfigMap，則 eksctl 會在您使用命令新增身分映射時建立它。

Pod 要求

- Pod 身分識別 - 透過 EKS Pod 身分識別設定的 Pod 會從 EKS 驗證 API 獲取憑證。如果沒有傳出網際網路存取權，則必須針對 EKS 驗證 API 建立並使用 VPC 端點：com.amazonaws.region-code.eks-auth。若要了解 EKS 與 EKS 驗證 VPC 端點的相關詳細資訊，請參閱 the section called “ AWS PrivateLink”。
- IRSA - 為服務帳戶設定 IAM 角色的 Pod 會從 AWS 安全字符服務 (AWS STS) API 呼叫取得憑證。如果沒有傳出網際網路存取，您必須在 VPC 中建立和使用 AWS STS VPC 端點。大多數 AWS v1 SDKs 預設使用全域 AWS STS 端點 (sts.amazonaws.com)，這不使用 AWS STS VPC 端點。若要使用 AWS STS VPC 端點，您可能需要將 SDK 設定為使用區域 AWS STS 端點 (sts.*region-code*.amazonaws.com)。如需詳細資訊，請參閱the section called “STS 端點”。

  若要設定 IRSA，您也需要從 VPC 內部到達叢集的 OIDC 探索/JWKS 端點。例如，當您建立叢集的 IAM OIDC 身分提供者或執行自己的權杖驗證程式時，您會到達它。如果沒有傳出網際網路存取，請為叢集 OIDC 端點建立並使用 VPC 端點：com.amazonaws.*region-code*.oidc-eks。這與 AWS STS VPC 端點不同。STS 從內部擷取叢集的 JWKS AWS，因此 OIDC 端點會私有化 VPC 來源的 OIDC 流量，而 STS 端點則會私有化AssumeRoleWithWebIdentity呼叫。如需詳細資訊，請參閱the section called “使用 AWS PrivateLink 存取叢集 OIDC 端點”。
- 對於 Pod 需要存取的任何 AWS 服務，叢集的 VPC 子網路必須具有 VPC 介面端點。如需詳細資訊，請參閱使用介面 VPC 端點存取 AWS 服務。下表列出了一些常用的服務和端點。如需端點的完整清單，請參閱《 AWSAWS PrivateLink 指南》中的與PrivateLink 整合的 服務。 AWS PrivateLink

  我們建議您為 VPC 端點啟用私有 DNS 名稱，讓工作負載可以繼續使用公有 AWS 服務端點而不會發生問題。

服務	Endpoint
Amazon EC2	com.amazonaws.*region-code* .ec2
Amazon Elastic Container Registry (用於提取容器映像)	com.amazonaws.*region-code* .ecr.api、com.amazonaws.*region-code* .ecr.dkr 和 com.amazonaws.*region-code* .s3

服務	Endpoint
Amazon Application Load Balancer 與 Network Load Balancer	com.amazonaws.*region-code* .elasticloadbalancing
（選用）AWS X-Ray（傳送至 AWS X-Ray 的追蹤需要）	com.amazonaws.*region-code* .xray
(選用) Amazon SSM (需要用於 SSM 代理程式以處理節點管理任務。SSH 的替代方案)	com.amazonaws.*region-code* .ssm
Amazon CloudWatch Logs (需要用於將節點與 Pod 日誌傳送至 Amazon CloudWatch Logs)	com.amazonaws.*region-code* .logs
AWS Security Token Service（針對服務帳戶使用 IAM 角色時需要）	com.amazonaws.*region-code* .sts
Amazon EKS 叢集 OIDC 端點（從 VPC 內部設定服務帳戶的 IAM 角色時需要）	com.amazonaws.*region-code* .oidc-eks
Amazon EKS 驗證 (使用 Pod 身分識別關聯時需要)	com.amazonaws.*region-code* .eks-auth
Amazon EKS	com.amazonaws.*region-code* .eks
Amazon Route 53	com.amazonaws.route53

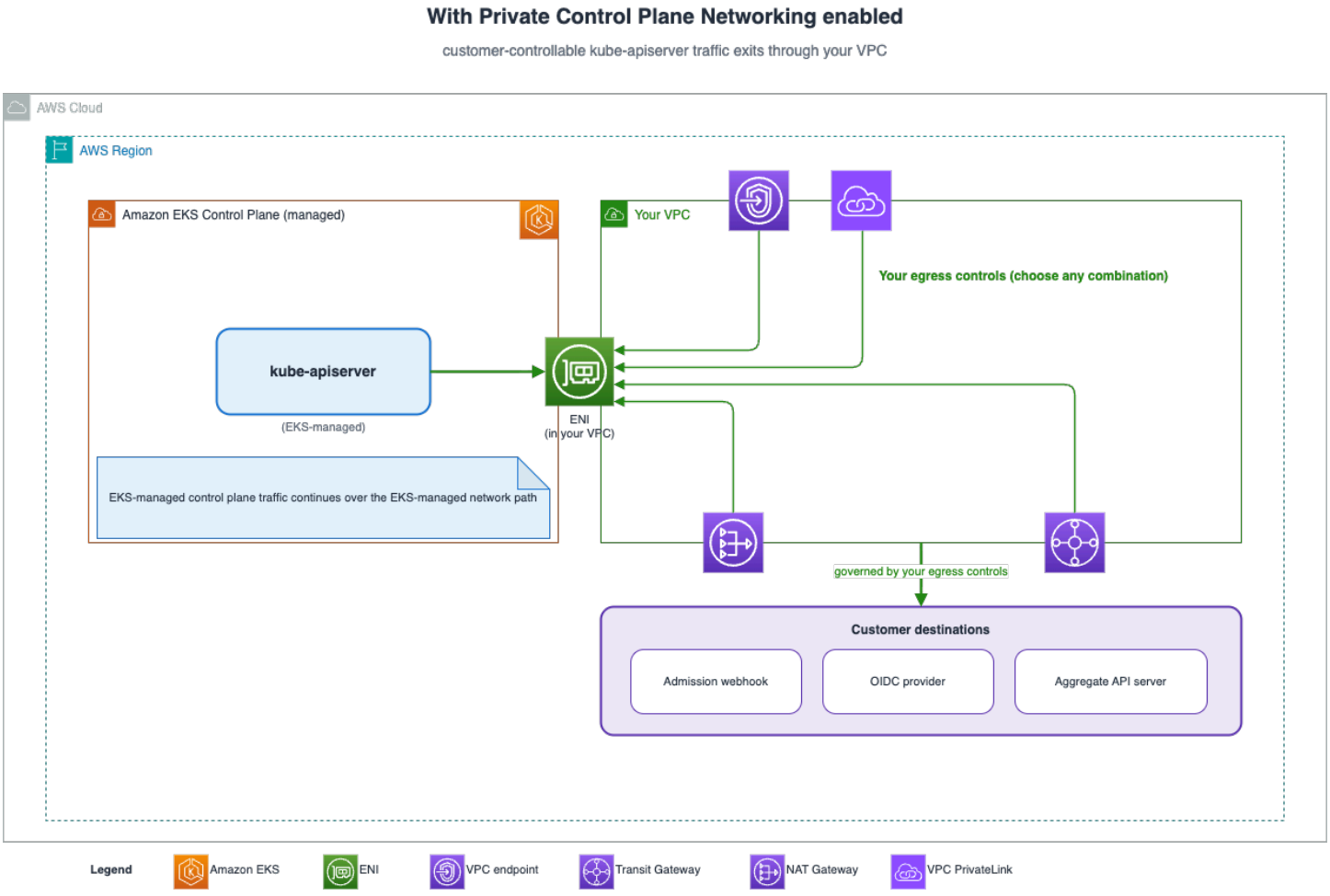
- 任何自我管理節點都必須部署至具有您所需 VPC 介面端點的子網路。如果您建立受管節點群組，VPC 介面端點安全群組必須允許子網路的 CIDR，或者您必須將建立的節點安全群組新增至 VPC 介面端點安全群組。
- EFS 儲存 - 如果您的 Pod 使用 Amazon EFS 磁碟區，則必須先變更EFS驅動程式的 kustomization.yaml 檔案，才能將容器映像設定為使用與 Amazon EKS 叢集相同的 AWS 區域。
- 若使用 EKS 最佳化 AMI，您應啟用如上方資料表所示 `ec2` 端點。或者，您可手動設定節點 DNS 名稱。最佳化 AMI 藉助 EC2 API 來自動設定節點 DNS 名稱。
- 您可以使用AWS Load Balancer控制器，將 AWS Application Load Balancer (ALB) 和 Network Load Balancer 部署到您的私有叢集。部署時，您應使用命令列旗標將 `enable-shield`、`enable-waf` 和 `enable-wafv2` 設定為 false。不支援透過傳入物件的主機名稱進行憑證探索。這是因為控制器需要連線到沒有 VPC 介面端點的 AWS Certificate Manager。

控制器支援具有 IP 目標的 Network Load Balancer，這些目標與 Fargate 一起使用。如需詳細資訊，請參閱the section called “應用程式負載平衡”及the section called “建立 Network Load Balancer”。

- 支援 Cluster Autoscaler。部署 Cluster Autoscaler Pod 時，請確保命令列包含 `--aws-use-static-instance-list=true`。如需詳細資訊，請參閱 GitHub 上的使用靜態執行個體清單。工作者節點 VPC 也必須包含 AWS STS VPC 端點和自動擴展 VPC 端點。
- 有些容器軟體產品會使用 API 呼叫來存取 AWS Marketplace Metering Service 來監控用量。私有叢集不允許這些呼叫，因此您無法將這些容器類型用於私有叢集。

設定控制平面輸出路由

根據預設，Amazon EKS 會管理從 Kubernetes 控制平面到 VPC 資源的輸出聯網。使用控制平面輸出路由來變更此行為，並自行管理網路路徑。這可讓您完全控制來自控制平面彈性網路界面 (ENIs) 的流量如何到達您的 VPC 資源。您可以透過自己的 NAT 閘道、防火牆或檢查設備進行路由。



輸出路由模式

Amazon EKS 支援下列控制平面輸出路由模式：

Mode	說明
AWS_MANAGED	預設行為。Amazon EKS 會管理控制平面 ENIs 的輸出路徑。您不需要為控制平面流量設定 NAT 閘道或其他路由基礎設施。
CUSTOMER_ROUTED	您可以從 VPC 子網路中的控制平面管理輸出路徑。您有責任確保控制平面可以到達所需的端點（例如 Webhook 伺服器、OIDC 提供者和其他資源）。您提供輸出路徑，例如 NAT 閘道、NAT 執行個體、傳輸閘道或防火牆設備。您也可以設定允許此流量的路由表、網路 ACL 和安全群組規則。

> **Important**
>
> 在 CUSTOMER_ROUTED模式下，您有責任確保從控制平面進行適當的網路連線。VPC 網路中的設定錯誤可能會導致控制平面操作失敗。這些設定錯誤包括缺少輸出路徑、限制性網路 ACLs 或不正確的安全群組。受影響的操作包括許可 Webhook 呼叫和 OIDC 身分驗證。

先決條件

您的 VPC 和子網路必須符合標準 Amazon EKS 聯網需求。如需詳細資訊，請參閱the section called “VPC 和子網要求”。

在 CUSTOMER_ROUTED模式下，Kubernetes API 伺服器會透過跨帳戶網路界面傳送面向客戶的流量傳出。Amazon EKS 已在子網路中建立這些界面，用於控制平面到節點的通訊。此流量包含對許可 Webhook 和 OIDC 供應商的呼叫。Amazon EKS 不會建立個別的輸出網路介面。此模式會變更現有界面的使用方式。存放這些界面的子網路必須符合下列要求：

- 子網路必須具有控制平面需要到達的端點路由（例如 Webhook 伺服器和 OIDC 供應商）。對於 VPC 外部的端點，這通常表示輸出裝置的預設路由。預設路由0.0.0.0/0適用於 IPv4，::/0適用於 IPv6。輸出裝置可以是 NAT 閘道、NAT 執行個體、防火牆或傳輸閘道至集中式輸出 VPC。輸出裝置的選擇由您選擇；Amazon EKS 只需要路徑即可運作。

- 跨帳戶網路介面上的安全群組必須允許工作負載所需的連接埠上的傳出流量（例如，Webhook 和 OIDC 提供者的連接埠 443)。
- 子網路上的網路 ACLs 必須允許傳出流量，以及傳回流量的對應傳入暫時性連接埠範圍。

在 CUSTOMER_ROUTED模式下，控制平面會使用 VPC 的 DNS 組態解析主機名稱。這可讓控制平面到達 Route 53 私有託管區域中的端點，以及透過 Route 53 Resolver 端點轉送的內部部署 DNS。

- 您的 VPC DHCP 選項集必須包含AmazonProvidedDNS在其網域名稱伺服器清單中。這是控制平面解析 VPC 內 DNS 名稱的必要項目。如果您的叢集使用具有公有 DNS 名稱的外部 Webhook 端點或 OIDC 供應商，解析程式也必須解析公有主機名稱。確保解析程式可以同時處理 VPC 和公有 DNS 解析。

下表摘要說明控制平面在 CUSTOMER_ROUTED 模式下透過 VPC 傳送的流量：

流量	目標	站點	備註
許可 Webhook	Webhook 端點（客戶定義的 URLs)	443（通常是）	只有在 Webhook 已設定時。如果端點是外部，則透過輸出裝置離開 VPC。
OIDC 探索	OIDC 發行者 URL	443	只有在已設定 OIDC 供應商時。如果發行者位於外部，則透過您的輸出裝置離開 VPC。
彙總 API 伺服器	客戶 API 伺服器端點	443	僅在設定時。如果端點是外部，則透過輸出裝置離開 VPC。
Kubelet API	工作者節點 IP 地址	10250	這是控制平面與節點之間透過叢集 ENI 的流量；它不會周遊您的輸出裝置。它需要路由表、安全群組和網路 ACLs 允許控制平面和節點之間跨叢集 ENI 的流量。

> **Note**
>
> 只有此資料表中列出的流量會受到輸出組態影響。EKS 受管控制平面流量（例如與 etcd、CloudWatch Logs 和內部 EKS 服務的通訊）會透過 AWS 受管網路路徑繼續，不受 VPC 組態影響。

建立具有客戶路由輸出的叢集

您可以在建立新叢集時指定控制平面輸出模式。

Example

AWS CLI

執行下列命令。將*#####*取代為您自己的值。

```
aws eks create-cluster \
    --name my-cluster \
    --role-arn arn:aws:iam::111122223333:role/myAmazonEKSClusterRole \
    --resources-vpc-config "subnetIds=subnet-ExampleID1,subnet-
ExampleID2,securityGroupIds=sg-ExampleID1,controlPlaneEgressMode=CUSTOMER_ROUTED" \
    --kubernetes-network-config "ipFamily=ipv4" \
    --region region-code
```

您可以將 `ipFamily=ipv6`用於 IPv6 叢集。搭配 `CUSTOMER_ROUTED` 模式使用 IPv6 時，請確定子網路除了 IPv4 流量的 NAT 閘道之外，還有 IPv6 流量的輸出限定網際網路閘道。

Example

AWS 管理主控台

1. 開啟 Amazon EKS 主控台。
2. 選擇 Add cluster (新增叢集)，然後選擇 Create (建立)。
3. 在聯網頁面上，針對控制平面輸出，選取客戶路由。
4. 完成剩餘的叢集組態，然後選擇建立。

For AWS CloudFormation，在 `ControlPlaneEgressMode: CUSTOMER_ROUTED`中設定 `ResourcesVpcConfig`。未來版本的AWS 提供者將可使用此欄位的 Terraform 支援。

> **Note**
>
> 切換到 CUSTOMER_ROUTED 是單向操作。在叢集上啟用客戶路由輸出之後，您就無法還原為 AWS_MANAGED。

更新現有叢集

您可以使用 update-cluster-config命令變更現有叢集上的控制平面輸出模式。

```
aws eks update-cluster-config \
    --name my-cluster \
    --resources-vpc-config "controlPlaneEgressMode=CUSTOMER_ROUTED" \
    --region region-code
```

監控更新狀態：

```
aws eks describe-update \
    --name my-cluster \
    --update-id update-id \
    --region region-code
```

當狀態顯示 時，更新即完成Successful。更新類型為 ControlPlaneEgressUpdate。更新通常會在 10 分鐘內完成。

> **Important**
>
> 切換到 CUSTOMER_ROUTED 是單向操作。在叢集上啟用客戶路由輸出之後，您就無法還原為 AWS_MANAGED。
> 切換之前，請確認您的 VPC 符合 中的要求the section called “先決條件”。如果控制平面在更新後失去與所需端點的連線，許可 Webhook 呼叫和 OIDC 身分驗證等操作可能會失敗。

IPv6 考量因素

如果您使用客戶路由輸出執行 IPv6 叢集，則需要同時設定 IPv4 和 IPv6 輸出路徑。

使用輸出執行 IPv6 叢集 (ipFamily=ipv6) CUSTOMER_ROUTED 時：

- 控制平面 ENIs 會同時指派 IPv4 和 IPv6 地址。
- 您必須同時設定 IPv4 和 IPv6 輸出路徑：
  - IPv4：輸出裝置的預設路由 (`0.0.0.0/0`)（例如 NAT 閘道）。
  - IPv6：到 IPv6 輸出裝置的`::/0`路由（例如，僅限輸出網際網路閘道）。
- 安全群組和 NACLs 必須允許兩個 IP 版本的流量。
- 如果您的 OIDC 提供者或 Webhook 端點僅限 IPv4-only，請確定 IPv4 NAT 正常運作。

考量事項

當您使用客戶路由控制平面輸出時，請記住下列幾點：

- 您的責任：在 CUSTOMER_ROUTED模式下，您擁有從控制平面到外部端點的網路路徑。如果該路徑中斷，控制相依於它的平面操作（例如許可 Webhook 呼叫和 OIDC 身分驗證）可能會失敗，直到您還原連線為止。
- VPC 內部流量不受影響：控制平面與節點之間的流量（例如，連接埠 10250 上的 kubelet API) 透過叢集 ENI 並不取決於您的輸出裝置。
- EKS Auto Mode：控制平面輸出路由在標準和自動模式叢集上的運作方式相同，因為控制平面架構相同。
- EKS 功能：EKS 功能（例如 ArgoCD、ACK 和 KRO) 會在不同的 AWS 受管基礎設施中執行。此功能不會透過 VPC 路由來自 EKS 功能控制器的流量。
- 可觀測性：如果您在 VPC 或叢集子網路上啟用 VPC 流程日誌，則可以觀察透過 VPC 路由的輸出流量。這包括對 Webhook 和 OIDC 端點的呼叫。如果未啟用 VPC 流程日誌，則不會記錄此流量。

IAM 條件索引鍵

Amazon EKS 支援 `eks:controlPlaneEgressMode`條件金鑰。您可以在 IAM 政策或服務控制政策 (SCPs) 中使用此金鑰，以控制發起人在建立或更新叢集時可以指定哪些輸出模式發起人。

條件索引鍵適用於下列動作：

- `eks:CreateCluster`
- `eks:UpdateClusterConfig`

例如，除非發起人指定 ，否則下列 SCP 會拒絕叢集建立和組態更新CUSTOMER_ROUTED：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "RequireCustomerRoutedControlPlane",
      "Effect": "Deny",
      "Action": [
        "eks:CreateCluster",
        "eks:UpdateClusterConfig"
      ],
      "Resource": "*",
      "Condition": {
        "StringNotEquals": {
          "eks:controlPlaneEgressMode": "CUSTOMER_ROUTED"
        }
      }
    }
  ]
}
```

使用此政策可強制執行組織中所有新的和更新的叢集都使用CUSTOMER_ROUTED輸出模式。

OIDC 提供者組態

如果您的叢集使用 OIDC 身分提供者，控制平面必須透過 HTTPS （連接埠 443) 到達 OIDC 探索端點。這適用於服務帳戶的 IAM 角色，或您為叢集身分驗證建立關聯的 OIDC 身分提供者。沒有 OIDC 特定的設定；它使用您在先決條件中設定的相同輸出路徑。若要允許它：

1. 確認控制平面子網路具有涵蓋 OIDC 端點的路由 （通常是輸出裝置的預設路由，例如 NAT 閘道）。
2. 確認叢集安全群組允許傳出 TCP 443。
3. 確認子網路 NACLs傳出 TCP 443 和傳入暫時性傳回流量 （連接埠 1024–65535)。

端點取決於您的提供者：

- Amazon EKS OIDC 提供者 （預設）： `oidc.eks.region-code.amazonaws.com`
- 自訂 OIDC 供應商：您設定的發行者 URL。

如果 OIDC 身分驗證失敗，如需疑難排解步驟，請參閱 the section called “無法連線 OIDC 提供者”。

驗證連線

設定CUSTOMER_ROUTED輸出後，請確認控制平面可以到達您的 VPC 資源：

1. 檢查目前的輸出模式：確認叢集使用您預期的模式。

```
aws eks describe-cluster --name my-cluster \
    --query "cluster.resourcesVpcConfig.controlPlaneEgressMode" \
    --region region-code
```

2. 檢查叢集狀態：叢集應處於 ACTIVE 狀態。

```
aws eks describe-cluster --name my-cluster --query "cluster.status" --region region-
code
```

3. 測試 Webhook 連線：如果您已設定許可 Webhook，請建立可觸發 Webhook 並確認 Webhook 成功的資源。
4. 驗證節點註冊：啟動節點並確認其成功加入叢集。

```
kubectl get nodes
```

5. 檢查 OIDC：如果針對服務帳戶 (IRSA) 使用 IAM 角色，請確認 Pod 可以擔任其 IAM 角色。

如需常見問題的故障診斷，請參閱 the section called “對輸出進行故障診斷”。

在 GitHub 上編輯此頁面

對控制平面輸出問題進行故障診斷

使用CUSTOMER_ROUTED控制平面輸出模式時，您必須負責控制平面 ENIs 的網路連線。此頁面涵蓋常見問題及其解決方案。

偵測失敗的 Webhook

當控制平面無法連線到 Webhook 伺服器或 OIDC 供應商時，症狀通常會顯示為 Webhook 逾時。若要確認，請建立或修改觸發 Webhook 的資源，並檢查錯誤：

```
kubectl apply -f my-resource.yaml
```

連線或 DNS 失敗通常會傳回類似以下的錯誤：

```
Error from server (InternalError): error when creating "my-resource.yaml": Internal
 error occurred:
failed calling webhook "my-webhook.example.com": failed to call webhook:
Post "https://my-webhook.example.com/validate?timeout=10s": context deadline exceeded
```

您也可以檢查叢集中最近的事件是否有 Webhook 錯誤：

```
kubectl get events --all-namespaces --field-selector reason=FailedCreate
```

- 如果錯誤是逾時 (`context deadline exceeded`) 或連線遭拒，則控制平面無法連線到 Webhook 端點。請參閱the section called “沒有輸出路由到所需的端點”、the section called “封鎖 Webhook 或控制平面流量NACLs”和the section called “防止存取的安全群組”。
- 如果錯誤提及 DNS 或沒有此類主機失敗，則控制平面無法解析端點。請參閱 the section called “DHCP 選項集重新整理失敗”。

沒有輸出路由到所需的端點

徵狀：

- 許可 Webhook 逾時。
- OIDC 提供者探索失敗。
- 叢集建立或更新停滯。

原因：

控制平面網路介面子網路沒有控制平面需要到達的端點的工作路由。最常見的是，子網路路由表缺少輸出裝置的預設路由。或者，該裝置設定錯誤。輸出裝置通常是 NAT 閘道。不過，它可以是 NAT 執行個體、防火牆或代理設備，或是集中式輸出 VPC 的傳輸閘道。

解決方案：

1. 識別您的叢集用於控制平面網路介面的子網路：

```
aws eks describe-cluster --name my-cluster \
    --query "cluster.resourcesVpcConfig.subnetIds"
```

2. 針對每個子網路，檢查相關聯的路由表：

```
aws ec2 describe-route-tables \
    --filters "Name=association.subnet-id,Values=subnet-ExampleID1"
```

3. 確認您的輸出裝置指向 `0.0.0.0/0`（或涵蓋端點的路由）的路由存在。如果遺失，請新增路由。下列範例新增 NAT 閘道路由；取代您自己的輸出目標（例如傳輸閘道或網路界面）：

```
aws ec2 create-route \
    --route-table-id rtb-ExampleID \
    --destination-cidr-block 0.0.0.0/0 \
    --nat-gateway-id nat-ExampleID
```

封鎖 Webhook 或控制平面流量NACLs

徵狀：

- 許可 Webhook 呼叫逾時（錯誤：`failed calling webhook`)。
- 建立或修改使用變動或驗證 Webhook 的 Kubernetes 資源時發生間歇性失敗。

原因：

控制平面 ENI 子網路上的網路 ACLs 會封鎖 Webhook 端點的傳出流量，或封鎖傳入暫時性連接埠傳回流量。

解決方案：

1. 識別與您的控制平面子網路相關聯的 NACLs：

```
aws ec2 describe-network-acls \
    --filters "Name=association.subnet-id,Values=subnet-ExampleID1"
```

2. 確保存在下列規則：

Direction	通訊協定	連接埠範圍	目的地/來源	Action
傳出	TCP	443	0.0.0.0/0（或 Webhook CIDR)	允許
傳出	TCP	10250	VPC CIDR	允許

Direction	通訊協定	連接埠範圍	目的地/來源	Action
傳入	TCP	1024–65535	0.0.0.0/0	允許（暫時性傳回流量）

> **Note**
>
> NACLs 是無狀態的。您必須明確允許傳入規則中暫時性連接埠 (1024–65535) 上的傳回流量。
> 這些規則涵蓋兩個不同的路徑。連接埠 443 規則適用於傳出流量到 Webhook 和 OIDC 端點，這會透過您的輸出裝置離開 VPC。連接埠 10250 規則適用於 kubelet API，其會在控制平面和節點之間保留在您的 VPC 內。遺失的輸出裝置不會影響連接埠 10250，但限制性網路 ACL 可以封鎖它。

防止存取的安全群組

徵狀：

- Webhook 呼叫失敗。
- 控制平面無法連線到節點上的 kubelet API（連接埠 10250)。
- `kubectl exec`、`kubectl logs`或 `kubectl port-forward` 失敗。

原因：

連接到控制平面 ENIs 的安全群組 (叢集安全群組) 不允許必要連接埠上的傳出流量。

解決方案：

1. 識別叢集安全群組：

```
aws eks describe-cluster --name my-cluster \
    --query "cluster.resourcesVpcConfig.clusterSecurityGroupId"
```

2. 驗證傳出規則允許：

通訊協定	站點	目標
TCP	443	0.0.0.0/0 (webhook 端點、OIDC 供應商)
TCP	10250	節點安全群組或 VPC CIDR (kubelet API)

3. 如果傳出規則具有限制性，請新增所需流量的規則：

```
aws ec2 authorize-security-group-egress \
    --group-id sg-ExampleClusterSG \
    --protocol tcp \
    --port 443 \
    --cidr 0.0.0.0/0
```

> **Note**
>
> 如果您有嚴格的輸出要求，而且您知道 Webhook 和 OIDC 端點的 IP 範圍，您可以將連接埠 443 規則範圍限定為這些特定的 CIDRs，而不是 `0.0.0.0/0`。連接埠 10250 (kubelet API) 規則為 VPC 內部；將其範圍限定於節點安全群組或 VPC CIDR，而非網際網路。

DHCP 選項集重新整理失敗

徵狀：

- DNS 解析從控制平面失敗。
- 需要 DNS 查詢 (OIDC 探索、Webhook 解析) 的叢集操作失敗。
- 問題會在 VPC DHCP 選項變更後或控制平面更新後出現。

原因：

VPC DHCP 選項集已變更。或者，它不包括在其網域名稱伺服器`AmazonProvidedDNS`中。它也可能缺少另一個解析程式，可以解析控制平面所需的名稱。控制平面會自動偵測 DHCP 選項集變更，並套用新的 DNS 設定，通常在一小時內。只有在叢集 IAM 角色授予所需的 Amazon EC2 讀取許可時，控制平面才能執行此操作。

解決方案：

1. 驗證 VPC 的 DHCP 選項集：

```
aws ec2 describe-vpcs --vpc-ids vpc-ExampleID \
    --query "Vpcs[0].DhcpOptionsId" \
    --region region-code
```

```
aws ec2 describe-dhcp-options --dhcp-options-ids dopt-ExampleID --region region-code
```

2. 確認 `domain-name-servers` 包含 `AmazonProvidedDNS`(Amazon 提供的 DNS 解析程式，這是 VPC IPv4 CIDR 的基礎加上兩個），或另一個解析程式可以解析控制平面所需的名稱。
3. 確認叢集 IAM 角色授予 `ec2:DescribeVpcs`和 `ec2:DescribeDhcpOptions`。如果沒有這些許可，控制平面將無法讀取更新的 DHCP 選項，也無法重新整理其 DNS 設定。如需詳細資訊，請參閱 Amazon EKS 叢集 IAM 角色。
4. DHCP 選項變更後，控制平面最多需要一小時的時間來自動偵測和套用新設定。不需要叢集更新或執行個體替換。如果 DNS 解析在一小時後仍然失敗，且上述許可已就緒，請聯絡 AWS Support。

IPv6 路由問題

徵狀：

- IPv6 叢集無法連線到外部 OIDC 或 Webhook 端點。
- 節點註冊可透過 IPv4 運作，但 IPv6 服務失敗。

原因：

子網路路由表缺少僅輸出網際網路閘道的`::/0`路由，或安全群組/NACLs不允許 IPv6 流量。

解決方案：

1. 驗證輸出限定網際網路閘道是否存在，並連接到 VPC：

```
aws ec2 describe-egress-only-internet-gateways \
    --filters "Name=attachment.vpc-id,Values=vpc-ExampleID"
```

2. 檢查控制平面子網路的路由表是否具有`::/0`路由：

```
aws ec2 describe-route-tables \
```

```
    --filters "Name=association.subnet-id,Values=subnet-ExampleID1" \
    --query "RouteTables[0].Routes[?DestinationIpv6CidrBlock=='::/0']"
```

3. 如果遺失，請新增路由：

```
aws ec2 create-route \
    --route-table-id rtb-ExampleID \
    --destination-ipv6-cidr-block ::/0 \
    --egress-only-internet-gateway-id eigw-ExampleID
```

4. 確保 NACLs 和安全群組允許連接埠 443 和傳入暫時性連接埠上的 IPv6 傳出。

無法連線 OIDC 提供者

徵狀：

- `IAM roles for service accounts` (IRSA) 失敗 — Pod 無法擔任角色。
- 叢集事件會顯示 OIDC 探索錯誤。

原因：

控制平面無法到達 OIDC 提供者端點（例如 `oidc.eks.region-code.amazonaws.com`），因為輸出遭到封鎖。

解決方案：

1. 確認輸出路徑和路由表允許傳出 HTTPS 流量。如需輸出路由遺失或設定錯誤時的疑難排解步驟，請參閱 the section called “沒有輸出路由到所需的端點”。
2. 確認叢集安全群組允許傳出 TCP 443 `0.0.0.0/0`（請參閱 the section called “防止存取的安全群組”）。

在 GitHub 上編輯此頁面

透過 Karpenter 與 Cluster Autoscaler 擴展叢集運算資源

自動擴展功能可自動將您的資源向內外擴展，滿足不斷變化的需求。這是 Kubernetes 的主要功能，否則需要大量人力資源才能手動執行。

EKS 自動模式

Amazon EKS 自動模式會自動擴展叢集運算資源。若現有節點無法容納 Pod，EKS 自動模式會建立新節點。另外，EKS 自動模式也會合併工作負載，並刪除多餘節點。EKS 自動模式的實現基礎是 Karpenter。

如需詳細資訊，請參閱：

- EKS 自動模式
- the section called “建立節點集區”
- the section called “部署 inflate 工作負載”

其他解決方案

Amazon EKS 支援另外兩種自動擴展產品：

Karpenter

Karpenter 是一款靈活的高效能 Kubernetes 叢集自動擴展程式，可協助提升應用程式的可用性和叢集效率。Karpenter 會在一分鐘內啟動適當大小的運算資源 (例如 Amazon EC2 執行個體)，以回應不斷變化的應用程式負載。透過將 Kubernetes 與 AWS 整合，Karpenter 可以佈建精確符合工作負載需求的即時運算資源。Karpenter 會根據叢集工作負載的特定需求，自動佈建新的運算資源。其中包括運算、儲存、加速和排程需求。Amazon EKS 支援使用 Karpenter 的叢集，然而 Karpenter 可與所有符合標準的 Kubernetes 叢集搭配使用。如需詳細資訊，請參閱 Karpenter 文件。

> **Important**
>
> Karpenter 屬於開源軟體，AWS 客戶需自行負責在其 Kubernetes 叢集中安裝、設定與管理。若在 Amazon EKS 叢集中執行未經修改的相容版本 Karpenter，AWS 將提供技術支援。與其他客戶管理的軟體相同，客戶必須確保 Karpenter 控制器的可用性與安全性，並在升級 Karpenter 或其執行所在的 Kubernetes 叢集時，執行適當的測試程序。此外，AWS 未針對 Karpenter 提供服務水準協議 (SLA)，客戶需自行確保 Karpenter 啟動的 EC2 執行個體符合其業務需求。

Cluster Autoscaler

當 Pod 故障或重新排程到其他節點時，Kubernetes Cluster Autoscaler 會自動調整叢集中的節點數量。Cluster Autoscaler 使用 Auto Scaling 群組。如需詳細資訊，請參閱 AWS 上的 Cluster Autoscaler。

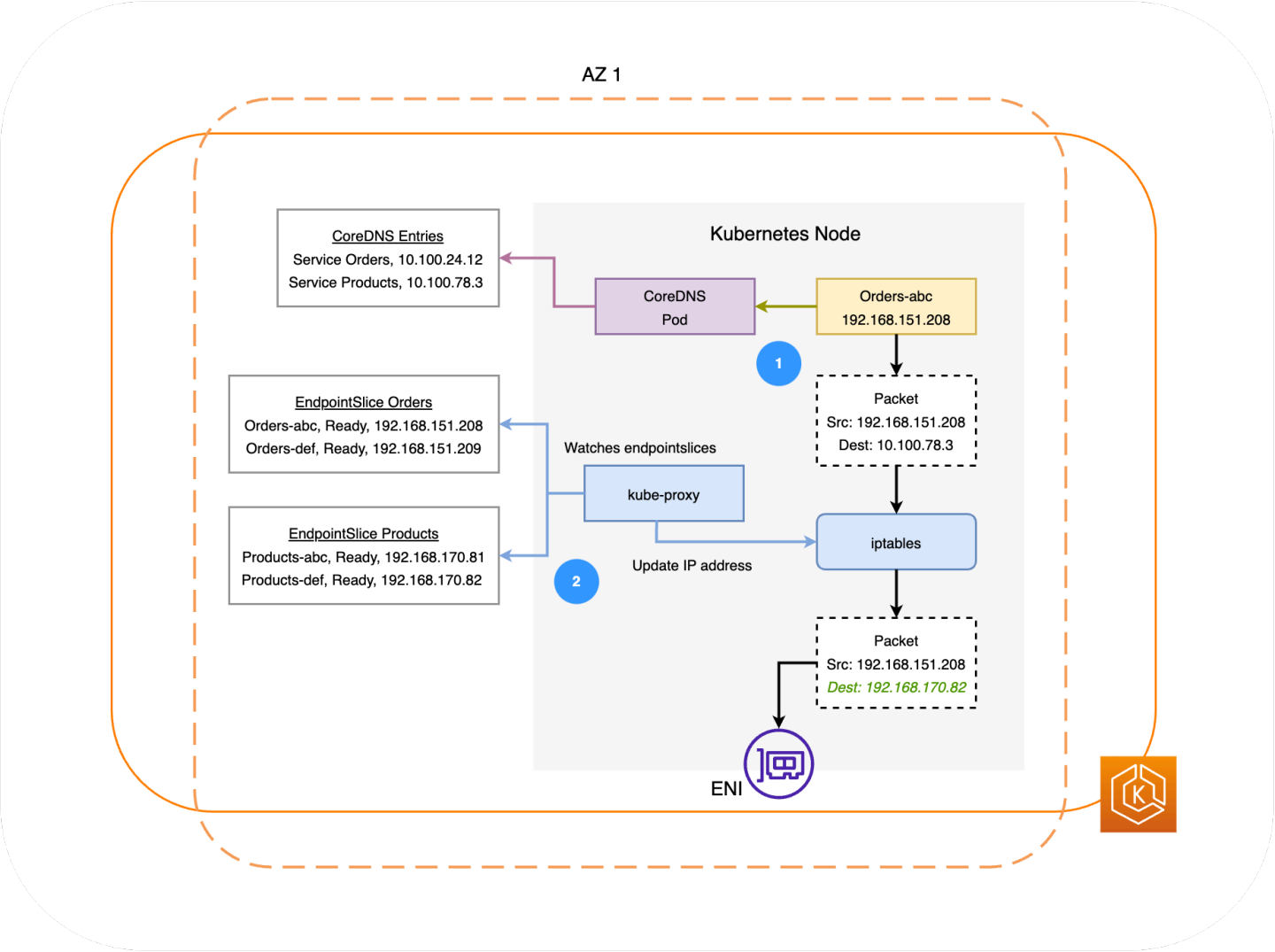
了解 Amazon EKS 中的 Amazon 應用程式復原控制器 (ARC) 區域轉移

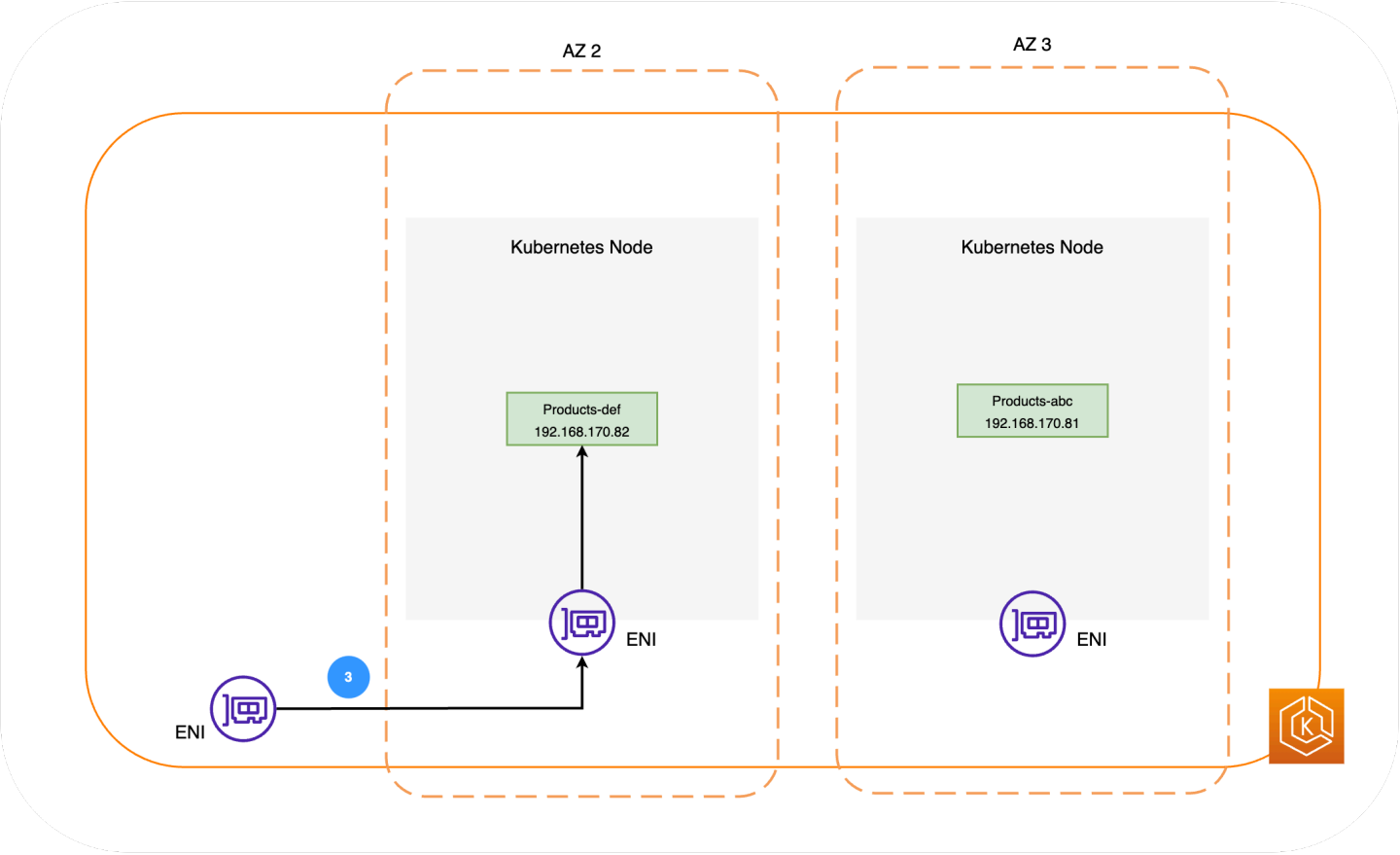
Kubernetes 提供各種原生功能，可支援您針對運作狀態降級或可用區域 (AZ) 受損等事件，讓應用程式具備更強的復原能力。在 Amazon EKS 叢集中執行工作負載時，您可使用 Amazon 應用程式復原控制器 (ARC) 區域轉移或區域自動轉移，進一步改善應用程式環境的容錯能力與應用程式復原能力。ARC 區域轉移旨在作為一項臨時措施，讓您將資源的流量遠離受損的 AZ，直到區域轉移到期或您將它取消為止。您可在必要時擴充區域轉移。

您可以為 EKS 叢集啟動區域轉移，也可以透過啟用區域自動轉移 AWS 來允許 為您轉移流量。藉助此區域轉移，可將叢集中的東向西網路流量更新為只考慮正常運作 AZ 中工作節點上執行之 Pod 的網路端點。另外，任何 ALB 或 NLB 在處理 EKS 叢集中應用程式的輸入流量時，都會將流量路由自動至運作狀態良好的可用區域中的目標。若客戶尋求實現最高可用性目標，在可用區域受損時，能夠在復原之前將所有流量從受損的可用區域移出非常重要。因此，您亦可啟用具備 ARC 區域轉移的 ALB 或 NLB。

了解 Pod 間的東西網路流量

下面的圖表闡述了兩個範例工作負載：訂單與產品。此範例的用途是顯示不同可用區域中的工作負載與 Pod 如何通訊。


AZ 1
CoreDNS Entries
Service Orders, 10.100.24.12
Service Products, 10.100.78.3
Kubernetes Node
CoreDNS
Pod
Orders-abc
192.168.151.208
1
Packet
Src: 192.168.151.208
Dest: 10.100.78.3
EndpointSlice Orders
Orders-abc, Ready, 192.168.151.208
Orders-def, Ready, 192.168.151.209
Watches endpointslices
kube-proxy
iptables
EndpointSlice Products
Products-abc, Ready, 192.168.170.81
Products-def, Ready, 192.168.170.82
2
Update IP address
Packet
Src: 192.168.151.208
Dest: 192.168.170.82
ENI



1. 為了讓訂單與產品通訊，訂單必須先解析目的地服務的 DNS 名稱。訂單與 CoreDNS 通訊，以便擷取該服務的虛擬 IP 位址 (叢集 IP)。訂單解析產品服務名稱之後，即會傳送流量至該目標 IP 位址。
2. kube-proxy 可在叢集的每個節點上執行，以及持續監看 EndpointSlices 的服務。建立服務後，EndpointSlice 控制器在後台建立及管理 EndpointSlice。每個 EndpointSlice 皆有端點清單或資料表，其中包含 Pod 位址子集，及其正在執行的節點。kube-proxy 可在節點上，使用 `iptables` 為這些 Pod 端點的每一個設定路由規則。此外，kube-proxy 還負責基本形式的負載平衡，以便重新引導目的地為服務叢集 IP 位址的流量，轉而直接傳送至 Pod 的 IP 位址。kube-proxy 可在傳出連線上重寫目的地 IP 位址，藉此來執行此操作。
3. 如先前圖表所示，隨後藉助各自節點上的 ENI，將網路封包傳送至可用區域 2 中的產品 Pod。

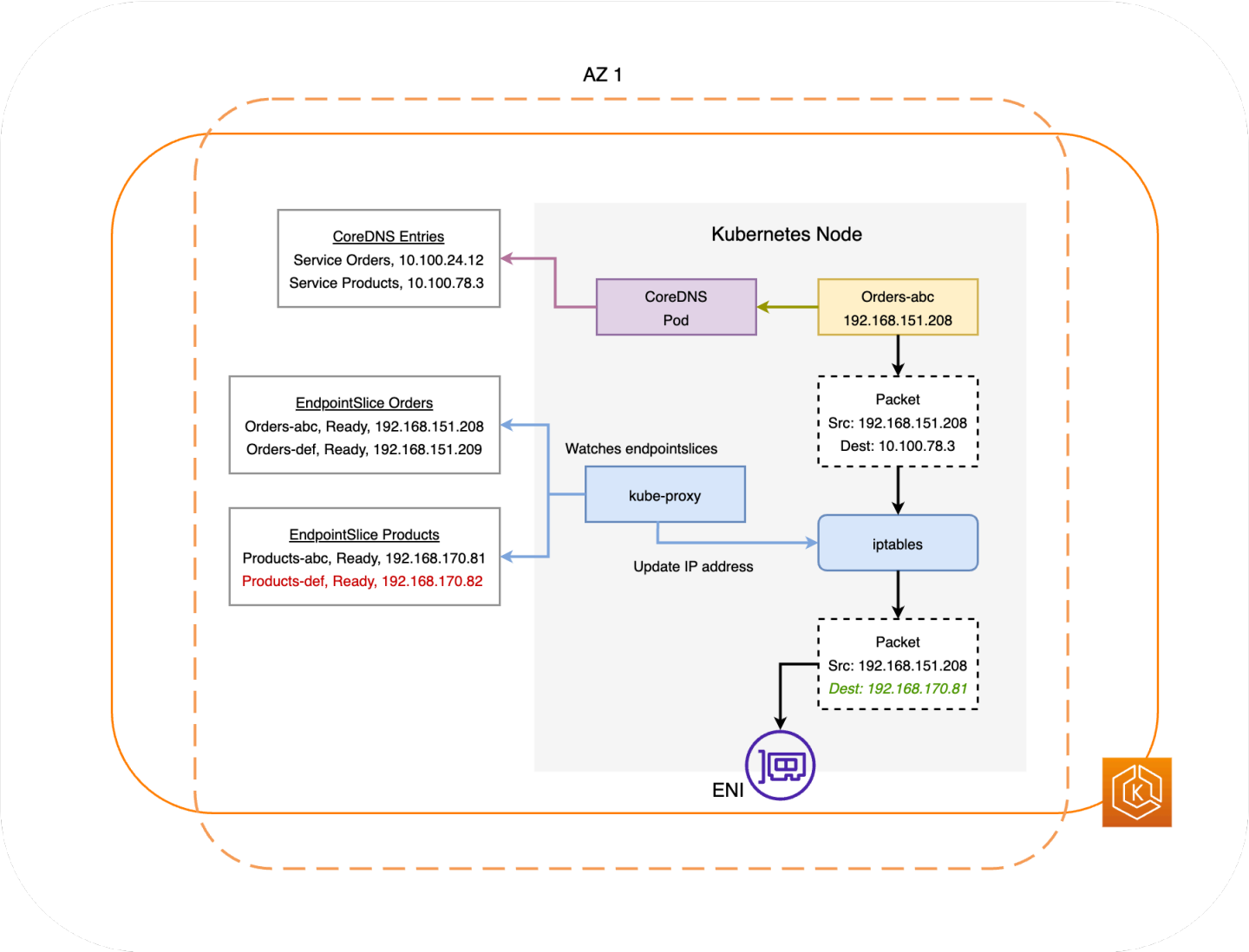
了解 Amazon EKS 中的 ARC 區域轉移

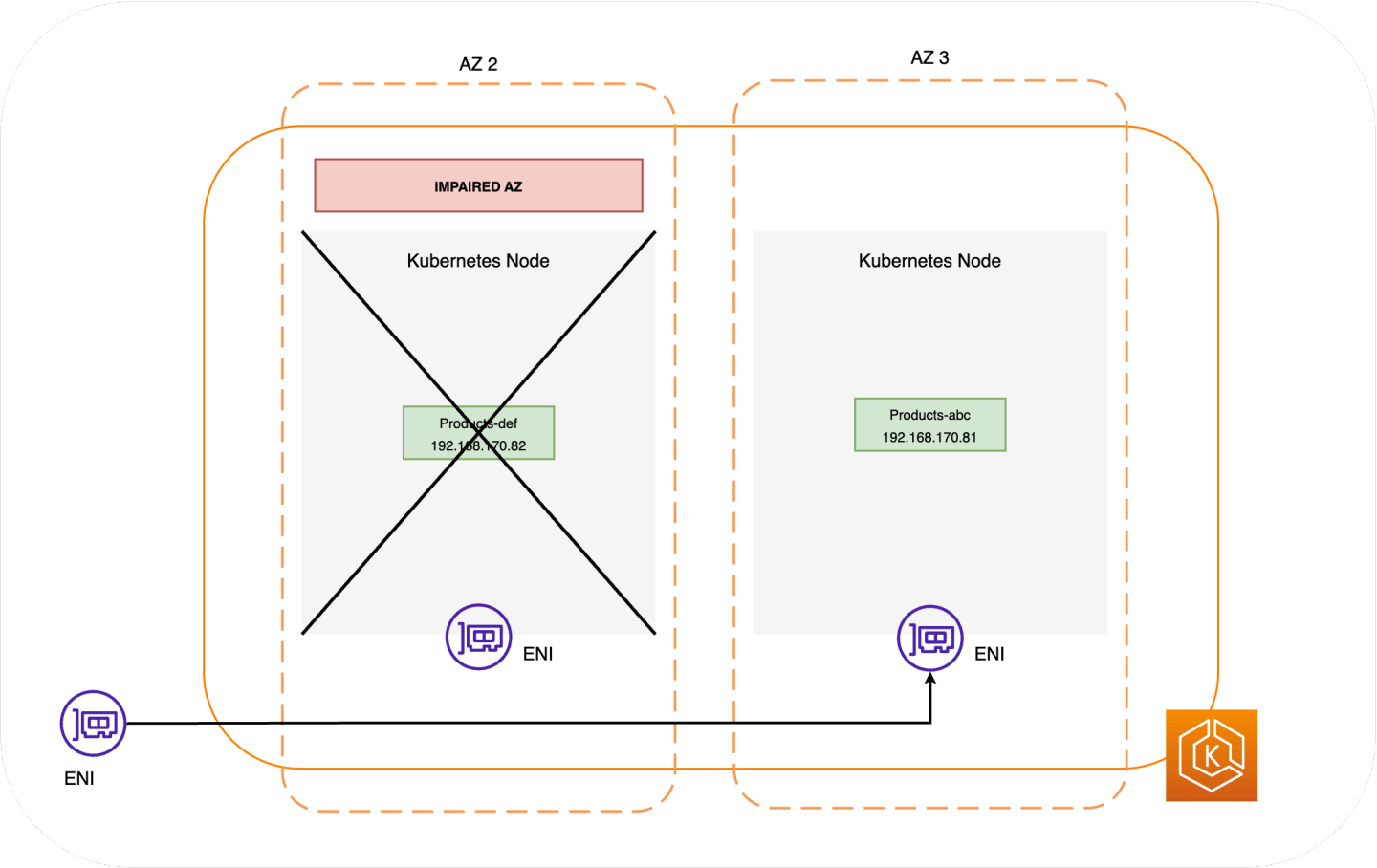
若您的環境中有可用區域受損，您可針對 EKS 叢集環境起始區域轉移。或者，您可以允許 使用區域自動轉移來 AWS 管理轉移流量。使用區域自動轉移， AWS 可監控整體 AZ 運作狀態，並透過自動將流量從叢集環境中受損的 AZ 轉移來回應潛在的 AZ 損害。

Amazon EKS 叢集使用 ARC 啟用區域轉移後，您可以使用 ARC 主控台、CLI 或區域轉移和區域自動轉移 APIs AWS 來啟動區域轉移或啟用區域自動轉移。EKS 區域轉移期間會自動執行以下操作：

- 封鎖受影響可用區域中的所有節點。此操作可阻止 Kubernetes 排程器將新的 Pod 排程到運作狀態不佳可用區域中的節點。
- 如果您使用的是 EKS Auto Mode，EKS Auto Mode 會自動停止在受損的 AZ 中佈建新節點。EKS Auto Mode 也會暫停自願中斷動作，例如會影響受損 AZ 的合併和偏離，這會停止受損 AZ 中的自願執行個體終止，以防止重試風暴。具有嚴格排程需求的 Pod，以受損的 AZ 為目標（例如持久性磁碟區繫結、節點親和性或嚴格的拓撲分散限制）不會導致對該 AZ 的新節點啟動嘗試。
- 若您使用受管節點群組，可用區域重新平衡則會暫停，且 Auto Scaling 群組會更新，進而確保新的 EKS 資料平面節點僅在運作狀態良好的可用區域啟動。
- 不會終止運作狀態不佳可用區域中的節點，亦不會從節點中移出 Pod。這可確保區域轉移到期或取消時，流量可安全地返回可用區域，以便獲取完整容量。
- EndpointSlice 控制器可在受損的可用區域中尋找全部 Pod 端點，然後從相關的 EndpointSlices 中將其移除。此操作可確保僅鎖定運作狀態良好可用區域中的 Pod 端點，來接收網路流量。若區域轉移取消或到期，EndpointSlice 控制器將更新 EndpointSlices，以便在還原的可用區域中納入這些端點。

下面的圖表提供了一個高階概觀，闡述 EKS 區域轉移如何確保叢集環境中僅鎖定運作狀態良好的 Pod 端點。


AZ 1
Kubernetes Node
CoreDNS Entries
Service Orders, 10.100.24.12
Service Products, 10.100.78.3
CoreDNS
Pod
Orders-abc
192.168.151.208
Packet
Src: 192.168.151.208
Dest: 10.100.78.3
EndpointSlice Orders
Orders-abc, Ready, 192.168.151.208
Orders-def, Ready, 192.168.151.209
Watches endpointslices
kube-proxy
EndpointSlice Products
Products-abc, Ready, 192.168.170.81
Products-def, Ready, 192.168.170.82
Update IP address
iptables
Packet
Src: 192.168.151.208
Dest: 192.168.170.81
ENI



EKS 區域轉移要求

> **Important**
>
> 區域轉移會將所有叢集內流量移離可用區域。如果您的工作負載未分散在具有足夠複本AZs，啟動區域轉移本身可能會對您的應用程式造成可用性影響，因為沒有運作狀態良好的端點可接收流量。在啟用區域自動轉移或啟動手動區域轉移之前，您應該驗證叢集環境是否可以使用較少的可用區域來操作。

為了將區域轉移與 EKS 成功地配合使用，必須事先設定叢集環境，以便具備復原能力應對可用區域受損。下面的清單列示了可協助確保復原能力的組態選項。

- 跨多個可用區域佈建叢集的工作節點
- 佈建足夠的運算容量來適應單一可用區域移除
- 在每個可用區域預先擴展包括 CoreDNS 在內的 Pod
- 將多個 Pod 複本分佈到所有可用區域，有助於確保從單一可用區域移出時，您仍有充足的容量

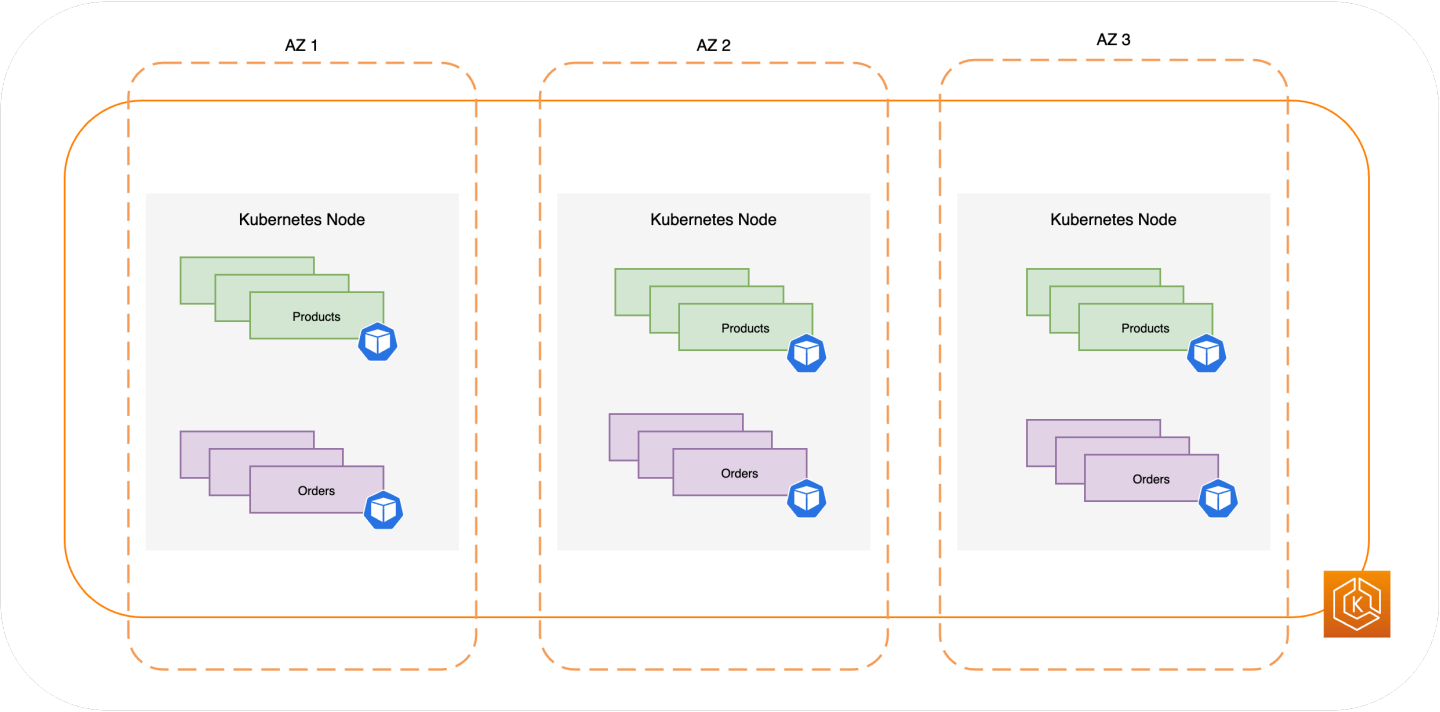
- 在相同的可用區域主機代管相依或相關的 Pod
- 手動啟動從可用區域區域移出，藉此測試在沒有一個可用區域的情況下，您的叢集環境是否如預期運作。或者，您可啟用區域自動轉移，以及依賴於自動轉移實際執行。手動或實際區域轉移測試，無須在 EKS 中運作區域轉移，但強烈建議這樣做。

跨多個可用區域佈建 EKS 工作節點

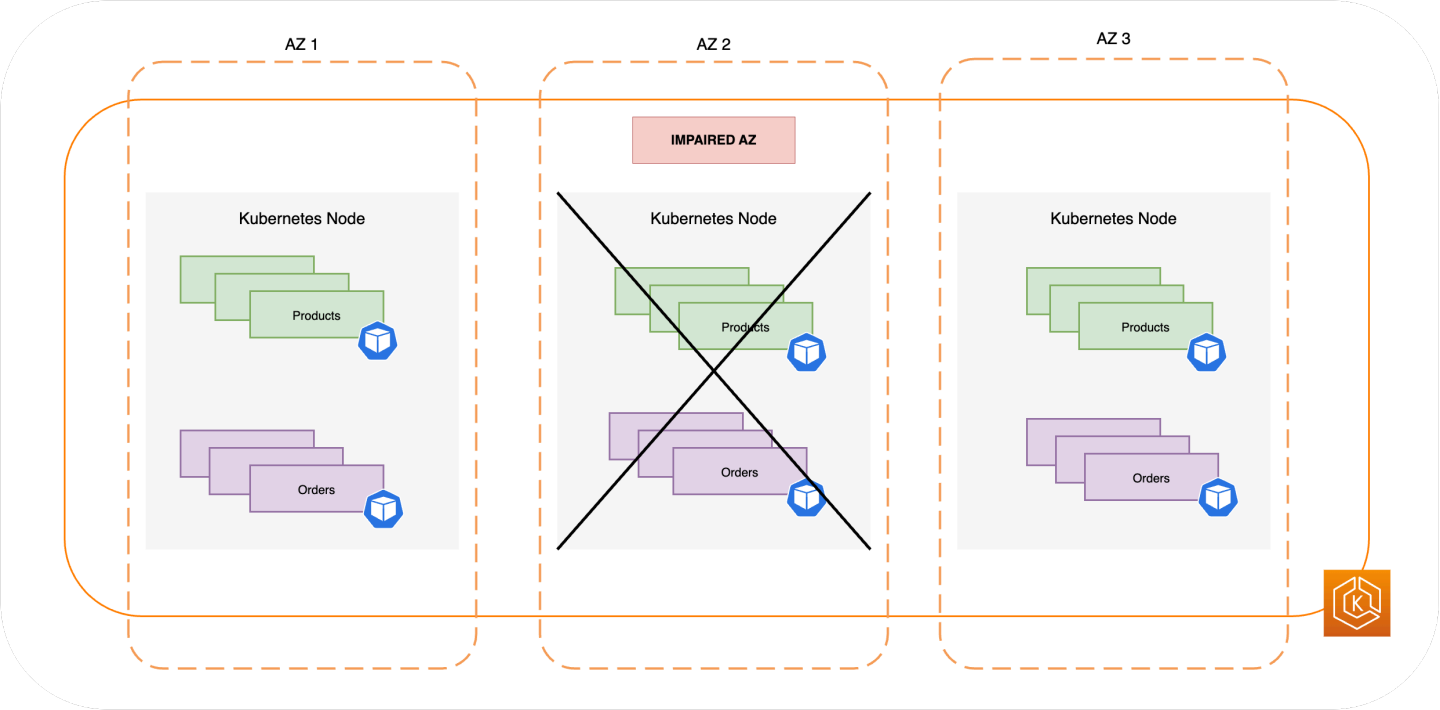
AWS 區域具有多個不同的位置，具有實體資料中心，稱為可用區域 (AZs)。設置可用區域旨在彼此進行實體隔離，以避免可能同時影響整個區域。佈建 EKS 叢集後，建議您將工作節點部署至區域中的多個可用區域。此操作有助於讓叢集環境具備更強的復原能力來應對單一可用區域受損，還可確保在其他可用區域執行的應用程式具有高可用性。當您開始從受影響的可用區域轉移區域時，EKS 環境的叢集內網路會自動更新為僅使用運作狀態良好的可用AZs，以協助維持叢集的高可用性。

確保 EKS 環境具有多可用區域設定，能夠增強系統的整體可靠性。然而，多可用區域環境會影響應用程式資料的傳輸及處理方式，進而影響環境的網路費用。具體來說，頻繁輸出跨區域流量 (在可用區域間分散的流量) 可能會對網路相關成本產生重大影響。您可運用不同的策略，來控制 EKS 叢集中 Pod 間的跨區域流量，進而降低關聯成本。若要了解如何在執行高可用性 EKS 環境時最佳化網路成本的詳細資訊，請參閱這些最佳實務。

下面的圖表闡述了三個運作狀態良好可用區域的高可用性 EKS 環境。



下面的圖表闡述三個可用區域的 EKS 環境如何具備復原能力來應對可用區域受損，以及確保高可用性，因為還有兩個運作狀態良好的可用區域。



佈建足夠的運算容量來應對單一可用區域移除

如需最佳化 EKS 資料平面中運算基礎結構的資源使用率與成本，最佳實務是使運算容量符合工作負載需求。然而，若您的所有工作節點容量已滿，則您需要在排程新的 Pod 之前，將新的工作節點新增至 EKS 資料平面。若執行關鍵工作負載，最佳實務一般是在線上使用備援容量來執行，以便處理負載突增及節點運作狀態問題等情境。若您打算使用區域轉移，需要規劃在發生受損情況時移除整個可用區域的容量。這意味著必須調整備援運算容量，即使其中一個可用區域離線，也足以處理負載。

擴展運算資源後，需要一些時間來處理將新的節點新增至 EKS 資料平面的程序。這可能會影響應用程式的即時效能與可用性，在區域受損的情況下尤其如此。EKS 環境應能在缺少一個可用區域的情況下吸收負載，而不會導致最終使用者或用戶端的體驗降級。這意味著，需要新 Pod 與在工作節點實際排程的間隔時間，需要最大限度地減少或消除延遲。

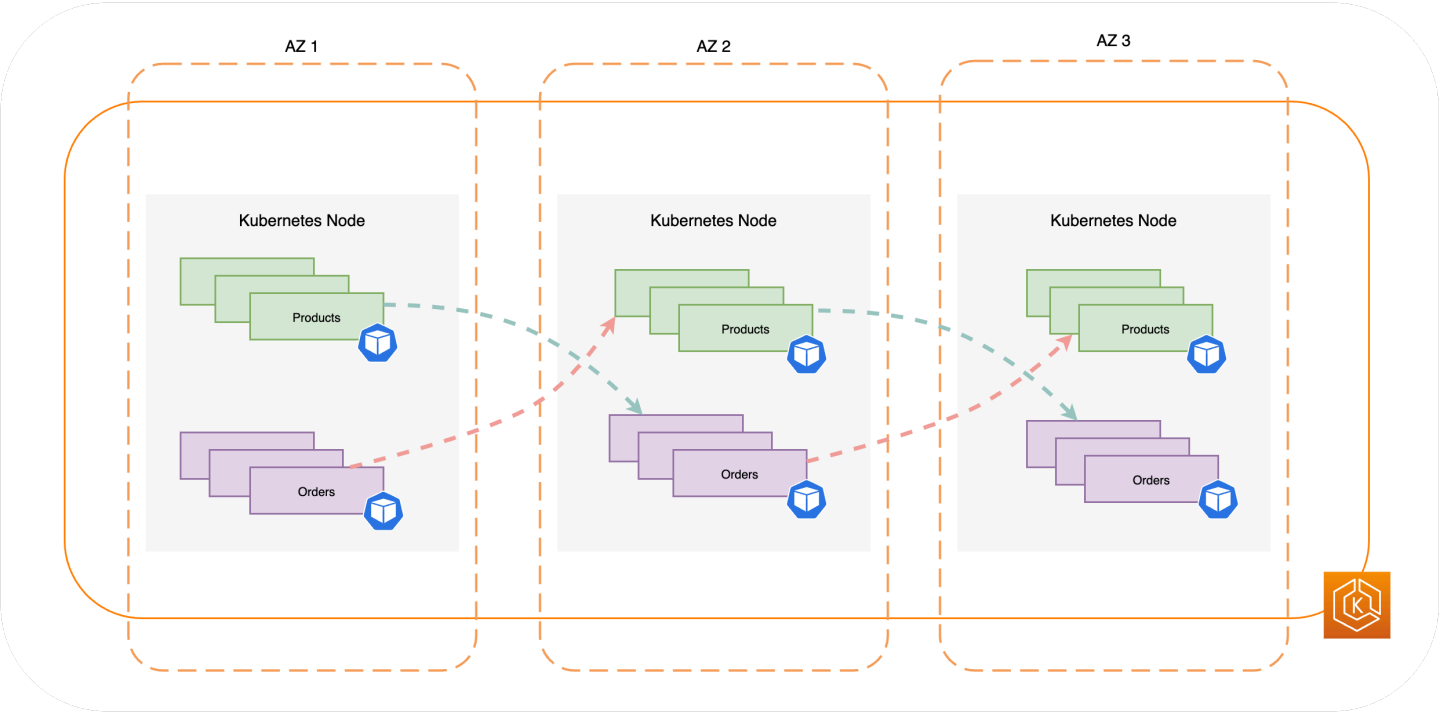
另外，若出現區域受損，您應竭力緩解在運算容量約束條件中執行的風險，以阻止將新的必要節點新增至運作狀態良好可用區域中的 EKS 資料平面。

為了降低這些潛在負面影響的風險，我們建議您在每個 AZs 中的某些工作者節點中過度佈建運算容量。執行此操作，Kubernetes 排程器就預先有容量可用於新 Pod 置放，在環境中缺少其中一個可用區域時尤其重要。

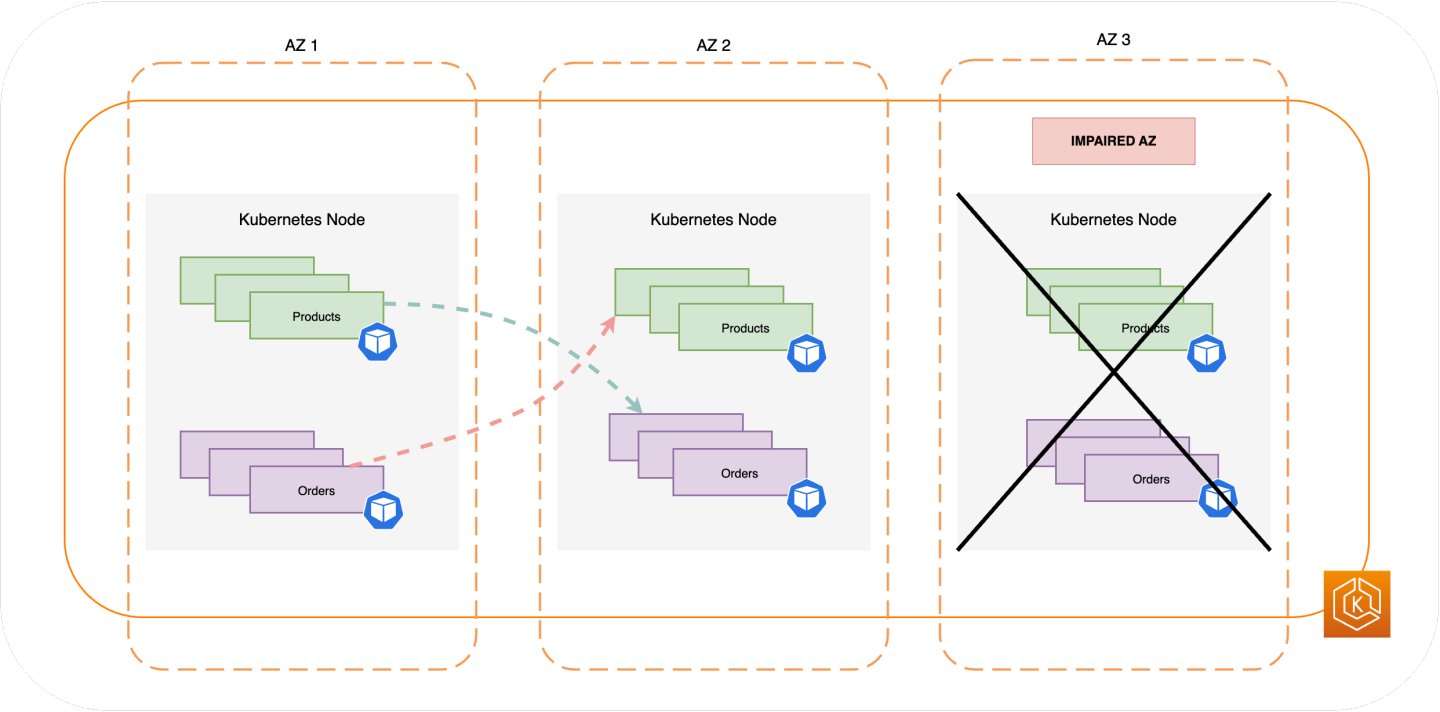
跨可用區域執行及分佈多個 Pod 複本

Kubernetes 藉由執行單一應用程式的多個執行個體 (Pod 複本)，可讓您預先擴展您的工作負載。針對應用程式執行多個 Pod 複本，能夠消除單一故障點，以及減少單一複本上的資源壓力來提高整體效能。然而，若要讓應用程式同時具備高可用性與更佳容錯能力，建議您執行應用程式的多個複本，然後將複本分佈到不同的故障網域，亦稱為拓撲網域。此情境中的故障網域為可用區域。藉助拓撲分佈約束條件，您可設定應用程式，使其具備預先存在的靜態穩定性。之後，若可用區域受損，您的環境將會在運作狀態良好的可用區域擁有足夠的複本，以便立即處理任何流量尖峰或激增。

下面的圖表闡述了 EKS 環境，在所有可用區域運作狀態良好時，該環境具有東西向流量。



下面的圖表闡述了具有東西向流量的 EKS 環境，其中單一可用區域發生故障，並且您已開始區域轉移。



以下程式碼片段展示了如何在 Kubernetes 中利用多個複本設定工作負載的範例。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: orders
spec:
  replicas: 9
  selector:
    matchLabels:
      app: orders
  template:
    metadata:
      labels:
        app: orders
        tier: backend
    spec:
      topologySpreadConstraints:
      - maxSkew: 1
        topologyKey: "topology.kubernetes.io/zone"
        whenUnsatisfiable: ScheduleAnyway
        labelSelector:
          matchLabels:
            app: orders
```

最重要的是，您應執行 DNS 伺服器軟體 (CoreDNS/kube-dns) 的多個複本，若預設未設定，則應套用類似的拓撲分佈約束條件。若有單一可用區域受損，這可協助確保您在運作狀態良好的可用區域有足夠的 DNS Pod，以便繼續處理叢集中其他通訊 Pod 的服務探索請求。CoreDNS EKS 附加元件具有適用於 CoreDNS Pod 的預設設定，若多個可用區域有可用的節點，可確保這些節點將分佈到叢集的各個可用區域。您可依據意願，使用您自己的自訂組態來取代預設設定。

若您透過 Helm 安裝 CoreDNS，可更新 values.yaml 檔案中的 `replicaCount`，以便確保在每個可用區域有充足的複本。另外，若要確保這些複本會分佈到叢集環境的不同可用區域，請確認您更新了同一 `values.yaml` 檔案中的 `topologySpreadConstraints` 屬性。以下程式碼片段闡述了如何設定 CoreDNS 來執行此動作。

CoreDNS Helm values.yaml

```
replicaCount: 6
topologySpreadConstraints:
  - maxSkew: 1
    topologyKey: topology.kubernetes.io/zone
    whenUnsatisfiable: ScheduleAnyway
    labelSelector:
      matchLabels:
        k8s-app: kube-dns
```

若可用區域受損，您可藉助適用於 CoreDNS 的自動擴展系統，來吸收 CoreDNS Pod 上增加的負載。您所需的 DNS 執行個體數目視乎叢集中執行的工作負載數目而定。CoreDNS 與 CPU 繫結，可使用水平 Pod 自動擴展器 (HPA) 依據 CPU 來進行擴展。下面展示了一個範例，您可透過修改來滿足您的需求。

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: coredns
  namespace: default
spec:
  maxReplicas: 20
  minReplicas: 2
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: coredns
  targetCPUUtilizationPercentage: 50
```

或者，EKS 可在 CoreDNS 的 EKS 附加元件版本中管理 CoreDNS 部署的自動擴展。此 CoreDNS 自動擴展器持續監控叢集狀態，包括節點數和 CPU 核心數。依據該資訊，控制器可動態調整 EKS 叢集中 CoreDNS 部署的複本數目。

如需在 CoreDNS EKS 附加元件中啟用自動擴展組態，請使用以下組態設定：

```
{
  "autoScaling": {
    "enabled": true
  }
}
```

您亦可使用 NodeLocal DNS，或是使用叢集比例自動擴展器來擴展 CoreDNS。如需詳細資訊，請參閱水平擴展 CoreDNS。

在同一可用區域設定相依的 Pod

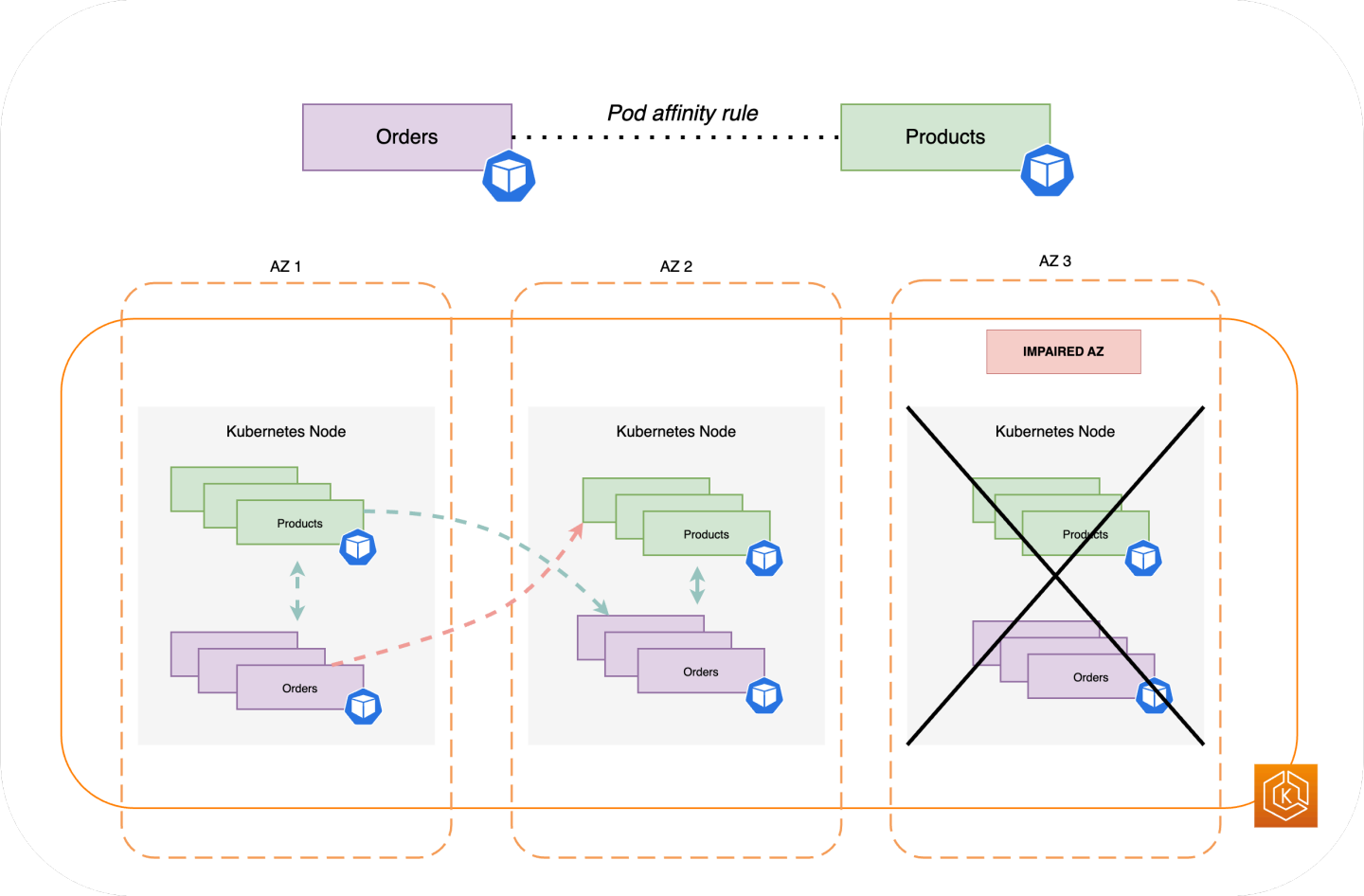
通常，應用程式有各種工作負載，需要彼此通訊來成功完成端對端程序。若這些不同的應用程式分佈到不同的可用區域，並且未在同一可用區域進行主機代管，則單一可用區域受損可能會影響端對端程序。舉例來說，若應用程式 A 在可用區域 1 與可用區域 2 有多個複本，但應用程式 B 在可用區域 3 有其所有複本，則缺少可用區域 3 會影響兩個工作負載 (應用程式 A 與應用程式 B) 間的端對端程序。若結合拓撲分佈約束條件與 Pod 親和性，則可將 Pod 分佈到所有可用區域，藉此增強應用程式的復原能力。另外，這可設定某些 Pod 間的關係，來確保其主機代管。

藉助 Pod 親和性規則，您可定義工作負載間的關係，來影響 Kubernetes 排程器的行為，以便將 Pod 在同一工作節點或同一可用區域進行主機代管。您亦可設定排程約束條件的嚴格度。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: products
  namespace: ecommerce
  labels:
    app.kubernetes.io/version: "0.1.6"
spec:
  template:
    spec:
      serviceAccountName: graphql-service-account
      affinity:
        podAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
```

```
        - labelSelector:
            matchExpressions:
            - key: app
              operator: In
              values:
              - orders
          topologyKey: "kubernetes.io/hostname"
```

下面的圖表展示了多個 Pod 藉助 Pod 親和性規則在同一節點進行主機代管。



測試叢集環境是否可應對可用區域缺失

在完成前面章節所述的要求之後，下一步是測試是否有充足的運算和工作負載容量來應對可用區域缺失。您可在 EKS 中，手動啟動區域轉移來執行此動作。或者，您可啟用區域自動轉移及設定實際執行，這亦會測試在叢集環境缺少一個可用區域的情況下，應用程式是否如預期運作。

常見問答集

這項功能有什麼用？

藉由在您的 EKS 叢集中使用 ARC 區域轉移或區域自動轉移，您可自動執行從受損的可用區域移出叢集內網路流量的快速復原程序，以便更好地確保 Kubernetes 應用程式可用性。藉助 ARC，在發生受損可用區域事件期間，您可避免可能引起延長復原時段的冗長且複雜的步驟。

此功能如何與其他 AWS 服務搭配使用？

EKS 與 ARC 整合，可提供您完成復原操作的主要界面 AWS。若要確保叢集內流量從受損的可用區域適當移出，EKS 會對執行於 Kubernetes 資料平面的 Pod 的網路端點清單做出修改。若您使用 Elastic Load Balancing，將外部流量路由至叢集，可在 ARC 註冊負載平衡器，然後在這些負載平衡器上開始區域轉移，來阻止流量流入降級的可用區域。如果您使用的是 EKS Auto 模式，EKS Auto Mode 會自動將節點佈建限制為運作狀態良好的可用AZs。此外，區域轉移還適用於 EKS 受管節點群組建立的 Amazon EC2 Auto Scaling 群組。若要阻止受損的可用區域用於新的 Kubernetes Pod 或節點啟動，EKS 會從 Auto Scaling 群組中移除受損的可用區域。

此功能與預設 Kubernetes 防護有何差異？

此功能可與若干 Kubernetes 內建防護功能配合使用，可協助客戶確保應用程式的復原能力。您可設定 Pod 整備與存活探查，來決定 Pod 應接收流量的時間。若這些探查失敗，Kubernetes 會將這些 Pod 作為服務目標進行移除，並且不再將流量傳送至 Pod。雖然這很實用，但客戶設定這些運作狀態檢查並不容易，因此若可用區域降級，需要保證這些探查會失敗。ARC 區域轉移功能提供了額外的安全網路，若 Kubernetes 的原生保護不足，它能協助您完全隔離降級的可用區域。此外，區域轉移還為您提供了一種簡便的方式，來測試架構的運作整備與復原能力。

可以代表我 AWS 開始區域轉移嗎？

是，若您需要以全自動化方式使用 ARC 區域轉移，您可啟用 ARC 區域自動轉移。透過區域自動轉移，您可以依賴 AWS 來監控 EKS 叢集的AZs運作狀態，並在偵測到可用區域受損時自動啟動區域轉移。

若我使用此功能，並且工作節點與工作負載未預先擴展，會發生什麼？

若您未預先調整擴展，並且在區域轉移期間依賴於佈建額外的節點或 Pod，則有延遲復原的風險。新增節點至 Kubernetes 資料平面的程序需要一些時間，這可能會影響應用程式的即時效能與可用性，在區域受損時尤其如此。另外，若發生區域受損，您可能會遇到運算容量約束，這可能會阻止新的必要節點新增至運作狀態良好的可用區域。

如果您使用的是 EKS Auto Mode，EKS Auto Mode 會自動在運作狀態良好的AZs佈建新節點，以滿足不可排程的 Pod 需求。不過，新節點仍需要一些時間才能啟動並準備就緒。為了獲得最快的復原，建議您在多個 AZs 中預先擴展工作負載。

若工作負載未預先擴展，並且分佈到叢集中的所有可用區域，對於僅執行於受影響可用區域中工作節點的應用程式，區域受損可能會影響其可用性。當工作負載在運作狀態不佳的可用區域擁有全部端點的情況下，若要緩解應用程式可用性完全中斷的風險，EKS 設置有失效安全機制，來將流量傳送至受損區域的 Pod 端點。然而，強烈建議您預先擴展應用程式，然後將其分佈到所有可用區域，以便在發生區域問題時確保可用性。

若我執行具狀態應用程式，會如何運作？

若您正在執行具狀態應用程式，必須依據您的使用案例和架構來評估其容錯能力。若您有作用中/待命架構或模式，可能會有作用中執行個體位於受損的可用區域。若在應用程式層級未啟用待命，執行時可能會遇到應用程式問題。若在運作狀態良好的可用區域啟動新的 Kubernetes Pod，可能會遇到問題，因為這些 Pod 無法連接至已繫結受損可用區域的持續性磁碟區。

此功能是否適用於 EKS Auto Mode？

是。當您在 EKS Auto Mode 叢集上啟用區域轉移時，EKS Auto Mode 會自動回應區域轉移事件。在區域轉移期間，EKS Auto Mode 會停止在受損 AZ 中佈建新節點、暫停會影響受損 AZ 的自願中斷動作（例如合併和偏離），以及避免為具有嚴格排程需求的 Pod 啟動容量，以受損 AZ 為目標。除了在叢集上啟用區域轉移之外，您不需要任何其他組態。

此功能是否適用於自我管理的 Karpenter？

自我管理的 Karpenter 支援可在具有 Karpenter 1.12 版或更新版本的 EKS 中使用 ARC 區域轉移和區域自動轉移。

此功能是否可與 EKS Fargate 配合使用？

此功能不與 EKS Fargate 配合使用。依預設，若 EKS Fargate 發現區域運作狀態事件，Pod 更偏好在其他可用區域執行。

EKS 受管 Kubernetes 控制平面是否會受到影響？

否，依預設，Amazon EKS 在多個可用區域內執行和擴展 Kubernetes 控制平面以確保高可用性。ARC 區域轉移與區域自動轉移僅可在 Kubernetes 資料平面運作。

此新功能是否有任何關聯成本？

您可在 EKS 叢集中使用 ARC 區域轉移與區域自動轉移，並且不會收取額外費用。但是，您會繼續支付佈建執行個體費用，強烈建議您預先擴展 Kubernetes 資料平面，再使用此功能。您應考慮在成本與應用程式可用性間達成平衡。

其他資源

- 區域轉移的運作方式
- ARC 中的區域轉移最佳實務
- 支援區域轉移和區域自動轉移的資源與情境
- 在 Amazon EKS 上運作彈性工作負載
- 透過確定 Pod 優先順序和過度佈建消除 Kubernetes 節點擴展延遲
- 為高 DNS 流量擴展 CoreDNS Pod

啟用 EKS 區域轉移來避免可用區域受損

Amazon 應用程式復原控制器 (ARC) 可協助您管理及協調跨可用區域 (AZ) 進行應用程式復原，並且可搭配包括 Amazon EKS 在內的多項服務一起使用。憑藉 ARC 區域轉移的 EKS 支援，您可從受損的可用區域轉移叢集內的網路流量。您也可以授權 AWS 監控 AZs 的運作狀態，並代表您暫時將網路流量移離運作狀態不佳的 AZ。

EKS 區域轉移運作方式：

1. 透過 Amazon 應用程式復原控制器 (ARC) 啟用 EKS 叢集。這會在叢集層級使用 Amazon EKS 主控台、CLI、CloudFormation AWS 或 eksctl 完成。
2. 啟用後，您可以使用 ARC 主控台、 AWS CLI 或區域轉移和區域自動轉移 APIs 來管理區域轉移或區域自動轉移。

請注意，在 ARC 註冊 EKS 叢集之後，您仍需設定 ARC。舉例來說，您可藉助 ARC 主控台來設定區域自動轉移。

若要了解 EKS 區域轉移運作方式，以及如何設計工作負載來處理受損可用區域的相關詳細資訊，請參閱 the section called “了解區域轉移”。

考量事項

- 建議在各項區域轉移操作間等候至少 60 秒，以確認每項請求均得到適當處理。

  嘗試快速連續執行區域轉移時（相隔 60 秒內），Amazon EKS 服務可能無法正確處理所有輪班請求。這是因為更新叢集區域狀態的目前輪詢機制。若您需要執行多次區域轉移，確認系統在各項操作間留有充足的時間來處理每項變更。

什麼是 Amazon 應用程式復原控制器？

Amazon 應用程式復原控制器 (ARC) 可協助您針對執行於 AWS的應用程式做好準備，以及加速完成復原。區域轉移可讓您從可用區域 (AZ) 受損快速復原，方法是暫時將支援資源的流量從可用區域移出至 AWS 區域中運作狀態良好的AZs。

進一步了解 Amazon 應用程式復原控制器 (ARC)

什麼是區域轉移？

區域轉移是 ARC 的一項功能，可讓您將 EKS 叢集或 Elastic Load Balancer 等資源的流量移離 AWS 區域中的可用區域，以快速緩解問題並快速復原應用程式。舉例來說，由於部署不良導致延遲問題或可用區域受損，您可選擇轉移流量。區域轉移不需要進階組態步驟。不過，您的工作負載必須符合特定需求，才能在輪班開始之前發揮區域彈性。如需詳細資訊，請參閱the section called “了解區域轉移”。

進一步了解 ARC 區域轉移

什麼是區域自動轉移？

區域自動轉移是 ARC 中的功能，可讓您授權 代表您 AWS 將流量從 AZ 轉移到 AWS 區域中運作狀態良好的 AZs。當內部遙測顯示 區域中的一個 AZ 受損，而可能影響客戶時， 會 AWS 啟動自動轉移。內部遙測包含來自多個來源的指標，包括 AWS 網路，以及 Amazon EC2 和 Elastic Load Balancing 服務。

AWS 當指標顯示不再存在問題或潛在問題時， 會結束自動轉移。

進一步了解 ARC 區域自動轉移

EKS 在區域轉移期間會做什麼？

EKS 可更新聯網組態，進而避免將流量引導至受損的可用區域。特定的運算行為取決於叢集的資料平面組態：

- EKS 自動模式：EKS 自動模式會自動停止在受損 AZ 中佈建新節點、停止可能影響受損 AZ 的自願節點中斷（例如整合），以及避免為具有嚴格排程需求的 Pod 啟動容量，以受損 AZ 為目標（例如持久性磁碟區繫結或嚴格的拓撲分散限制）。除了在叢集上啟用區域轉移之外，您不需要任何其他組態。
- 自我管理 Karpenter (v1.12+)：如果您使用 `--enable-zonal-shift`旗標執行 Karpenter v1.12 或更高版本，Karpenter 會停止在受損 AZ 中佈建容量，並停止依賴受損 AZ 的自願中斷。

- 受管節點群組：EKS 只會透過暫停可用區域重新平衡和更新 Auto Scaling 群組，在區域轉移期間於運作狀態良好的 AZs 中啟動新節點。

當轉移到期或取消時，網路組態和運算行為會還原，以包含先前偵測到運作狀態不佳的 AZ。

進一步了解 EKS 區域轉移。

向 Amazon Application Recovery Controller (ARC) 註冊 EKS 叢集AWS （主控台）

1. 找到在 ARC 註冊的 EKS 叢集名稱與區域。
2. 導覽至該區域的 EKS 主控台，然後選取您的叢集。
3. 在叢集資訊頁面上，選取概觀索引標籤。
4. 在區域轉移標題項下，選取管理按鈕。
5. 針對 EKS 區域轉移，選取啟用或停用。

現在即可在 ARC 註冊 EKS 叢集。

如果您想要 AWS 偵測並避免可用區域受損，則需要設定 ARC 區域自動轉移。舉例來說，您可在 ARC 主控台中執行此動作。

後續步驟

- 了解如何啟用區域自動轉移。
- 了解如何手動開始區域轉移。

了解存取控制在 Amazon EKS 中的運作方式

了解如何管理對 Amazon EKS 叢集的存取權限。使用 Amazon EKS 需要了解 Kubernetes 和 AWS Identity and Access Management (AWS IAM) 如何處理存取控制。

本節包括：

the section called “Kubernetes API 存取”：了解如何讓應用程式或使用者對 Kubernetes API 進行身分驗證。您可使用存取項目、aws-auth ConfigMap 或外部 OIDC 提供商。

the section called “存取叢集資源” — 了解如何設定 AWS 管理主控台 以與您的 Amazon EKS 叢集通訊。使用主控台檢視叢集中的 Kubernetes 資源，如命名空間、節點和 Pod。

the section called “修改叢集資源” — 了解修改 Kubernetes 資源所需的許可。

the section called “透過 kubectl 存取叢集”：了解如何設定 kubectl 以與您的 Amazon EKS 叢集進行通訊。使用 AWS CLI 建立 kubeconfig 檔案。

the section called “工作負載存取 AWS ” — 了解如何將 Kubernetes 服務帳戶與 IAM AWS 角色建立關聯。您可以使用 Pod 身分識別或服務帳戶的 IAM 角色 (IRSA)。

一般任務

- 授予開發人員對 Kubernetes API 的存取權限。在 中檢視 Kubernetes 資源 AWS 管理主控台。
  - 解決方案：使用存取項目將 Kubernetes RBAC 許可與 IAM AWS 使用者或角色建立關聯。
- 設定 kubectl 以使用 AWS 登入資料與 Amazon EKS 叢集通訊。
  - 解決方案：使用 AWS CLI 建立 kubeconfig 檔案。
- 使用外部身分識別提供商 (例如 Ping Identity) 來驗證使用者對 Kubernetes API 的存取。
  - 解決方案：連結外部 OIDC 提供商。
- 授予 Kubernetes 叢集上的工作負載呼叫 AWS APIs的能力。
  - 解決方案：使用 Pod Identity 將 AWS IAM 角色與 Kubernetes 服務帳戶建立關聯。

背景介紹

- 了解 Kubernetes 服務帳戶的工作原理。
- 檢閱 Kubernetes 角色型存取控制 (RBAC) 模型

- 如需管理 AWS 資源存取權的詳細資訊，請參閱 AWS IAM 使用者指南。或者，使用 IAM 在 AWS 上進行免費入門訓練。

EKS 自動模式的考量事項

EKS Auto Mode 與 EKS Pod Identity 和 EKS 存取項目整合。

- EKS 自動模式使用存取項目來授予 EKS 控制平面 Kubernetes 許可。例如，存取政策讓 EKS 自動模式能夠讀取有關網路端點和服務的資訊。
  - 您無法在 EKS 自動模式叢集上停用存取項目。
  - 您可選擇性啟用 `aws-auth ConfigMap`。
  - EKS 自動模式的存取項目將會自動設定。您可以檢視這些存取項目，但無法進行修改。
  - 如果您使用 NodeClass 建立自訂節點 IAM 角色，則需要使用 AmazonEKSAutoNodePolicy 存取政策為該角色建立存取項目。
- 如果您想要授予 AWS 服務的工作負載許可，請使用 EKS Pod Identity。
  - 您無需在 EKS 自動模式叢集上安裝 Pod 身分識別代理程式。

授予 IAM 使用者和角色對 Kubernetes APIs存取權

您的叢集具有 Kubernetes API 端點。Kubectl 會使用此 API。您可以使用兩種類型的身分來驗證此 API：

- AWS Identity and Access Management (IAM) 主體 （角色或使用者） – 此類型需要對 IAM 進行身分驗證。使用者可以 AWS 使用透過身分來源提供的憑證，以 IAM 使用者身分或使用聯合身分登入 。如果管理員先前已設定使用 IAM 角色的聯合身分，則使用者只能夠以聯合身分登入。當使用者使用聯合 AWS 存取 時，他們間接擔任角色。當使用者使用此類身分時，您：
  - 可以為其指派 Kubernetes 許可，從而讓其能夠使用您叢集上的 Kubernetes 物件。如需如何向 IAM 主體指派許可以便相應主體能夠存取您叢集上的 Kubernetes 物件的詳細資訊，請參閱 the section called “存取項目”。
  - 可以指派 IAM 許可給他們，讓他們可以使用 Amazon EKS API、 AWS CLI AWS 管理主控台、AWS CloudFormation 或 與您的 Amazon EKS 叢集及其資源搭配使用`eksctl`。如需詳細資訊，請參閱《服務授權參考》中的 Amazon Elastic Kubernetes Service 定義的動作一節。
  - 節點透過承擔 IAM 角色來加入叢集。適用於 Kubernetes 的AWS IAM 驗證器在 Amazon EKS 控制平面上執行，讓使用者能夠使用 IAM 主體存取您的叢集。

- 您自己的 OpenID Connect (OIDC) 提供商中的使用者：此類型需要向 OIDC 提供商進行身分驗證。如需向 Amazon EKS 叢集設定您自己的 OIDC 提供商的詳細資訊，請參閱 the section called “連結 OIDC 提供商”。當使用者使用此類身分時，您：
  - 可以為其指派 Kubernetes 許可，從而讓其能夠使用您叢集上的 Kubernetes 物件。
  - 無法為他們指派 IAM 許可，讓他們可以使用 Amazon EKS API、 AWS CLI AWS 管理主控台、AWS CloudFormation 或 來使用您的 Amazon EKS 叢集及其資源`eksctl`。

您可以在叢集中使用這兩種類型的身分。IAM 身分驗證方法無法停用。OIDC 身分驗證方法為選用。

建立 IAM 身分與 Kubernetes 許可的關聯

適用於 Kubernetes 的AWS IAM 驗證器安裝在叢集的控制平面上。它啟用您允許存取叢集上的 Kubernetes 資源的 AWS Identity and Access Management (IAM) 主體 (角色和使用者)。您可以使用以下方法之一允許 IAM 主體存取叢集上的 Kubernetes 物件：

- 建立存取項目：如果叢集版本等於或高於叢集 Kubernetes 版本先決條件部分中列出的平台版本，則建議您使用此選項。

  使用存取項目，從叢集外部管理 IAM 主體的 Kubernetes 許可。您可以使用 EKS API、 AWS 命令列界面、 AWS SDKs、 AWS CloudFormation 和 來新增和管理對叢集的存取 AWS 管理主控台。這意味著您可以使用建立叢集時使用的工具來管理使用者。

  若要開始使用，請依照變更身分驗證模式來使用存取項目，然後遷移現有的 aws-auth ConfigMap 項目來存取項目。
- 新增項目到 **`aws-auth ConfigMap`**：如果叢集的平台版本低於先決條件部分中列出的版本，則必須使用此選項。如果叢集的平台版本等於或高於叢集 Kubernetes 版本先決條件部分中列出的平台版本，並且您已將項目新增至 `ConfigMap`，則建議您將這些項目移轉到存取項目。但是，您無法移轉 Amazon EKS 新增至 `ConfigMap` 的項目，例如用於受管節點群組或 Fargate 設定檔的 IAM 角色項目。如需詳細資訊，請參閱the section called “Kubernetes API 存取”。
  - 如果必須使用 `aws-auth ConfigMap` 選項，則可以使用 `eksctl create iamidentitymapping` 命令將項目新增至 `ConfigMap`。如需詳細資訊，請參閱 `eksctl` 文件中的 Manage IAM users and roles 一節。

設定叢集驗證模式

每個叢集都有自己的身分驗證模式。身分驗證模式決定您可以使用哪些方法來允許 IAM 主體存取叢集上的 Kubernetes 物件。身分驗證模式有 3 種。

> **Important**
>
> 啟用存取項目方法後，即無法將其停用。
> 如果在叢集建立期間未啟用 ConfigMap 方法，則稍後無法啟用。在引入存取項目之前建立的所有叢集均已啟用 ConfigMap 方法。
> 如果您搭配叢集使用混合節點，則必須使用 API 或 API_AND_CONFIG_MAP 叢集身分驗證模式。

叢集內的 aws-auth ConfigMap

這是 Amazon EKS 叢集的原始身分驗證模式。建立叢集的 IAM 主體是可以使用 kubectl 存取叢集的初始使用者。初始使用者必須將其他使用者新增至 aws-auth ConfigMap 的清單中，並為這些使用者指派相應許可。其他使用者無法管理或移除初始使用者，因為 ConfigMap 中沒有要管理的項目。

ConfigMap 和存取項目

使用這種身分驗證模式時，您可以使用這兩種方法將 IAM 主體新增至叢集。請注意，每個方法都會存放個別的項目；例如，如果您從 AWS CLI 新增存取項目，aws-authConfigMap則不會更新 。

僅存取項目

透過此身分驗證模式，您可以使用 EKS API、 AWS 命令列界面、 AWS SDKs、 AWS CloudFormation 和 AWS 管理主控台 來管理 IAM 主體對叢集的存取。

每個存取項目都有一個類型，您可以使用存取範圍的組合將主體限制為特定命名空間，並使用存取政策來設定預先設定的可重複使用許可政策。您也可以使用 STANDARD 類型和 Kubernetes RBAC 群組來指派自訂許可。

身分驗證方式	方法
僅 ConfigMap （CONFIG_MAP ）	aws-auth ConfigMap
EKS API 和 ConfigMap （API_AND_CONFIG_MAP ）	EKS API、 AWS 命令列界面、 AWS SDKs、 AWS CloudFormation 和 AWS 管理主控台 中的存取項目 aws-auth ConfigMap

身分驗證方式	方法
僅 EKS API (API)	EKS API、 AWS 命令列界面、 AWS SDKs、AWS CloudFormation 和 中的存取項目 AWS 管理主控台

> **Note**
>
> Amazon EKS 自動模式需要存取項目。

使用 EKS 存取項目授予 IAM 使用者 Kubernetes 的存取權

本節旨在說明如何使用存取項目與政策，管理 Amazon Elastic Kubernetes Service (EKS) 中 IAM 主體對 Kubernetes 叢集的存取權。您將找到有關變更驗證模式、從舊版 `aws-auth` ConfigMap 項目移轉、建立、更新、刪除存取項目、將政策與項目關聯、檢閱預先定義政策許可，以及安全存取管理之重要先決條件與考量事項的詳細資訊。

概觀

EKS 存取項目是授予使用者存取 Kubernetes API 的最佳方式。例如，您可使用存取項目授予開發人員使用 kubectl 的權限。本質上，EKS 存取項目會將一組 Kubernetes 許可與 IAM 身分 (如 IAM 角色) 建立關聯。例如，開發人員可擔任某個 IAM 角色，並使用該角色向 EKS 叢集進行身分驗證。

功能

- 集中式身分驗證與授權：直接透過 Amazon EKS API 控制對 Kubernetes 叢集的存取，無需在 AWS 和 Kubernetes API 之間切換，以管理使用者許可。
- 精細許可管理：使用存取項目與政策為 AWS IAM 主體定義精細許可，包括修改或撤銷建立者的叢集管理員存取權。
- IaC 工具整合：支援 AWS CloudFormation、Terraform 與 AWS CDK 等基礎結構即程式碼工具，以便在叢集建立期間定義存取組態。
- 錯誤設定復原：可透過 Amazon EKS API 復原叢集存取權，無需直接存取 Kubernetes API。
- 降低負擔並增強安全性：集中化作業以降低管理負擔，同時運用 AWS IAM 功能 (如 CloudTrail 稽核日誌與多因素驗證)。

附加許可的方式

您可透過兩種方式將 Kubernetes 許可附加至存取項目：

- 使用存取政策。存取政策是由 AWS 維護的預先定義 Kubernetes 許可範本。如需詳細資訊，請參閱 the section called “檢閱存取政策”。
- 引用 Kubernetes 群組。若您將 IAM 身分與 Kubernetes 群組建立關聯，即可建立授予該群組許可的 Kubernetes 資源。如需詳細資訊，請參閱 Kubernetes 文件中的使用 RBAC 授權。

考量事項

在現有叢集上啟用 EKS 存取項目時，請留意下列事項：

- 舊版叢集行為：對於在存取項目引入前建立的叢集 (其初始平台版本早於平台版本需求中指定的版本)，EKS 會自動建立反映既有許可的存取項目。此項目包含了最初建立叢集的 IAM 身分，以及在叢集建立期間授予該身分的管理許可。
- 處理舊版 **`aws-auth`** ConfigMap：若您的叢集依賴舊版 `aws-auth` ConfigMap 進行存取管理，則在啟用存取項目時，僅會自動建立原叢集建立者的存取項目。新增至 ConfigMap 的其他角色或許可 (如供開發人員或服務使用的自訂 IAM 角色) 不會自動移轉。為了解決這個問題，請手動建立對應的存取項目。

開始使用

1. 確定您要使用的 IAM 身分和存取政策。
   - the section called “檢閱存取政策”
2. 在您的叢集上啟用 EKS 存取項目。確認您擁有受支援的平台版本。
   - the section called “身分驗證方式”
3. 建立存取項目，將 IAM 身分與 Kubernetes 許可建立關聯。
   - the section called “建立存取項目”
4. 使用該 IAM 身分向叢集進行驗證。
   - the section called “設定 AWS CLI”
   - the section called “設定 kubectl 和 eksctl”

將存取政策與存取項目建立關聯

您可以向類型為 STANDARD 的存取項目指派一個或多個存取政策。Amazon EKS 會自動向其他類型的存取項目授予在叢集中正常運作所需的許可。Amazon EKS 存取政策包含 Kubernetes 許可，而不是 IAM 許可。在將存取政策與存取項目關聯之前，請確認您熟悉每個存取政策中包含的 Kubernetes 許可。如需詳細資訊，請參閱the section called “檢閱存取政策”。如果沒有一個存取政策符合您的要求，則不要將存取政策與存取項目關聯。而是為存取項目指定一個或多個群組名稱，並建立和管理 Kubernetes 角色型存取控制物件。如需詳細資訊，請參閱the section called “建立存取項目”。

- 現有的存取項目。若要建立服務角色，請參閱the section called “建立存取項目”。
- Identity AWS and Access Management 角色或具有下列許可的使用者：`ListAccessEntries`、`DescribeAccessEntry`、`UpdateAccessEntryListAccessPolicies`、`AssociateAccessPolicy`和 `DisassociateAccessPolicy`。如需詳細資訊，請參閱《服務授權參考》中的 Amazon Elastic Kubernetes Service 定義的動作一節。

在將存取政策與存取項目關聯之前，請考量以下要求：

- 每個存取項目可以關聯多個存取政策，但只能將每個政策與一個存取項目關聯一次。如果您將一個存取項目與多個存取政策關聯，則該存取項目的 IAM 主體擁有所有關聯的存取政策中包含的所有許可。
- 您可以將存取政策的適用範圍限定為叢集上的所有資源，或透過指定一個或多個 Kubernetes 命名空間的名稱來限制其適用範圍。您可以將萬用字元用於命名空間名稱。例如，如果要將存取政策的適用範圍限定為以 `dev-` 開頭的所有命名空間，則可以指定 `dev-*` 作為命名空間名稱。確認叢集上存在相應命名空間，並且拼字與叢集上的實際命名空間名稱相符。Amazon EKS 不會確認叢集上命名空間的拼字或命名空間是否存在。
- 將存取政策關聯到存取項目後，您可以變更存取政策的存取範圍。如果您已將存取政策的適用範圍限定為 Kubernetes 命名空間，則可以根據需要新增和移除關聯的命名空間。
- 如果您將存取政策關聯到也指定了群組名稱的存取項目，則相應 IAM 主體擁有所有關聯存取政策中的所有許可。它還擁有在指定群組名稱的任何 `Role` 和 `RoleBinding` 物件中指定的任何 `Role` 或 `ClusterRole` 物件中的所有許可。
- 如果您執行 `kubectl auth can-i --list` 命令，您不會看到與您在執行該命令時所使用的 IAM 主體的存取項目關聯的存取政策指派的任何 Kubernetes 許可。只有當您已在繫結至為存取項目指定的群組名稱或使用者名稱的 Kubernetes `Role` 或 `ClusterRole` 物件中授予 Kubernetes 許可時，此命令才會顯示相應許可。
- 如果您在與叢集上的 Kubernetes 物件互動時模擬 Kubernetes 使用者或群組，例如將 `kubectl` 命令與 `--as` *`username`* 或 `--as-group` *`group-name`* 結合使用，則將強制使用 RBAC 授權。

因此，相應 IAM 主體沒有與相應存取項目關聯的任何存取政策指派的許可。相應 IAM 主體模擬的使用者或群組擁有的 Kubernetes 許可，僅限於您在繫結到該群組名稱或使用者名稱的 Kubernetes `Role` 或 `ClusterRole` 物件中授予的 Kubernetes 許可。為了讓相應 IAM 主體擁有關聯的存取政策中的許可，請勿模擬 Kubernetes 使用者或群組。相應 IAM 主體還將擁有您在繫結至為存取項目指定的群組名稱或使用者名稱的 Kubernetes `Role` 或 `ClusterRole` 物件中授予的任何許可。如需詳細資訊，請參閱 Kubernetes 文件中的使用者模擬一節。

您可以使用 AWS 管理主控台 或 AWS CLI 將存取政策與存取項目建立關聯。

AWS 管理主控台

1. 開啟 Amazon EKS 主控台。
2. 選擇要將存取政策關聯到的存取項目所在的叢集的名稱。
3. 選擇存取索引標籤。
4. 如果存取項目的類型為標準，則可以將 Amazon EKS存取政策與之關聯或取消關聯。如果存取項目的類型不為標準，則此選項不可用。
5. 選擇關聯存取政策。
6. 對於政策名稱，選取具有您希望相應 IAM 主體擁有的許可的政策。若要查看每個政策包含的許可，請參閱 the section called “檢閱存取政策”。
7. 對於存取範圍，選擇存取範圍。如果您選擇叢集，則該存取政策中的許可將授予相應 IAM 主體對所有 Kubernetes 命名空間中的資源的存取權。如果您選擇 Kubernetes 命名空間，則可以選擇新增命名空間。在顯示的命名空間欄位中，您可以輸入叢集上的 Kubernetes 命名空間的名稱。如果您希望相應 IAM 主體擁有跨多個命名空間的許可，則可以輸入多個命名空間。
8. 選擇新增存取政策。

AWS CLI

1. 在您的裝置或 AWS CloudShell 上安裝和設定的 AWS 命令列界面 (AWS CLI) 版本 `1.27.160` `2.12.3`或更新版本。若要檢查您目前的版本，請使用 `aws --version | cut -d / -f2 | cut -d ' ' -f1`。適用於 macOS 的 `yum`、 `apt-get`或 Homebrew 等套件管理員通常是最新版本 CLI AWS 後面的幾個版本。若要安裝最新版本，請參閱《 AWS 命令列界面使用者指南》中的使用 aws 設定安裝 和快速組態。 https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-quickstart.html#cli-configure-quickstart-config在 AWS CloudShell 中安裝的 AWS CLI 版本也可能是最新版本後面的幾個版本。若要更新它，請參閱《CloudShell AWS 使用者指南》中的將 CLI 安裝到您的主目錄。 AWS CloudShell

2. 檢視可用的存取政策。

```
aws eks list-access-policies --output table
```

範例輸出如下。

```
-------------------------------------------------------------------------------------------------
|                                          ListAccessPolicies
                  |
+-----------------------------------------------------------------------------------------------
+
||                                           accessPolicies
                 ||
|+-------------------------------------------------------------------
+-------------------------------+|
||                                arn                                |
 name              ||
|+-------------------------------------------------------------------
+-------------------------------+|
||  {arn-aws}eks::aws:cluster-access-policy/AmazonEKSAdminPolicy        |
 AmazonEKSAdminPolicy         ||
||  {arn-aws}eks::aws:cluster-access-policy/AmazonEKSClusterAdminPolicy |
 AmazonEKSClusterAdminPolicy  ||
||  {arn-aws}eks::aws:cluster-access-policy/AmazonEKSEditPolicy         |
 AmazonEKSEditPolicy          ||
||  {arn-aws}eks::aws:cluster-access-policy/AmazonEKSViewPolicy         |
 AmazonEKSViewPolicy          ||
|+-------------------------------------------------------------------
+-------------------------------+|
```

若要查看每個政策包含的許可，請參閱 the section called “檢閱存取政策”。

3. 檢視現有的存取項目。使用您叢集的名稱取代 *my-cluster*。

```
aws eks list-access-entries --cluster-name my-cluster
```

範例輸出如下。

```
{
    "accessEntries": [
        "arn:aws: iam::111122223333:role/my-role",
        "arn:aws: iam::111122223333:user/my-user"
```

```
    ]
}
```

4. 將存取政策與存取項目關聯。以下範例將 AmazonEKSViewPolicy 存取政策與一個存取項目關聯。每當 *my-role* IAM 角色嘗試存取叢集上的 Kubernetes 物件時，Amazon EKS 將授權該角色使用政策中的許可來存取且僅存取 *my-namespace1* 和 *my-namespace2* Kubernetes 命名空間中的 Kubernetes 物件。將 *my-cluster* 取代為您的叢集名稱、將 *111122223333* 取代為您的 AWS 帳戶 ID，並將 *my-role* 取代為您希望 Amazon EKS 授權存取 Kubernetes 叢集物件的 IAM 角色名稱。

```
aws eks associate-access-policy --cluster-name my-cluster --principal-arn arn:aws:
 iam::111122223333:role/my-role \
    --access-scope type=namespace,namespaces=my-namespace1,my-namespace2 --policy-arn
 arn:aws: eks::aws:cluster-access-policy/AmazonEKSViewPolicy
```

如果您希望相應 IAM 主體擁有適用於整個叢集的許可，請用 `type=cluster` 取代 `type=namespace,namespaces=`*my-namespace1*`,`*my-namespace2* 。如果要將多個存取政策關聯到該存取項目，請多次執行該命令，並每次使用不同的存取政策。每個關聯的存取政策都有自身的適用範圍。

> **Note**
>
> 如果您稍後想要變更關聯的存取政策的適用範圍，請再次執行先前的命令並指定新的適用範圍。例如，如果您想要移除 *my-namespace2*，則只需使用 `type=namespace,namespaces=`*my-namespace1* 再次執行命令。如果您想將適用範圍從 `namespace` 變更為 `cluster`，您需要使用 `type=cluster` 再次執行該命令，而不使用 `type=namespace,namespaces=`*my-namespace1*`,`*my-namespace2* 。

5. 確定哪些存取政策與某個存取項目關聯。

```
aws eks list-associated-access-policies --cluster-name my-cluster --principal-arn
 arn:aws: iam::111122223333:role/my-role
```

範例輸出如下。

```
{
    "clusterName": "my-cluster",
    "principalArn": "arn:aws: iam::111122223333:role/my-role",
    "associatedAccessPolicies": [
```

```
        {
            "policyArn": "arn:aws: eks::aws:cluster-access-policy/
AmazonEKSViewPolicy",
            "accessScope": {
                "type": "cluster",
                "namespaces": []
            },
            "associatedAt": "2023-04-17T15:25:21.675000-04:00",
            "modifiedAt": "2023-04-17T15:25:21.675000-04:00"
        },
        {
            "policyArn": "arn:aws: eks::aws:cluster-access-policy/
AmazonEKSAdminPolicy",
            "accessScope": {
                "type": "namespace",
                "namespaces": [
                    "my-namespace1",
                    "my-namespace2"
                ]
            },
            "associatedAt": "2023-04-17T15:02:06.511000-04:00",
            "modifiedAt": "2023-04-17T15:02:06.511000-04:00"
        }
    ]
}
```

在前面的範例中，相應存取項目的 IAM 主體擁有相應叢集上所有命名空間的檢視許可，以及兩個 Kubernetes 命名空間的管理員許可。

6. 將存取政策與存取項目取消關聯。在此範例中，AmazonEKSAdminPolicy 政策與存取項目取消關聯。但是，相應 IAM 主體保留 AmazonEKSViewPolicy 存取政策中對 *my-namespace1* 和 *my-namespace2* 命名空間中物件的許可，因為該存取政策並未與該存取項目取消關聯。

```
aws eks disassociate-access-policy --cluster-name my-cluster --principal-arn arn:aws:
 iam::111122223333:role/my-role \
    --policy-arn arn:aws: eks::aws:cluster-access-policy/AmazonEKSAdminPolicy
```

要列出所有可用的存取政策，請參閱 the section called “檢閱存取政策”。

遷移現有 aws-auth ConfigMap 項目至存取項目

如果您已將項目新增至叢集上的 aws-auth ConfigMap，則建議您為 aws-auth ConfigMap 中的現有項目建立存取項目。建立存取項目後，您可以從 ConfigMap 中移除對應項目。您無法將存取政策與 aws-auth ConfigMap 中的項目建立關聯。如果您想要將存取政策與 IAM 主體建立關聯，請建立存取項目。

> **Important**
>
> - 當叢集處於 API_AND_CONFIGMAP 身分驗證模式，且在 aws-auth ConfigMap 和存取項目中具有相同 IAM 角色的映射時，該角色將會使用存取項目的映射進行身分驗證。存取項目的優先順序高於相同 IAM 主體的 ConfigMap 項目。
> - 在將 Amazon EKS 為受管節點群組或 Fargate 設定檔建立的現有 aws-auth ConfigMap 項目移除至您的叢集之前，請仔細檢查 Amazon EKS 叢集中是否存在這些特定資源的正確存取項目。如果您在沒有相同存取項目的情況下移除 Amazon EKS 在 ConfigMap 中建立的項目，叢集將無法正常運作。

先決條件

- 熟悉存取項目和存取政策。如需詳細資訊，請參閱the section called “存取項目”及the section called “關聯存取政策”。
- 現有叢集的平台版本等於或高於 the section called “存取項目” 物件主題之「先決條件」部分中列出的版本。
- 裝置或 AWS CloudShell 上安裝的eksctl命令列工具版本 0.215.0或更新版本。如需有關安裝或更新 eksctl 的指示，請參閱 eksctl 文件中的安裝一節。
- 修改 kube-system 命名空間中的 aws-auth ConfigMap 的 Kubernetes 許可。
- Identity AWS and Access Management 角色或具有下列許可的使用者： CreateAccessEntry和 ListAccessEntries。如需詳細資訊，請參閱《服務授權參考》中的 Amazon Elastic Kubernetes Service 定義的動作一節。

eksctl

1. 檢視 aws-auth ConfigMap 中的現有項目。使用您叢集的名稱取代 *my-cluster*。

```
eksctl get iamidentitymapping --cluster my-cluster
```

範例輸出如下。

```
ARN
           USERNAME                                      GROUPS
                       ACCOUNT
arn:aws: iam::111122223333:role/EKS-my-cluster-Admins
            Admins                                        system:masters
arn:aws: iam::111122223333:role/EKS-my-cluster-my-namespace-Viewers
            my-namespace-Viewers                          Viewers
arn:aws: iam::111122223333:role/EKS-my-cluster-self-managed-ng-1
            system:node:{{EC2PrivateDNSName}}       system:bootstrappers,system:nodes
arn:aws: iam::111122223333:user/my-user
            my-user
arn:aws: iam::111122223333:role/EKS-my-cluster-fargateprofile1
                              system:node:{{SessionName}}
 system:bootstrappers,system:nodes,system:node-proxier
arn:aws: iam::111122223333:role/EKS-my-cluster-managed-ng
            system:node:{{EC2PrivateDNSName}}       system:bootstrappers,system:nodes
```

2. 為在上一個輸出中傳回的您建立的任何 ConfigMap 項目的 the section called “建立存取項目”。建立存取項目時，請確保為 ARN、USERNAME、GROUPS 和 ACCOUNT 指定與輸出中傳回的值相同的值。在範例輸出中，您將為最後兩個項目之外的所有項目建立存取項目，因為這些項目是由 Amazon EKS 為 Fargate 描述檔和受管節點群組建立的。
3. 從 ConfigMap 中刪除您建立的任何存取項目對應的項目。如果您沒有從 ConfigMap 中刪除對應項目，則對應 IAM 主體 ARN 的存取項目的設定將覆寫 ConfigMap 項目。將 *111122223333* 取代為 AWS 您的帳戶 ID，並將 *EKS-my-cluster-my-namespace-Viewers* 取代為 中項目中角色的名稱ConfigMap。如果您要移除的項目用於 IAM 使用者而不是 IAM 角色，請用 user 取代 role，並用使用者名稱取代 *EKS-my-cluster-my-namespace-Viewers*。

```
eksctl delete iamidentitymapping --arn arn:aws: iam::111122223333:role/EKS-my-
cluster-my-namespace-Viewers --cluster my-cluster
```

檢閱存取政策許可

存取政策包含 rules，其中包含 Kubernetes verbs (許可) 和 resources。存取政策不包含 IAM 許可或資源。與 Kubernetes Role 和 ClusterRole 物件類似，存取政策僅包含 allow rules。您無法修改存取政策的內容。您無法建立自己的存取政策。如果存取政策中的許可無法滿足您的需求，您可以建立 Kubernetes RBAC 物件並為存取項目指定群組名稱。如需詳細資訊，請參閱the section called

"建立存取項目"。存取政策中包含的許可類似於 Kubernetes 面向使用者之叢集角色中的許可。如需詳細資訊，請參閱 Kubernetes 文件中的 User-facing roles 一節。

列出所有政策

使用此頁面列出的任一存取政策，或使用 CLI AWS 擷取所有可用存取政策的清單：

```
aws eks list-access-policies
```

預期輸出應如下所示 (為簡潔起見已省略部分內容)：

```
{
    "accessPolicies": [
        {
            "name": "AmazonAIOpsAssistantPolicy",
            "arn": "arn:aws:eks::aws:cluster-access-policy/AmazonAIOpsAssistantPolicy"
        },
        {
            "name": "AmazonARCRegionSwitchScalingPolicy",
            "arn": "arn:aws:eks::aws:cluster-access-policy/
AmazonARCRegionSwitchScalingPolicy"
        },
        {
            "name": "AmazonEKSAdminPolicy",
            "arn": "arn:aws:eks::aws:cluster-access-policy/AmazonEKSAdminPolicy"
        },
        {
            "name": "AmazonEKSAdminViewPolicy",
            "arn": "arn:aws:eks::aws:cluster-access-policy/AmazonEKSAdminViewPolicy"
        },
        {
            "name": "AmazonEKSAutoNodePolicy",
            "arn": "arn:aws:eks::aws:cluster-access-policy/AmazonEKSAutoNodePolicy"
        }
        // Additional policies omitted
    ]
}
```

AmazonEKSAdminPolicy

此存取政策包含向相應 IAM 主體授予對資源的大部分許可的許可。當將之與某個存取項目關聯時，其存取範圍通常是一個或多個 Kubernetes 命名空間。如果您希望相應 IAM 主體對叢集上的所有資源具有

管理員存取權，請將 the section called “AmazonEKSClusterAdminPolicy” 存取政策與相應存取項目關聯。

ARN – arn:aws: eks::aws:cluster-access-policy/AmazonEKSAdminPolicy

Kubernetes API 群組	Kubernetes 資源	Kubernetes 動詞 (許可)
apps	daemonsets , deployments , deployments/rollback , deployments/scale , replicasets , replicasets/scale, statefulsets , statefulsets/scale	create, delete, deletecollection , patch, update
apps	controllerrevisions , daemonsets , daemonsets/status , deployments , deployments/scale , deployments/status , replicasets , replicasets/scale , replicasets/status , statefulsets , statefulsets/scale , statefulsets/status	get, list, watch
authorization.k8s.io	localsubjectaccessreviews	create
autoscaling	horizontalpodautoscalers	create, delete, deletecollection , patch, update
autoscaling	horizontalpodautoscalers , horizontalpodautoscalers/status	get, list, watch

Kubernetes API 群組	Kubernetes 資源	Kubernetes 動詞 (許可)
batch	cronjobs, jobs	create, delete, deletecollection , patch, update
batch	cronjobs, cronjobs/status , jobs, jobs/status	get, list, watch
discovery.k8s.io	endpointslices	get, list, watch
extensions	daemonsets , deployments , deployments/rollback , deployments/scale , ingresses , networkpolicies , replicasets , replicasets/scale , replicationcontrollers/scale	create, delete, deletecollection , patch, update
extensions	daemonsets , daemonsets/status , deployments , deployments/scale , deployments/status , ingresses , ingresses/status , networkpolicies , replicasets , replicasets/scale , replicasets/status , replicationcontrollers/scale	get, list, watch
networking.k8s.io	ingresses , ingresses/status , networkpolicies	get, list, watch
networking.k8s.io	ingresses , networkpolicies	create, delete, deletecollection , patch, update

Kubernetes API 群組	Kubernetes 資源	Kubernetes 動詞 (許可)
policy	poddisruptionbudgets	create, delete, deletecollection , patch, update
policy	poddisruptionbudgets , poddisruptionbudgets/status	get, list, watch
rbac.authorization.k8s.io	rolebindings , roles	create, delete, deletecollection , get, list, patch, update, watch
	configmaps , endpoints , persistentvolumeclaims , persistentvolumeclaims/status , pods, replicationcontrollers , replicationcontrollers/scale , serviceaccounts , services, services/status	get, list, watch
	pods/attach , pods/exec , pods/portforward , pods/proxy , secrets, services/proxy	get, list, watch

Kubernetes API 群組	Kubernetes 資源	Kubernetes 動詞 (許可)
	configmaps , events, persistentvolumeclaims , replicationcontrollers , replicationcontrollers/scale , secrets, serviceaccounts , services, services/proxy	create, delete, deletecollection , patch, update
	pods, pods/attach , pods/exec , pods/portforward , pods/proxy	create, delete, deletecollection , patch, update
	serviceaccounts	impersonate
	bindings, events, limitranges , namespaces/status , pods/log, pods/status , replicationcontrollers/status , resourcequotas , resourcequotas/status	get, list, watch
	namespaces	get, list, watch

AmazonEKSClusterAdminPolicy

此存取政策包含授予相應 IAM 主體對叢集的管理員存取權的許可。當將之與某個存取項目關聯時，其存取範圍通常是叢集，而不是某個 Kubernetes 命名空間。如果您希望限制相應 IAM 主體的管理範圍，請考慮將 the section called “AmazonEKSAdminPolicy” 存取政策與相應存取項目關聯。

ARN – arn:aws: eks::aws:cluster-access-policy/AmazonEKSClusterAdminPolicy

Kubernetes API 群組	Kubernetes nonResourceURLs	Kubernetes 資源	Kubernetes 動詞 (許可)
*		*	*
	*		*

AmazonEKSAdminViewPolicy

此存取政策包含授予相應 IAM 主體在叢集中列出/檢視所有資源的許可。請注意，這包含 Kubernetes 秘密。

ARN – `arn:aws: eks::aws:cluster-access-policy/AmazonEKSAdminViewPolicy`

Kubernetes API 群組	Kubernetes 資源	Kubernetes 動詞 (許可)
*	*	`get`, `list`, `watch`

AmazonEKSEditPolicy

此存取政策包含允許相應 IAM 主體編輯大多數 Kubernetes 資源的許可。

ARN – `arn:aws: eks::aws:cluster-access-policy/AmazonEKSEditPolicy`

Kubernetes API 群組	Kubernetes 資源	Kubernetes 動詞 (許可)
`apps`	`daemonsets` , `deployments` , `deployments/rollback` , `deployments/scale` , `replicasets` , `replicasets/scale`, `statefulsets` , `statefulsets/scale`	`create`, `delete`, `deletecollection` , `patch`, `update`
`apps`	`controllerrevisions` , `daemonsets` , `daemonsets/status` , `deploymen`	`get`, `list`, `watch`

Kubernetes API 群組	Kubernetes 資源	Kubernetes 動詞 (許可)
	ts , deployments/scale , deployments/status , replicasets , replicasets/scale , replicasets/status , statefulsets , statefulsets/scale , statefulsets/status	
autoscaling	horizontalpodautoscalers , horizontalpodautoscalers/status	get, list, watch
autoscaling	horizontalpodautoscalers	create, delete, deletecollection , patch, update
batch	cronjobs, jobs	create, delete, deletecollection , patch, update
batch	cronjobs, cronjobs/status , jobs, jobs/status	get, list, watch
discovery.k8s.io	endpointslices	get, list, watch
extensions	daemonsets , deployments , deployments/rollback , deployments/scale , ingresses , networkpolicies , replicasets , replicasets/scale , replicationcontrollers/scale	create, delete, deletecollection , patch, update

Kubernetes API 群組	Kubernetes 資源	Kubernetes 動詞 (許可)
extensions	daemonsets , daemonsets/status , deployments , deployments/scale , deployments/status , ingresses , ingresses/status , networkpolicies , replicasets , replicasets/scale , replicasets/status , replicationcontrollers/scale	get, list, watch
networking.k8s.io	ingresses , networkpolicies	create, delete, deletecollection , patch, update
networking.k8s.io	ingresses , ingresses/status , networkpolicies	get, list, watch
policy	poddisruptionbudgets	create, delete, deletecollection , patch, update
policy	poddisruptionbudgets , poddisruptionbudgets/status	get, list, watch
	namespaces	get, list, watch
	pods/attach , pods/exec , pods/portforward , pods/proxy , secrets, services/proxy	get, list, watch
	serviceaccounts	impersonate

Kubernetes API 群組	Kubernetes 資源	Kubernetes 動詞 (許可)
	pods, pods/attach , pods/exec , pods/port forward , pods/proxy	create, delete, deletecollection , patch, update
	configmaps , events, persistentvolumeclaims , replicationcontrollers , replicationcontrollers/scale , secrets, serviceaccounts , services, services/proxy	create, delete, deletecollection , patch, update
	configmaps , endpoints , persistentvolumeclaims , persistentvolumeclaims/status , pods, replicationcontrollers , replicationcontrollers/scale , serviceaccounts , services, services/status	get, list, watch
	bindings, events, limitranges , namespaces/status , pods/log, pods/status , replicationcontrollers/status , resourcequotas , resourcequotas/status	get, list, watch

AmazonEKSViewPolicy

此存取政策包含允許相應 IAM 主體檢視大多數 Kubernetes 資源的許可。

ARN – `arn:aws: eks::aws:cluster-access-policy/AmazonEKSViewPolicy`

Kubernetes API 群組	Kubernetes 資源	Kubernetes 動詞 (許可)
apps	controllerrevisions , daemonsets , daemonsets/status , deployments , deployments/scale , deployments/status , replicasets , replicasets/scale , replicasets/status , statefulsets , statefulsets/scale , statefulsets/status	get, list, watch
autoscaling	horizontalpodautoscalers , horizontalpodautoscalers/status	get, list, watch
batch	cronjobs, cronjobs/status , jobs, jobs/status	get, list, watch
discovery.k8s.io	endpointslices	get, list, watch
extensions	daemonsets , daemonsets/status , deployments , deployments/scale , deployments/status , ingresses , ingresses/status , networkpolicies , replicasets , replicasets/scale ,	get, list, watch

Kubernetes API 群組	Kubernetes 資源	Kubernetes 動詞 (許可)
	replicasets/status , replicationcontrollers/scale	
networking.k8s.io	ingresses , ingresses/status , networkpolicies	get, list, watch
policy	poddisruptionbudgets , poddisruptionbudgets/status	get, list, watch
	configmaps , endpoints , persistentvolumeclaims , persistentvolumeclaims/status , pods, replicationcontrollers , replicationcontrollers/scale , serviceaccounts , services, services/status	get, list, watch
	bindings, events, limitranges , namespaces/status , pods/log, pods/status , replicationcontrollers/status , resourcequotas , resourcequotas/status	get, list, watch
	namespaces	get, list, watch

AmazonEKSPodIdentityPolicy

此存取政策包含允許 IAM 主體列出 Kubernetes Pod 的許可。此政策由 Amazon EKS 服務連結角色在內部使用。

> **Note**
>
> 此政策僅針對 AWS 服務連結角色指定，無法與客戶受管角色搭配使用。

ARN – `arn:aws: eks::aws:cluster-access-policy/AmazonEKSPodIdentityPolicy`

Kubernetes API 群組	Kubernetes 資源	Kubernetes 動詞 (許可)
	`pods`	`list`

AmazonEKSSecretAdminPolicy

此存取政策包含允許 IAM 主體完整存取 Kubernetes Secrets 的許可。

ARN – `arn:aws: eks::aws:cluster-access-policy/AmazonEKSSecretAdminPolicy`

Kubernetes API 群組	Kubernetes 資源	Kubernetes 動詞 (許可)
	`secrets`	*

AmazonEKSSecretReaderPolicy

此存取政策包含允許 IAM 主體讀取 Kubernetes 秘密的許可。

ARN – `arn:aws: eks::aws:cluster-access-policy/AmazonEKSSecretReaderPolicy`

Kubernetes API 群組	Kubernetes 資源	Kubernetes 動詞 (許可)
	`secrets`	`get`, `list`, `watch`

AmazonEKSAutoNodePolicy

ARN – `arn:aws: eks::aws:cluster-access-policy/AmazonEKSAutoNodePolicy`

此政策包含下列許可，這些許可能讓 Amazon EKS 元件完成下列任務：

- `kube-proxy` – 監控網路端點和服務，並管理相關事件。這可啟用整個叢集的網路代理功能。
- `ipamd` – 管理 AWS VPC 聯網資源和容器網路介面 (CNI)。這可讓 IP 位址管理常駐程式處理 Pod 聯網。
- `coredns` – 存取端點與服務等服務探索資源。這可在叢集內啟用 DNS 解析。
- `ebs-csi-driver` – 使用 Amazon EBS 磁碟區的儲存相關資源。這允許持續性磁碟區的動態佈建與附加。
- `neuron` – 監控 AWS Neuron 裝置的節點和 Pod。這可讓您管理 AWS Inferentia 和 Trainium 加速器。
- `node-monitoring-agent` – 存取節點診斷與事件。這實現了叢集運作狀態監控與診斷收集。

每個元件都使用專用服務帳戶，且僅限於其特定功能所需的許可。

如果您在 NodeClass 中手動指定節點 IAM 角色，則需要建立將新節點 IAM 角色與此存取政策建立關聯的存取項目。

AmazonEKSBlockStoragePolicy

> **Note**
>
> 此政策僅針對 AWS 服務連結角色指定，無法與客戶受管角色搭配使用。

ARN – `arn:aws: eks::aws:cluster-access-policy/AmazonEKSBlockStoragePolicy`

此政策包含允許 Amazon EKS 管理儲存操作的領導者選擇和協調資源的許可：

- `coordination.k8s.io` – 建立和管理用於領導者選擇的租約物件。這使得 EKS 儲存元件能夠透過領導者選擇機制在整個叢集中協調其活動。

該政策的範圍限定於 EKS 儲存元件使用的特定租約資源，避免與叢集中其他協調資源發生存取衝突。

當啟用自動模式時，Amazon EKS 會自動為叢集 IAM 角色建立帶有此存取政策的存取項目，以確保區塊儲存功能正常運作所需的許可到位。

AmazonEKSLoadBalancingPolicy

ARN – `arn:aws: eks::aws:cluster-access-policy/AmazonEKSLoadBalancingPolicy`

此政策包含允許 Amazon EKS 管理領導者選擇資源以進行負載平衡的許可：

- `coordination.k8s.io` – 建立和管理用於領導者選擇的租約物件。此功能可讓 EKS 負載平衡元件透過選擇領導者，協調多個複本間的活動。

此政策範圍專門針對負載平衡租約資源，確保協調運作順暢，同時防止存取叢集中的其他租約資源。

當啟用自動模式時，Amazon EKS 會自動為叢集 IAM 角色建立帶有此存取政策的存取項目，以確保聯網功能正常運作所需的許可到位。

AmazonEKSNetworkingPolicy

ARN – `arn:aws: eks::aws:cluster-access-policy/AmazonEKSNetworkingPolicy`

此政策包含允許 Amazon EKS 管理領導者選擇資源以進行聯網的許可：

- `coordination.k8s.io` – 建立和管理用於領導者選擇的租約物件。這可讓 EKS 聯網元件可透過選擇領導者來協調 IP 位址分配活動。

此政策範圍專門針對聯網租約資源，確保協調運作順暢，同時防止存取叢集中的其他租約資源。

當啟用自動模式時，Amazon EKS 會自動為叢集 IAM 角色建立帶有此存取政策的存取項目，以確保聯網功能正常運作所需的許可到位。

AmazonEKSComputePolicy

> **Note**
>
> 此政策僅針對 AWS 服務連結角色指定，無法與客戶受管角色搭配使用。

ARN – `arn:aws: eks::aws:cluster-access-policy/AmazonEKSComputePolicy`

此政策包含允許 Amazon EKS 管理領導者選擇資源以進行運算操作的許可：

- `coordination.k8s.io` – 建立和管理用於領導者選擇的租約物件。這讓 EKS 運算元件能透過選擇領導者來協調節點擴展活動。

政策範圍專門針對運算管理租約資源，同時允許叢集中所有租約資源的基本讀取權限 (get、watch)。

當啟用自動模式時，Amazon EKS 會自動為叢集 IAM 角色建立帶有此存取政策的存取項目，以確保運算管理功能正常運作所需的許可到位。

AmazonEKSBlockStorageClusterPolicy

> **Note**
>
> 此政策僅針對 AWS 服務連結角色指定，無法與客戶受管角色搭配使用。

ARN – `arn:aws: eks::aws:cluster-access-policy/AmazonEKSBlockStorageClusterPolicy`

此政策會授予 Amazon EKS 自動模式的區塊儲存功能所需的許可。它能有效管理 Amazon EKS 叢集內的區塊儲存資源。政策包含下列許可：

CSI 驅動程式管理：

- 建立、讀取、更新和刪除 CSI 驅動程式，特別是區塊儲存。

磁碟區管理：

- 列出、監看、建立、更新、修補及刪除持續性磁碟區。
- 列出、監看及更新持續性磁碟區宣告。
- 修補持續性磁碟區宣告狀態。

節點與 Pod 互動：

- 讀取節點與 Pod 資訊。
- 管理與儲存操作相關的事件。

儲存類別與屬性：

- 讀取儲存類別與 CSI 節點。
- 讀取磁碟區屬性類別。

磁碟區連接：

- 列出、監看及修改磁碟區連接及其狀態。

快照操作：

- 管理磁碟區快照、快照內容及快照類別。
- 處理磁碟區群組快照及相關資源的操作。

此政策旨在為執行於自動模式的 Amazon EKS 叢集提供全面的區塊儲存管理支援。它結合了區塊儲存磁碟卷的佈建、連接、調整大小與快照等多種操作許可。

當啟用自動模式時，Amazon EKS 會自動為叢集 IAM 角色建立帶有此存取政策的存取項目，以確保區塊儲存功能正常運作所需的許可到位。

AmazonEKSComputeClusterPolicy

> **Note**
>
> 此政策僅針對 AWS 服務連結角色指定，無法與客戶受管角色搭配使用。

ARN – `arn:aws: eks::aws:cluster-access-policy/AmazonEKSComputeClusterPolicy`

此政策會授予 Amazon EKS 自動模式的運算管理功能所需的許可。它能高效協同運作與擴展 Amazon EKS 叢集內的運算資源。政策包含下列許可：

節點管理：

- 建立、讀取、更新、刪除及管理 NodePools 與 NodeClaims 的狀態。
- 管理 NodeClass，包括建立、修改與刪除。

排程與資源管理：

- 對 Pod、節點、持續性磁碟區、持續性磁碟區宣告、複寫控制器和命名空間的讀取權限。
- 對儲存類別、CSI 節點及磁碟區連接的讀取權限。
- 列出及監看部署、常駐程式集、複本集與有狀態集。

- 讀取 Pod 中斷預算。

事件處理：

- 建立、讀取及管理叢集事件。

節點解除佈建與 Pod 移出：

- 更新、修補及刪除節點。
- 必要時建立 Pod 移出及刪除 Pod。

自訂資源定義 (CRD) 管理：

- 建立新 CRD。
- 管理與節點管理相關的特定 CRD (NodeClasses、NodePools、NodeClaims 及 NodeDiagnostics)。

此政策旨在為執行於自動模式的 Amazon EKS 叢集提供全面的運算管理支援。它結合了節點佈建、排程、擴展與資源最佳化等多種操作的許可。

當啟用自動模式時，Amazon EKS 會自動為叢集 IAM 角色建立帶有此存取政策的存取項目，以確保運算管理功能正常運作所需的許可到位。

AmazonEKSLoadBalancingClusterPolicy

ARN – `arn:aws: eks::aws:cluster-access-policy/AmazonEKSLoadBalancingClusterPolicy`

此政策會授予 Amazon EKS 自動模式的負載平衡功能所需的許可。它能實現 Amazon EKS 叢集內負載平衡資源的高效管理與組態。政策包含下列許可：

事件與資源管理：

- 建立與修補事件。
- 對 Pod、節點、端點及命名空間的讀取權限。
- 更新 Pod 狀態。

服務與傳入管理：

- 服務及其狀態的完整管理。
- 全面控制傳入及其狀態。
- 端點配量與傳入類別的讀取權限。

目標群組連結：

- 建立與修改目標群組連結及其狀態。
- 對傳入類別參數的讀取權限。

自訂資源定義 (CRD) 管理：

- 建立與讀取所有 CRD。
- targetgroupbindings.eks.amazonaws.com 與 ingressclassparams.eks.amazonaws.com CRD 的特定管理。

Webhook 組態：

- 建立與讀取變動及驗證 Webhook 組態。
- 管理 eks-load-balancing-webhook 組態。

此政策旨在為執行於自動模式的 Amazon EKS 叢集提供全面的負載平衡管理支援。它結合了各種操作的許可，包括服務公開、輸入路由，以及與 AWS 負載平衡服務的整合。

當啟用自動模式時，Amazon EKS 會自動為叢集 IAM 角色建立帶有此存取政策的存取項目，以確保負載平衡功能正常運作所需的許可到位。

AmazonEKSNetworkingClusterPolicy

ARN – `arn:aws: eks::aws:cluster-access-policy/AmazonEKSNetworkingClusterPolicy`

AmazonEKSNetworkingClusterPolicy

此政策會授予 Amazon EKS 自動模式的聯網功能所需的許可。它能實現 Amazon EKS 叢集內聯網資源的高效管理與組態。政策包含下列許可：

節點與 Pod 管理：

- 對 NodeClass 及其狀態的讀取權限。
- 對 NodeClaims 及其狀態的讀取權限。
- 對 Pod 的讀取權限。

CNI 節點管理：

- 對 CNINodes 及其狀態的許可，包含建立、讀取、更新、刪除及修補。

自訂資源定義 (CRD) 管理：

- 建立與讀取所有 CRD。
- 對 cninodes.eks.amazonaws.com CRD 的特定管理 (更新、修補、刪除)。

管理事件：

- 建立與修補事件。

此政策旨在為執行於自動模式的 Amazon EKS 叢集提供全面的聯網管理支援。它結合了節點聯網組態、CNI (容器網路介面) 管理及相關自訂資源處理等多種操作的許可。

此政策允許聯網元件與節點相關資源互動、管理 CNI 特定的節點組態，並處理叢集中聯網操作關鍵的自訂資源。

當啟用自動模式時，Amazon EKS 會自動為叢集 IAM 角色建立帶有此存取政策的存取項目，以確保聯網功能正常運作所需的許可到位。

AmazonEKSHybridPolicy

> **Note**
>
> 此政策僅針對 AWS 服務連結角色指定，無法與客戶受管角色搭配使用。

此存取政策包含了授予 EKS 存取叢集節點的許可。當將之與某個存取項目關聯時，其存取範圍通常是叢集，而不是某個 Kubernetes 命名空間。此政策供 Amazon EKS 混合節點使用。

ARN – `arn:aws: eks::aws:cluster-access-policy/AmazonEKSHybridPolicy`

Kubernetes API 群組	Kubernetes nonResourceURLs	Kubernetes 資源	Kubernetes 動詞 (許可)
*		nodes	list

AmazonEKSClusterInsightsPolicy

> **Note**
> 此政策僅針對 AWS 服務連結角色指定，無法與客戶受管角色搭配使用。

ARN – arn:aws: eks::aws:cluster-access-policy/AmazonEKSClusterInsightsPolicy

此政策授予 Amazon EKS Cluster Insights 功能的唯讀許可。政策包含下列許可：

節點存取：- 列出與檢視叢集節點 - 讀取節點狀態資訊

DaemonSet 存取：- 對 kube-proxy 組態的讀取權限

此政策由 EKS 服務為 Cluster Insights 自動管理。如需詳細資訊，請參閱the section called “叢集洞察”。

AWSBackupFullAccessPolicyForBackup

ARN – arn:aws: eks::aws:cluster-access-policy/AWSBackupFullAccessPolicyForBackup

AWSBackupFullAccessPolicyForBackup

此政策授予 AWS Backup 管理和建立 EKS 叢集備份所需的許可。此政策包含以下許可：

Kubernetes API 群組	Kubernetes 資源	Kubernetes 動詞 (許可)
*	*	list, get

AWSBackupFullAccessPolicyForRestore

ARN – arn:aws: eks::aws:cluster-access-policy/AWSBackupFullAccessPolicyForRestore

AWSBackupFullAccessPolicyForRestore

此政策授予 AWS Backup 管理和還原 EKS 叢集備份所需的許可。此政策包含以下許可：

Kubernetes API 群組	Kubernetes 資源	Kubernetes 動詞 (許可)
*	*	list, get, create
rbac.authorization.k8s.io	roles, clusterroles	create, update, escalate, bind

AmazonEKSACKPolicy

ARN – arn:aws: eks::aws:cluster-access-policy/AmazonEKSACKPolicy

此政策授予 AWS Controllers for Kubernetes (ACK) 功能從 Kubernetes 管理 AWS 資源所需的許可。政策包含下列許可：

ACK 自訂資源管理：

- 完整存取 50 種以上服務的所有 ACK AWS 服務自訂資源，包括 S3、RDS、DynamoDB、Lambda、EC2 等。
- 建立、讀取、更新和刪除 ACK 自訂資源定義。

命名空間存取：

- 資源組織的命名空間讀取存取權。

領導者選擇：

- 建立和讀取領導者選擇的協調租用。
- 更新和刪除特定 ACK 服務控制器租用。

管理事件：

- 建立和修補 ACK 操作的事件。

此政策旨在支援透過 Kubernetes APIs全面的 AWS 資源管理。Amazon EKS 會自動為您在建立 ACK 功能時提供的功能 IAM 角色建立具有此存取政策的存取項目。

Kubernetes API 群組	Kubernetes 資源	Kubernetes 動詞 (許可)
	namespaces	get, watch, list
services.k8s.aws , acm.services.k8s.aws , acmpca.services.k8s.aws , apigateway.services.k8s.aws , apigatewayv2.services.k8s.aws , applicationautoscaling.services.k8s.aws , athena.services.k8s.aws , bedrock.services.k8s.aws , bedrockagent.services.k8s.aws , bedrockagentcorecontrol.services.k8s.aws , cloudfront.services.k8s.aws , cloudtrail.services.k8s.aws , cloudwatch.services.k8s.aws , cloudwatchlogs.services.k8s.aws , codeartifact.services.k8s.aws , cognitoidentityprovider.ser	*	*

Kubernetes API 群組	Kubernetes 資源	Kubernetes 動詞 (許可)
vices.k8s.aws , documentdb.service s.k8s.aws ,dynamodb. services.k8s.aws , ec2.services.k8s.aws , ecr.services.k8s.aws , ecrpublic.services .k8s.aws ,ecs.servi ces.k8s.aws , efs.services.k8s.aws , eks.services.k8s.aws , elasticache.servic es.k8s.aws ,elbv2.ser vices.k8s.aws , emrcontainers.serv ices.k8s.aws , eventbridge.servic es.k8s.aws ,iam.servi ces.k8s.aws , kafka.services.k8s .aws ,keyspaces .services.k8s.aws , kinesis.services.k 8s.aws ,kms.servi ces.k8s.aws , lambda.services.k8 s.aws ,memorydb. services.k8s.aws , mq.services.k8s.aw s ,networkfirewall.se rvices.k8s.aws , opensearchservice. services.k8s.aws , organizations.serv		

Kubernetes API 群組	Kubernetes 資源	Kubernetes 動詞 (許可)
ices.k8s.aws , pipes.services.k8s.aws , prometheusservice.services.k8s.aws , ram.services.k8s.aws , rds.services.k8s.aws , recyclebin.services.k8s.aws , route53.services.k8s.aws , route53resolver.services.k8s.aws , s3.services.k8s.aws , s3control.services.k8s.aws , sagemaker.services.k8s.aws , secretsmanager.services.k8s.aws , ses.services.k8s.aws , sfn.services.k8s.aws , sns.services.k8s.aws , sqs.services.k8s.aws , ssm.services.k8s.aws , wafv2.services.k8s.aws		
coordination.k8s.io	leases	create, get, list, watch
coordination.k8s.io	leases （僅限特定 ACK 服務控制器租用）	delete, update, patch
	events	create, patch
apiextensions.k8s.io	customresourcedefinitions	*

AmazonEKSArgoCDClusterPolicy

ARN – `arn:aws: eks::aws:cluster-access-policy/AmazonEKSArgoCDClusterPolicy`

此政策會授予 Argo CD 功能所需的叢集層級許可，以探索資源和管理叢集範圍的物件。政策包含下列許可：

命名空間管理：

- 建立、讀取、更新和刪除應用程式命名空間管理的命名空間。

自訂資源定義管理：

- 管理 Argo CD 特定 CRDs（應用程式、AppProjects、ApplicationSets)。

API 探索：

- 用於資源探索的 Kubernetes API 端點讀取存取權。

此政策旨在支援叢集層級 Argo CD 操作，包括命名空間管理和 CRD 安裝。Amazon EKS 會自動為您在建立 Argo CD 功能時提供的功能 IAM 角色建立具有此存取政策的存取項目。

Kubernetes API 群組	Kubernetes nonResourceURLs	Kubernetes 資源	Kubernetes 動詞 (許可)
		`namespaces`	`create`, `get`, `update`, `patch`, `delete`
`apiextensions.k8s.io`		`customresourcedefinitions`	`create`
`apiextensions.k8s.io`		`customresourcedefinitions` （僅限 Argo CD CRDs)	`get`, `update`, `patch`, `delete`

Kubernetes API 群組	Kubernetes nonResourceURLs	Kubernetes 資源	Kubernetes 動詞 (許可)
	/api, /api/*, /apis, /apis/*		get

AmazonEKSArgoCDPolicy

ARN – arn:aws: eks::aws:cluster-access-policy/AmazonEKSArgoCDPolicy

此政策會授予 Argo CD 功能部署和管理應用程式所需的命名空間層級許可。政策包含下列許可：

秘密管理：

- 完整存取 Git 登入資料和叢集秘密的秘密。

ConfigMap 存取：

- 如果客戶嘗試使用不支援的 Argo CD ConfigMaps，請讀取 ConfigMaps 的存取權以傳送警告。

管理事件：

- 讀取和建立事件以進行應用程式生命週期追蹤。

Argo CD 資源管理：

- 應用程式、ApplicationSets 和 AppProjects 的完整存取權。
- 管理 Argo CD 資源的最終處理者和狀態。

此政策旨在支援命名空間層級的 Argo CD 操作，包括應用程式部署和管理。Amazon EKS 會自動針對您在 Argo CD 功能建立範圍為 Argo CD 命名空間時提供的功能 IAM 角色，使用此存取政策建立存取項目。

Kubernetes API 群組	Kubernetes 資源	Kubernetes 動詞 (許可)
	secrets	*

Kubernetes API 群組	Kubernetes 資源	Kubernetes 動詞 (許可)
	configmaps	get, list, watch
	events	get, list, watch, patch, create
argoproj.io	applications , applications/finalizers , applications/status , applicationsets , applicationsets/finalizers , applicationsets/status , appprojects , appprojects/finalizers , appprojects/status	*

AmazonEKSKROPolicy

ARN – arn:aws: eks::aws:cluster-access-policy/AmazonEKSKROPolicy

此政策會授予 kro (Kube Resource Orchestrator) 功能建立和管理自訂 Kubernetes APIs 所需的許可。政策包含下列許可：

kro 資源管理：

- 完整存取所有 kro 資源，包括 ResourceGraphDefinitions 和自訂資源執行個體。

自訂資源定義管理：

- 為 ResourceGraphDefinitions 定義的自訂 APIs建立、讀取、更新和刪除 CRDs。

領導者選擇：

- 建立和讀取領導者選擇的協調租用。
- 更新和刪除 kro 控制器租用。

管理事件：

- 建立和修補 kro 操作的事件。

此政策旨在透過 kro 支援全面的資源合成和自訂 API 管理。Amazon EKS 會自動為您在建立 kro 功能時提供的功能 IAM 角色建立具有此存取政策的存取項目。

Kubernetes API 群組	Kubernetes 資源	Kubernetes 動詞 (許可)
kro.run	*	*
apiextensions.k8s.io	customresourcedefinitions	*
coordination.k8s.io	leases	create, get, list, watch
coordination.k8s.io	leases （僅限 Kro 控制器租用）	delete, update, patch
	events	create, patch

存取政策更新

檢視有關存取政策更新 (自引進以來) 的詳細資訊。如需有關此頁面變更的自動提醒，請訂閱 文件歷史紀錄 中的 RSS 摘要。

變更	描述	日期
新增 EKS 功能的政策	發佈 AmazonEKSACKPolicy 、AmazonEKSArgoCDPolicy 、 AmazonEKSArgoCDClusterPolicy 和 AmazonEKSKROPolicy 以管理 EKS 功能	2025 年 11 月 22 日
加 AmazonEKSSecretReaderPolicy	為秘密的唯讀存取新增新政策	2025 年 11 月 6 日

變更	描述	日期
新增 EKS Cluster Insights 政策	發布 AmazonEKSClusterInsightsPolicy	2024 年 12 月 2 日
新增 Amazon EKS 混合政策	發布 AmazonEKSHybridPolicy	2024 年 12 月 2 日
新增 Amazon EKS 自動模式政策	這些存取政策提供叢集 IAM 角色和節點 IAM 角色呼叫 Kubernetes APIs許可。 AWS 使用這些許可來自動化儲存、運算和聯網資源的例行任務。	2024 年 12 月 2 日
加 AmazonEKSAdminViewPolicy	新增用於擴展檢視存取權限的新政策，包含如 Secrets 等資源。	2024 年 4 月 23 日
引進存取政策。	Amazon EKS 引進了存取政策。	2023 年 5 月 29 日

變更驗證模式以使用存取項目

若要開始使用存取項目，您必須將叢集的身分驗證模式變更為 API_AND_CONFIG_MAP 或 API 模式。這會新增用於存取項目的 API。

AWS 主控台

1. 開啟 Amazon EKS 主控台。
2. 選擇要在其中建立存取項目的叢集名稱。
3. 選擇存取索引標籤。
4. 身分驗證模式顯示叢集目前的身分驗證模式。如果模式顯示 EKS API，則表示您已經可以新增存取項目，您可以略過餘下步驟。
5. 選擇管理存取。
6. 對於叢集身分驗證模式，請選取帶有 EKS API 的模式。請注意，您無法將身分驗證模式變更回移除 EKS API 和存取項目的模式。

7. 選擇儲存變更。Amazon EKS 開始更新叢集，叢集的狀態變更為「正在更新」，並且將變更記錄在更新歷史記錄索引標籤中。
8. 等待叢集狀態變回「作用中」。當叢集的狀態為「作用中」時，您可以依照 the section called “建立存取項目” 中的步驟為 IAM 主體新增對叢集的存取權。

AWS CLI

1. 依據《AWS 命令列介面使用者指南》中的安裝內容所述，安裝 AWS CLI。
2. 執行下列命令。使用您叢集的名稱取代 *my-cluster*。如果您想永久停用 ConfigMap 方法，則請用 API 取代 API_AND_CONFIG_MAP。

   Amazon EKS 開始更新叢集，叢集的狀態變更為「正在更新」，並且變更記錄在 aws eks list-updates 中。

   ```
   aws eks update-cluster-config --name my-cluster --access-config
    authenticationMode=API_AND_CONFIG_MAP
   ```
3. 等待叢集狀態變回「作用中」。當叢集的狀態為「作用中」時，您可以依照 the section called “建立存取項目” 中的步驟為 IAM 主體新增對叢集的存取權。

要求使用的平台版本

如需使用存取項目，叢集的平台版本必須與下表所列版本相同或更新，或是晚於表格中列出版本的 Kubernetes 版本。若未列示 Kubernetes 版本，則所有平台版本皆支援這些存取項目。

Kubernetes 版本	平台版本
未列示	全部支援
1.30	eks.2
1.29	eks.1
1.28	eks.6

如需詳細資訊，請參閱 platform-versions。

建立存取項目

在建立存取項目之前，請考慮以下事項：

- 正確設定的身分驗證模式。請參閱 the section called “身分驗證方式”。
- 存取項目包含一個 (且僅一個) 現有 IAM 主體的 Amazon Resource Name (ARN)。一個 IAM 主體不能包含在多個存取項目中。對於您指定的 ARN 的其他考量：
  - IAM 最佳實務建議使用具有短期憑證的 IAM 角色存取叢集，而不是使用具有長期憑證的 IAM 使用者進行存取。如需詳細資訊，請參閱《IAM 使用者指南》中的要求人類使用者使用聯合身分提供者來 AWS 使用臨時憑證存取 。
  - 如果 ARN 用於 IAM 角色，則它可以包含路徑。`aws-auth ConfigMap` 項目中的 ARN 則不能包含路徑。例如， ARN 可以是 `arn:aws: iam::<111122223333>:role/<development/apps/my-role>` 或 `arn:aws: iam::<111122223333>:role/<my-role>`。
  - 如果存取項目的類型是 以外的任何項目 `STANDARD`（請參閱有關 類型的下一個考量），則 ARN 必須位於叢集所在的相同 AWS 帳戶中。如果類型為 `STANDARD`，則 ARN 可以是與您叢集所在的帳戶相同或不同的 AWS 帳戶。
  - 建立存取項目後，您無法變更該 IAM 主體。
  - 如果您刪除具有此 ARN 的 IAM 主體，則存取項目並不會自動刪除。建議您刪除帶有您刪除之 IAM 主體的 ARN 的存取項目。如果您在不刪除存取項目的情況下重新建立 IAM 主體，則即使它具有相同的 ARN，存取項目也將無法運作。這是因為即使重新建立的 IAM 主體的 ARN 相同，對於重新建立的 IAM 主體， `roleID`或 `userID`（您可以使用 `aws sts get-caller-identity` AWS CLI 命令查看） 與原始 IAM 主體的 ARN 不同。即使您看不到存取項目的 IAM 主體 `roleID` 或 `userID`，Amazon EKS 也會將其與存取項目一起儲存。
- 每個存取項目都有一個類型。存取項目的類型取決於與其關聯的資源類型，並不定義許可。如果您不指定類型，則 Amazon EKS 會自動將類型設為 `STANDARD`
  - `EC2_LINUX` - 用於與 Linux 或 Bottlerocket 自我管理的節點搭配使用的 IAM 角色
  - `EC2_WINDOWS` - 用於與 Windows 自我管理的節點搭配使用的 IAM 角色
  - `FARGATE_LINUX` - 用於搭配 AWS Fargate (Fargate) 使用的 IAM 角色
  - `HYBRID_LINUX` - 用於與混合節點搭配使用的 IAM 角色
  - `STANDARD` - 未指定類型時為預設類型
  - `EC2` - 適用於 EKS 自動模式自訂節點類別。如需詳細資訊，請參閱the section called “建立節點類別存取項目”。
  - 建立存取項目後，您無法變更類型。

- 無需為用於受管節點群組或 Fargate 設定檔的 IAM 角色建立存取項目。EKS 將建立存取項目 (如果啟用)，或更新身分驗證組態映射 (如果存取項目不可用)
- 如果某個存取項目的類型為 STANDARD，則可以為其指定一個使用者名稱。如果您沒有指定使用者名稱的值，Amazon EKS 會根據存取項目的類型以及您指定的 IAM 主體是 IAM 角色還是 IAM 使用者，設定下列其中一個值。除非您有特定原因需要自己指定使用者名稱，否則建議您不要指定使用者名稱，而是讓 Amazon EKS 自動為您產生該使用者名稱。如果您自己指定使用者名稱：
  - 該名稱不能以 `system:`、`eks:`、`aws:`、`amazon:` 或 `iam:` 開頭。
  - 如果該使用者名稱用於 IAM 角色，則建議您在末尾新增 `{{SessionName}}` 或 `{{SessionNameRaw}}`。如果將 `{{SessionName}}` 或 `{{SessionNameRaw}}` 新增至使用者名稱，則必須在 {{SessionName}} 之前必須有冒號。擔任此角色時，擔任角色時指定的 AWS STS 工作階段名稱名稱會自動傳遞至叢集，並會出現在 CloudTrail 日誌中。例如，使用者名稱不能為 `john{{SessionName}}`。使用者名稱必須為 `:john{{SessionName}}` 或 `jo:hn{{SessionName}}`。冒號只需位於 `{{SessionName}}` 之前。以下資料表中 Amazon EKS 產生的使用者名稱包含 ARN。由於 ARN 包含冒號，因此它符合此項要求。如果使用者名稱中不包含 `{{SessionName}}`，則不需要冒號。請注意，在 `{{SessionName}}` 中，特殊字元 "@" 在工作階段名稱中會被取代為 "-"。`{{SessionNameRaw}}` 會保留工作階段名稱中的所有特殊字元。

IAM 主體類型	Type	Amazon EKS 自動設定的使用者名稱值
使用者	STANDARD	使用者的 ARN。範例：`arn:aws: iam::<111122223333>:user/<my-user>`
Role	STANDARD	角色被擔任時的 STS ARN。Amazon EKS 將 `{{SessionName}}` 附加到角色。 範例： `arn:aws:sts::<111122223333>:assumed-role/<my-role>/{{SessionName}}`

IAM 主體類型	Type	Amazon EKS 自動設定的使用者名稱值
		如果您指定的角色的 ARN 包含路徑，則 Amazon EKS 會將其從產生的使用者名稱中移除。
Role	`EC2_LINUX` 或 `EC2_WINDOWS`	`system:node:{{EC2PrivateDNSName}}`
Role	`FARGATE_LINUX`	`system:node:{{SessionName}}`
Role	`HYBRID_LINUX`	`system:node:{{SessionName}}`

建立存取項目後，您可以對使用者名稱進行變更。

- 如果某個存取項目的類型為 `STANDARD`，且您希望使用 Kubernetes RBAC 授權，則可以在該存取項目中新增一個或多個群組名稱。建立存取項目後，您可以新增和移除群組名稱。為了使 IAM 主體能夠存取叢集上的 Kubernetes 物件，您必須建立和管理 Kubernetes 角色型授權 (RBAC) 物件。在您的叢集上建立 Kubernetes `RoleBinding` 或 `ClusterRoleBinding` 物件，並將群組名稱指定為 `kind: Group` 的 `subject`。Kubernetes 會授權 IAM 主體存取您在 Kubernetes `Role` 或 `ClusterRole` 物件中指定的任何叢集物件，這些物件您也在繫結的 `roleRef` 中指定了。如果您指定群組名稱，則建議您熟悉 Kubernetes role-based 角色型授權 (RBAC) 物件。如需詳細資訊，請參閱 Kubernetes 文件中的使用 RBAC 授權。

> **Important**
>
> Amazon EKS 不會確認叢集上存在的任何 Kubernetes RBAC 物件是否包含您指定的任何群組名稱。例如，如果您為目前不存在的群組建立存取項目，Amazon EKS 會接受組態而不傳回錯誤，但在建立相符的 Kubernetes RBAC 資源之前，IAM 主體不會有任何許可。

您可以將 Amazon EKS 存取政策關聯到存取項目，而不是讓 Kubernetes 授權 IAM 主體存取叢集上的 Kubernetes 物件。Amazon EKS 授權 IAM 主體使用存取政策中的許可存取叢集上的 Kubernetes

物件。您可以將存取政策的許可範圍限定為您指定的 Kubernetes 命名空間。使用存取政策不需要您管理 Kubernetes RBAC 物件。如需詳細資訊，請參閱the section called “關聯存取政策”。

- 如果您建立類型為 EC2_LINUX 或 EC2_WINDOWS 的存取項目，則建立該存取項目的 IAM 主體必須具有 iam:PassRole 許可。如需詳細資訊，請參閱《IAM 使用者指南》中的授予使用者將角色傳遞至 AWS 服務的許可。
- 與標準 IAM 行為類似，存取項目的建立和更新採用最終一致模式，在初始 API 呼叫成功返回後可能需要幾秒鐘才能生效。您設計的應用程式必須能夠處理這些可能的延遲問題。建議您不要在應用程式的關鍵、高可用性程式碼路徑中包含存取項目建立或更新動作。而應在不常運作的、單獨的初始化或設定常式中進行 更改。另外，在生產工作流程套用這些變更之前，請務必確認變更已傳播完畢。
- 存取項目不支援服務連結角色。您無法建立主體 ARN 是服務連結角色的存取項目。您可以透過其 ARN 識別服務連結角色，其格式為 arn:aws: iam::*:role/aws-service-role/*。

您可以使用 AWS 管理主控台 或 CLI AWS 建立存取項目。

AWS 管理主控台

1. 開啟 Amazon EKS 主控台。
2. 選擇要在其中建立存取項目的叢集名稱。
3. 選擇存取索引標籤。
4. 選擇建立存取項目。
5. 對於 IAM 主體，選取現有 IAM 角色或使用者。IAM 最佳實務建議使用具有短期憑證的 IAM 角色存取叢集，而不是使用具有長期憑證的 IAM 使用者進行存取。如需詳細資訊，請參閱《IAM 使用者指南》中的要求人類使用者使用聯合身分提供者來 AWS 使用臨時憑證存取 。
6. 對於類型，如果存取項目針對用於自我管理 Amazon EC2 節點的節點角色，請選取 EC2 Linux 或 EC2 Windows。否則，請接受預設值 (標準)。
7. 如果您選擇的類型是標準並且您想要指定使用者名稱，則請輸入使用者名稱。
8. 如果您選擇的類型是標準並且您想要對 IAM 主體使用 Kubernetes RBAC 授權，則請為群組指定一個或多個名稱。如果不指定任何群組名稱並希望使用 Amazon EKS 授權，則您可以在後續步驟中或在建立存取項目後關聯存取政策。
9. (選用) 您可以使用標籤為存取項目指派標籤。例如，為了更輕鬆地找到具有相同標籤的所有資源而指定標籤。
10. 選擇下一步。
11. 在新增存取政策頁面上，如果您選擇的類型是標準並且希望 Amazon EKS 授予該 IAM 主體對叢集上的 Kubernetes 物件的許可，請完成下列步驟。否則請選擇 Next (下一步)。/

a. 對於政策名稱，請選擇存取政策。您無法檢視存取政策的許可，但其包含與 Kubernetes 面向使用者之 `ClusterRole` 物件中的許可類似的許可。如需詳細資訊，請參閱 Kubernetes 文件中的 User-facing roles 一節。

b. 請選擇下列其中一個選項：

- 叢集：如果您希望 Amazon EKS 授予該 IAM 主體存取政策中針對叢集上所有 Kubernetes 物件的許可，請選擇此選項。
- Kubernetes 命名空間：如果您希望 Amazon EKS 授予該 IAM 主體存取政策中針對叢集上特定 Kubernetes 命名空間中所有 Kubernetes 物件的許可，則請選擇此選項。對於命名空間，請輸入叢集上 Kubernetes 命名空間的名稱。如果要新增其他命名空間，則請選擇新增命名空間並輸入命名空間名稱。

c. 如果要新增其他政策，則請選擇新增政策。您可以對每個政策設定不同的範圍，但每個政策只能新增一次。

d. 選擇下一步。

12檢查存取項目的組態。如果有任何內容看起來不正確，請選擇上一步以返回上一步並修正錯誤。如果組態正確，請選擇建立。

AWS CLI

1. 安裝 AWS CLI，如 AWS 命令列界面使用者指南中的安裝中所述。
2. 要建立存取項目，您可以使用以下任何範例來建立存取項目：

- 為自我管理的 Amazon EC2 Linux 節點群組建立存取項目。將 *my-cluster* 取代為您的叢集名稱、將 *111122223333* 取代為您的 AWS 帳戶 ID，並將 *EKS-my-cluster-self-managed-ng-1* 取代為您的節點 IAM 角色名稱。如果節點群組是 Windows 節點群組，請用 `EC2_WINDOWS` 取代 *EC2_LINUX*。

```
aws eks create-access-entry --cluster-name my-cluster --principal-arn arn:aws:
 iam::111122223333:role/EKS-my-cluster-self-managed-ng-1 --type EC2_LINUX
```

  當指定 `STANDARD` 以外的類型時，則不能使用 `--kubernetes-groups` 選項。您無法將存取政策與此存取項目關聯，因為其類型是 `STANDARD` 以外的值。

- 建立這樣的一個存取項目：允許一個未用於 Amazon EC2 自我管理節點群組且您希望 Kubernetes 透過其授予對叢集的存取權的 IAM 角色。將 *my-cluster* 取代為您的叢集名稱，將 *111122223333* 取代為您的 AWS 帳戶 ID，將 *my-role* 取代為您的 IAM 角色名稱。用您在叢集上的 Kubernetes `RoleBinding` 或 `ClusterRoleBinding` 物件中指定的群組名稱取代 *Viewers*。

```
aws eks create-access-entry --cluster-name my-cluster --principal-arn arn:aws:
 iam::111122223333:role/my-role --type STANDARD --username Viewers --kubernetes-
groups Viewers
```

- 建立一個允許 IAM 使用者向叢集進行身分驗證的存取項目。在此提供此範例只是為了說明這種可能性。IAM 最佳實務建議使用具有短期憑證的 IAM 角色存取叢集，而不是使用具有長期憑證的 IAM 使用者進行存取。如需詳細資訊，請參閱《IAM 使用者指南》中的要求人類使用者使用聯合身分提供者來 AWS 使用臨時憑證存取 。

  ```
  aws eks create-access-entry --cluster-name my-cluster --principal-arn arn:aws:
   iam::111122223333:user/my-user --type STANDARD --username my-user
  ```

  如果您希望此使用者對叢集的存取許可高於 Kubernetes API 探索角色中的許可，則您需要將存取政策關聯到該存取項目，因為不使用 `--kubernetes-groups` 選項。如需詳細資訊，請參閱 Kubernetes 文件中的 the section called “關聯存取政策” 以及 API discovery roles 兩節。

更新存取項目

您可以使用 AWS 管理主控台 或 AWS CLI 來更新存取項目。

AWS 管理主控台

1. 開啟 Amazon EKS 主控台。
2. 選擇要在其中建立存取項目的叢集名稱。
3. 選擇存取索引標籤。
4. 選擇要更新的存取項目。
5. 選擇編輯。
6. 對於使用者名稱，您可以變更現有值。
7. 對於群組，您可以移除現有群組名稱或新增群組名稱。如果存在以下群組名稱，請勿將其移除：system:nodes 或 system:bootstrappers。移除這些群組可能會導致叢集無法正常運作。如果您不指定任何群組名稱並希望使用 Amazon EKS 授權，請在後續步驟中關聯存取政策。
8. 您可以使用標籤為存取項目指派標籤。例如，為了更輕鬆地找到具有相同標籤的所有資源而指定標籤。您也可以移除現有的標籤。
9. 選擇儲存變更。
10. 如果您想要將存取政策關聯到項目，請參閱 the section called “關聯存取政策”。

AWS CLI

1. 依據《AWS 命令列介面使用者指南》中的安裝內容所述，安裝 AWS CLI。
2. 如需更新存取項目，使用您的叢集名稱取代 *my-cluster*，使用您的 AWS 帳戶 ID 取代 *111122223333*，並使用您的 IAM 角色名稱取代 *EKS-my-cluster-my-namespace-Viewers*。

```
aws eks update-access-entry --cluster-name my-cluster --principal-arn
 arn:aws:iam::111122223333:role/EKS-my-cluster-my-namespace-Viewers --kubernetes-
groups Viewers
```

如果存取項目類型是 STANDARD 以外的值，則不能使用 `--kubernetes-groups` 選項。您也無法將存取政策與 STANDARD 以外的類型的存取項目關聯。

刪除存取項目

如果您發現某個存取項目被錯誤刪除，則您可以隨時重新建立。如果您要刪除的存取項目與任何存取政策關聯，則這些關聯將自動刪除。在刪除存取項目之前，您不必取消其與存取政策的關聯。

您可以使用 AWS 管理主控台 或 AWS CLI 刪除存取項目。

AWS 管理主控台

1. 開啟 Amazon EKS 主控台。
2. 選擇要從中刪除存取項目之叢集的名稱。
3. 選擇存取索引標籤。
4. 在存取項目清單中，選擇要刪除的存取項目。
5. 選擇 Delete (刪除)。
6. 在確認對話方塊中，選擇 Delete (刪除)。

AWS CLI

1. 依據《AWS 命令列介面使用者指南》中的安裝內容所述，安裝 AWS CLI。
2. 要刪除存取項目，用您的叢集名稱取代 *my-cluster*，用您的 AWS 帳戶 ID 取代 *111122223333*，並用您不想再讓其存取叢集的 IAM 角色的名稱取代 *my-role*。

```
aws eks delete-access-entry --cluster-name my-cluster --principal-arn
 arn:aws:iam::111122223333:role/my-role
```

針對 EKS 存取項目設定自訂使用者名稱

針對 Amazon EKS 建立存取項目時，您可使用自動生成的使用者名稱，或者指定自訂使用者名稱。此頁面詳細說明了這兩種選項，並引導您設定自訂使用者名稱。

概觀

存取項目中的使用者名稱用於辨識 Kubernetes 日誌與稽核線索中的 IAM 主體。Amazon EKS 預設會依據 IAM 身分的 ARN，來生成使用者名稱，但您可根據需要指定自訂使用者名稱。

預設使用者名稱生成

若您未指定使用者名稱的值，Amazon EKS 會依據 IAM 身分自動生成使用者名稱：

- 對於 IAM 使用者：
  - EKS 將 Kubernetes 使用者名稱設定為 IAM 使用者的 ARN
  - 範例：

    ```
    {arn-aws}iam::<111122223333>:user/<my-user>
    ```
- 對於 IAM 角色：
  - EKS 依據 IAM 角色 ARN 來設定 Kubernetes 使用者名稱
  - 角色被擔任時的 STS ARN。Amazon EKS 將 `{{SessionName}}` 附加到角色。如果您指定的角色的 ARN 包含路徑，則 Amazon EKS 會將其從產生的使用者名稱中移除。
  - 範例：

    ```
    {arn-aws}sts::<111122223333>:assumed-role/<my-role>/{{SessionName}}
    ```

除非您有特定原因需要自己指定使用者名稱，否則建議您不要指定使用者名稱，而是讓 Amazon EKS 自動為您產生該使用者名稱。

設定自訂使用者名稱

建立存取項目後，您可藉助 `--username` 參數來指定自訂使用者名稱：

```
aws eks create-access-entry --cluster-name <cluster-name> --principal-arn <iam-
identity-arn> --type STANDARD --username <custom-username>
```

自訂使用者名稱要求

若指定自訂使用者名稱：

- 使用者名稱的開頭不能為 `system:`、`eks:`、`aws:`、`amazon:` 或 `iam:`。
- 如果該使用者名稱用於 IAM 角色，則建議您在末尾新增 `{{SessionName}}` 或 `{{SessionNameRaw}}`。
  - 如果將 `{{SessionName}}` 或 `{{SessionNameRaw}}` 新增至使用者名稱，則必須在 {{SessionName}} 之前必須有冒號。

使用存取政策和 CLI 為 IAM AWS 角色或使用者建立存取項目

建立使用 AWS受管 EKS 存取政策的 Amazon EKS 存取項目，以授予 IAM 身分標準化許可來存取和管理 Kubernetes 叢集。

概觀

Amazon EKS 中的存取項目會定義了 IAM 身分 (使用者和角色) 如何存取您的 Kubernetes 叢集並與其互動。透過建立具有 EKS 存取政策的存取項目，您可以：

- 授予特定 IAM 使用者或角色存取 EKS 叢集的許可
- 使用提供標準化、預先定義許可集的 AWS受管 EKS 存取政策來控制許可
- 將許可範圍限定於特定命名空間或整個叢集
- 簡化存取管理，無需修改 `aws-auth` ConfigMap 或建立 Kubernetes RBAC 資源
- 使用 AWS整合的 Kubernetes 存取控制方法，涵蓋常見的使用案例，同時維護安全最佳實務

此方法建議用於大多數使用案例，因為它提供 AWS受管、標準化的許可，而無需手動 Kubernetes RBAC 組態。EKS 存取政策無需手動設定 Kubernetes RBAC 資源，並提供涵蓋常見使用案例的預定義許可集。

先決條件

- 您的叢集身分驗證模式必須設定為啟用存取項目。如需詳細資訊，請參閱the section called “身分驗證方式”。

- 安裝和設定 AWS CLI，如 AWS 命令列界面使用者指南中的安裝中所述。

步驟 1：定義存取項目

1. 尋找您要授予許可的 IAM 身分的 ARN，如使用者或角色。
   - 每個 IAM 身分只能有一個 EKS 存取項目。
2. 確定您希望 Amazon EKS 存取政策許可僅適用於特定的 Kubernetes 命名空間，還是跨整個叢集。
   - 如果您想將許可限制在特定命名空間，請記下命名空間名稱。
3. 為 IAM 身分選擇所需的 EKS 存取政策。此政策授予叢集內許可。記下政策的 ARN。
   - 有關政策列表，請參閱可用的存取政策。
4. 判斷自動產生的使用者名稱是否適合該存取項目，或者您是否需要手動指定使用者名稱。
   - AWS 根據 IAM 身分自動產生此值。您可設定自訂使用者名稱。這在 Kubernetes 日誌中可見。
   - 如需詳細資訊，請參閱the section called “設定自訂使用者名稱”。

步驟 2：建立存取項目

規劃存取項目之後，請使用 AWS CLI 來建立該項目。

以下範例涵蓋大多數使用案例。檢視全部組態選項的 CLI 參考。

您將在下一步驟中附加存取政策。

```
aws eks create-access-entry --cluster-name <cluster-name> --principal-arn <iam-
identity-arn> --type STANDARD
```

步驟 3：關聯存取政策

根據您是否希望政策限制於指定的 Kubernetes 命名空間，指令會有所不同。

您需要存取政策的 ARN。檢視可用的存取政策。

建立無命名空間範圍的政策

```
aws eks associate-access-policy --cluster-name <cluster-name> --principal-arn <iam-
identity-arn> --access-scope type=cluster --policy-arn <access-policy-arn>
```

使用命名空間範圍建立

```
aws eks associate-access-policy --cluster-name <cluster-name> --principal-arn <iam-
identity-arn> \
    --access-scope type=namespace,namespaces=my-namespace1,my-namespace2 --policy-arn
 <access-policy-arn>
```

後續步驟

- 建立 kubeconfig，以便您可以將 kubectl 與 IAM 身分搭配使用

透過 AWS CLI 使用 Kubernetes 群組建立存取項目

建立使用 Kubernetes 群組進行授權且需要手動 RBAC 組態的 Amazon EKS 存取項目。

> **Note**
>
> 對於大多數使用案例，我們建議使用 EKS 存取政策，而非本頁所述的 Kubernetes 群組方法。EKS 存取政策提供更簡單、更加整合 AWS 的方法來管理存取，無需手動 RBAC 組態。僅當您需要比 EKS 存取政策提供更精細的控制時，才使用 Kubernetes 群組方法。

概觀

存取項目定義了 IAM 身分 (使用者和角色) 如何存取您的 Kubernetes 叢集。Kubernetes 群組方法透過標準 Kubernetes RBAC 群組，授予 IAM 使用者或角色存取 EKS 叢集的許可。此方法需建立並管理 Kubernetes RBAC 資源 (Role、RoleBinding、ClusterRole 及 ClusterRoleBinding)，建議在下列情境使用：需要高度自訂的許可集合、有複雜的授權需求，或希望在混合式 Kubernetes 環境中維持一致的存取控制模式。

本主題不涵蓋為「用於讓 Amazon EC2 執行個體加入 EKS 叢集」的 IAM 身分建立存取項目。

先決條件

- 您的叢集身分驗證模式必須設定為啟用存取項目。如需詳細資訊，請參閱 the section called “身分驗證方式”。
- 依據《AWS 命令列介面使用者指南》中的安裝內容所述，安裝與設定 AWS CLI。
- 建議熟悉 Kubernetes RBAC。如需詳細資訊，請參閱 Kubernetes 文件中的使用 RBAC 授權。

步驟 1：定義存取項目

1. 尋找您要授予許可的 IAM 身分的 ARN，如使用者或角色。
   - 每個 IAM 身分只能有一個 EKS 存取項目。
2. 決定要與此 IAM 身分建立關聯的 Kubernetes 群組。
   - 您將需要建立或使用現有的 Kubernetes `Role/ClusterRole` 和 `RoleBinding/ClusterRoleBinding` 資源，這些資源需引用這些群組。
3. 判斷自動產生的使用者名稱是否適合該存取項目，或者您是否需要手動指定使用者名稱。
   - AWS 會根據 IAM 身分自動產生此值。您可設定自訂使用者名稱。這在 Kubernetes 日誌中可見。
   - 如需詳細資訊，請參閱 the section called “設定自訂使用者名稱”。

步驟 2：使用 Kubernetes 群組建立存取項目

規劃存取項目之後，使用 AWS CLI 以適當的 Kubernetes 群組建立該項目。

```
aws eks create-access-entry --cluster-name <cluster-name> --principal-arn <iam-
identity-arn> --type STANDARD --kubernetes-groups <groups>
```

取代：

- `<cluster-name>` 為您的 EKS 叢集名稱
- `<iam-identity-arn>` 為 IAM 使用者或角色的 ARN
- `<groups>` 為以逗號分隔的 Kubernetes 群組清單 (例如 "system:developers,system:readers")

檢視全部組態選項的 CLI 參考。

步驟 3：設定 Kubernetes RBAC

為了讓 IAM 主體能夠存取您叢集上的 Kubernetes 物件，您必須建立和管理 Kubernetes 角色型存取控制 (RBAC) 物件：

1. 建立定義許可的 Kubernetes `Role` 或 `ClusterRole` 物件。
2. 在您的叢集上建立 Kubernetes `RoleBinding` 或 `ClusterRoleBinding` 物件，並將群組名稱指定為 `kind: Group` 的 `subject`。

有關在 Kubernetes 中設定群組和許可的詳細資訊，請參閱 Kubernetes 文件中的使用 RBAC 授權。

後續步驟

- 建立 kubeconfig，以便您可以將 kubectl 與 IAM 身分搭配使用

透過 ConfigMap 授予 IAM 使用者存取 Kubernetes 的權限

> **Important**
>
> `aws-auth ConfigMap` 已棄用。如需管理 Kubernetes API 存取權的建議方法，請參閱 the section called “存取項目”。

適用於 Kubernetes 的AWS IAM 驗證器在 Amazon EKS 控制平面上執行，可以使用 IAM 主體啟用對叢集的存取權。驗證器會從 `aws-auth ConfigMap` 獲得其組態資訊。如需所有 `aws-auth ConfigMap` 設定的詳細資訊，請參閱 GitHub 上的完整組態格式。

將 IAM 主體新增至 Amazon EKS 叢集

當您建立 Amazon EKS 叢集時，建立該叢集的 IAM 主體會由 Amazon EKS 控制平面中叢集的角色型存取控制 (RBAC) 組態來自動授予 `system:masters` 許可。此主體不會出現在任何可見的組態中，因此請務必記下最初建立叢集的主體。若要賦予其他 IAM 主體與叢集互動的能力，您必須編輯 Kubernetes 中的 `aws-auth ConfigMap` 並用 `group` 的名稱建立一個 Kubernetes `rolebinding` 或 `clusterrolebinding`，其名稱是您在 `aws-auth ConfigMap` 中指定的名稱。

> **Note**
>
> 如需有關 Kubernetes 角色型存取控制 (RBAC) 組態的詳細資訊，請參閱 Kubernetes 文件中的使用 RBAC 授權。

1. 判斷 `kubectl` 正使用哪些憑證來存取叢集。您可以在電腦上透過一下命令查看 `kubectl` 正在使用的憑證。如果不使用預設路徑，則使用 `kubeconfig` 檔案的路徑取代 *`~/.kube/config`*。

   ```
   cat ~/.kube/config
   ```

   範例輸出如下。

   ```
   [...]
   ```

```
contexts:
- context:
    cluster: my-cluster.region-code.eksctl.io
    user: admin@my-cluster.region-code.eksctl.io
  name: admin@my-cluster.region-code.eksctl.io
current-context: admin@my-cluster.region-code.eksctl.io
[...]
```

在前述範例輸出中，已針對名為 *my-cluster* 的叢集設定名為 *admin* 的使用者的憑證。若該叢集是由這名使用者建立，則其已擁有您叢集的存取權。若叢集不是由此使用者建立，則您需要完成剩餘的步驟，才能為其他 IAM 主體啟用叢集存取權。IAM 最佳實務建議您將許可授予角色而非使用者。您可以使用以下命令查看目前哪些其他主體可以存取您的叢集：

```
kubectl describe -n kube-system configmap/aws-auth
```

範例輸出如下。

```
Name:         aws-auth
Namespace:    kube-system
Labels:       <none>
Annotations:  <none>

Data
====
mapRoles:
----
- groups:
  - system:bootstrappers
  - system:nodes
  rolearn: arn:aws: iam::111122223333:role/my-node-role
  username: system:node:{{EC2PrivateDNSName}}


BinaryData
====

Events:  <none>
```

上一個範例是預設的 aws-auth ConfigMap。只有節點執行個體角色有權存取叢集。

2. 請確定您現有的 Kubernetes roles 和 rolebindings 或 clusterroles 和 clusterrolebindings (您可以將 IAM 主體映射至此)。如需有關這些資源的詳細資訊，請參閱 Kubernetes 文件中的使用 RBAC 授權。

   a. 檢視您現有的 Kubernetes roles 或 clusterroles。Roles 的作用域為 namespace，但 clusterroles 的作用域限定為叢集。

   ```
   kubectl get roles -A
   ```

   ```
   kubectl get clusterroles
   ```

   b. 檢視先前輸出中傳回的任何 role 或 clusterrole 的詳細資訊，並確認其具有您希望 IAM 主體在叢集中擁有的許可 (rules)。

   將 *role-name* 取代為上一個命令的輸出中傳回的 role 名稱。將 *kube-system* 取代為 role 的命名空間。

   ```
   kubectl describe role role-name -n kube-system
   ```

   將 *cluster-role-name* 取代為上一個命令的輸出中傳回的 clusterrole 名稱。

   ```
   kubectl describe clusterrole cluster-role-name
   ```

   c. 檢視您現有的 Kubernetes rolebindings 或 clusterrolebindings。Rolebindings 的作用域為 namespace，但 clusterrolebindings 的作用域限定為叢集。

   ```
   kubectl get rolebindings -A
   ```

   ```
   kubectl get clusterrolebindings
   ```

   d. 檢視任何 rolebinding 或 clusterrolebinding 的詳細資訊，並確認其具有上一步中列為 roleRef 的 role 或 clusterrole，以及為 subjects 列出的群組名稱。

   將 *role-binding-name* 取代為上一個命令的輸出中傳回的 rolebinding 名稱。將 *kube-system* 取代為 rolebinding 的 namespace。

   ```
   kubectl describe rolebinding role-binding-name -n kube-system
   ```

   範例輸出如下。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: eks-console-dashboard-restricted-access-role-binding
  namespace: default
subjects:
- kind: Group
  name: eks-console-dashboard-restricted-access-group
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: eks-console-dashboard-restricted-access-role
  apiGroup: rbac.authorization.k8s.io
```

將 *cluster-role-binding-name* 取代為上一個命令的輸出中傳回的 clusterrolebinding 名稱。

```
kubectl describe clusterrolebinding cluster-role-binding-name
```

範例輸出如下。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: eks-console-dashboard-full-access-binding
subjects:
- kind: Group
  name: eks-console-dashboard-full-access-group
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: eks-console-dashboard-full-access-clusterrole
  apiGroup: rbac.authorization.k8s.io
```

3. 編輯 aws-auth ConfigMap。您可以使用例如 eksctl 的工具來更新 ConfigMap，或者您可以透過編輯來手動更新它。

> **Important**
>
> 我們建議您使用 eksctl 或其他工具來編輯 ConfigMap。如需有關您可以使用的其他工具的資訊，請參閱《Amazon EKS 最佳實務指南》中的使用工具對 aws-auth ConfigMap 進行變更。格式錯誤的 aws-auth ConfigMap可能會導致您失去叢集存取權。

- 檢視使用 eksctl 編輯 configmap 的步驟。
- 檢視手動編輯 configmap ma 的步驟。

使用 eksctl 編輯 ConfigMap

1. 您需要在裝置0.215.0或 AWS CloudShell 上安裝版本 或更新版本的eksctl命令列工具。如需有關安裝或更新 eksctl 的指示，請參閱 eksctl 文件中的安裝一節。
2. 在 ConfigMap 檢視目前的映射項目。使用您叢集的名稱取代 *my-cluster*。將 *region-code* 取代為您的叢集所在的 AWS 區域。

   ```
   eksctl get iamidentitymapping --cluster my-cluster --region=region-code
   ```

   範例輸出如下。

   ```
   ARN
               USERNAME                                GROUPS
    ACCOUNT
   arn:aws: iam::111122223333:role/eksctl-my-cluster-my-nodegroup-
   NodeInstanceRole-1XLS7754U3ZPA    system:node:{{EC2PrivateDNSName}}
    system:bootstrappers,system:nodes
   ```

3. 為角色新增映射項目。將 *my-group* 取代為您的角色名稱。將 *eks-console-dashboard-full-access-group* 取代為您的 Kubernetes RoleBinding 或 ClusterRoleBinding 物件中指定的群組名稱。使用您的帳戶 ID 取代 *111122223333*。您可以將 *admin* 取代為您選擇的任何名稱。

   ```
   eksctl create iamidentitymapping --cluster my-cluster --region=region-code \
       --arn arn:aws: iam::111122223333:role/my-role --username admin --group eks-
   console-dashboard-full-access-group \
       --no-duplicate-arns
   ```

> **Important**
>
> 角色 ARN 不能包含例如 role/my-team/developers/my-role 的路徑。ARN 的格式必須是 arn:aws: iam::*111122223333*:role/*my-role* 。在此範例中，需移除 my-team/developers/。

範例輸出如下。

```
[...]
2022-05-09 14:51:20 [#]  adding identity "{arn-aws}iam::111122223333:role/my-role" to
 auth ConfigMap
```

4. 為使用者新增映射項目。IAM 最佳實務建議您將許可授予角色而非使用者。將 *my-user* 取代為您的使用者名稱。將 *eks-console-dashboard-restricted-access-group* 取代為您的 Kubernetes RoleBinding 或 ClusterRoleBinding 物件中指定的群組名稱。使用您的帳戶 ID 取代 *111122223333*。您可以將 *my-user* 取代為您選擇的任何名稱。

```
eksctl create iamidentitymapping --cluster my-cluster --region=region-code \
    --arn arn:aws: iam::111122223333:user/my-user --username my-user --group eks-
console-dashboard-restricted-access-group \
    --no-duplicate-arns
```

範例輸出如下。

```
[...]
2022-05-09 14:53:48 [#]  adding identity "arn:aws: iam::111122223333:user/my-user" to
 auth ConfigMap
```

5. 再次檢視 ConfigMap 中的映射項目。

```
eksctl get iamidentitymapping --cluster my-cluster --region=region-code
```

範例輸出如下。

```
ARN
            USERNAME                                GROUPS
      ACCOUNT
```

```
arn:aws: iam::111122223333:role/eksctl-my-cluster-my-nodegroup-
NodeInstanceRole-1XLS7754U3ZPA    system:node:{{EC2PrivateDNSName}}
 system:bootstrappers,system:nodes
arn:aws: iam::111122223333:role/admin
               my-role                                    eks-console-dashboard-full-
access-group
arn:aws: iam::111122223333:user/my-user
            my-user                                  eks-console-dashboard-restricted-
access-group
```

手動編輯 ConfigMap

1. 開啟 ConfigMap 進行編輯。

```
kubectl edit -n kube-system configmap/aws-auth
```

> **Note**
>
> 若您收到錯誤訊息，指出「Error from server (NotFound): configmaps "aws-auth" not found」，請使用將 aws-auth ConfigMap 套用至您的叢集中的程序套用庫存 ConfigMap。

2. 將您的 IAM 主體新增至 ConfigMap。IAM 群組不是 IAM 主體，因此無法新增至 ConfigMap。
   - 新增 IAM 角色 (例如：為聯合身分使用者)：在 data 下位於 ConfigMap 中的 mapRoles 區段新增角色詳細資訊。若此區段在檔案不存在，則將其新增。每個項目支援以下參數：
     - rolearn：要新增的 IAM 角色之 ARN。此值不能包含路徑。例如，您無法指定 ARN (如 arn:aws: iam::*111122223333*:role/my-team/developers/*role-name*)。ARN 需要改為 arn:aws: iam::*111122223333*:role/*role-name* 。
     - username：Kubernetes 內映射至 IAM 角色的使用者名稱。
     - 群組：要將角色映射到的 Kubernetes 群組的群組或清單。群組可以是預設群組，也可以是 clusterrolebinding 或 rolebinding 中指定的群組。如需詳細資訊，請參閱 Kubernetes 文件中的預設角色和角色連結。
   - 若要新增 IAM 使用者：IAM 最佳實務建議您將許可授予角色而非使用者。將使用者詳細資訊新增至 ConfigMap 的 mapUsers 區段 (在 data 下方)。若此區段在檔案不存在，則將其新增。每個項目支援以下參數：
     - userarn：要新增之 IAM 使用者的 ARN。

- username：Kubernetes 內應設至 IAM 使用者的使用者名稱。
- 群組：要將使用者映射到的 Kubernetes 群組的群組或清單。群組可以是預設群組，也可以是 clusterrolebinding 或 rolebinding 中指定的群組。如需詳細資訊，請參閱 Kubernetes 文件中的預設角色和角色連結。

=== username和 的範本變數 groups

username 和 mapUsers項目中的 mapRoles和 groups 欄位都支援 AWS IAM Authenticator 在身分驗證時取代的範本變數。下列範本變數可供使用：

{{AccountID}}

已驗證 IAM 主體的 12 位數 AWS 帳戶 ID。

{{SessionName}}

STS 角色工作階段名稱，以 取代@字元-。對於 EC2 執行個體角色，這是執行個體 ID。對於聯合角色，這是聯合身分。對於使用 擔任的角色sts:AssumeRole，這是發起人設定的RoleSessionName參數值。

{{SessionNameRaw}}

不取代任何角色的原始 STS 角色工作階段名稱。

{{AccessKeyID}}

用來驗證請求的 AWS 存取金鑰 ID。

{{EC2PrivateDNSName}}

EC2 執行個體的私有 DNS 名稱。此變數需要工作階段名稱為有效的 EC2 執行個體 ID，並發出 ec2:DescribeInstances API 呼叫。

Important

使用 擔任角色時，{{SessionName}}和 {{SessionNameRaw}}值由使用者控制sts:AssumeRole。發起人將 RoleSessionName 參數設定為幾乎任意的字串。如果您在 username欄位中使用這些變數，可以擔任映射角色的發起人可以模擬任何不符合預留字首（例如 system:、eks:、aws:、 amazon:或）的 Kubernetes 使用者名稱iam:。如果您在 groups欄位中使用這些變數，發起人可以透過製作工作階段名稱來選擇自己的

Kubernetes 群組成員資格。只有在您信任可擔任該角色的所有委託人`groups`時，才能使用 `{{SessionName}}` `{{SessionNameRaw}}` `username`或 。

例如，下列的 YAML 區塊包含：

- `mapRoles` 區段，將 IAM 節點執行個體映射至 Kubernetes 群組，讓節點可以將自己註冊到叢集，以及 `my-console-viewer-role` IAM 角色，映射至 Kubernetes 群組，可以檢視所有叢集的所有 Kubernetes 資源。如需 `my-console-viewer-role` IAM 角色所需的 IAM 和 Kubernetes 群組許可的清單，請參閱 the section called “所需的許可”。
- 將 `admin` IAM 使用者從預設 AWS 帳戶映射至 `system:masters` Kubernetes 群組，以及將`my-user`使用者從映射至可檢視特定命名空間 Kubernetes 資源之 Kubernetes 群組的不同 AWS 帳戶映射的`mapUsers`區段。如需 `my-user` IAM 使用者所需的 IAM 和 Kubernetes 群組許可的清單，請參閱 the section called “所需的許可”。

根據需要新增或移除行，並將所有範例值取代為您自己的值。

```
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file will
 be
# reopened with the relevant failures.
#
apiVersion: v1
data:
  mapRoles: |
    - groups:
      - system:bootstrappers
      - system:nodes
      rolearn: arn:aws: iam::111122223333:role/my-role
      username: system:node:{{EC2PrivateDNSName}}
    - groups:
      - eks-console-dashboard-full-access-group
      rolearn: arn:aws: iam::111122223333:role/my-console-viewer-role
      username: my-console-viewer-role
  mapUsers: |
    - groups:
      - system:masters
      userarn: arn:aws: iam::111122223333:user/admin
      username: admin
    - groups:
```

```
    - eks-console-dashboard-restricted-access-group
    userarn: arn:aws: iam::444455556666:user/my-user
    username: my-user
```

1. 儲存檔案並結束您的文字編輯器。

將 aws-auth ConfigMap 套用至您的叢集

在建立受管節點群組或以 eksctl 建立節點群組時，便會自動建立並套用 aws-auth ConfigMap 至您的叢集。最初建立的目的是要允許將節點加入您的叢集，但您也可以使用此 ConfigMap 來新增角色型存取控制 (RBAC) 以存取 IAM 主體。若尚未啟動自我管理節點且尚未套用 aws-auth ConfigMap 至叢集，則您可以使用以下程序來完成此操作。

1. 檢查您是否已套用 aws-auth ConfigMap。

   ```
   kubectl describe configmap -n kube-system aws-auth
   ```

   若您收到錯誤訊息，指出 "Error from server (NotFound): configmaps "aws-auth" not found"，請使用以下步驟的程序來套用儲存的 ConfigMap。

2. 下載、編輯和套用 AWS 驗證器組態映射。

   a. 下載組態對應。

   ```
   curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/
   aws-auth-cm.yaml
   ```

   b. 在 aws-auth-cm.yaml 檔案中，將 rolearn 設定至與您節點關聯的 IAM 角色之 Amazon Resource Name (ARN)。您可以使用文字編輯器來完成此操作，或者透過取代 *my-node-instance-role* 並執行下列命令：

   ```
   sed -i.bak -e 's|<ARN of instance role (not instance profile)>|my-node-instance-
   role|' aws-auth-cm.yaml
   ```

   請勿修改此檔案中的任何其他行。

> **Important**
>
> 角色 ARN 不能包含例如 `role/my-team/developers/my-role` 的路徑。ARN 的格式必須是 `arn:aws: iam::111122223333:role/my-role` 。在此範例中，需移除 `my-team/developers/`。

您可以檢查節點群組的 AWS CloudFormation 堆疊輸出，並尋找下列值：

- InstanceRoleARN：適用於以 `eksctl` 建立的節點群組
- NodeInstanceRole – 適用於使用 中的 Amazon EKS vended AWS CloudFormation 範本建立的節點群組 AWS 管理主控台

c. 套用組態。此命令可能需要幾分鐘的時間來完成。

```
kubectl apply -f aws-auth-cm.yaml
```

> **Note**
>
> 如果您收到任何授權或資源類型錯誤，請參閱故障診斷主題中的the section called “未經授權或存取遭拒 (kubectl)”。

3. 查看節點的狀態，並等待他們到達 `Ready` 狀態。

```
kubectl get nodes --watch
```

輸入 `Ctrl+C` 傳回 Shell 提示。

透過外部 OIDC 提供商授予使用者存取 Kubernetes 的權限

Amazon EKS 支援使用 OpenID Connect (OIDC) 身分提供者作為向您的叢集對使用者進行身分驗證的方法。OIDC 身分提供者可以與 AWS Identity and Access Management (IAM) 搭配使用或作為替代方案。如需使用 IAM 的詳細資訊，請參閱 the section called “Kubernetes API 存取”。設定叢集身分驗證後，您可以建立 Kubernetes `roles` 和 `clusterroles` 指派許可給角色，然後使用 Kubernetes `rolebindings` 和 `clusterrolebindings` 將角色繫結至身分。如需詳細資訊，請參閱 Kubernetes 文件中的使用 RBAC 授權。

- Kubernetes 不提供 OIDC 身分提供者。您可以使用現有的公有 OIDC 身分提供者，也可以執行您自己的身分提供者。如需認證提供商的清單，請參閱 OpenID 網站上的 OpenID 認證。
- OIDC 身分提供者的發行者 URL 必須可公開存取，以便 Amazon EKS 可以探索簽署金鑰。Amazon EKS 不支援具有自我簽署憑證的 OIDC 身分提供者。
- 您無法停用叢集上的 IAM 身分驗證器，因為將節點連接到叢集仍然需要此身分驗證器。
- Amazon EKS 叢集仍然必須由 AWS IAM 主體建立，而非 OIDC 身分提供者使用者。這是因為叢集建立者會與 Amazon EKS API 進行互動，而不是與 Kubernetes API 進行互動。
- 如果控制平面的 CloudWatch Logs 已開啟，則已進行提供商身分驗證的 OIDC 身分使用者會列在叢集的稽核日誌中。如需詳細資訊，請參閱the section called “啟用或停用控制平面日誌”。
- 您無法 AWS 管理主控台 透過 OIDC 供應商的帳戶登入 。您只能the section called “存取叢集資源”使用 AWS Identity and Access Management AWS 管理主控台 帳戶登入 。

您可以與叢集建立關聯的 OIDC 身分提供者數量取決於叢集的 Kubernetes 版本。在執行 Kubernetes 1.32 版或更新版本的叢集上，您最多可以關聯 10 個 OIDC 身分提供者。所有 OIDC 提供者組態的合併大小必須小於 12 KB。在執行 1.32 以前版本的叢集上，您只能關聯一個 OIDC 身分提供者。

- 發行者 URL (`issuerUrl`) 和名稱 (`name`) 在與叢集相關聯的所有 OIDC 身分提供者中必須是唯一的。您無法多次關聯相同的供應商。
- 關聯 OIDC 身分提供者是叢集更新。叢集會進入 `UPDATING` 狀態，而變更可能需要幾分鐘的時間才能完全套用至叢集的 API 伺服器。您可以使用 DescribeUpdate 操作追蹤更新進度。

> **Note**
>
> 控制平面會從發行者 URL 擷取提供者的簽署金鑰，因此發行者必須可從控制平面連線。如果您的叢集使用客戶路由控制平面輸出 (`controlPlaneEgressMode=CUSTOMER_ROUTED`)，請確定發行者端點可透過您在 VPC 中設定的輸出路徑來連線。如需詳細資訊，請參閱the section called “控制平面輸出”。

關聯 OIDC 身分提供者

在可以將 OIDC 身分提供者與叢集建立關聯之前，您需要提供商的下列資訊：

發行者 URL

OIDC 身分提供者的 URL，可讓 API 伺服器探索用於驗證權杖的公有簽署金鑰。URL 必須以 `https://` 開頭並且應該對應至提供商之 OIDC ID 字符中的 `iss` 宣告。根據 OIDC 標準，允許路徑元件，但不允許查詢參數。通常，URL 只包含一個主機名稱，如 `https://server.example.org` 或 `https://example.com`。此 URL 應指向 `.well-known/openid-configuration` 以下的層級，必須可透過網際網路公開存取。發行者 URL 在與叢集相關聯的所有 OIDC 身分提供者中必須是唯一的。

用戶端 ID (也稱為對象)

向 OIDC 身分提供者提出驗證請求的用戶端應用程式 ID。

您可以使用 `eksctl` 或 AWS 管理主控台關聯身分提供者。

使用 eksctl 關聯身分提供者

1. 使用下列內容建立名為 `associate-identity-provider.yaml` 的檔案。使用自己的取代範例值。`identityProviders` 區段中的值是從您的 OIDC 身分提供者取得的。值僅對 `identityProviders` 下的 `name`、`type`、`issuerUrl` 和 `clientId` 設定為必要項目。

```
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: my-cluster
  region: your-region-code

identityProviders:
  - name: my-provider
    type: oidc
    issuerUrl: https://example.com
    clientId: kubernetes
    usernameClaim: email
    usernamePrefix: my-username-prefix
    groupsClaim: my-claim
    groupsPrefix: my-groups-prefix
    requiredClaims:
      string: string
    tags:
      env: dev
```

> **Important**
>
> 請勿指定 `system:`，或該字串的任何部分，對於 `groupsPrefix` 或 `usernamePrefix`。

2. 建立供應商。

```
eksctl associate identityprovider -f associate-identity-provider.yaml
```

3. 若要讓 `kubectl` 與叢集和 OIDC 身分提供者搭配使用，請參閱 Kubernetes 文件中的使用 kubectl。

使用 AWS 主控台關聯身分提供者

1. 開啟 Amazon EKS 主控台。
2. 選取叢集，然後選取存取索引標籤。
3. 在 OIDC 身分提供者區段中，選取關聯身分提供者。
4. 在 Associate OIDC Identity Provider (關聯 OIDC 身分提供者) 頁面上，輸入或選取下列選項，然後選取 Associate (關聯)。
   - 對於 Name (名稱)，輸入提供商的唯一名稱。
   - 對於 Issuer URL (發行者 URL)，輸入您的提供商 URL。此 URL 必須可透過網際網路存取。
   - 對於用戶端 ID，輸入 OIDC 身分提供者的用戶端 ID (也稱為受眾)。
   - 對於 Username claim (使用者名稱宣告)，輸入要用作使用者名稱的宣告。
   - 對於群組宣告，輸入要作為使用者群組使用的宣告。
   - (選用) 選取 Advanced options (進階選項)，輸入或選取下列資訊。
     - Username prefix (使用者名稱字首)：輸入要在使用者名稱宣告前面加上的字首。字首會加入使用者名稱宣告，以防止與現有名稱發生衝突。如果未提供值，且使用者名稱是非 `email` 的值，字首預設為 Issuer URL (發行者 URL) 的值。您可以使用 值`-`來停用所有字首。請勿指定 `system:` 或該字串的任何部分。
     - Groups prefix (群組字首)：輸入要在群組宣告前面加上的字首。字首會加上群組宣告，以防止與現有名稱 （ 例如) 發生衝突`system:groups`。例如，值 `oidc:` 會建立群組名稱，例如 `oidc:engineering` 和 `oidc:infra`。請勿指定 `system:` 或該字串的任何部分。
     - Required claims (必要宣告)：選取 Add claim (新增宣告)，然後在用戶端 ID 字符中輸入一或多個描述必要宣告的鍵值對。鍵值對在 ID 字符中描述了必要宣告。如果設定，則會驗證每個宣告是否存在於具有相符值的 ID 字符中。

a. 若要讓 kubectl 與叢集和 OIDC 身分提供者搭配使用，請參閱 Kubernetes 文件中的使用 kubectl。

多個身分提供者的最佳實務

使用使用者名稱和群組字首：為每個 OIDC 身分提供者設定使用者名稱字首 (usernamePrefix) 和群組字首 (groupsPrefix)，以減少來自不同提供者的身分重疊的機會。如果您不使用字首，請使用每個供應商唯一的使用者名稱和群組宣告。任一種方法都會確保兩個供應商無法產生相同的 Kubernetes 使用者名稱或群組名稱。

驗證權杖宣告：使用必要的宣告 (requiredClaims) 來指定 ID 權杖中必須存在的鍵值對，以便您的叢集只接受您授權的權杖。

限制每個權杖中的宣告：將您的身分提供者設定為僅發行具有叢集所需宣告的權杖。限制宣告可減少字符洩露時公開的資訊。

範例 IAM 政策

如果您想要防止 OIDC 身分提供者與叢集產生關聯，請建立下列 IAM 政策並將其與 Amazon EKS 管理員的 IAM 帳戶關聯。如需詳細資訊，請參閱《IAM 使用者指南》中的建立 IAM 政策和新增 IAM 身分許可以及《服務授權參考》中的動作。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "denyOIDC",
            "Effect": "Deny",
            "Action": [
                "eks:AssociateIdentityProviderConfig"
            ],
            "Resource": "arn:aws:eks:us-west-2:111122223333:cluster/*"

        },
        {
            "Sid": "eksAdmin",
            "Effect": "Allow",
            "Action": [
                "eks:*"
            ],
```

```
            "Resource": "*"
        }
    ]
}
```

下列範例政策允許 OIDC 身分提供者關聯，如果 `clientID` 為 `kubernetes` 以及 `issuerUrl` 為 `https://cognito-idp.us-west-2.amazonaws.com/*`。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowCognitoOnly",
            "Effect": "Deny",
            "Action": "eks:AssociateIdentityProviderConfig",
            "Resource": "arn:aws:eks:us-west-2:111122223333:cluster/my-instance",
            "Condition": {
                "StringNotLikeIfExists": {
                    "eks:issuerUrl": "https://cognito-idp.us-west-2.amazonaws.com/*"
                }
            }
        },
        {
            "Sid": "DenyOtherClients",
            "Effect": "Deny",
            "Action": "eks:AssociateIdentityProviderConfig",
            "Resource": "arn:aws:eks:us-west-2:111122223333:cluster/my-instance",
            "Condition": {
                "StringNotEquals": {
                    "eks:clientId": "kubernetes"
                }
            }
        },
        {
            "Sid": "AllowOthers",
            "Effect": "Allow",
            "Action": "eks:*",
            "Resource": "*"
        }
    ]
}
```

取消 OIDC 身分提供者與叢集的關聯

如果您取消 OIDC 身分提供者與叢集的關聯，則提供商中包含的使用者將無法再存取叢集。不過，您仍然可以透過 IAM 主體存取叢集。

取消 OIDC 身分提供者的關聯是叢集更新。叢集會進入 UPDATING 狀態，而變更可能需要幾分鐘的時間才能完全套用至叢集的 API 伺服器。您可以使用 DescribeUpdate 操作追蹤更新進度。如果您的叢集有多個相關聯的 OIDC 身分提供者，您必須一次取消它們的關聯。請等待每個叢集更新完成，再取消下一個提供者的關聯。

1. 開啟 Amazon EKS 主控台。
2. 在 OIDC 身分提供者區段中，選取 Disassociate (解除關聯)，輸入身分提供者名稱，然後選取 `Disassociate`。

在 中檢視 Kubernetes 資源 AWS 管理主控台

您可以檢視部署至使用 AWS 管理主控台的叢集的 Kubernetes 資源。您無法使用 CLI AWS 或 eksctl 檢視 Kubernetes 資源。若要使用命令列工具檢視 Kubernetes 資源，請使用 kubectl。

> **Note**
>
> 若要在 的運算索引標籤上檢視資源索引標籤和節點區段 AWS 管理主控台，您使用的 IAM 主體必須具有特定的 IAM 和 Kubernetes 許可。如需詳細資訊，請參閱the section called “所需的許可”。

1. 開啟 Amazon EKS 主控台。
2. 在 Clusters (叢集) 清單中，選取包含您要檢視的 Kubernetes 資源的叢集。
3. 選取 Resources (資源) 標籤。
4. 選取您要檢視資源的 Resource Type (資源類型) 群組，例如 Workloads (工作負載)。您可以看到該群組中的資源類型清單。
5. 選取資源類型，例如 Workloads (工作負載) 群組中的 Deployments (部署)。您可以看到資源類型的說明、一個 Kubernetes 文件連結以獲取有關資源類型的詳細資訊，以及部署在叢集上的該類型的資源列表。如果清單是空的，則不會將該類型的資源部署至您的叢集。
6. 請選取資源以檢視其詳細資訊。請試試看下列範例：

- 選取 Workloads (工作負載) 群組，選取 Deployments (部署) 資源類型，然後選取 coredns 資源。當您選取資源時，依預設，您位於 Structured view (結構式檢視)。針對某些資源類型，您會在 Structured view (結構式檢視) 看到 Pods 區段。本區段列出由工作負載管理的 Pod。您可以選取任何列出的 Pod 來檢視關於 Pod 的資訊。並非所有資源類型都顯示於 Structured View (結構式檢視)。如果選擇資源頁面右上角的 Raw view (Raw 檢視)，您可以看到來自資源 Kubernetes API 的完整 JSON 回應。
- 選取 Cluster (叢集) 群組，然後選取 Nodes (節點) 資源類型。您將看到叢集中所有節點的清單。節點可以是任何 Amazon EKS 節點類型。這個列表與您選取叢集的 Compute (運算) 標籤時，在 Nodes (節點)看到的列表為同一列表。從清單中選取節點資源。在 Structured view (結構式檢視) 中，您還會看到 Pods 區段。本區段顯示在節點上運行的所有 Pod。

所需的許可

若要在 的運算索引標籤上檢視資源索引標籤和節點區段 AWS 管理主控台，您使用的 IAM 主體必須具有特定的最低 IAM 和 Kubernetes 許可。您必須正確設定 IAM 和 Kubernetes RBAC 許可。完成以下步驟，將所需的許可指派給您的 IAM 主體。

1. 請確保將 `eks:AccessKubernetesApi` 和檢視 Kubernetes 資源所需的其他必要 IAM 許可指派給您正在使用的 IAM 主體。如需如何編輯 IAM 主體許可的詳細資訊，請參閱《IAM 使用者指南》中的控制對主體的存取。如需如何編輯角色許可的詳細資訊，請參閱《IAM 使用者指南》中的變更角色許可 (主控台)。

   以下範例政策包含主體欲檢視帳戶中所有叢集的 Kubernetes 資源所需必要許可。將 *111122223333* 取代為 AWS 您的帳戶 ID。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "eks:ListFargateProfiles",
                "eks:DescribeNodegroup",
                "eks:ListNodegroups",
                "eks:ListUpdates",
                "eks:AccessKubernetesApi",
                "eks:ListAddons",
                "eks:DescribeCluster",
                "eks:DescribeAddonVersions",
```

```
                "eks:ListClusters",
                "eks:ListIdentityProviderConfigs",
                "iam:ListRoles"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "ssm:GetParameter",
            "Resource": "arn:aws:ssm:*:111122223333:parameter/*"
        }
    ]
}
```

若要檢視已連線叢集中的節點，Amazon EKS connector IAM 角色應能模擬叢集中的主體。這可讓 Amazon EKS Connector 將主體映射至 Kubernetes 使用者。

2. 使用 EKS 存取項目設定 Kubernetes RBAC 許可。

   什麼是 EKS 存取項目？

   EKS 存取項目是授予 IAM 主體（使用者和角色）存取 Kubernetes 叢集的簡化方式。存取項目不會手動管理 Kubernetes RBAC 資源和 aws-auth ConfigMap，而是使用 提供的受管政策自動處理 IAM 和 Kubernetes 許可之間的映射 AWS。如需存取項目的詳細資訊，請參閱 the section called "存取項目"。如需可用存取政策及其許可的相關資訊，請參閱存取政策許可。

   您可透過兩種方式將 Kubernetes 許可附加至存取項目：

   - 使用存取政策：存取政策是由 維護的預先定義 Kubernetes 許可範本 AWS。這些提供常見使用案例的標準化許可集。
   - 參考 Kubernetes 群組：如果您將 IAM 身分與 Kubernetes 群組建立關聯，您可以建立授予群組許可的 Kubernetes 資源。如需詳細資訊，請參閱 Kubernetes 文件中的使用 RBAC 授權。

     a. 使用 CLI 為您的 IAM AWS 主體建立存取項目。使用您叢集的名稱取代 *my-cluster*。使用您的帳戶 ID 取代 *111122223333*。

     ```
     aws eks create-access-entry \
         --cluster-name my-cluster \
         --principal-arn arn:aws: iam::111122223333:role/my-console-viewer-role
     ```

     範例輸出如下。

```
{
    "accessEntry": {
        "clusterName": "my-cluster",
        "principalArn": "arn:aws: iam::111122223333:role/my-console-viewer-
role",
        "kubernetesGroups": [],
        "accessEntryArn": "arn:aws: eks:region-code:111122223333:access-
entry/my-cluster/role/111122223333/my-console-viewer-role/
abc12345-1234-1234-1234-123456789012",
        "createdAt": "2024-03-15T10:30:45.123000-07:00",
        "modifiedAt": "2024-03-15T10:30:45.123000-07:00",
        "tags": {},
        "username": "arn:aws: iam::111122223333:role/my-console-viewer-role",
        "type": "STANDARD"
    }
}
```

b. 將政策與存取項目建立關聯。若要檢視 Kubernetes 資源，請使用 AmazonEKSViewPolicy：

```
aws eks associate-access-policy \
    --cluster-name my-cluster \
    --principal-arn arn:aws: iam::111122223333:role/my-console-viewer-role \
    --policy-arn arn:aws: eks::aws:cluster-access-policy/AmazonEKSViewPolicy \
    --access-scope type=cluster
```

範例輸出如下。

```
{
    "clusterName": "my-cluster",
    "principalArn": "arn:aws: iam::111122223333:role/my-console-viewer-role",
    "associatedAt": "2024-03-15T10:31:15.456000-07:00"
}
```

對於命名空間特定的存取，您可以將政策範圍限定為特定命名空間：

```
aws eks associate-access-policy \
    --cluster-name my-cluster \
    --principal-arn arn:aws: iam::111122223333:role/my-console-viewer-role \
    --policy-arn arn:aws: eks::aws:cluster-access-policy/AmazonEKSViewPolicy \
    --access-scope type=namespace,namespaces=default,kube-system
```

c. 確認已成功建立存取項目：

```
aws eks describe-access-entry \
    --cluster-name my-cluster \
    --principal-arn arn:aws: iam::111122223333:role/my-console-viewer-role
```

d. 列出相關聯的政策以確認政策關聯：

```
aws eks list-associated-access-policies \
    --cluster-name my-cluster \
    --principal-arn arn:aws: iam::111122223333:role/my-console-viewer-role
```

範例輸出如下。

```
{
    "associatedAccessPolicies": [
        {
            "policyArn": "arn:aws: eks::aws:cluster-access-policy/
AmazonEKSViewPolicy",
            "accessScope": {
                "type": "cluster"
            },
            "associatedAt": "2024-03-15T10:31:15.456000-07:00",
            "modifiedAt": "2024-03-15T10:31:15.456000-07:00"
        }
    ]
}
```

CloudTrail 可見性

檢視 Kubernetes 資源時，您會在 CloudTrail 日誌中看到下列操作名稱：

- `AccessKubernetesApi` - 讀取或檢視資源時

此 CloudTrail 事件提供對 Kubernetes 資源讀取存取權的稽核線索。

> **Note**
>
> 此操作名稱會出現在 CloudTrail 日誌中，僅供稽核之用。它不是 IAM 動作，無法在 IAM 政策陳述式中使用。若要透過 IAM 政策控制 Kubernetes 資源的讀取存取權，請使用 `eks:AccessKubernetesApi`許可，如 the section called “所需的許可”一節所示。

授予 AWS 服務對 Kubernetes APIs寫入存取權

所需的許可

若要讓 AWS 服務在 Amazon EKS 叢集中的 Kubernetes 資源上執行寫入操作，您必須同時授予 `eks:AccessKubernetesApi`和 `eks:MutateViaKubernetesApi` IAM 許可。

例如，Amazon SageMaker HyperPod 使用這些許可從 SageMaker AI Studio 啟用模型部署。如需詳細資訊，請參閱《Amazon SageMaker AI 開發人員指南》中的設定選用的 JavaScript SDK 許可。

> **Important**
>
> 建立、更新和刪除等寫入操作需要兩個許可，如果缺少任一許可，寫入操作將會失敗。

CloudTrail 可見性

在 Kubernetes 資源上執行寫入操作時，您會在 CloudTrail 日誌中看到特定操作名稱：

- `createKubernetesObject` - 建立新資源時
- `updateKubernetesObject` - 修改現有資源時
- `deleteKubernetesObject` - 移除資源時

這些 CloudTrail 事件提供對 Kubernetes 資源所做的所有修改的詳細稽核線索。

> **Note**
>
> 這些操作名稱會出現在 CloudTrail 日誌中，僅供稽核之用。它們不是 IAM 動作，無法在 IAM 政策陳述式中使用。若要透過 IAM 政策控制對 Kubernetes 資源的寫入存取，請使用 `eks:MutateViaKubernetesApi`許可，如 the section called “所需的許可”一節所示。

透過建立 kubeconfig 檔案將 kubectl 連線至 EKS 叢集

> **Tip**
>
> 註冊即將舉行的 Amazon EKS 研討會。

在本主題中，您將為您的叢集建立 kubeconfig 檔案 (或更新現有的檔案)。

kubectl 命令列工具會使用 kubeconfig 檔案中的組態資訊，以與叢集的 API 伺服器進行通訊。如需詳細資訊，請參閱 Kubernetes 文件中的使用 kubeconfig 檔案組織叢集存取。

Amazon EKS 使用 aws eks get-token 命令搭配 kubectl 進行叢集身分驗證。根據預設，CLI AWS 會使用與使用下列命令傳回的相同登入資料：

```
aws sts get-caller-identity
```

- 現有 Amazon EKS 叢集。若要部署叢集，請參閱 開始使用。
- kubectl 命令列工具安裝在您的裝置或 AWS CloudShell 上。版本可以與您的叢集 Kubernetes 版本相同，或是為最多比該版本更舊一版或更新一版的次要版本。例如，如果您的叢集版本為 1.29，則可以搭配使用 kubectl 1.28、1.29 或 1.30 版。若要安裝或升級 kubectl，請參閱 the section called “設定 kubectl 和 eksctl”。
- 在您的裝置或 AWS CloudShell 上安裝和設定的 AWS 命令列界面 (AWS CLI) 版本 1.27.160 2.12.3或更新版本。若要檢查您目前的版本，請使用 aws --version | cut -d / -f2 | cut -d ' ' -f1。適用於 macOS 的 yum、 apt-get或 Homebrew 等套件管理員通常是最新版本 CLI AWS 後面的幾個版本。若要安裝最新版本，請參閱《 AWS 命令列界面使用者指南》中的使用 aws 設定安裝 和快速組態。 https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-quickstart.html#cli-configure-quickstart-config在 AWS CloudShell 中安裝的 AWS CLI 版本也可能是最新版本後面的幾個版本。若要更新它，請參閱《CloudShell AWS 使用者指南》中的將 CLI 安裝到您的主目錄。 AWS CloudShell
- 具有對您指定的叢集使用 eks:DescribeCluster API 動作許多的 IAM 使用者或角色。如需詳細資訊，請參閱the section called “身分型政策”。如果您使用自己的 OpenID Connect 提供商提供的身分來存取叢集，則請參閱 Kubernetes 文件中的 Using kubectl 一節以了解如何建立或更新 kubeconfig 檔案。

使用 EKS 主控台連線 (CloudShell)

您可以直接從 AWS 管理主控台 使用 AWS CloudShell 的 連線到叢集，而不需要任何本機設定。

1. 在以下網址開啟 Amazon EKS 主控台：https://console.aws.amazon.com/eks/home#/clusters。
2. 選擇您要連線的叢集名稱。
3. 在叢集詳細資訊頁面上，從右上角的動作列中選擇連線。
4. AWS CloudShell 會開啟，並預先為您的叢集`kubectl`設定 。

連線後，您可以立即執行`kubectl`命令：

```
kubectl get nodes
kubectl get pods -A
```

CloudShell 工作階段包括 `kubectl`、 CLI 和標準 CloudShell AWS 公用程式。此功能支援具有公有和私有 API 伺服器端點的叢集。對於具有私有端點的叢集，CloudShell 會自動啟動 VPC 環境以連接叢集的私有 API 伺服器，並提示您輸入環境的名稱。

使用 CLI AWS 建立**`kubeconfig`**檔案

- 在您的裝置或 AWS CloudShell 上安裝和設定的 AWS 命令列界面 (AWS CLI) 版本 `1.27.160` `2.12.3`或更新版本。若要檢查您目前的版本，請使用 `aws --version | cut -d / -f2 | cut -d ' ' -f1`。適用於 macOS 的 `yum`、 `apt-get`或 Homebrew 等套件管理員通常是最新版本 CLI AWS 後面的幾個版本。若要安裝最新版本，請參閱《 AWS 命令列界面使用者指南》中的使用 aws 設定安裝 和快速組態。 https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-quickstart.html#cli-configure-quickstart-config在 AWS CloudShell 中安裝的 AWS CLI 版本也可能是最新版本後面的幾個版本。若要更新它，請參閱《CloudShell AWS 使用者指南》中的將 CLI 安裝到您的主目錄。 AWS CloudShell
- 對您指定的叢集使用 `eks:DescribeCluster` API 動作的許可。如需詳細資訊，請參閱the section called “身分型政策”。
  1. 為您的叢集建立或更新 `kubeconfig` 檔案。將 *`region-code`* 取代為您的叢集所在的 AWS 區域，並將 *`my-cluster`* 取代為您的叢集名稱。

     ```
     aws eks update-kubeconfig --region region-code --name my-cluster
     ```

根據預設，產生的組態檔案會在主目錄預設的 kubeconfig 路徑下 (.kube)，或與位在該處的現有 config 檔案合併。您可以使用 --kubeconfig 選項指定其他路徑。

您可以使用 --role-arn 選項指定 IAM 角色 ARN，用於在您發出 kubectl 命令時進行身分驗證。否則，會使用預設 CLI 或 SDK 登入資料鏈中的 IAM 主體。 AWS 您可以執行 AWS aws sts get-caller-identity命令來檢視預設的 CLI 或 SDK 身分。

如需所有可用的選項，請執行 aws eks update-kubeconfig help命令，或參閱 AWS CLI 命令參考中的 update-kubeconfig。

2. 測試組態。

```
kubectl get svc
```

範例輸出如下。

```
NAME             TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
svc/kubernetes   ClusterIP   10.100.0.1   <none>        443/TCP   1m
```

如果您收到任何授權或資源類型錯誤，請參閱故障診斷主題中的the section called “未經授權或存取遭拒 (kubectl)”。

AWS 使用 Kubernetes 服務帳戶授予 Kubernetes 工作負載存取權

Kubernetes 服務帳戶為在 Pod 中執行的程序提供身分。如需詳細資訊，請參閱 Kubernetes 文件中的 Managing Service Accounts (管理服務帳戶)。如果您的 Pod 需要存取 AWS 服務，您可以將服務帳戶對應至 AWS Identity and Access Management 身分，以授予該存取權。如需詳細資訊，請參閱 the section called “具有 IRSA 的憑證” 或 the section called “Pod 身分識別” 。

服務帳戶字符

在 Kubernetes 版本中預設會啟用 BoundServiceAccountTokenVolume 功能。此功能允許在 Kubernetes 上執行的工作負載請求對象、時間和金鑰綁定的 JSON Web 字符，從而提升服務帳戶字符的安全性。服務帳戶字符的過期時間為一小時。在舊版 Kubernetes 版本中，字符沒有過期。這表示仰賴這些字符的客戶端必須在一小時內重新整理字符。以下 Kubernetes 用戶端 SDK 會在要求的時間範圍內自動重新整理字符：

- Go 版本 0.15.7 和更新版本

- Python 版本 `12.0.0` 和更新版本
- Java 版本 `9.0.0` 和更新版本
- JavaScript 版本 `0.10.3` 和更新版本
- Ruby `master` 分支
- Haskell 版本 `0.3.0.0`
- C# `7.0.5` 版及更新版本

如果工作負載使用較早的客戶端版本，則必須進行更新。為使客戶端順暢移轉到更新的限時服務帳戶字符，Kubernetes 對服務帳戶字符新增了超過預設的一小時的延長到期期間。針對 Amazon EKS 叢集，延長的到期期間為 90 天。您的 Amazon EKS 叢集的 Kubernetes API 伺服器拒絕超過 90 天的字符的請求。我們建議您檢查應用程式及其相依性，以確保 Kubernetes 用戶端開發套件與先前列出的版本相同或為更新版本。

當 API 服務器收到使用超過一小時的字符的請求時，它會使用 `annotations.authentication.k8s.io/stale-token` 註釋 API 稽核日誌事件。註釋的值如下範例所示：

```
subject: system:serviceaccount:common:fluent-bit, seconds after warning threshold:
 4185802.
```

如果您的叢集啟用了控制平面記錄，則註釋位於稽核日誌中。您可以使用下列 CloudWatch Logs Insights 查詢以識別使用過時字符的 Amazon EKS 叢集中的所有 Pod：

```
fields @timestamp
|filter @logStream like /kube-apiserver-audit/
|filter @message like /seconds after warning threshold/
|parse @message "subject: *, seconds after warning threshold:*\"" as subject,
 elapsedtime
```

`subject` 指的是該 Pod 使用的服務帳戶。`elapsedtime` 表示讀取最新字符後的經過時間（以秒為單位）。當 `elapsedtime` 超過 90 天 (7,776,000 秒)，對 API 伺服器的請求將遭到拒絕。您應主動更新應用程式的 Kubernetes 用戶端 SDK，以使用先前列出的自動重新整理字符的版本之一。如果您使用的服務帳戶字符接近 90 天，並且沒有足夠的時間在字符過期前更新客戶端 SDK 版本，則可以終止現有 Pod 並建立新 Pod。這會導致重新擷取服務帳戶字符，進而給您額外的 90 天來更新的客戶端版本 SDK。

如果 Pod 為部署的一部分，則在維持高可用性的同時終止 Pod 的建議方法是使用以下命令執行推出。使用您的部署名稱替換 *my-deployment* (我的部署)。

```
kubectl rollout restart deployment/my-deployment
```

叢集附加元件

下列叢集附加元件已更新以使用自動重新擷取服務帳戶字符的 Kubernetes 客戶端 SDK。建議您確保已在叢集上安裝列出的版本或更新版本。

- 適用於 Kubernetes 的 Amazon VPC CNI 外掛程式及指標輔助外掛程式 `1.8.0` 版及更新版本。若要檢查目前版本或更新版本，請參閱 the section called “Amazon VPC CNI” 和 cni-metrics-helper。
- CoreDNS `1.8.4` 版及更新版本。若要檢查目前版本或更新版本，請參閱 the section called “CoreDNS”。
- AWS Load Balancer控制器版本 `2.0.0`和更新版本。若要檢查目前版本或更新版本，請參閱 the section called “ AWS Load Balancer控制器”。
- 目前的 `kube-proxy` 版本。若要檢查目前版本或更新版本，請參閱 the section called “kube-proxy”。
- AWS 適用於 Fluent Bit 版本 `2.25.0` 或更新版本。若要更新您的目前版本，請參閱 GitHub 上的 Releases (版本)。
- Fluentd 映像 1.14.6-1.2 版或更新版本，以及適用於 Kubernetes 中繼資料的 Fluentd 篩選器外掛程式 2.11.1 版或更新版本。

將 AWS Identity and Access Management 許可授予 Amazon Elastic Kubernetes Service 叢集上的工作負載

Amazon EKS 提供兩種方式，將 AWS Identity and Access Management 許可授予在 Amazon EKS 叢集中執行的工作負載：服務帳戶的 IAM 角色，以及 EKS Pod 身分。

服務帳戶的 IAM 角色

> 服務帳戶的 IAM 角色 (IRSA) 會設定在 上執行的 Kubernetes 應用程式 AWS ，具有精細的 IAM 許可，以存取各種其他 AWS 資源，例如 Amazon S3 儲存貯體、Amazon DynamoDB 資料表等。您可以在同一個 Amazon EKS 叢集中同時執行多個應用程式，並確保每個應用程式只有其所需的最低許可集。IRSA 旨在支援 支援的各種 Kubernetes 部署選項， AWS 例如 Amazon EKS、Amazon EKS Anywhere、Red Hat OpenShift Service on AWS，以及 Amazon EC2 執行個體上的自我管理

Kubernetes 叢集。因此，IRSA 是使用 IAM 等基礎 AWS 服務建置的，並且不直接依賴 Amazon EKS 服務和 EKS API。如需詳細資訊，請參閱the section called “具有 IRSA 的憑證”。

EKS Pod 身分

EKS Pod Identity 為叢集管理員提供簡化的工作流程，以驗證應用程式存取各種其他 AWS 資源，例如 Amazon S3 儲存貯體、Amazon DynamoDB 資料表等。EKS Pod 身分識別僅適用於 EKS，因此可簡化叢集管理員設定 Kubernetes 應用程式以取得 IAM 許可的方式。您現在可以直接透過 EKS API 和 AWS CLI 輕鬆設定這些許可 AWS 管理主控台，而且在任何 Kubernetes 物件中，叢集內不需要採取任何動作。叢集管理員不需要在 EKS 和 IAM 服務之間切換，也不需要使用特殊權限 IAM 操作來設定應用程式所需的許可。IAM 角色現在可跨多個叢集使用，無需在建立新叢集時更新角色信任政策。EKS Pod 身分識別提供的 IAM 憑證包含角色工作階段標籤，並具有叢集名稱、命名空間、服務帳戶名稱等屬性。角色工作階段標籤可讓管理員根據相符的標籤，允許存取 AWS 資源，以撰寫可在服務帳戶間運作的單一角色。如需詳細資訊，請參閱the section called “Pod 身分識別”。

比較 EKS Pod 身分識別和 IRSA

在高階，EKS Pod Identity 和 IRSA 可讓您將 IAM 許可授予在 Kubernetes 叢集上執行的應用程式。但是，它們在設定方式、支援的限制和啟用的功能完全不同。下面，我們比較兩種解決方案的一些關鍵面向。

> **Note**
>
> AWS 建議盡可能使用 EKS Pod 身分來授予 Pod AWS 資源的存取權。如需詳細資訊，請參閱the section called “Pod 身分識別”。

屬性	EKS Pod 身分識別	IRSA
角色擴充性	您必須設定每個角色一次，才能與新推出的 Amazon EKS 服務主體 `pods.eks.amazonaws.com` 建立信任。完成此一次性步驟之後，您不需要每次在新叢集中使用角色的信任政策時更新該政策。	每次要在新叢集中使用 IAM 角色時，您都必須使用新的 EKS 叢集 OIDC 提供者端點更新該角色的信任政策。

屬性	EKS Pod 身分識別	IRSA
叢集可擴展性	EKS Pod 身分識別不需要使用者設定 IAM OIDC 提供者，因此不適用此限制。	每個 EKS 叢集具有與其相關聯的 OpenID Connect (OIDC) 發行者 URL。若要使用 IRSA，則必須為 IAM 中的每個 EKS 叢集建立唯一的 OpenID Connect 提供者。IAM 每個 AWS 帳戶的預設全域限制為 100 個 OIDC 供應商。如果您計劃為每個具有 IRSA AWS 的帳戶擁有超過 100 個 EKS 叢集，則您將達到 IAM OIDC 提供者限制。
角色可擴展性	EKS Pod 身分識別不需要使用者在信任政策中定義 IAM 角色和服務帳戶之間的信任關係，因此不適用此限制。	在 IRSA 中，您可以在角色的信任政策中定義 IAM 角色和服務帳戶之間的信任關係。根據預設，信任政策大小的長度為 2048。這表示您通常可以在單一信任政策中定義 4 個信任關係。雖然您可以提高信任政策長度限制，但在單一信任政策中，您通常只能使用最多 8 個信任關係。
STS API 配額用量	EKS Pod Identity 可簡化將 AWS 登入資料交付至 Pod 的程序，而且不需要您的程式碼直接使用 AWS Security Token Service (STS) 進行呼叫。EKS 服務會處理角色假設，並將登入資料交付至使用 Pod 中 AWS 開發套件撰寫的應用程式，而不需要您的 Pod 與 AWS STS 通訊或使用 STS API Quota。	在 IRSA 中，使用 Pod 中的 AWS SDK 撰寫的應用程式會使用權杖來呼叫 AWS Security Token Service (STS) 上的 `AssumeRoleWithWebIdentity` API。視軟體 AWS 開發套件上程式碼的邏輯而定，您的程式碼可能會對 AWS STS 進行不必要的呼叫，並接收限流錯誤。

屬性	EKS Pod 身分識別	IRSA
角色可重複使用性	AWS EKS Pod Identity 提供的 STS 臨時登入資料包含角色工作階段標籤，例如叢集名稱、命名空間、服務帳戶名稱。角色工作階段標籤可讓管理員撰寫單一 IAM 角色，這些角色可以與多個服務帳戶搭配使用，具有不同的有效許可，方法是允許根據連接至他們的標籤存取 AWS 資源。這也稱為屬性型存取控制 (ABAC)。如需詳細資訊，請參閱the section called “授予 Pods 存取權”。	AWS 不支援 STS 工作階段標籤。您可以在叢集之間重複使用角色，但是每個 Pod 都會收到該角色的所有許可。
支援的環境	EKS Pod 身分識別僅適用於 Amazon EKS。	IRSA 可用於 Amazon EKS、Amazon EKS Anywhere、Red Hat OpenShift Service on AWS，以及 Amazon EC2 執行個體上的自我管理 Kubernetes 叢集。
支援 EKS 版本	所有支援的 EKS 叢集版本。如需特定平台版本，請參閱 the section called “EKS Pod 身分識別叢集版本”。	所有支援的 EKS 叢集版本。

服務帳戶的 IAM 角色

> **Tip**
>
> 註冊即將舉行的 Amazon EKS 研討會。

Pod 容器中的應用程式可以使用 AWS SDK 或 AWS CLI，使用 AWS Identity and Access Management (IAM) 許可向 AWS 服務提出 API 請求。應用程式必須使用 AWS 登入資料簽署其 AWS

API 請求。服務帳戶的 IAM 角色 (IRSA) 提供管理應用程式憑證的功能，類似 Amazon EC2 執行個體設定檔將憑證提供給 Amazon EC2 執行個體的方式。您可以將 IAM 角色與 Kubernetes 服務帳戶建立關聯，並將 Pod 設定為使用服務帳戶，而不是建立 AWS 登入資料並將其分發至容器或使用 Amazon EC2 執行個體的 角色。您無法將 IAM 角色用於具有 Amazon EKS on AWS Outposts 本機叢集的服務帳戶。

服務帳戶的 IAM 角色提供下列優點：

- 最低權限：您可以將 IAM 許可範圍限定為服務帳戶，只有使用該服務帳戶的 Pod 才能存取這些許可。有此功能也就不需要第三方解決方案，例如 `kiam` 或 `kube2iam`。
- 憑證隔離：若 Amazon EC2 執行個體中繼資料服務 (IMDS) 存取權受限，Pod 的容器僅可擷取與容器所用服務帳戶關聯的 IAM 角色憑證。容器一律無法存取其他 Pod 中其他容器使用的憑證。若 IMDS 不受限制，Pod 的容器亦可存取 Amazon EKS 節點 IAM 角色，且容器或許能存取相同節點上其他 Pod 的 IAM 角色的憑證。如需詳細資訊，請參閱限制存取指派給工作節點的執行個體設定檔。

> **Note**
>
> 使用 設定的 Pod `hostNetwork: true` 一律具有 IMDS 存取權，但啟用時， AWS SDKs和 CLI 將使用 IRSA 登入資料。

- 可稽核性 – 可透過 AWS CloudTrail 存取和事件記錄，以協助確保回溯性稽核。

> **Important**
>
> 容器並非安全邊界，而且使用服務帳戶的 IAM 角色亦不會對此有所變更。若 Pod 指派到相同的節點，將會分享核心及可能的其他資源，具體取決於 Pod 組態。儘管執行於單獨節點上的 Pod 將會在運算層隔離，但有些節點應用程式在 Kubernetes API 中擁有超出個別執行個體範圍的額外許可。部分範例包括 `kubelet`、`kube-proxy`、CSI 儲存驅動程式，或者您本人的 Kubernetes 應用程式。

完成下列程序，為服務帳戶啟用 IAM 角色：

1. 為叢集建立 IAM OIDC 提供商：對於每個叢集，您只需完成一次本程序。

> **Note**
>
> 如果您叢集的 VPC 沒有傳出網際網路存取權，而且您尚未設定叢集 OIDC 端點的私有存取權，則從 VPC 內部到達該端點的操作，例如使用 建立 OIDC 供應商`eksctl`，將無法解析 OIDC 發行者主機名稱。以下是範例錯誤訊息：
>
> ```
> server cant find oidc.eks.region.amazonaws.com: NXDOMAIN
> ```
>
> 若要從 VPC 私下到達叢集 OIDC 端點，請在啟用私有 DNS 的情況下為其建立 VPC 介面端點 (`com.amazonaws.region-code.oidc-eks`)。如需詳細資訊，請參閱the section called “ AWS PrivateLink”。
> 或者，您可以在 VPC 外部執行 命令 （ 例如 in AWS CloudShell)，或建立分割期限條件式解析程式。如需範例，請參閱 GitHub 上的 Amazon EKS 功能請求。

2. 將 IAM 角色指派給 Kubernetes Service 帳戶：對您希望應用程式擁有的每種非重複可組合完成此程序。
3. 將 Pod 設定為使用 Kubernetes 服務帳戶 – 為每個需要存取 AWS 服務的 Pod 完成此程序。
4. 搭配 AWS SDK 使用 IRSA – 確認工作負載使用支援版本的 AWS SDK，以及工作負載使用預設登入資料鏈。

IAM, Kubernetes 和 OpenID Connect (OIDC) 背景資訊

在 2014 年， AWS Identity and Access Management 新增了對使用 OpenID Connect (OIDC) 的聯合身分的支援。此功能可讓您向支援的身分提供者驗證 AWS API 呼叫，並接收有效的 OIDC JSON Web 字符 (JWT)。您可以將此權杖傳遞至 AWS STS `AssumeRoleWithWebIdentity` API 操作，並接收 IAM 臨時角色登入資料。您可以使用這些登入資料與任何 AWS 服務互動，包括 Amazon S3 和 DynamoDB。

每個 JWT 權杖均由一個簽署金鑰對簽署。這些金鑰是由 Amazon EKS 管理的 OIDC 供應商提供服務，且私有金鑰每 7 天就會輪換一次。Amazon EKS 會保留公有金鑰，直到它們過期為止。如果您連接外部 OIDC 用戶端，請注意，您必須在公有金鑰到期之前重新整理簽署金鑰。了解如何擷取簽署金鑰以驗證 OIDC 權杖。

Kubernetes 長期以來使用服務帳戶作為自己的內部身分系統。Pod 可以使用只有 Kubernetes API 伺服器可以驗證的自動掛載字符 (非 OIDC JWT)，向 Kubernetes API 伺服器進行驗證。這些舊的服務帳戶字符不會過期，而輪換簽署金鑰是一項困難的程序。Kubernetes 版本 `1.12` 進一步支援新的

ProjectedServiceAccountToken 功能。此功能是 OIDC JSON Web 字符，同時包含服務帳戶身分，並支援可設定的對象。

Amazon EKS 為每個叢集託管公有 OIDC 探索端點，其中包含 ProjectedServiceAccountToken JSON Web 字符的簽署金鑰，讓外部系統 (例如 IAM) 可以驗證和接受 Kubernetes 發出的 OIDC 字符。

為您的叢集建立 IAM OIDC 身分提供者

您的叢集具有與其相關聯的 OpenID Connect (OIDC) 發行者 URL。若要針對服務帳戶使用 AWS Identity and Access Management (IAM) 角色，叢集的 OIDC 發行者 URL 必須存在 IAM OIDC 供應商。

先決條件

- 現有 Amazon EKS 叢集。若要部署叢集，請參閱 開始使用。
- 在您的裝置或 AWS CloudShell 上安裝和設定的 AWS 命令列界面 (AWS CLI) 版本 1.27.160 2.12.3或更新版本。若要檢查您目前的版本，請使用 aws --version | cut -d / -f2 | cut -d ' ' -f1。適用於 macOS 的 yum、 apt-get或 Homebrew 等套件管理員通常是最新版本 CLI AWS 後面的幾個版本。若要安裝最新版本，請參閱《 AWS 命令列界面使用者指南》中的使用 aws 設定安裝 和快速組態。 https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-quickstart.html#cli-configure-quickstart-config在 AWS CloudShell 中安裝的 AWS CLI 版本也可能是最新版本後面的幾個版本。若要更新它，請參閱《CloudShell AWS 使用者指南》中的將 CLI 安裝到您的主目錄。 AWS CloudShell
- kubectl 命令列工具安裝在您的裝置或 AWS CloudShell 上。版本可以與您的叢集 Kubernetes 版本相同，或是為最多比該版本更舊一版或更新一版的次要版本。例如，如果您的叢集版本為 1.29，則可以搭配使用 kubectl 1.28、1.29 或 1.30 版。若要安裝或升級 kubectl，請參閱 the section called “設定 kubectl 和 eksctl”。
- 包含叢集組態的現有 kubectl config 檔案。若要建立 kubectl config 檔案，請參閱the section called “透過 kubectl 存取叢集”。

您可以使用 eksctl 或 AWS 管理主控台為叢集建立 IAM OIDC 提供商。

建立 OIDC 提供商 (eksctl)

1. 裝置或 AWS CloudShell 上安裝的eksctl命令列工具版本 0.215.0或更新版本。如需有關安裝或更新 eksctl 的指示，請參閱 eksctl 文件中的安裝一節。
2. 確定叢集的 OIDC 發行者 ID。

擷取叢集的 OIDC 發行者 ID 並將其存放在變數中。使用您自己的值取代 <my-cluster>。

```
cluster_name=<my-cluster>
oidc_id=$(aws eks describe-cluster --name $cluster_name --query
 "cluster.identity.oidc.issuer" --output text | cut -d '/' -f 5)
echo $oidc_id
```

3. 判斷您的帳戶中是否已經有擁有叢集發行者 ID 的 IAM OIDC 提供商。

```
aws iam list-open-id-connect-providers | grep $oidc_id | cut -d "/" -f4
```

如果有輸出傳回，則表示您已經有叢集 IAM OIDC 提供商，且可以略過下一步驟。如果未傳回任何輸出，則您必須為叢集建立 IAM OIDC 提供商。

4. 使用下列命令為您的叢集建立 IAM OIDC 身分提供者。

```
eksctl utils associate-iam-oidc-provider --cluster $cluster_name --approve
```

> **Note**
>
> 如果您叢集的 VPC 沒有傳出網際網路存取權，而且您尚未設定叢集 OIDC 端點的私有存取權，則從 VPC 內部到達該端點的操作，例如使用 建立 OIDC 供應商eksctl，將無法解析 OIDC 發行者主機名稱。以下是範例錯誤訊息：
>
> ```
> ** server cant find oidc.eks.<region-code>.amazonaws.com: NXDOMAIN
> ```

若要從 VPC 私下到達叢集 OIDC 端點，請在啟用私有 DNS 的情況下為其建立 VPC 介面端點 (com.amazonaws.<region-code>.oidc-eks)。如需詳細資訊，請參閱使用 AWS PrivateLink 存取叢集 OIDC 端點。

或者，您可以在 VPC 外部執行 命令 （ 例如 in AWS CloudShell)，或建立分割期限條件式解析程式。如需範例，請參閱 GitHub 上的 Amazon EKS 功能請求。

建立 OIDC 供應商AWS （ 主控台 ）

1. 開啟 Amazon EKS 主控台。
2. 在左側窗格中，選取 Clusters (叢集)，然後在 Clusters (叢集) 頁面上選取您的叢集名稱。

3. 在 Overview (概觀) 標籤的 Details (詳細資訊) 區段中，記下 OpenID Connect provider URL (OpenID Connect 供應商 URL) 的值。
4. 前往 https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。
5. 在左側導覽窗格中，選擇 Access management (存取管理) 下的 Identity Providers (身分提供者)。如果列出的提供商與您叢集的 URL 相符，則表示您已經擁有叢集提供商。如果沒有列出符合叢集 URL 的提供商，則您必須建立一個。
6. 若要建立供應商，請選擇 Add provider (新增供應商)。
7. 對於提供商類型，選取 OpenID Connect。
8. 對於 Provider URL (提供商 URL)，輸入叢集的 OIDC 提供商 URL。
9. 對於 Audience (對象)，輸入 `sts.amazonaws.com`。
10. (選用) 新增任何標籤，例如可識別適用於此提供商的叢集的標籤。
11. 選擇 Add provider (新增提供者)。

後續步驟：the section called “指派 IAM 角色”

將 IAM 角色指派給 Kubernetes 服務帳戶

本主題說明如何設定 Kubernetes 服務帳戶以擔任 AWS Identity and Access Management (IAM) 角色。然後，任何設定為使用服務帳戶的 Pod 都可以存取該角色有權存取的任何 AWS 服務。

先決條件

- 現有的叢集。如果您沒有，則可以按照其中一個 開始使用 指南來建立。
- 叢集的現有 IAM OpenID Connect (OIDC) 提供商。若要了解是否已經擁有，或是了解如何建立，請參閱the section called “IAM OIDC 提供商”。
- 裝置或 AWS CloudShell 上安裝和設定的 AWS 命令列界面 (AWS CLI) 版本 `1.27.160` `2.12.3`或更新版本。若要檢查您目前的版本，請使用 `aws --version | cut -d / -f2 | cut -d ' ' -f1`。適用於 macOS 的 `yum`、 `apt-get`或 Homebrew 等套件管理員通常是最新版本 CLI AWS 後面的幾個版本。若要安裝最新版本，請參閱《 AWS 命令列界面使用者指南》中的使用 aws 設定安裝 和快速組態。 https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-quickstart.html#cli-configure-quickstart-config安裝在 AWS CloudShell 中的 AWS CLI 版本也可能是最新版本後面的幾個版本。若要更新它，請參閱《CloudShell AWS 使用者指南》中的將 CLI 安裝到您的主目錄。 AWS CloudShell
- `kubectl` 命令列工具安裝在您的裝置或 AWS CloudShell 上。版本可以與您的叢集 Kubernetes 版本相同，或是為最多比該版本更舊一版或更新一版的次要版本。例如，如果您的叢集版本為 `1.29`，

則可以搭配使用 kubectl 1.28、1.29 或 1.30 版。若要安裝或升級 kubectl，請參閱 the section called “設定 kubectl 和 eksctl”。

- 包含叢集組態的現有 kubectl config 檔案。若要建立 kubectl config 檔案，請參閱the section called “透過 kubectl 存取叢集”。

步驟 1：建立 IAM 政策

如果您要將現有 IAM 政策與 IAM 角色建立關聯，請跳到下一步驟。

1. 建立 IAM 政策。您可以建立自己的政策，或複製已授予部分所需許可的 AWS 受管政策，並根據您的特定需求進行自訂。如需詳細資訊，請參閱《IAM 使用者指南》中的建立 IAM 政策。
2. 建立檔案，其中包含您希望 Pod 存取 AWS 之服務的許可。如需所有 AWS 服務的所有動作清單，請參閱服務授權參考。

   您可以執行以下命令來建立允許唯讀存取 Amazon S3 儲存貯體的範例政策檔案。您可以選擇性地將組態資訊或引導指令碼存放在此儲存貯體中，而 Pod 中的容器可從儲存貯體讀取檔案，並載入至您的應用程式。如果您要建立此範例政策，請將以下內容複製到您的裝置。將 *my-pod-secrets-bucket* 取代為您的儲存貯體名稱並執行命令。

   ```
   {
       "Version":"2012-10-17",
       "Statement": [
           {
               "Effect": "Allow",
               "Action": "s3:GetObject",
               "Resource": "arn:aws:s3:::my-pod-secrets-bucket"
           }
       ]
   }
   ```

3. 建立 IAM 政策。

   ```
   aws iam create-policy --policy-name my-policy --policy-document file://my-policy.json
   ```

步驟 2：建立並關聯 IAM 角色

建立 IAM 角色並與 Kubernetes 服務帳戶建立關聯。您可以使用 eksctl或 AWS CLI。

建立並關聯角色 (eksctl)

此 eksctl 命令會在指定的命名空間中建立 Kubernetes 服務帳戶、建立具有指定名稱的 IAM 角色 (若不存在)、將現有的 IAM 政策 ARN 附加至該角色，並以 IAM 角色 ARN 註釋服務帳戶。請務必將此命令中的範例預留位置值取代為您的特定值。如需有關安裝或更新 eksctl 的指示，請參閱 eksctl 文件中的安裝一節。

```
eksctl create iamserviceaccount --name my-service-account --namespace default --cluster
 my-cluster --role-name my-role \
    --attach-policy-arn arn:aws: iam::111122223333:policy/my-policy --approve
```

Important

如果角色或服務帳戶已存在，之前的命令可能會失敗。 eksctl具有您可以在這些情況下提供的不同選項。如需詳細資訊，請執行 eksctl create iamserviceaccount --help。

建立角色並建立關聯 (AWS CLI)

如果您有要擔任 IAM 角色的現有 Kubernetes 服務帳戶，則可略過此步驟。

1. 建立 Kubernetes 服務帳戶。將以下內容複製到您的裝置。視需要將 *my-service-account* 取代為您要使用的名稱，將 *default* 取代為其他命名空間。如果您變更 *default*，命名空間必須已經存在。

   ```
   cat >my-service-account.yaml <<EOF
   apiVersion: v1
   kind: ServiceAccount
   metadata:
     name: my-service-account
     namespace: default
   EOF
   kubectl apply -f my-service-account.yaml
   ```

2. 使用下列命令將 AWS 您的帳戶 ID 設定為 環境變數。

   ```
   account_id=$(aws sts get-caller-identity --query "Account" --output text)
   ```

3. 使用下列命令將叢集的 OIDC 身分提供者設定為環境變數。使用您叢集的名稱取代 *my-cluster*。

```
oidc_provider=$(aws eks describe-cluster --name my-cluster --region $AWS_REGION --
query "cluster.identity.oidc.issuer" --output text | sed -e "s/^https:\/\///")
```

4. 設定命名空間和服務帳戶名稱的變數。將 *my-service-account* 取代為要擔任角色的 Kubernetes 服務帳戶。將 *default* 取代為服務帳戶的命名空間。

```
export namespace=default
export service_account=my-service-account
```

5. 執行下列命令以建立 IAM 角色的信任政策檔案。如果您想要允許命名空間中的所有服務帳戶使用該角色，請將以下內容複製到您的裝置。將 *StringEquals* 取代為 StringLike，並將 *$service_account* 取代為 *。您可以在 StringEquals 和 StringLike 條件中新增多個項目，以允許多個服務帳戶或命名空間擔任角色。若要允許來自與叢集所在 AWS 帳戶不同的 帳戶的角色擔任該角色，請參閱 the section called “跨帳戶 IAM” 以取得詳細資訊。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::123456789012:oidc-provider/$oidc_provider"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "$oidc_provider:aud": "sts.amazonaws.com",
          "$oidc_provider:sub": "system:serviceaccount:$namespace:$service_account"
        }
      }
    }
  ]
}
```

6. 建立角色。將 *my-role* 取代為您的 IAM 角色名稱，並將 *my-role-description* 取代為您角色的描述。

```
aws iam create-role --role-name my-role --assume-role-policy-document file://trust-
relationship.json --description "my-role-description"
```

7. 將 IAM 政策連接至您的角色。將 *my-role* 取代為您的 IAM 角色名稱，將 *my-policy* 取代為您建立的現有政策名稱。

```
aws iam attach-role-policy --role-name my-role --policy-arn=arn:aws: iam::
$account_id:policy/my-policy
```

8. 使用您希望服務帳戶擔任之 IAM 角色的 Amazon Resource Name (ARN) 標註您的服務帳戶。使用現有 IAM 角色的名稱取代 *my-role*。假設您允許來自與叢集所在 AWS 帳戶不同的 帳戶的角色，以在上一個步驟中擔任該角色。然後，請務必從另一個 AWS 帳戶指定帳戶和角色。如需詳細資訊，請參閱the section called “跨帳戶 IAM”。

```
kubectl annotate serviceaccount -n $namespace $service_account eks.amazonaws.com/
role-arn=arn:aws: iam::$account_id:role/my-role
```

9. （選用）設定服務帳戶 AWS 的安全字符服務端點。 AWS 建議使用區域 AWS STS 端點而非全域端點。這樣可以減少延遲、提供內建備援，並增加工作階段字符的有效性。

步驟 3：確認組態

1. 確認 IAM 角色的信任政策設定正確。

```
aws iam get-role --role-name my-role --query Role.AssumeRolePolicyDocument
```

範例輸出如下。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::111122223333:oidc-provider/oidc.eks.us-
east-1.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "oidc.eks.us-east-1.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:sub": "system:serviceaccount:default:my-service-
account",
```

```
                    "oidc.eks.us-east-1.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:aud": "sts.amazonaws.com"
                }
            }
        }
    ]
}
```

2. 確認您在上一步連接至角色的政策已連接至該角色。

```
aws iam list-attached-role-policies --role-name my-role --query
 "AttachedPolicies[].PolicyArn" --output text
```

範例輸出如下。

```
 arn:aws: iam::111122223333:policy/my-policy
```

3. 設定變數以存放您要使用之政策的 Amazon Resource Name (ARN)。將 *my-policy* 取代為您要確認許可的政策名稱。

```
export policy_arn=arn:aws: iam::111122223333:policy/my-policy
```

4. 檢視預設政策版本。

```
aws iam get-policy --policy-arn $policy_arn
```

範例輸出如下。

```
{
    "Policy": {
        "PolicyName": "my-policy",
        "PolicyId": "EXAMPLEBIOWGLDEXAMPLE",
        "Arn": "arn:aws: iam::111122223333:policy/my-policy",
        "Path": "/",
        "DefaultVersionId": "v1",
        [...]
    }
}
```

5. 檢視政策內容，以確定政策包含您的 Pod 需要的所有許可。視需要將以下命令中的 *1* 取代為上一個輸出中傳回的版本。

```
aws iam get-policy-version --policy-arn $policy_arn --version-id v1
```

範例輸出如下。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::my-pod-secrets-bucket"
        }
    ]
}
```

如果您在上一步建立了範例政策，則輸出結果相同。如果您建立了不同的政策，則 *example* 內容有所不同。

6. 確認 Kubernetes 服務帳戶標註了角色。

```
kubectl describe serviceaccount my-service-account -n default
```

範例輸出如下。

```
Name:                my-service-account
Namespace:           default
Annotations:         eks.amazonaws.com/role-arn: arn:aws: iam::111122223333:role/my-
role
Image pull secrets:  <none>
Mountable secrets:   my-service-account-token-qqjfl
Tokens:              my-service-account-token-qqjfl
[...]
```

後續步驟

- the section called “指派到 Pod”

設定 Pod 來使用 Kubernetes 服務帳戶

如果 Pod 需要存取 AWS 服務，則必須將其設定為使用 Kubernetes 服務帳戶。服務帳戶必須與具有存取 AWS 服務許可的 AWS Identity and Access Management (IAM) 角色相關聯。

- 現有的叢集。如果您沒有，則可以使用其中一個 開始使用 指南來建立一個。
- 叢集的現有 IAM OpenID Connect (OIDC) 提供商。若要了解是否已經擁有，或是了解如何建立，請參閱the section called “IAM OIDC 提供商”。
- 與 IAM 角色關聯的現有 Kubernetes 服務帳戶。服務帳戶必須標註 IAM 角色的 Amazon Resource Name (ARN)。角色必須具有關聯的 IAM 政策，其中包含您希望 Pod 必須使用 AWS 服務的許可。如需有關服務帳戶和角色之建立和設定方式的詳細資訊，請參閱the section called “指派 IAM 角色”。
- 在您的裝置或 AWS CloudShell 上安裝和設定的 AWS 命令列界面 (AWS CLI) 版本 `1.27.160` `2.12.3`或更新版本。若要檢查您目前的版本，請使用 `aws --version | cut -d / -f2 | cut -d ' ' -f1`。適用於 macOS 的 `yum`、 `apt-get`或 Homebrew 等套件管理員通常是最新版本 CLI AWS 後面的幾個版本。若要安裝最新版本，請參閱《 AWS 命令列界面使用者指南》中的使用 aws 設定安裝 和快速組態。 https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-quickstart.html#cli-configure-quickstart-config在 AWS CloudShell 中安裝的 AWS CLI 版本也可能是最新版本後面的幾個版本。若要更新它，請參閱《CloudShell AWS 使用者指南》中的將 CLI 安裝到您的主目錄。 AWS CloudShell
- `kubectl` 命令列工具安裝在您的裝置或 AWS CloudShell 上。版本可以與您的叢集 Kubernetes 版本相同，或是為最多比該版本更舊一版或更新一版的次要版本。例如，如果您的叢集版本為 `1.29`，則可以搭配使用 `kubectl` `1.28`、`1.29` 或 `1.30` 版。若要安裝或升級 `kubectl`，請參閱 the section called “設定 kubectl 和 eksctl”。
- 包含叢集組態的現有 `kubectl config` 檔案。若要建立 `kubectl config` 檔案，請參閱the section called “透過 kubectl 存取叢集”。

  1. 使用以下命令，以建立部署清單檔案，讓您可部署 Pod 以確認組態。以您自己的值取代範例值。

```
cat >my-deployment.yaml <<EOF
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
spec:
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
```

```
      labels:
        app: my-app
    spec:
      serviceAccountName: my-service-account
      containers:
      - name: my-app
        image: public.ecr.aws/nginx/nginx:X.XX
EOF
```

2. 將清單檔案部署到叢集。

```
kubectl apply -f my-deployment.yaml
```

3. 確認您的 Pod 有必要的環境變數。

   a. 檢視使用上一步中的部署作業進行部署的 Pod。

   ```
   kubectl get pods | grep my-app
   ```

   範例輸出如下。

   ```
   my-app-6f4dfff6cb-76cv9   1/1     Running   0          3m28s
   ```

   b. 檢視 Pod 所使用 IAM 角色的 ARN。

   ```
   kubectl describe pod my-app-6f4dfff6cb-76cv9 | grep AWS_ROLE_ARN:
   ```

   範例輸出如下。

   ```
   AWS_ROLE_ARN: arn:aws: iam::111122223333:role/my-role
   ```

   角色 ARN 必須與您用來標註現有服務帳戶的角色 ARN 相符。如需有關標註服務帳戶的詳細資訊，請參閱the section called “指派 IAM 角色”。

   c. 確認 Pod 有 Web 身分字符檔案掛載。

   ```
   kubectl describe pod my-app-6f4dfff6cb-76cv9 | grep AWS_WEB_IDENTITY_TOKEN_FILE:
   ```

   範例輸出如下。

```
AWS_WEB_IDENTITY_TOKEN_FILE:  /var/run/secrets/eks.amazonaws.com/serviceaccount/
token
```

kubelet 請求並代表 Pod 存放字符。根據預設，如果字符超過總存留時間的 80% 或 24 小時，kubelet 會重新整理字符。您可以使用 Pod 規格中的設定來修改任何帳戶 (預設服務帳戶除外) 的過期持續時間。如需詳細資訊，請參閱 Kubernetes 文件中的服務帳戶字符磁碟區投影。

叢集上的 Amazon EKS Pod 身分識別 Webhook 會監視使用具有以下註釋之服務帳戶的 Pod：

```
eks.amazonaws.com/role-arn: arn:aws: iam::111122223333:role/my-role
```

Webhook 會將先前的環境變數套用至這些 Pod。您的叢集不需要使用 Webhook 來設定環境變數和字符檔案掛載。您可以手動將 Pod 設定為具有這些環境變數。支援的 AWS SDK 版本會先在登入資料鏈提供者中尋找這些環境變數。符合此條件的 Pod 會使用角色憑證。

4. 確認您的 Pod 可以使用您在連接至角色的 IAM 政策中指派的許可與 AWS 服務互動。

> **Note**
>
> 當 Pod 使用與服務帳戶相關聯的 IAM 角色 AWS 登入資料時，該 Pod 容器中的 AWS CLI 或其他 SDKs 會使用該角色提供的登入資料。如果您沒有對提供給 Amazon EKS 節點 IAM 角色的憑證限制存取權，則 Pod 仍然可以存取這些認證。如需詳細資訊，請參閱限制存取指派給工作節點的執行個體設定檔。

如果您的 Pod 無法如預期與服務互動，請完成以下步驟以確認所有項目都已正確設定。

a. 確認您的 Pod 使用支援透過 OpenID Connect Web 身分字符檔案擔任 IAM 角色的 AWS SDK 版本。如需詳細資訊，請參閱the section called “支援的 SDK”。

b. 確認部署使用服務帳戶。

```
kubectl describe deployment my-app | grep "Service Account"
```

範例輸出如下。

```
Service Account:  my-service-account
```

c. 如果您的 Pod 仍然無法存取服務，請檢閱為 Kubernetes 服務帳戶指派 IAM 角色中描述的步驟，以確認您的角色和服務帳戶設定正確。

設定服務帳戶 AWS 的安全字符服務端點

如果您使用 Kubernetes 服務帳戶搭配服務帳戶的 IAM 角色，則可以設定服務帳戶所使用的 AWS Security Token Service 端點類型。

AWS 建議使用區域 AWS STS 端點，而不是全域端點。這樣可以減少延遲、提供內建備援，並增加工作階段字符的有效性。 AWS Security Token Service 必須在執行 Pod 的區域中處於作用中 AWS 狀態。此外，如果 AWS 區域中的服務發生故障，您的應用程式必須具有不同 AWS 區域的內建備援。如需詳細資訊，請參閱《IAM 使用者指南AWS 》中的在 區域中管理 AWS STS。

- 現有的叢集。如果您沒有，則可以使用其中一個 開始使用 指南來建立一個。
- 叢集的現有 IAM OIDC 提供商。如需詳細資訊，請參閱the section called “IAM OIDC 提供商”。
- 現有的 Kubernetes 服務帳戶設定為與服務帳戶的 Amazon EKS IAM 功能搭配使用。

> **Note**
>
> 若要讓 IRSA 登入資料請求在無網際網路輸出的 VPC 中私有，請建立 AWS STS 介面端點 (`com.amazonaws.region-code.sts`) 並設定區域端點，如本主題所述。全域端點 (`sts.amazonaws.com`) 會略過 VPC 端點。若要私下連接叢集 OIDC 探索/JWKS 端點（用於提供者設定和字符驗證器），請建立單獨的 VPC 端點 (`com.amazonaws.region-code.oidc-eks`)。如需詳細資訊，請參閱the section called “使用 AWS PrivateLink 存取叢集 OIDC 端點”。

以下範例均使用由 Amazon VPC CNI 外掛程式所使用的 aws-node Kubernetes 服務帳戶。您可將*###*取代為您自己的服務帳戶、Pod、命名空間和其他資源。

1. 根據您要變更端點的服務帳戶，選取使用該服務帳戶的 Pod。決定 Pod AWS 執行的區域。將 *aws-node-6mfgv* 取代為您的 Pod 名稱，以及將 *kube-system* 取代為您 Pod 的命名空間。

   ```
   kubectl describe pod aws-node-6mfgv -n kube-system |grep Node:
   ```

   範例輸出如下。

```
ip-192-168-79-166.us-west-2/192.168.79.166
```

在先前的輸出中，Pod 正在 us-west-2 AWS Region 中的節點上執行。

2. 判斷 Pod 的服務帳戶所使用的端點類型。

```
kubectl describe pod aws-node-6mfgv -n kube-system |grep AWS_STS_REGIONAL_ENDPOINTS
```

範例輸出如下。

```
AWS_STS_REGIONAL_ENDPOINTS: regional
```

如果目前端點是全域範圍，則輸出中會傳回 `global`。如果未傳回任何輸出，則預設端點類型正在使用中且未被覆寫。

3. 您可以使用下列其中一個命令，將服務帳戶使用的端點類型從預設類型變更為不同的類型。將 *aws-node* 取代為您服務帳戶的名稱，以及將 *kube-system* 取代為服務帳戶的命名空間。
   - 如果您的預設或目前端點類型是全域範圍，且您想將其範圍變更為區域：

     ```
     kubectl annotate serviceaccount -n kube-system aws-node eks.amazonaws.com/sts-regional-endpoints=true
     ```

     如果您是使用服務帳戶的 IAM 角色來產生在 Pod 容器中執行的應用程式中的預簽署 S3 URL，則區域端點的 URL 格式會與以下範例類似：

     ```
     https://bucket.s3.us-west-2.amazonaws.com/path?...&X-Amz-Credential=your-access-key-id/date/us-west-2/s3/aws4_request&...
     ```

   - 如果您的預設或目前端點類型是區域範圍，且您想將其範圍變更為全域：

     ```
     kubectl annotate serviceaccount -n kube-system aws-node eks.amazonaws.com/sts-regional-endpoints=false
     ```

     如果您的應用程式明確向 AWS STS 全域端點提出請求，而且您未覆寫在 Amazon EKS 叢集中使用區域端點的預設行為，則請求將會失敗並顯示錯誤。如需詳細資訊，請參閱the section called “Pod 容器會接收到下列錯誤：An error occurred (SignatureDoesNotMatch) when calling the GetCallerIdentity operation: Credential should be scoped to a valid region”。

如果您是使用服務帳戶的 IAM 角色來產生在 Pod 容器中執行的應用程式中的預簽署 S3 URL，則全域端點的 URL 格式會與以下範例類似：

```
https://bucket.s3.amazonaws.com/path?...&X-Amz-Credential=your-access-key-id/date/
us-west-2/s3/aws4_request&...
```

如果您的自動化預期特定格式的預先簽章 URL，或者您的應用程式或使用預先簽章 URLs下游相依性對目標 AWS 區域具有預期，則請進行必要的變更，以使用適當的 AWS STS 端點。

4. 刪除並重新建立任何與服務帳戶相關聯的現有 Pod，以套用憑證環境變數。變動 Webhook 不會將這些變數套用到已在執行中的 Pod。您可使用所設定標註的 Pod 資訊，來取代 *Pod*、*kube-system* 和 *-l k8s-app=aws-node*。

```
kubectl delete Pods -n kube-system -l k8s-app=aws-node
```

5. 確認所有 Pod 已重新啟動。

```
kubectl get Pods -n kube-system -l k8s-app=aws-node
```

6. 檢視其中一個 Pod 的環境變數。確認 AWS_STS_REGIONAL_ENDPOINTS 值是您在上一個步驟中所設定的值。

```
kubectl describe pod aws-node-kzbtr -n kube-system |grep AWS_STS_REGIONAL_ENDPOINTS
```

範例輸出如下。

```
AWS_STS_REGIONAL_ENDPOINTS=regional
```

使用 IRSA 向其他帳戶進行身分驗證

您可以從另一個帳戶的叢集建立身分提供者，或使用鏈結的 AssumeRole 操作，設定跨帳戶 IAM 許可。在下列範例中，帳戶 A 擁有 Amazon EKS 叢集，該叢集支援服務帳戶的 IAM 角色。在該叢集上執行的 Pod 必須取得帳戶 B 的 IAM 許可。

- 選項 1 更簡單，但需要帳戶 B 為帳戶 A 的叢集建立和管理 OIDC 身分提供者。
- 選項 2 會在帳戶 A 中保留 OIDC 管理，但需要透過兩個AssumeRole呼叫鏈結角色。

選項 1：從另一個帳戶的叢集建立身分提供者

在此範例中，帳戶 A 將為帳戶 B 提供其叢集的 OpenID Connect (OIDC) 發行者 URL。帳戶 B 遵循為您的叢集建立 IAM OIDC 提供商和 the section called “指派 IAM 角色” 中的指示，使用來自帳戶 A 叢集的 OIDC 發行者 URL。然後，叢集管理員會註釋帳戶 A 叢集中的服務帳戶，以使用帳戶 B (*444455556666*) 中的角色。

```
apiVersion: v1
kind: ServiceAccount
metadata:
  annotations:
    eks.amazonaws.com/role-arn: arn:aws: iam::444455556666:role/account-b-role
```

選項 2：使用鏈結**AssumeRole**操作

在此方法中，每個帳戶都會建立 IAM 角色。帳戶 B 的角色信任帳戶 A，而帳戶 A 的角色使用 OIDC 聯合從叢集取得登入資料。然後，Pod 會使用 CLI AWS 設定檔將兩個角色鏈結在一起。

步驟 1：在帳戶 B 中建立目標角色

帳戶 B (*444455556666*) 會建立具有帳戶 A 叢集中 Pod 所需許可的 IAM 角色。帳戶 B 會將所需的許可政策連接至此角色，然後新增下列信任政策。

帳戶 B 角色的信任政策 — 此政策允許帳戶 A 的特定 IRSA 角色擔任此角色。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:root"
      },
      "Action": "sts:AssumeRole",
      "Condition": {}
    }
  ]
}
```

> **Important**
>
> 如需最低權限，請將 `Principal` ARN 取代為帳戶 A 中的特定角色 ARN，而不是使用帳戶根 (`arn:aws:iam::111122223333:root`)。使用帳戶根允許帳戶 A 中的任何 IAM 主體擔任此角色。

步驟 2：在帳戶 A 中建立 IRSA 角色

帳戶 A (*111122223333*) 會建立具有信任政策的角色，該政策會從使用叢集 OIDC 發行者地址建立的身分提供者取得憑證。

帳戶 A 角色的信任政策 (OIDC 聯合） — 此政策允許 EKS 叢集的 OIDC 供應商為此角色發出憑證。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::111122223333:oidc-provider/oidc.eks.us-
east-1.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "oidc.eks.us-east-1.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE:aud":
 "sts.amazonaws.com"
        }
      }
    }
  ]
}
```

> **Important**
>
> 針對最低權限，請新增`sub`宣告`StringEquals`的條件，將此角色限制為特定的 Kubernetes 服務帳戶。如果沒有`sub`條件，叢集中的任何服務帳戶都可以擔任此角色。`sub` 值使用 格式 `system:serviceaccount:`*NAMESPACE*`:`*SERVICE_ACCOUNT_NAME* 。例如，若要限制在`default`命名空間`my-service-account`中名為 的服務帳戶：

```
"oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE:sub":
 "system:serviceaccount:default:my-service-account"
```

步驟 3：將 AssumeRole 許可連接至帳戶 A 的角色

帳戶 A 會將許可政策連接至步驟 2 中建立的角色。此政策允許角色擔任帳戶 B 的角色。

帳戶 A 角色的許可政策 — 此政策`sts:AssumeRole`授予帳戶 B 的目標角色。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Resource": "arn:aws:iam::444455556666:role/account-b-role"
        }
    ]
}
```

步驟 4：將 Pod 設定為鏈結角色

Pod 擔任帳戶 B 角色的應用程式碼使用兩個描述檔：`account_b_role` 和 `account_a_role`。`account_b_role` 描述檔使用 `account_a_role` 描述檔作為其來源。對於 AWS CLI，`~/.aws/config`檔案類似於以下內容。

```
[profile account_b_role]
source_profile = account_a_role
role_arn=arn:aws: iam::444455556666:role/account-b-role

[profile account_a_role]
web_identity_token_file = /var/run/secrets/eks.amazonaws.com/serviceaccount/token
role_arn=arn:aws: iam::111122223333:role/account-a-role
```

若要指定其他 AWS SDKs的鏈結設定檔，請參閱您正在使用的 SDK 文件。如需詳細資訊，請參閱要建置的工具 AWS。

搭配 AWS SDK 使用 IRSA

使用憑證

若要針對服務帳戶 (IRSA) 使用來自 IAM 角色的登入資料，您的程式碼可以使用任何 AWS SDK 來透過 SDK 為 AWS 服務建立用戶端，並根據預設，軟體開發套件會在位置鏈中搜尋要使用的 AWS Identity and Access Management 登入資料。如果您在建立用戶端或初始化 SDK 時未指定登入資料提供者，則會使用服務帳戶登入資料的 IAM 角色。

這樣是有效的，因為服務帳戶的 IAM 角色已新增為預設憑證鏈中的一個步驟。如果您的工作負載目前使用登入資料鏈中較早的登入資料，即使您為相同工作負載的服務帳戶設定 IAM 角色，這些登入資料仍會繼續使用。

軟體開發套件會使用 `AssumeRoleWithWebIdentity`動作，自動交換服務帳戶 OIDC 字符，以取得來自 AWS Security Token Service 的臨時憑證。Amazon EKS 和此 SDK 動作會繼續輪換臨時憑證，方法是在臨時憑證到期前進行更新。

將 IAM 角色用於服務帳戶時，Pod 中的容器必須使用支援透過 OpenID Connect Web 身分字符檔案擔任 IAM 角色的 AWS SDK 版本。請確定您使用 SDK 的下列版本 或更新版本 AWS ：

- Java (第 2 版) – 2.10.11
- Java – 1.12.782
- AWS 適用於 Go 的 SDK v1 – 1.23.13
- AWS SDK for Go v2 – 所有版本支援
- Python (Boto3) – 1.9.220
- Python (botocore) – 1.12.200
- AWS CLI – 1.16.232
- 節點：2.525.0 和 3.27.0
- Ruby – 3.58.0
- C++ – 1.7.174
- .NET：3.3.659.1 – 您也必須包含 `AWSSDK.SecurityToken`。
- PHP – 3.110.7

許多熱門的 Kubernetes 附加元件，例如 Cluster Autoscaler、Route Internet traffic with AWS Load Balancer Controller 和 Amazon VPC CNI plugin for Kubernetes 支援服務帳戶的 IAM 角色。

為了確保您使用支援的 SDK，在建立您的容器時，請遵循在 AWS上建立的工具中您偏好之 SDK 的安裝指示。

考量事項

Java

使用 Java 時，您必須在類別路徑上包含 `sts` 模組。如需詳細資訊，請參閱 Java SDK 文件中的 WebIdentityTokenFileCredentialsProvider。

擷取簽署金鑰以驗證 OIDC 權杖

Kubernetes 會為每個 Kubernetes Service 帳戶發出一個 `ProjectedServiceAccountToken`。此權杖是 OIDC 權杖，同時也是一種 JSON Web 權杖 (JWT)。Amazon EKS 為每個叢集託管公有 OIDC 端點，其中包含權杖的簽署金鑰，讓外部系統可以對其進行驗證。

若要驗證 `ProjectedServiceAccountToken`，您需要擷取 OIDC 公有簽署金鑰，也稱為 JSON Web Key Set (JWKS)。在應用程式中使用這些金鑰，以驗證權杖。例如，您可以使用 PyJWT Python 程式庫，以利用這些金鑰來驗證權杖。如需 `ProjectedServiceAccountToken` 的詳細資訊，請參閱 the section called “IAM, Kubernetes 和 OpenID Connect (OIDC) 背景資訊”。

> **Note**
>
> 如果您的驗證器在沒有傳出網際網路存取的情況下在 VPC 中執行，您可以透過為叢集 OIDC 端點 () 建立 a AWS PrivateLink 介面端點，私下擷取探索文件和 JWKS`com.amazonaws.region-code.oidc-eks`。如需詳細資訊，請參閱the section called “使用 AWS PrivateLink 存取叢集 OIDC 端點”。

先決條件

- 叢集的現有 AWS Identity and Access Management (IAM) OpenID Connect (OIDC) 提供者。若要判定您是否已經擁有一個，或是要建立一個，請參閱 the section called “IAM OIDC 提供商”。
- AWS CLI — 用於使用 AWS 服務的命令列工具，包括 Amazon EKS。如需詳細資訊，請參閱《 AWS 命令列界面使用者指南》中的安裝。安裝 CLI AWS 之後，建議您也進行設定。如需詳細資訊，請參閱《 AWS 命令列界面使用者指南》中的具有 aws 設定的快速組態。

程序

1. 使用 CLI 擷取 Amazon EKS 叢集的 AWS OIDC URL。

```
$ aws eks describe-cluster --name my-cluster --query 'cluster.identity.oidc.issuer'
```

```
"https://oidc.eks.us-west-2.amazonaws.com/id/8EBDXXXX00BAE"
```

2. 使用 curl 或類似工具檢索公有簽署金鑰。結果為 JSON Web Key Set (JWKS)。

> **Important**
>
> Amazon EKS 會限流對 OIDC 端點的呼叫。您應該快取公有簽署金鑰。遵守回應中包含的 `cache-control` 標題。

> **Important**
>
> Amazon EKS 每七天輪換一次 OIDC 簽署金鑰。

```
$ curl https://oidc.eks.us-west-2.amazonaws.com/id/8EBDXXXX00BAE/keys
{"keys":
[{"kty":"RSA","kid":"2284XXXX4a40","use":"sig","alg":"RS256","n":"wklbXXXXMVfQ","e":"AQAB"}]
```

了解 EKS Pod Identity 如何授予 Pod 對 AWS 服務的存取權

Pod 容器中的應用程式可以使用 AWS SDK 或 AWS CLI，使用 AWS Identity and Access Management (IAM) 許可向 AWS 服務提出 API 請求。應用程式必須使用 AWS 登入資料簽署其 AWS API 請求。

EKS Pod 身分識別提供管理應用程式憑證的功能，類似 Amazon EC2 執行個體設定檔將憑證提供給 Amazon EC2 執行個體的方式。您可以將 IAM 角色與 Kubernetes 服務帳戶建立關聯，並將 Pod 設定為使用服務帳戶，而不是建立 AWS 登入資料並將其分發至容器或使用 Amazon EC2 執行個體的角色。

每個 EKS Pod 身分識別關聯會將角色對應至指定叢集內命名空間中的服務帳戶。如果您在多個叢集中具有相同的應用程式，則可以在每個叢集中建立相同的關聯，而不必修改角色的信任政策。

如果 Pod 使用具有關聯的服務帳戶，則 Amazon EKS 會在 Pod 的容器中設定環境變數。環境變數設定 AWS SDKs，包括 AWS CLI，以使用 EKS Pod Identity 憑證。

EKS Pod 身分識別的優勢

EKS Pod 身分識別提供下列優勢：

- 最低權限：您可以將 IAM 許可範圍限定為服務帳戶，只有使用該服務帳戶的 Pod 才能存取這些許可。有此功能也就不需要第三方解決方案，例如 `kiam` 或 `kube2iam`。
- 憑證隔離：若 Amazon EC2 執行個體中繼資料服務 (IMDS) 存取權受限，Pod 的容器僅可擷取與容器所用服務帳戶關聯的 IAM 角色憑證。容器一律無法存取其他 Pod 中其他容器使用的憑證。若 IMDS 不受限制，Pod 的容器亦可存取 Amazon EKS 節點 IAM 角色，且容器或許能存取相同節點上其他 Pod 的 IAM 角色的憑證。如需詳細資訊，請參閱限制存取指派給工作節點的執行個體設定檔。

> **Note**
>
> 使用 設定的 Pod `hostNetwork: true`一律具有 IMDS 存取權，但啟用時， AWS SDKs 和 CLI 將使用 Pod Identity 登入資料。

- 可稽核性 – 可透過 AWS CloudTrail 存取和事件記錄，以協助促進回溯性稽核。

> **Important**
>
> 容器並非安全邊界，而且使用 Pod 身分識別亦不會對此有所變更。若 Pod 指派到相同的節點，將會分享核心及可能的其他資源，具體取決於 Pod 組態。儘管執行於單獨節點上的 Pod 將會在運算層隔離，但有些節點應用程式在 Kubernetes API 中擁有超出個別執行個體範圍的額外許可。部分範例包括 `kubelet`、`kube-proxy`、CSI 儲存驅動程式，或者您本人的 Kubernetes 應用程式。

相較於 the section called “具有 IRSA 的憑證”，EKS Pod 身分識別是比較簡單的方法，因為此方法不使用 OIDC 身分識別提供者。EKS Pod 身分識別具有下列增強功能：

- 獨立運作 – 在許多組織中，建立 OIDC 身分識別提供者和管理 Kubernetes 叢集是不同團隊的責任。EKS Pod 身分識別具有完整的職責分離，其中 EKS Pod 身分識別關聯的所有組態都在 Amazon EKS 中完成，而 IAM 許可的所有組態則都在 IAM 中完成。
- 可重複使用性 – EKS Pod 身分識別會針對服務帳戶的 IAM 角色使用的每個叢集使用單一 IAM 主體，而不是使用不同的主體。您的 IAM 管理員會將下列主體新增至任何角色的信任政策，以便供 EKS Pod 身分識別使用。

```
            "Principal": {
                "Service": "pods.eks.amazonaws.com"
            }
```

- 可擴展性 — EKS Pod Identity 中的 EKS Auth 服務會擔任每組臨時登入資料，而不是您在每個 Pod 中執行的每個 AWS SDK。然後，在每個節點上執行的 Amazon EKS Pod 身分識別代理程式會向 SDK 發出憑證。因此，每個節點的負載會減少為一次，而且不會在每個 Pod 中重複。如需該程序的詳細資訊，請參閱 the section called “運作方式”。

如需比較兩種替代方案的詳細資訊，請參閱 the section called “工作負載存取 AWS ”。

設定 EKS Pod 身分識別的概觀

完成下列程序以開啟 EKS Pod 身分識別：

1. the section called “設定代理程式”：您對每個叢集只完成此程序一次。若叢集已啟用 EKS 自動模式，則無須完成此步驟。
2. the section called “指派 IAM 角色”：對您希望應用程式擁有的每種非重複可組合完成此程序。
3. the section called “Pod 服務帳戶” — 為每個需要存取 AWS 服務的 Pod 完成此程序。
4. the section called “支援的 SDK” — 確認工作負載使用支援版本的 AWS SDK，以及工作負載使用預設登入資料鏈。

限制

- 每個叢集可擁有最多 5，000 個 EKS Pod 身分識別關聯，將 IAM 角色映射至 Kubernetes 服務帳戶。

考量事項

- IAM 角色關聯：叢集中的每個 Kubernetes 服務帳戶都可以與叢集相同 AWS 帳戶中的一個 IAM 角色建立關聯。如需變更角色，編輯 EKS Pod 身分識別關聯。針對跨帳戶存取權，使用 IAM 角色向角色委派存取權。若要進一步了解，請參閱《IAM 使用者指南》中的使用 IAM 角色在 AWS 帳戶之間委派存取權。
- EKS Pod 身分識別代理程式：需要 Pod 身分識別代理程式以使用 EKS Pod 身分識別。代理程式以叢集節點上的 Kubernetes `DaemonSet` 執行，僅向相同節點上的 Pod 提供憑證。它使用節點的 `hostNetwork`、佔用連接埠`80`和連結本機地址 `2703` (`169.254.170.23`適用於 IPv4、`[fd00:ec2::23]`適用於 IPv6)。若 IPv6 在叢集中停用，則為 Pod 身分識別代理程式停用 IPv6。如需了解詳細資訊，請參閱在 EKS Pod 身分識別代理程式中停用 IPv6。
- 最終一致性：EKS Pod 身分識別關聯最終會一致，並在 API 呼叫之後可能會延遲幾秒。請避免在關鍵、高可用性程式碼路徑中建立或更新關聯。取而代之的是，在單獨、不太頻繁的初始化或設定例程中執行這些動作。如需了解詳細資訊，請參閱 EKS 最佳實務指南中的每個 Pod 的安全群組。
- 代理與安全群組考量：針對使用代理的 Pod，新增 `169.254.170.23` (IPv4) 與 `[fd00:ec2::23]` (IPv6) 至 `no_proxy/NO_PROXY` 環境變數，以免向 EKS Pod 身分識別代理程式發出的請求失敗。如果搭配 AWS VPC CNI 使用 Pod 的安全群組，請將 `ENABLE_POD_ENI`旗標設為「true」，並將 `POD_SECURITY_GROUP_ENFORCING_MODE`旗標設為「standard」。如需了解詳細資訊，請參閱將安全群組指派給個別 Pod。

EKS Pod 身分識別叢集版本

如需使用 EKS Pod 身分識別，叢集的平台版本必須與下表所列版本相同或更新，或是晚於表格中列出版本的 Kubernetes 版本。如需尋找適用於 Kubernetes 版本的 Amazon EKS Pod 身分識別代理程式建議版本，請參閱 the section called “確認相容性”。

Kubernetes 版本	平台版本
Kubernetes 版本未列出	所有平台版本支援
1.28	eks.4

EKS Pod 身分識別限制

EKS Pod 身分識別適用於下列情況：

- 上一個主題 the section called “EKS Pod 身分識別叢集版本” 中列出的 Amazon EKS 叢集版本。
- 叢集中為 Linux Amazon EC2 執行個體的工作節點。

EKS Pod 身分識別不適用於下列情況：

- AWS Outpost。
- Amazon EKS Anywhere。
- 您在 Amazon EC2 上建立和執行的 Kubernetes 叢集。EKS Pod 身分識別元件僅適用於 Amazon EKS。

您無法將 EKS Pod 身分識別與下列項目搭配使用：

- 在 Linux Amazon EC2 執行個體以外的任何位置執行的 Pod。不支援在 AWS Fargate (Fargate) 上執行的 Linux 和 Windows Pod。不支援在 Windows Amazon EC2 執行個體上執行的 Pod。

了解 EKS Pod 身分識別的運作方式

Amazon EKS Pod 身分識別關聯提供管理應用程式憑證的功能，類似 Amazon EC2 執行個體設定檔將憑證提供給 Amazon EC2 執行個體的方式。

Amazon EKS Pod 身分識別透過其他 EKS 驗證 API 和在每個節點上執行的代理程式 Pod，為您的工作負載提供憑證。

在您的附加元件中，例如 Amazon EKS 附加元件和自我管理控制器、運算子和其他附加元件，作者需要更新其軟體，才能使用最新的 AWS SDKs。如需 EKS Pod 身分識別與 Amazon EKS 產生的附加元件之間的相容性清單，請參閱上節 the section called “EKS Pod 身分識別限制”。

在程式碼中使用 EKS Pod 身分識別

在程式碼中，您可以使用 AWS SDKs來存取 AWS 服務。您可以編寫程式碼，以使用 SDK 為 AWS 服務建立用戶端，根據預設， SDK 會在位置鏈中搜尋要使用的 AWS Identity and Access Management 憑證。找到有效的憑證後，系統就會停止搜尋。如需使用的預設位置的詳細資訊，請參閱《 AWS SDKs和工具參考指南》中的登入資料提供者鏈結。

EKS Pod 身分識別已新增至容器憑證提供者，系統會在預設憑證鏈中的某個步驟搜尋此提供者。如果您的工作負載目前使用憑證鏈中較早的憑證，那麼即使您為相同工作負載設定 EKS Pod 身分識別關聯，系統仍會繼續使用這些憑證。如此一來，您就可以先建立關聯，然後再移除舊憑證，以安全地從其他類型的憑證遷移。

容器憑證提供者會從每個節點上執行的代理程式提供臨時憑證。在 Amazon EKS 中，代理程式為 Amazon EKS Pod 身分識別代理程式，而在 Amazon Elastic Container Service 上，代理程式則為 `amazon-ecs-agent`。SDK 使用環境變數來尋找要連線的代理程式。

相反地，服務帳戶的 IAM 角色提供 Web 身分字符， AWS 開發套件必須使用 與 AWS Security Token Service 交換`AssumeRoleWithWebIdentity`。

EKS Pod 身分識別代理程式如何搭配 Pod 使用

1. 當 Amazon EKS 啟動使用具有 EKS Pod 身分識別關聯之服務帳戶的全新 Pod 時，叢集會將下列內容新增至 Pod 資訊清單：

```
    env:
    - name: AWS_CONTAINER_AUTHORIZATION_TOKEN_FILE
      value: "/var/run/secrets/pods.eks.amazonaws.com/serviceaccount/eks-pod-
identity-token"
    - name: AWS_CONTAINER_CREDENTIALS_FULL_URI
      value: "http://169.254.170.23/v1/credentials"
    volumeMounts:
    - mountPath: "/var/run/secrets/pods.eks.amazonaws.com/serviceaccount/"
      name: eks-pod-identity-token
  volumes:
```

```
  - name: eks-pod-identity-token
    projected:
      defaultMode: 420
      sources:
      - serviceAccountToken:
          audience: pods.eks.amazonaws.com
          expirationSeconds: 86400 # 24 hours
          path: eks-pod-identity-token
```

2. Kubernetes 會選取要在哪個節點上執行 Pod。然後，節點上的 Amazon EKS Pod 身分識別代理程式會使用 AssumeRoleForPodIdentity 動作以從 EKS 驗證 API 擷取臨時憑證。
3. EKS Pod Identity Agent 可讓您在容器內執行 AWS SDKs 使用這些登入資料。
4. 您可以在應用程式中使用 SDK，無需指定憑證提供者來使用預設憑證鏈。或者，您可以指定容器憑證提供者。如需使用的預設位置的詳細資訊，請參閱《 AWS SDKs和工具參考指南》中的登入資料提供者鏈結。
5. SDK 會使用環境變數來連線至 EKS Pod 身分識別代理程式，並擷取憑證。

> **Note**
>
> 如果您的工作負載目前使用憑證鏈中較早的憑證，那麼即使您為相同工作負載設定 EKS Pod 身分識別關聯，系統仍會繼續使用這些憑證。

設定 Amazon EKS Pod 身分識別代理程式

Amazon EKS Pod 身分識別關聯提供管理應用程式憑證的功能，類似 Amazon EC2 執行個體設定檔將憑證提供給 Amazon EC2 執行個體的方式。

Amazon EKS Pod 身分識別透過其他 EKS 驗證 API 和在每個節點上執行的代理程式 Pod，為您的工作負載提供憑證。

> **Tip**
>
> 您無須在 EKS 自動模式叢集上安裝 EKS Pod 身分識別代理程式。EKS 自動模式中已內建此功能。

考量事項

- 依預設，EKS Pod 身分識別代理程式將預先安裝在 EKS 自動模式叢集上。如需詳細資訊，請參閱 EKS 自動模式。
- 依預設，EKS Pod 身分識別代理程式將接聽 IPv4 與 IPv6 位址，以便 Pod 請求憑證。代理程式針對 IPv4 使用迴路 (localhost) IP 位址 169.254.170.23，針對 IPv6 使用 localhost IP 位址 [fd00:ec2::23]。
- 若您停用 IPv6 位址，或者以其他方式阻止 localhost IPv6 IP 位址，將無法啟動代理程式。如需在不能使用 IPv6 的節點上啟動代理程式，遵循 the section called “停用 IPv6” 中的步驟來停用 IPv6 組態。

建立 Amazon EKS Pod 身分識別代理程式

代理程式先決條件

> **Important**
>
> EKS Pod Identity Agent 執行的節點必須能夠存取 EKS 驗證 API。如果您為節點使用私有子網路，則必須為 EKS 身分驗證 API 設定一個 AWS PrivateLink 介面端點。如需詳細資訊，請參閱the section called “ AWS PrivateLink”。

- 現有 Amazon EKS 叢集。若要部署叢集，請參閱 開始使用。叢集版本和平台版本必須與 EKS Pod 身分識別叢集版本中列出的版本相同或為更新版本。
- 節點角色具有代理程式在 EKS 驗證 API 中執行 AssumeRoleForPodIdentity 動作的許可。您可使用 AWS 受管政策：AmazonEKSWorkerNodePolicy，或者新增與下列所示類似的自訂政策：

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "eks-auth:AssumeRoleForPodIdentity"
            ],
            "Resource": "*"
        }
    ]
```

```
}
```

此動作可能會受到標籤的限制，以限制使用代理程式的 Pod 可以擔任哪些角色。

- 這些節點可以從 Amazon ECR 連上和下載映像。適用於附加元件的容器映像位於檢視適用於 Amazon EKS 附加元件的 Amazon 容器映像登錄檔中列示的登錄檔中。

  請注意，您可以在 的選用組態設定中，以及在 CLI `imagePullSecrets` `--configuration-values`的 中 AWS 管理主控台，變更映像位置並提供 EKS AWS 附加元件。

使用 AWS 主控台設定代理程式

1. 開啟 Amazon EKS 主控台。
2. 在左側導覽窗格中，選取叢集，然後選取您要為其設定 EKS Pod 身分識別代理程式附加元件之叢集的名稱。
3. 選擇附加元件索引標籤。
4. 選擇取得更多附加元件。
5. 選取 EKS Pod 身分識別代理程式之附加元件方塊右上方的方塊，然後選擇下一步。
6. 在設定選取的附加元件設定頁面上，選取版本下拉式清單中的任何版本。
7. (選用) 展開選用組態設定以輸入其他組態。例如，您可以提供替代容器映像位置和 `ImagePullSecrets`。具有已接受索引鍵的 JSON 結構描述會顯示在附加元件組態結構描述中。

   在組態值中輸入組態金鑰和值。
8. 選擇下一步。
9. 確認 EKS Pod 身分識別代理程式 Pod 正在您的叢集上執行。

```
kubectl get pods -n kube-system | grep 'eks-pod-identity-agent'
```

範例輸出如下。

```
eks-pod-identity-agent-gmqp7                                          1/1     Running
   1 (24h ago)   24h
eks-pod-identity-agent-prnsh                                          1/1     Running
   1 (24h ago)   24h
```

您現在可以在叢集中使用 EKS Pod 身分識別關聯。如需詳細資訊，請參閱the section called “指派 IAM 角色”。

使用 CLI AWS 設定代理程式

1. 執行下列 AWS CLI 命令。使用您叢集的名稱取代 `my-cluster`。

```
aws eks create-addon --cluster-name my-cluster --addon-name eks-pod-identity-agent --
addon-version v1.0.0-eksbuild.1
```

> **Note**
>
> EKS Pod 身分識別代理程式不會針對服務帳戶的 IAM 角色使用 `service-account-role-arn`。您必須為 EKS Pod 身分識別代理程式提供節點角色的許可。

2. 確認 EKS Pod 身分識別代理程式 Pod 正在您的叢集上執行。

```
kubectl get pods -n kube-system | grep 'eks-pod-identity-agent'
```

範例輸出如下。

```
eks-pod-identity-agent-gmqp7                                          1/1     Running
   1 (24h ago)   24h
eks-pod-identity-agent-prnsh                                          1/1     Running
   1 (24h ago)   24h
```

您現在可以在叢集中使用 EKS Pod 身分識別關聯。如需詳細資訊，請參閱the section called “指派 IAM 角色”。

指派 IAM 角色至 Kubernetes 服務帳戶

本主題說明如何設定 Kubernetes 服務帳戶，以使用 EKS Pod Identity 擔任 AWS Identity and Access Management (IAM) 角色。然後，任何設定為使用服務帳戶的 Pod 都可以存取該角色有權存取的任何 AWS 服務。

若要建立 EKS Pod 身分關聯，只有一個步驟；您可以透過 AWS 管理主控台、 AWS CLI、 AWS SDKs AWS CloudFormation 和其他工具在 EKS 中建立關聯。在任何 Kubernetes 物件中，叢集內部的關聯沒有任何資料或中繼資料，也不會將任何註解新增至服務帳戶。

先決條件

- 現有的叢集。如果您沒有，則可以按照其中一個 開始使用 指南來建立。

- 建立關聯的 IAM 主體必須具有 `iam:PassRole`。
- 在您的裝置或 AWS CloudShell 上安裝和設定的最新版本 AWS CLI。您可以使用 `aws --version | cut -d / -f2 | cut -d ' ' -f1` 來檢查您的目前版本。適用於 macOS 的 yum、`apt-get`或 Homebrew 等套件管理員通常是最新版本 CLI AWS 後面的幾個版本。若要安裝最新版本，請參閱《 AWS 命令列界面使用者指南》中的使用 aws 設定安裝 和快速組態。 https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-quickstart.html#cli-configure-quickstart-config安裝在 AWS CloudShell 中的 AWS CLI 版本也可能是最新版本後面的幾個版本。若要更新它，請參閱《 AWS CloudShell AWS 使用者指南》中的將 CLI 安裝到您的主目錄。
- `kubectl` 命令列工具安裝在您的裝置或 AWS CloudShell 上。版本可以與您的叢集 Kubernetes 版本相同，或是為最多比該版本更舊一版或更新一版的次要版本。例如，如果您的叢集版本為 `1.29`，則可以搭配使用 `kubectl` `1.28`、`1.29` 或 `1.30` 版。若要安裝或升級 `kubectl`，請參閱 the section called “設定 kubectl 和 eksctl”。
- 包含叢集組態的現有 `kubectl config` 檔案。若要建立 `kubectl config` 檔案，請參閱the section called “透過 kubectl 存取叢集”。

建立 Pod Identity 關聯AWS （主控台）

1. 開啟 Amazon EKS 主控台。
2. 在左側導覽窗格中，選取叢集，然後選取您要為其設定 EKS Pod 身分識別代理程式附加元件之叢集的名稱。
3. 選擇存取索引標籤。
4. 在 Pod 身分識別關聯中，選擇建立。
5. 對於 IAM 角色，選取具有您希望工作負載擁有許可的 IAM 角色。

> **Note**
>
> 此清單僅包含具有下列信任政策的角色，允許 EKS Pod 身分識別使用這些角色。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowEksAuthToAssumeRoleForPodIdentity",
            "Effect": "Allow",
            "Principal": {
```

```
                "Service": "pods.eks.amazonaws.com"
            },
            "Action": [
                "sts:AssumeRole",
                "sts:TagSession"
            ]
        }
    ]
}
```

`sts:AssumeRole`：EKS Pod 身分識別會先使用 `AssumeRole` 來擔任 IAM 角色，然後將臨時憑證傳至您的 Pod。

`sts:TagSession` — EKS Pod Identity 使用 `TagSession` 在 STS AWS 的請求中包含工作階段標籤。

您可以在信任政策的條件索引鍵中使用這些標籤，用來限制哪些服務帳戶、命名空間和叢集可以使用此角色。

如需 Amazon EKS 條件金鑰的清單，請參閱服務授權參考中的 Amazon Elastic Kubernetes Service 定義的條件。若要了解您可以搭配哪些動作和資源使用條件索引鍵，請參閱 Amazon Elastic Kubernetes Service 定義的動作。

6. 針對 Kubernetes 命名空間，選取包含服務帳戶和工作負載的 Kubernetes 命名空間。或者，您可以使用名稱 (不存在於叢集中) 指定命名空間。
7. 針對 Kubernetes 服務帳戶，選取要使用的 Kubernetes 服務帳戶。Kubernetes 工作負載的清單檔案必須指定此服務帳戶。或者，您可以使用名稱 (不存在於叢集中) 指定服務帳戶。
8. （選用）選取停用工作階段標籤，以停用 Pod Identity 在擔任角色時自動新增的預設工作階段標籤。
9. （選用）切換設定工作階段政策以設定 IAM 政策，以套用超出連接至 IAM 角色之 IAM 政策中定義之許可的其他限制至此 Pod Identity 關聯。

> **Note**
>
> 只有在核取停用工作階段標籤設定時，才能套用工作階段政策。

10. (選用) 對於標籤，請選擇新增標籤，以在鍵值對中新增中繼資料。這些標籤會套用至關聯，可用於 IAM 政策。

    您可以多次重複此步驟以新增多個標籤。

11選擇建立。

建立 Pod 身分關聯 (AWS CLI)

1. 如果您要將現有 IAM 政策與 IAM 角色建立關聯，請跳到下一步驟。

   建立 IAM 政策。您可以建立自己的政策，或複製已授予部分所需許可的 AWS 受管政策，並根據您的特定需求進行自訂。如需詳細資訊，請參閱《IAM 使用者指南》中的建立 IAM 政策。

   a. 建立檔案，其中包含您希望 Pod 存取 AWS 之服務的許可。如需所有 AWS 服務的所有動作清單，請參閱服務授權參考。

      您可以執行以下命令來建立允許唯讀存取 Amazon S3 儲存貯體的範例政策檔案。您可以選擇性地將組態資訊或引導指令碼存放在此儲存貯體中，而 Pod 中的容器可從儲存貯體讀取檔案，並載入至您的應用程式。如果您要建立此範例政策，請將以下內容複製到您的裝置。將 *my-pod-secrets-bucket* 取代為您的儲存貯體名稱並執行命令。

      ```
      {
          "Version":"2012-10-17",
          "Statement": [
              {
                  "Effect": "Allow",
                  "Action": "s3:GetObject",
                  "Resource": "arn:aws:s3:::my-pod-secrets-bucket"
              }
          ]
      }
      ```

   b. 建立 IAM 政策。

      ```
      aws iam create-policy --policy-name my-policy --policy-document file://my-
      policy.json
      ```

2. 建立 IAM 角色並與 Kubernetes 服務帳戶建立關聯。

   a. 如果您有要擔任 IAM 角色的現有 Kubernetes 服務帳戶，則可略過此步驟。

      建立 Kubernetes 服務帳戶。將以下內容複製到您的裝置。視需要將 *my-service-account* 取代為您要使用的名稱，將 *default* 取代為其他命名空間。如果您變更 *default*，命名空間必須已經存在。

      ```
      cat >my-service-account.yaml <<EOF
      ```

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: my-service-account
  namespace: default
EOF
kubectl apply -f my-service-account.yaml
```

執行下列命令。

```
kubectl apply -f my-service-account.yaml
```

b. 執行下列命令以建立 IAM 角色的信任政策檔案。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowEksAuthToAssumeRoleForPodIdentity",
            "Effect": "Allow",
            "Principal": {
                "Service": "pods.eks.amazonaws.com"
            },
            "Action": [
                "sts:AssumeRole",
                "sts:TagSession"
            ]
        }
    ]
}
```

c. 建立角色。將 *my-role* 取代為您的 IAM 角色名稱，並將 *my-role-description* 取代為您角色的描述。

```
aws iam create-role --role-name my-role --assume-role-policy-document file://
trust-relationship.json --description "my-role-description"
```

d. 將 IAM 政策連接至您的角色。將 *my-role* 取代為您的 IAM 角色名稱，將 *my-policy* 取代為您建立的現有政策名稱。

```
aws iam attach-role-policy --role-name my-role --policy-arn=arn:aws:
 iam::111122223333:policy/my-policy
```

> **Note**
>
> 與服務帳戶的 IAM 角色不同，EKS Pod 身分識別不會在服務帳戶上使用註釋。

e. 執行下列命令以建立關聯。視需要使用叢集名稱取代 `my-cluster`，使用您要使用的名稱取代 *`my-service-account`*，使用其他命名空間取代 *`default`*。

```
aws eks create-pod-identity-association --cluster-name my-cluster --role-arn
 arn:aws: iam::111122223333:role/my-role --namespace default --service-account my-
service-account
```

範例輸出如下。

```
{
    "association": {
        "clusterName": "my-cluster",
        "namespace": "default",
        "serviceAccount": "my-service-account",
        "roleArn": "arn:aws: iam::111122223333:role/my-role",
        "associationArn": "arn:aws::111122223333:podidentityassociation/my-
cluster/a-abcdefghijklmnop1",
        "associationId": "a-abcdefghijklmnop1",
        "tags": {},
        "createdAt": 1700862734.922,
        "modifiedAt": 1700862734.922
    }
}
```

> **Note**
>
> 您可以使用名稱 (不存在於叢集中) 指定命名空間和服務帳戶。您必須建立命名空間、服務帳戶以及使用服務帳戶進行 EKS Pod 身分識別關聯才能運作的工作負載。

確認組態

1. 確認 IAM 角色的信任政策設定正確。

```
aws iam get-role --role-name my-role --query Role.AssumeRolePolicyDocument
```

範例輸出如下。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "pods.eks.amazonaws.com"
            },
            "Action": [
                "sts:AssumeRole",
                "sts:TagSession"
            ]
        }
    ]
}
```

2. 確認您在上一步連接至角色的政策已連接至該角色。

```
aws iam list-attached-role-policies --role-name my-role --query
 'AttachedPolicies[].PolicyArn' --output text
```

範例輸出如下。

```
 arn:aws: iam::111122223333:policy/my-policy
```

3. 設定變數以存放您要使用之政策的 Amazon Resource Name (ARN)。將 *my-policy* 取代為您要確認許可的政策名稱。

```
export policy_arn=arn:aws: iam::111122223333:policy/my-policy
```

4. 檢視預設政策版本。

```
aws iam get-policy --policy-arn $policy_arn
```

範例輸出如下。

```
{
    "Policy": {
```

```
        "PolicyName": "my-policy",
        "PolicyId": "EXAMPLEBIOWGLDEXAMPLE",
        "Arn": "arn:aws: iam::111122223333:policy/my-policy",
        "Path": "/",
        "DefaultVersionId": "v1",
        [...]
    }
}
```

5. 檢視政策內容，以確定政策包含您的 Pod 需要的所有許可。視需要將以下命令中的 *1* 取代為上一個輸出中傳回的版本。

```
aws iam get-policy-version --policy-arn $policy_arn --version-id v1
```

範例輸出如下。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::my-pod-secrets-bucket"
        }
    ]
}
```

如果您在上一步建立了範例政策，則輸出結果相同。如果您建立了不同的政策，則 *example* 內容有所不同。

後續步驟

the section called “Pod 服務帳戶”

使用 EKS Pod 身分目標 IAM 角色存取 AWS 資源

在 Amazon Elastic Kubernetes Service (Amazon EKS) 上執行應用程式時，您可能需要存取存在於不同 AWS 帳戶中 AWS 的資源。本指南說明如何使用 EKS Pod Identity 設定跨帳戶存取，這可讓您的 Kubernetes Pod 使用目標角色存取其他 AWS 資源。

先決條件

開始之前，請確保您已完成下列步驟：

- 設定 Amazon EKS Pod 身分識別代理程式
- 建立 EKS Pod 身分識別角色

運作方式

Pod Identity 可讓 EKS 叢集中的應用程式透過稱為角色鏈結的程序跨帳戶存取 AWS 資源。

建立 Pod Identity 關聯時，您可以提供兩個 IAM 角色：與 EKS 叢集位於相同帳戶中的 EKS Pod Identity 角色，以及包含您想要存取 AWS 之資源（例如 S3 儲存貯體或 RDS 資料庫）帳戶中的目標 IAM 角色。由於 IAM PassRole 需求，EKS Pod 身分角色必須位於 EKS 叢集的帳戶中，而目標 IAM 角色可以位於任何 AWS 帳戶中。PassRole 可讓 AWS 實體將角色擔任委派給其他服務。EKS Pod Identity 使用 PassRole 將角色連線至 Kubernetes 服務帳戶，要求角色和傳遞該角色的身分都與 EKS 叢集位於相同的 AWS 帳戶中。當您的應用程式 Pod 需要存取 AWS 資源時，它會向 Pod Identity 請求登入資料。Pod 身分識別隨後可自動依序執行兩項角色擔任：首先擔任 EKS Pod 身分識別角色，然後使用這些憑證來擔任目標 IAM 角色。此程序為您的 Pod 提供暫時登入資料，該登入資料具有目標角色中定義的許可，允許安全存取其他 AWS 帳戶中的資源。

快取考量

由於採用快取機制，對現有 Pod 身分識別關聯中 IAM 角色更新，可能不會在 EKS 叢集上執行的 Pod 中立即生效。Pod 身分識別代理程式可依據擷取憑證時關聯的組態來快取 IAM 憑證。若關聯僅包括 EKS Pod 身分識別角色，而不含目標 IAM 角色，快取憑證則可持續 6 小時。若關聯包括 EKS Pod 身分識別角色 ARN 與目標 IAM 角色，快取憑證則可持續 59 分鐘。修改現有的關聯，例如更新 EKS Pod 身分角色 ARN、新增目標 IAM 角色，或更新關聯上的工作階段政策，不會重設現有的快取。因此，代理程式不會識別更新，直至快取憑證資料重新整理。如需加速套用變更，您可重新建立現有 Pod；否則，您將需要等候快取到期。

步驟 1：建立目標 IAM 角色並與之關聯

在此步驟中，您將建立並設定目標 IAM 角色，藉此來建立安全的信任鏈。為了示範，我們將建立新的目標 IAM 角色，以在兩個 AWS 帳戶之間建立信任鏈：EKS 叢集帳戶中的 EKS Pod 身分角色（例如 `eks-pod-identity-primary-role`) 取得在您的目標帳戶中擔任目標 IAM 角色（例如 `eks-pod-identity-aws-resources`) 的許可，以允許存取 Amazon S3 儲存貯體等 AWS 資源。 AWS

建立目標 IAM 角色

1. 開啟 Amazon IAM 主控台。
2. 在頂端導覽列中，確認您已登入帳戶，其中包含目標 IAM 角色 AWS 的資源 （ 例如 S3 儲存貯體或 DynamoDB 資料表 ）。
3. 在左側導覽窗格中，選擇 Roles (角色)。
4. 選擇建立角色按鈕，然後選擇「可信任實體類型」下的 AWS 帳戶。
5. 選擇另一個 AWS 帳戶，輸入 AWS 您的帳戶號碼 (EKS Pod 身分角色所在的帳戶 ），然後選擇下一步。
6. 新增您想要關聯角色的許可政策 (例如 AmazonS3FullAccess)，然後選擇下一步。
7. 輸入角色名稱 (例如 `MyCustomIAMTargetRole`)，然後選擇建立角色。

更新目標 IAM 角色信任政策

1. 建立角色之後，您將返回到角色清單。尋找並選取您在之前步驟中建立的角色政策 (例如 `MyCustomIAMTargetRole`)。
2. 選取信任關係標籤。
3. 選擇 編輯信任政策。
4. 在政策編輯器中，使用您的信任政策來取代預設 JSON。將角色名稱和 IAM 角色 ARN 111122223333中的預留位置值取代為託管 EKS 叢集 AWS 的帳戶 ID。此外，您亦可選擇在角色信任政策中使用 PrincipalTags，以便僅授權指定叢集與命名空間的特定服務帳戶來擔任目標角色。例如：

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:root"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ],
      "Condition": {
        "StringEquals": {
```

```
          "aws:RequestTag/eks-cluster-arn": "arn:aws:eks:us-
east-1:111122223333:cluster/example-cluster",
          "aws:RequestTag/kubernetes-namespace": "ExampleNameSpace",
          "aws:RequestTag/kubernetes-service-account": "ExampleServiceAccountName"
        },
        "ArnEquals": {
          "aws:PrincipalARN": "arn:aws:iam::111122223333:role/eks-pod-identity-primary-
role"
        }
      }
    }
  ]
}
```

上述政策可讓`eks-pod-identity-primary-role` AWS 帳戶 111122223333 中具有相關 EKS Pod Identity Session 標籤的角色擔任此角色。

若您在 EKS Pod 身分識別中停用工作階段標籤，EKS Pod 身分識別亦可在擔任目標角色時，利用 Pod 的叢集、命名空間及服務帳戶相關資訊來設定 `sts:ExternalId`。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:root"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "sts:ExternalId": "region/111122223333/cluster-name/namespace/service-
account-name"
        },
        "ArnEquals": {
          "aws:PrincipalARN": "arn:aws:iam::111122223333:role/eks-pod-identity-primary-
role"
        }
      }
    },
    {
      "Effect": "Allow",
      "Principal": {
```

```
        "AWS": "arn:aws:iam::111122223333:root"
      },
      "Action": "sts:TagSession"
    }
  ]
}
```

藉助上述政策，可確保僅預期的叢集、命名空間及服務帳戶擔任目標角色。

更新 EKS Pod 身分識別角色的許可政策

在此步驟中，您將新增目標 IAM 角色 ARN 作為資源，藉此來更新與 Amazon EKS 叢集關聯的 EKS Pod 身分識別角色的許可政策。

1. 開啟 Amazon EKS 主控台。
2. 在左側導覽窗格中，選取叢集，然後選取 EKS 叢集的名稱。
3. 選擇存取索引標籤。
4. 在 Pod 身分識別關聯項下，選取 EKS Pod 身分識別角色。
5. 依序選擇許可、新增許可、建立內嵌政策。
6. 選擇 JSON。
7. 在政策編輯器中，使用您的許可政策來取代預設 JSON。使用您的目標 IAM 角色，來取代 IAM 角色 ARN 中的角色名稱及 222233334444 的預留位置。例如：

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sts:AssumeRole",
                "sts:TagSession"
            ],
            "Resource": "arn:aws:iam::222233334444:role/eks-pod-identity-aws-resources"
        }
    ]
}
```

步驟 2：關聯目標 IAM 角色與 Kubernetes 服務帳戶

在此步驟中，您將在 EKS 叢集中的目標 IAM 角色與 Kubernetes 服務帳戶間建立關聯。

1. 開啟 Amazon EKS 主控台。
2. 在左側導覽窗格中，選取叢集，然後選取您想要新增關聯的目標叢集的名稱。
3. 選擇存取索引標籤。
4. 在 Pod 身分識別關聯中，選擇建立。
5. 針對您想要其擔任角色的工作負載，選擇 IAM 角色中的 EKS Pod 身分識別角色。
6. 在目標 IAM 角色中選擇目標 IAM 角色，將由 EKS Pod 身分識別角色擔任。
7. 在 Kubernetes 命名空間欄位，輸入您想要建立關聯的命名空間名稱 (例如 `my-app-namespace`)。這將定義服務帳戶所在的位置。
8. 在 Kubernetes 服務帳戶欄位，輸入將會使用 IAM 憑證的服務帳戶名稱 (例如 `my-service-account`)。這將連結 IAM 角色至服務帳戶。
9. （選用）選取停用工作階段標籤，以停用 Pod Identity 在擔任角色時自動新增的預設工作階段標籤。
10. （選用）切換設定工作階段政策以設定 IAM 政策，以套用超出連接至目標 IAM 角色之 IAM 政策中定義之許可的其他限制至此 Pod Identity 關聯。

> **Note**
>
> 1. 只有在核取停用工作階段標籤設定時，才能套用工作階段政策。2. 如果您指定工作階段政策，則政策限制適用於目標 IAM 角色的許可，而不是與此 Pod Identity 關聯相關聯的 IAM 角色。

11選擇建立可建立關聯。

設定 Pod 以使用 AWS 服務帳戶存取 服務

如果 Pod 需要存取 AWS 服務，則必須將其設定為使用 Kubernetes 服務帳戶。服務帳戶必須與具有存取 AWS 服務許可的 AWS Identity and Access Management (IAM) 角色相關聯。

- 現有的叢集。如果您沒有，則可以使用其中一個 開始使用 指南來建立一個。
- 將服務帳戶與 IAM 角色建立關聯的現有 Kubernetes 服務帳戶和 EKS Pod 身分識別關聯。角色必須具有關聯的 IAM 政策，其中包含您希望 Pod 必須使用 AWS 服務的許可。如需有關服務帳戶和角色之建立和設定方式的詳細資訊，請參閱the section called “指派 IAM 角色”。

- 在您的裝置或 AWS CloudShell 上安裝和設定的最新版本 AWS CLI。您可以使用 `aws --version | cut -d / -f2 | cut -d ' ' -f1` 來檢查您的目前版本。適用於 macOS 的 yum、apt-get或 Homebrew 等套件管理員通常是最新版本 CLI AWS 後面的幾個版本。若要安裝最新版本，請參閱《 AWS 命令列界面使用者指南》中的使用 aws 設定安裝 和快速組態。 https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-quickstart.html#cli-configure-quickstart-config安裝在 AWS CloudShell 中的 AWS CLI 版本也可能是最新版本後面的幾個版本。若要更新它，請參閱《 AWS CloudShell AWS 使用者指南》中的將 CLI 安裝到您的主目錄。
- `kubectl` 命令列工具安裝在您的裝置或 AWS CloudShell 上。版本可以與您的叢集 Kubernetes 版本相同，或是為最多比該版本更舊一版或更新一版的次要版本。例如，如果您的叢集版本為 `1.29`，則可以搭配使用 `kubectl` `1.28`、`1.29` 或 `1.30` 版。若要安裝或升級 `kubectl`，請參閱 the section called “設定 kubectl 和 eksctl”。
- 包含叢集組態的現有 `kubectl config` 檔案。若要建立 `kubectl config` 檔案，請參閱the section called “透過 kubectl 存取叢集”。

  1. 使用以下命令，以建立部署清單檔案，讓您可部署 Pod 以確認組態。以您自己的值取代範例值。

     ```
     cat >my-deployment.yaml <<EOF
     apiVersion: apps/v1
     kind: Deployment
     metadata:
       name: my-app
     spec:
       selector:
         matchLabels:
           app: my-app
       template:
         metadata:
           labels:
             app: my-app
         spec:
           serviceAccountName: my-service-account
           containers:
           - name: my-app
             image: public.ecr.aws/nginx/nginx:X.XX
     EOF
     ```

  2. 將清單檔案部署到叢集。

     ```
     kubectl apply -f my-deployment.yaml
     ```

  3. 確認您的 Pod 有必要的環境變數。

a. 檢視使用上一步中的部署作業進行部署的 Pod。

```
kubectl get pods | grep my-app
```

範例輸出如下。

```
my-app-6f4dfff6cb-76cv9   1/1     Running   0          3m28s
```

b. 確認 Pod 具有服務帳戶字符檔案掛載。

```
kubectl describe pod my-app-6f4dfff6cb-76cv9 | grep
 AWS_CONTAINER_AUTHORIZATION_TOKEN_FILE:
```

範例輸出如下。

```
AWS_CONTAINER_AUTHORIZATION_TOKEN_FILE:  /var/run/secrets/
pods.eks.amazonaws.com/serviceaccount/eks-pod-identity-token
```

4. 確認您的 Pod 可以使用您在連接至角色的 IAM 政策中指派的許可與 AWS 服務互動。

> **Note**
>
> 當 Pod 使用與服務帳戶相關聯的 IAM 角色 AWS 登入資料時，該 Pod 容器中的 AWS CLI 或其他 SDKs 會使用該角色提供的登入資料。如果您沒有對提供給 Amazon EKS 節點 IAM 角色的憑證限制存取權，則 Pod 仍然可以存取這些認證。如需詳細資訊，請參閱限制存取指派給工作節點的執行個體設定檔。

如果您的 Pod 無法如預期與服務互動，請完成以下步驟以確認所有項目都已正確設定。

a. 確認您的 Pod 使用支援透過 EKS Pod Identity 關聯擔任 IAM 角色的 AWS SDK 版本。如需詳細資訊，請參閱the section called “支援的 SDK”。

b. 確認部署使用服務帳戶。

```
kubectl describe deployment my-app | grep "Service Account"
```

範例輸出如下。

```
Service Account:  my-service-account
```

根據標籤授予 Pod 對 AWS 資源的存取權

屬性型存取控制 (ABAC) 透過結合屬性的政策，來向使用者授予權限。EKS Pod 身分識別會將標籤連接至具有叢集名稱、命名空間與服務帳戶名稱等屬性的每個 Pod 的臨時憑證。這些角色工作階段標籤可讓管理員撰寫單一角色，以根據相符的標籤存取 AWS 資源，跨服務帳戶運作。透過新增對角色工作階段標籤的支援，您可以在叢集之間和叢集內的工作負載之間強制執行更嚴格的安全界限，同時重複使用相同的 IAM 角色和 IAM 政策。

使用標籤的範例政策

下面是一個 IAM 政策範例，該政策在對應物件以 EKS 叢集名稱標記時授予 `s3:GetObject` 許可。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectTagging"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "s3:ExistingObjectTag/eks-cluster-name": "${aws:PrincipalTag/eks-
cluster-name}"
                }
            }
        }
    ]
}
```

啟用或停用工作階段標籤

EKS Pod 身分識別會在擔任角色時，新增一組預先定義的工作階段標籤。這些工作階段標籤可讓管理員撰寫單一角色，允許根據相符標籤存取 AWS 資源，以跨資源運作。

啟用工作階段標籤

工作階段標籤可透過 EKS Pod 身分識別自動啟用，您無須執行任何操作。EKS Pod 身分識別預設會將預先定義的標籤集連接至工作階段。如需在政策中參考這些標籤，可在標籤索引鍵後使用語法 `${aws:PrincipalTag/`。例如 `${aws:PrincipalTag/kubernetes-namespace}`。

- `eks-cluster-arn`
- `eks-cluster-name`
- `kubernetes-namespace`
- `kubernetes-service-account`
- `kubernetes-pod-name`
- `kubernetes-pod-uid`

停用工作階段標籤

AWS 會將內嵌工作階段政策、受管政策 ARNs 和工作階段標籤壓縮為具有個別限制的封裝二進位格式。如果您收到 `PackedPolicyTooLarge` 錯誤，指出封裝二進位格式超過大小限制，您可嘗試停用 EKS Pod 身分識別新增的工作階段標籤來縮減該大小。如需停用這些工作階段標籤，遵循以下步驟操作：

1. 開啟 Amazon EKS 主控台。
2. 在左側導覽窗格中，選取叢集，然後選取您想要修改的叢集名稱。
3. 選擇存取索引標籤。
4. 在 Pod 身分識別關聯中，選擇您想要在關聯 ID 中修改的關聯 ID，然後選擇編輯。
5. 在工作階段標籤項下，選擇停用工作階段標籤。
6. 選擇儲存變更。

跨帳戶標籤

EKS Pod 身分識別新增的所有工作階段標籤都可轉移；標籤金鑰和值會傳遞至工作負載用於將角色切換至其他帳戶的任何 `AssumeRole` 動作。您可以在其他帳戶的政策中使用這些標籤，以限制跨帳戶情況中的存取。如需詳細資訊，請參閱《IAM 使用者指南》中的使用工作階段標籤鏈結角色。

自訂標籤

EKS Pod 身分識別無法將其他自訂標籤新增至其執行的 AssumeRole 動作。不過，您套用至 IAM 角色的標籤永遠都可以使用，但格式相同：${aws:PrincipalTag/ 後面是金鑰，例如 ${aws:PrincipalTag/MyCustomTag}。

> **Note**
>
> 在發生衝突的情況下，透過 sts:AssumeRole 請求新增至工作階段的標籤具有優先順序。舉例來說，假設：
>
> - EKS 擔任客戶角色時，Amazon EKS 將鍵 eks-cluster-name 和值 my-cluster 新增至工作階段，並且
> - 您將 eks-cluster-name 標籤新增至具有值 my-own-cluster 的 IAM 角色。
>
> 在這種情況下，前者優先，eks-cluster-name 標籤的值會是 my-cluster。

搭配 AWS SDK 使用 Pod 身分

使用 EKS Pod 身分識別憑證

若要使用來自 EKS Pod Identity 關聯的登入資料，您的程式碼可以使用任何 AWS SDK 來透過 SDK 為 AWS 服務建立用戶端，並根據預設，軟體開發套件會在位置鏈中搜尋要使用的 AWS Identity and Access Management 登入資料。如果您在建立用戶端或初始化 SDK 時未指定登入資料提供者，將使用 EKS Pod Identity 登入資料。

這樣是有效的，因為 EKS Pod 身分識別已新增至容器憑證提供者，系統會在預設憑證鏈中的某個步驟搜尋此提供者。如果您的工作負載目前使用憑證鏈中較早的憑證，那麼即使您為相同工作負載設定 EKS Pod 身分識別關聯，系統仍會繼續使用這些憑證。

如需 EKS Pod 身分識別運作方式的詳細資訊，請參閱 the section called “運作方式”。

使用 了解 EKS Pod Identity 如何授予 Pod 對 AWS 服務的存取權時，Pod 中的容器必須使用支援從 EKS Pod Identity Agent 擔任 IAM 角色的 AWS SDK 版本。請確定您使用 SDK 的下列版本 或更新版本 AWS ：

- Java (版本 2) – 2.21.30
- Java – 1.12.746

- Go v1 – v1.47.11
- Go v2 – release-2023-11-14
- Python (Boto3) – 1.34.41
- Python (botocore) – 1.34.41
- AWS CLI – 1.30.0

  AWS CLI – 2.15.0
- JavaScript v2 – 2.1550.0
- JavaScript v3 – v3.458.0
- Kotlin – v1.0.1
- Ruby – 3.188.0
- Rust – 版本-2024-03-13
- C++ – 1.11.263
- .NET – 3.7.734.0
- PowerShell – 4.1.502
- PHP – 3.289.0

為了確保您使用支援的 SDK，在建立您的容器時，請遵循在 AWS上建立的工具中您偏好之 SDK 的安裝指示。

若要了解支援 EKS Pod 身分識別的附加元件清單，請參閱 the section called “Pod 身分識別支援參考”。

在 EKS Pod 身分識別代理程式中停用 IPv6

AWS 管理主控台

1. 如需在 EKS Pod 身分識別代理程式中停用 IPv6，可新增以下組態至 EKS 附加元件的選用組態設定。
   a. 開啟 Amazon EKS 主控台。
   b. 在左側導覽窗格中，選取 Clusters (叢集)，然後選取您要為其設定附加元件之叢集的名稱。
   c. 選擇附加元件索引標籤。
   d. 選取 EKS Pod 身分識別代理程式附加元件方塊右上方的方塊，然後選擇編輯。
   e. 在設定 EKS Pod 身分識別代理程式頁面：

i. 選取您要使用的版本。建議您保留與上一步驟相同的版本，並執行單獨的動作來更新版本與組態。

ii. 展開選用組態設定。

iii. 請輸入組態值中包含索引鍵 "additionalArgs": 的巢狀 JSON 物件的 JSON 索引鍵 "agent": 和值。產生的文字必須是有效的 JSON 物件。如果此金鑰和值是文字方塊中唯一的資料，請以大括號 { } 括住該金鑰和值。以下範例顯示已啟用網路政策：

```
{
    "agent": {
        "additionalArgs": {
            "-b": "169.254.170.23"
        }
    }
}
```

此組態將 IPv4 位址設定為代理程式使用的唯一位址。

f. 如需透過取代 EKS Pod 身分識別代理程式 Pod 來套用新的組態，選擇儲存變更。

Amazon EKS 藉由推展適用於 EKS Pod 身分識別代理程式的 Kubernetes DaemonSet，來套用變更至 EKS 附加元件。您可以在 和 中的附加元件更新歷史記錄中追蹤推展的狀態 AWS 管理主控台 kubectl rollout status daemonset/eks-pod-identity-agent --namespace kube-system。

kubectl rollout 具有下列命令：

```
$ kubectl rollout

history  -- View rollout history
pause    -- Mark the provided resource as paused
restart  -- Restart a resource
resume   -- Resume a paused resource
status   -- Show the status of the rollout
undo     -- Undo a previous rollout
```

若推展時間太長，Amazon EKS 會復原推展，並且類型為附加元件更新及狀態為失敗的訊息，則會新增至附加元件的更新歷程記錄。若要調查任何問題，請從推展的歷史記錄開始，並在 EKS Pod Identity Agent Pod kubectl logs上執行，以查看 EKS Pod Identity Agent 的日誌。

2. 若更新歷程記錄中新項目的狀態為成功，則表示推展已完成，並且在所有 EKS Pod 身分識別代理程式 Pod 中，附加元件均會使用新的組態。

AWS CLI

1. 如需在 EKS Pod 身分識別代理程式中停用 IPv6，可新增以下組態至 EKS 附加元件的組態值。

   執行下列 AWS CLI 命令。將 my-cluster 取代為您的叢集名稱，並將 IAM 角色 ARN 取代為您正在使用的角色。

   ```
   aws eks update-addon --cluster-name my-cluster --addon-name eks-pod-identity-agent \
       --resolve-conflicts PRESERVE --configuration-values '{"agent":{"additionalArgs":
    { "-b": "169.254.170.23"}}}'
   ```

   此組態將 IPv4 位址設定為代理程式使用的唯一位址。

   Amazon EKS 藉由推展適用於 EKS Pod 身分識別代理程式的 Kubernetes DaemonSet，來套用變更至 EKS 附加元件。您可以在 和 中的附加元件更新歷史記錄中追蹤推展的狀態 AWS 管理主控台 `kubectl rollout status daemonset/eks-pod-identity-agent --namespace kube-system`。

   `kubectl rollout` 具有下列命令：

   ```
   kubectl rollout

   history  -- View rollout history
   pause    -- Mark the provided resource as paused
   restart  -- Restart a resource
   resume   -- Resume a paused resource
   status   -- Show the status of the rollout
   undo     -- Undo a previous rollout
   ```

   若推展時間太長，Amazon EKS 會復原推展，並且類型為附加元件更新及狀態為失敗的訊息，則會新增至附加元件的更新歷程記錄。若要調查任何問題，請從推展的歷史記錄開始，並在 EKS Pod Identity Agent Pod `kubectl logs`上執行，以查看 EKS Pod Identity Agent 的日誌。

透過 EKS Pod 身分識別所需的信任政策來建立 IAM 角色

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowEksAuthToAssumeRoleForPodIdentity",
            "Effect": "Allow",
            "Principal": {
                "Service": "pods.eks.amazonaws.com"
            },
            "Action": [
                "sts:AssumeRole",
                "sts:TagSession"
            ]
        }
    ]
}
```

sts:AssumeRole

EKS Pod 身分識別會先使用 AssumeRole 來擔任 IAM 角色，然後將臨時憑證傳至您的 Pod。

sts:TagSession

EKS Pod Identity 使用 TagSession 在對 AWS STS 的請求中包含工作階段標籤。

設定條件

您可以在信任政策的條件索引鍵中使用這些標籤，用來限制哪些服務帳戶、命名空間和叢集可以使用此角色。若要了解 Pod 身分識別新增的請求標籤清單，請參閱 the section called “啟用或停用工作階段標籤”。

例如，您可以限制哪些 Pod 可以擔任特定 Pod 身分 IAM 角色Namespace，ServiceAccount並使用下列具有新增 的信任政策Condition：

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowEksAuthToAssumeRoleForPodIdentity",
            "Effect": "Allow",
            "Principal": {
                "Service": "pods.eks.amazonaws.com"
            },
            "Action": [
```

```
                "sts:AssumeRole",
                "sts:TagSession"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/kubernetes-namespace": [
                        "Namespace"
                    ],
                    "aws:RequestTag/kubernetes-service-account": [
                        "ServiceAccount"
                    ]
                }
            }
        }
    ]
}
```

如需 Amazon EKS 條件金鑰的清單，請參閱服務授權參考中的 Amazon Elastic Kubernetes Service 定義的條件。若要了解您可以搭配哪些動作和資源使用條件索引鍵，請參閱 Amazon Elastic Kubernetes Service 定義的動作。

使用節點管理運算資源

Kubernetes 節點是執行容器化應用程式的機器。每個節點皆具有下列元件：

- 容器執行時期：負責執行容器的軟體。
- kubelet：確保容器運作狀態良好並在其關聯的 Pod 中執行。
- kube-proxy：維護允許與 Pod 通訊的網路規則。

如需詳細資訊，請參閱 Kubernetes 文件中的節點。

您的 Amazon EKS 叢集可在 EKS 自動模式受管理節點、自助管理節點、Amazon EKS 受管理節點群組、AWS Fargate與Amazon EKS 混合節點的任意組合上排程 Pod。若要進一步了解在叢集中部署的節點，請參閱 the section called “存取叢集資源”。

> **Note**
>
> 排除混合節點，節點必須與您在建立叢集時選取的子網路位於同一 VPC 中。不過，節點不需要位於相同的子網路中。

對比運算選項

下表提供了幾個準則，以便在決定哪些選項最符合您的要求時進行評估。自助管理節點是另一種選項，支援下表列出的所有標準，但需要更多手動維護工作。如需詳細資訊，請參閱the section called “自我管理的節點”。

> **Note**
>
> Bottlerocket 與此表格中的一般資訊有一些特定的差異。如需詳細資訊，請參閱 GitHub 上的 Bottlerocket 文件。

條件	EKS 受管節點群組	EKS 自動模式	Amazon EKS 混合節點
可以部署到 AWS Outposts	否	否	否
可以部署到 AWS Local Zone	是	否	否
可以執行需要 Windows 的容器	是	否	否
可以執行需要 Linux 的容器	是	是	是
可以執行需要 Inferentia 晶片的工作負載	是：僅限 Amazon Linux 節點	是	否
可以執行需要 GPU 的工作負載	是：僅限 Amazon Linux 節點	是	是
可以執行需要 Arm 處理器的工作負載	是	是	是
可以執行 AWS Bottlerocket	是	是	否
Pod 會與其他 Pod 共享 CPU、記憶體、儲存體和網路資源。	是	是	是
必須部署和管理 Amazon EC2 執行個體	是	否 - 了解 EC2 受管執行個體	是 – 內部部署實體或虛擬機器由您選擇的工具管理。
必須保護、維護和修補 Amazon EC2 執行個體的作業系統	是	否	是 – 在實體或虛擬機器上執行的作業系統由您選擇的工具管理。

條件	EKS 受管節點群組	EKS 自動模式	Amazon EKS 混合節點
可以在部署節點時提供引導引數，例如額外的 kubelet 引數。	是 – 以自訂 AMI 使用 `eksctl` 或啟動範本。	否 - 使用 `NodeClass` 設定節點	是 - 您可使用 nodeadm 自訂引導引數。請參閱 the section called “混合節點 nodeadm”。
可以將 IP 位址指派給來自不同於將 IP 位址指派給節點之 CIDR 區塊的 Pod。	是：以自訂 AMI 使用啟動範本。如需詳細資訊，請參閱the section called “啟動範本”。	否	是，請參閱 the section called “設定 CNI”。
可以對節點執行 SSH	是	否 - 了解如何對節點進行疑難排解	是
可以將自己的自訂 AMI 部署至節點	是：使用啟動範本	否	是
可以將您自己的自訂 CNI 部署至節點	是：以自訂 AMI 使用啟動範本	否	是
必須自行更新節點 AMI	是：如果已部署 Amazon EKS 最佳化 AMI，Amazon EKS 主控台會在有可用更新時通知您。您只要在主控台中按一下即可執行更新。如果已部署自訂 AMI，Amazon EKS 主控台不會在有更新可用時通知您。您必須自行執行更新。	否	是 – 在實體或虛擬機器上執行的作業系統由您選擇的工具管理。請參閱 the section called “準備作業系統”。

條件	EKS 受管節點群組	EKS 自動模式	Amazon EKS 混合節點
必須自行更新節點 Kubernetes 版本	是：如果已部署 Amazon EKS 最佳化 AMI，Amazon EKS 主控台會在有可用更新時通知您。您只要在主控台中按一下即可執行更新。如果已部署自訂 AMI，Amazon EKS 主控台不會在有更新可用時通知您。您必須自行執行更新。	否	是 - 您可以使用自選工具或 nodeadm 來管理混合節點的升級。請參閱 the section called “升級混合節點”。
可以透過 Pod 使用 Amazon EBS 儲存體	是	是，作為整合能力。了解如何建立儲存類別。	否
可以透過 Pod 使用 Amazon EFS 儲存體	是	是	否
可以搭配 Pod 使用 Amazon S3 檔案儲存	是	是	否
可以透過 Pod 使用 Amazon FSx for Lustre 儲存體	是	是	否
可以針對服務使用 Network Load Balancer	是	是	是 - 必須使用目標類型 `ip`。
Pod 可以在公有子網路中執行	是	是	否 - 在內部部署環境中執行的 Pod。

條件	EKS 受管節點群組	EKS 自動模式	Amazon EKS 混合節點
可以將不同的 VPC 安全群組指派給個別 Pod	是 – 僅限 Linux 節點	否	否
可以執行 Kubernetes DaemonSets	是	是	是
支援 Pod 資訊清單中的 `HostPort` 和 `HostNetwork`	是	是	是
AWS 區域可用性	所有 Amazon EKS 支援的區域	所有 Amazon EKS 支援的區域	所有 Amazon EKS 支援的區域，AWS GovCloud (US) 區域和中國區域除外。
可以在 Amazon EC2 專用主機上執行容器	是	否	否
定價	執行多個 Pod 的 Amazon EC2 執行個體的成本。如需詳細資訊，請參閱 Amazon EC2 定價。	在叢集中啟用 EKS 自動模式時，除了標準 EC2 執行個體費用之外，您還需要為使用自動模式運算功能啟動的執行個體支付個別費用。數量會隨啟動的執行個體類型和叢集所在的 AWS 區域而有所不同。如需詳細資訊，請參閱 Amazon EKS 定價。	每小時混合節點 vCPU 的成本。如需詳細資訊，請參閱 Amazon EKS 定價。

透過受管節點群組來簡化節點生命週期

Amazon EKS 受管節點群組會自動化 Amazon EKS Kubernetes 叢集節點 (Amazon EC2 執行個體) 的佈建和生命週期管理。

使用 Amazon EKS 受管節點群組時，不需要個別佈建或註冊提供運算容量的 Amazon EC2 執行個體，來執行 Kubernetes 應用程式。您可以透過單一操作來建立、自動更新或終止叢集的節點。節點更新和終止會自動耗盡節點，以確保您的應用程式保持可用。

每個受管節點均會佈建為 Amazon EKS 為您管理的 Amazon EC2 Auto Scaling 群組的一部分。包括執行個體和 Auto Scaling 群組在內的每個資源都會在您的 AWS 帳戶中執行。每個節點群組都可以跨您定義的多個可用區域執行。

受管節點群組還可以選擇性地利用節點自動修復，進而持續監控節點的運作狀態。它會自動對偵測到的問題進行回應，並盡可能取代節點。此功能有助於提高叢集的整體可用性，並且盡可能減少手動介入。如需詳細資訊，請參閱the section called “節點運作狀態和修復”。

您可以使用 Amazon EKS 主控台、`eksctl`、 AWS CLI、 AWS API 或基礎設施做為程式碼工具，包括 AWS CloudFormation，將受管節點群組新增至新的或現有的叢集。以受管節點群組身分啟動的節點，會自動標記以供 Kubernetes Cluster Autoscaler 自動探索。您可以使用節點群組將 Kubernetes 標籤套用至節點，並隨時加以更新。

使用 Amazon EKS 受管節點群組無需額外費用，您只需為佈建 AWS 的資源付費。其中包括 Amazon EC2 執行個體、Amazon EBS 磁碟區、Amazon EKS 叢集時數，以及任何其他 AWS 基礎設施。沒有最低費用，也沒有前期承諾。

若要開始使用新的 Amazon EKS 叢集和受管節點群組，請參閱 the section called “建立叢集 (主控台和 CLI)”。

若要將受管理的節點群組新增至現有叢集，請參閱 the section called “建立”。

受管節點群組概念

- Amazon EKS 受管節點群組會為您建立和管理 Amazon EC2 執行個體。
- 每個受管節點均會佈建為 Amazon EKS 為您管理的 Amazon EC2 Auto Scaling 群組的一部分。此外，包括 Amazon EC2 執行個體和 Auto Scaling 群組在內的每個資源都會在您的 AWS 帳戶中執行。
- Amazon EKS 會定期同步受管節點群組的擴展組態，以符合實際的 Auto Scaling 群組值。如果 Cluster Autoscaler 等外部演員修改 Auto Scaling 群組的大小， 最終`DescribeNodegroup`會反映這些變更。當您啟動節點群組更新或升級而不明確修改擴展組態時，工作流程會使用目前的

Auto Scaling 群組值，而不是節點群組的預存擴展組態。儲存的擴展組態只有在您明確地將其包含在UpdateNodegroupConfig請求中時才會優先。

- 受管節點群組的 Auto Scaling 群組會跨越您在建立群組時指定的每個子網路。
- Amazon EKS 會標記受管節點群組資源，以便將其設定為使用 Kubernetes Cluster Autoscaler。

> **Important**
>
> 如果您跨越多個由 Amazon EBS 磁碟區提供的可用區域執行狀態應用程式，並且使用 Kubernetes Cluster Autoscaler，您應該設定多個節點群組，每個群組的範圍皆為單一可用區域。另外，您應該啟用 `--balance-similar-node-groups` 功能。

- 部署受管節點時，您可以使用自訂啟動範本來獲得更高層級的靈活性和自訂能力。例如，可指定額外的 `kubelet` 引數並使用自訂 AMI。如需詳細資訊，請參閱the section called “啟動範本”。如果在第一次建立受管節點群組時未使用自訂啟動範本，會有一個自動產生的啟動範本。請勿手動修改此自動產生的範本，否則會發生錯誤。
- Amazon EKS 會遵循受管節點群組上 CVE 和安全修補程式的共同責任模型。受管節點執行 Amazon EKS 最佳化 AMI 時，Amazon EKS 負責在報告錯誤或問題時建置 AMI 的修補版本。不過，您必須負責將這些修補的 AMI 版本部署至受管節點群組。當受管節點執行自訂 AMI 時，您必須負責在報告錯誤或問題時建置 AMI 的修補版本，然後部署 AMI。如需詳細資訊，請參閱the section called “更新”。
- Amazon EKS 受管節點群組可以在公有和私有子網路中啟動。如果在 2020 年 4 月 22 日或之後啟動公有子網路中的受管節點群組，則子網路必須將 `MapPublicIpOnLaunch` 設定為「true」，執行個體才能成功加入叢集。如果公有子網路是在 2020 年 3 月 26 日或之後使用 `eksctl`或 Amazon EKS vended AWS CloudFormation 範本建立，則此設定已設為 true。如果公有子網路是在 2020 年 3 月 26 日之前所建立，則必須手動變更設定。如需詳細資訊，請參閱修改子網的公有 IPv4 定址屬性。
- 在私有子網路中部署受管節點群組時，您必須確保該群組可存取 Amazon ECR 以提取容器映像。您可以透過將 NAT 閘道連接至子網路的路由表，或新增以下 AWS PrivateLink VPC 端點來執行此操作：
  - Amazon ECR API 端點界面 – `com.amazonaws.`*`region-code`*`.ecr.api`
  - Amazon ECR Docker 登錄檔 API 端點界面 – `com.amazonaws.`*`region-code`*`.ecr.dkr`
  - Amazon S3 閘道端點 – `com.amazonaws.`*`region-code`*`.s3`

  如需了解其他常用的服務和端點，請參閱 the section called “私有叢集”。
- 受管節點群組無法部署在 AWS Outposts 上或 AWS Wavelength 中。受管節點群組可建立在 AWS Local Zones 上。如需詳細資訊，請參閱the section called “ AWS 本機區域”。

- 您可以在單一叢集內建立多個受管節點群組。例如，您可以針對某些工作負載建立具有標準 Amazon EKS 最佳化 Amazon Linux AMI 的節點群組，而針對需要 GPU 支援的工作負載使用 GPU 變體建立另一個節點群組。
- 如果受管節點群組遇到 Amazon EC2 執行個體狀態檢查故障問題，Amazon EKS 會傳回錯誤代碼，以協助您診斷問題。如需詳細資訊，請參閱the section called “受管節點群組錯誤代碼”。
- Amazon EKS 會將 Kubernetes 標籤新增至受管節點群組執行個體。這些 Amazon EKS 提供的標籤前面會加上 `eks.amazonaws.com`。
- 在終止或更新期間，Amazon EKS 會使用 Kubernetes API 自動耗盡節點。
- 使用 `AZRebalance` 終止節點或減少所需的節點計數時，不會遵守 Pod 中斷預算。這些動作會嘗試在該節點上移出 Pod。但如果需要超過 15 分鐘，則無論節點上的所有 Pod 是否終止，都會終止節點。若要延長期間直到節點終止，請將 lifecycle hook 新增至 Auto Scaling 群組。如需詳細資訊，請參閱 Amazon EC2 Auto Scaling 使用者指南中的新增 lifecycle hook。
- 若要在收到 Spot 中斷通知或容量重新平衡通知後正確執行耗盡程序，必須將 `CapacityRebalance` 設定為 `true`。
- 更新受管節點群組會遵守您為 Pod 設定的 Pod 中斷預算。如需詳細資訊，請參閱 the section called “更新行為詳細資訊”。
- 使用 Amazon EKS 受管節點群組不會產生額外成本。您只需為佈建 AWS 的資源付費。
- 如果要為節點加密 Amazon EBS 磁碟區，則可以使用啟動範本部署節點。若要在不使用啟動範本的情況下部署具有加密 Amazon EBS 磁碟區的受管節點，請加密帳戶中建立的所有新 Amazon EBS 磁碟區。如需詳細資訊，請參閱《Amazon EC2 使用者指南》中的預設加密。

受管節點群組容量類型

建立受管節點群組時，您可以選擇隨需或 Spot 容量類型。Amazon EKS 部署具有 Amazon EC2 Auto Scaling 群組的受管節點群組，該群組僅包含隨需執行個體或僅包含 Amazon EC2 Spot 執行個體。您可以將具有容錯能力之應用程式的 Pod 排程到 Spot 受管節點群組，以及將具有容錯能力之應用程式的 Pod 排程到單一 Kubernetes 叢集中的隨需節點群組。依預設，受管節點群組會部署隨需 Amazon EC2 執行個體。

On-Demand

透過隨需執行個體，您只需要按秒數支付運算容量開銷，無需簽訂長期合約。

預設情況下，如果未指定容量類型，則會使用隨需執行個體佈建受管節點群組。受管節點群組會代表您設定 Amazon EC2 Auto Scaling 群組，並套用下列設定：

- 佈建隨需容量的配置策略設定為 `prioritized`。在滿足隨需容量時，受管節點群組會 API 中傳遞執行個體類型的順序，決定先使用哪一種執行個體類型。例如，您可以按照下列順序指定三種執行個體類型：`c5.large`、`c4.large`，以及 `c3.large`。當您的隨需執行個體啟動時，受管節點群組會從 `c5.large` 開始，然後是 `c4.large`，再來是 `c3.large`，以滿足隨需容量。如需詳細資訊，請參閱《Amazon EC2 Auto Scaling 使用者指南》中的 Amazon EC2 Auto Scaling 群組。
- Amazon EKS 會將下列 Kubernetes 標籤新增至指定容量類型之受管節點群組中的所有節點：`eks.amazonaws.com/capacityType: ON_DEMAND`。您可以使用此標籤，在隨需節點上排程可設定狀態或可容錯的應用程式。

Spot

Amazon EC2 Spot 執行個體是備用的 Amazon EC2 容量，可提供隨需價格的極低折扣。EC2 需要取回容量時，就可能會以兩分鐘中斷通知的方式中斷 Amazon EC2 Spot 執行個體。如需詳細資訊，請參閱《Amazon EC2 使用者指南》中的Spot 執行個體。您可以使用 Amazon EC2 Spot 執行個體設定受管節點群組，以最佳化 Amazon EKS 叢集中執行之運算節點的成本。

若要在受管節點群組內使用 Spot 執行個體，請將容量類型設定為`spot`，以建立受管節點群組。受管節點群組代表您設定 Amazon EC2 Auto Scaling 群組，並套用下列 Spot 最佳實務：

- 為確保您的 Spot 節點已在最佳 Spot 容量集區中佈建，配置策略會設定為下列之一：
  - `price-capacity-optimized` (PCO)：在具有 Kubernetes 版本 `1.28` 或更新版本的叢集中建立新節點群組時，配置策略會設定為 `price-capacity-optimized`。但是，在 Amazon EKS 受管節點群組開始支援 PCO 之前，系統不會變更已使用 `capacity-optimized` 建立的節點群組的配置策略。
  - `capacity-optimized` (CO)：在具有 Kubernetes 版本 `1.27` 或更低版本的叢集中建立新節點群組時，配置策略會設定為 `capacity-optimized`。

  若要增加可用於配置容量的 Spot 容量集區數量，請將受管節點群組設定為使用多個執行個體類型。
- 已啟用 Amazon EC2 Spot 容量重新平衡功能，以便 Amazon EKS 可以正常地消耗和重新平衡 Spot 節點，在 Spot 節點中斷風險提高時，將應用程式中斷情況降至最低。如需詳細資訊，請參閱《Amazon EC2 Auto Scaling 使用者指南》中的 Amazon EC2 Auto Scaling 容量重新平衡。
  - Spot 節點收到重新平衡建議時，Amazon EKS 會自動嘗試啟動新的替代 Spot 節點。
  - 如果 Spot 兩分鐘中斷通知在取代 Spot 節點處於 `Ready` 狀態前到達，則 Amazon EKS 會開始消耗收到重新平衡建議的 Spot 節點。Amazon EKS 將竭盡全力耗盡節點。因此，無法保證 Amazon EKS 會在耗盡現有節點之前等待替代節點加入叢集。

- 啟動取代 Spot 節點，並且在 Kubernetes 處於 Ready 狀態時，Amazon EKS 會包圍隔離並消耗收到重新平衡建議的 Spot 節點。包圍隔離 Spot 節點可確保服務控制器不會將任何新的請求傳送到此 Spot 節點。它也會從狀況良好、作用中的 Spot 節點清單中移除。耗盡 Spot 節點可確保正在執行的 Pod 可以正常地移出。

- Amazon EKS 會將下列 Kubernetes 標籤新增至指定容量類型之受管節點群組中的所有節點：`eks.amazonaws.com/capacityType: SPOT`。您可以使用此標籤在 Spot 節點上排程可容錯的應用程式。

> **Important**
>
> EC2 會在終止 Spot 執行個體的兩分鐘前發出 Spot 中斷通知。不過，Spot 節點上的 Pod 可能不會收到正常關閉的完整 2 分鐘時段。當 EC2 發出通知時，Amazon EKS 開始移出 Pod 之前會有延遲。移除會依序發生以保護 Kubernetes API 伺服器，因此在多個同時 Spot 回收期間，某些 Pod 可能會收到延遲的移出通知。Pod 可能會在不接收終止訊號的情況下強制終止，特別是在具有高 Pod 密度的節點上、在並行回收期間或使用長終止寬限期時。對於 Spot 工作負載，我們建議您使用 30 秒或更短的終止寬限期來設計應用程式，避免長時間執行preStop 掛鉤，並監控 Pod 移出指標，以了解叢集中的實際寬限期。對於需要保證正常終止的工作負載，我們建議您改用隨需容量。

決定是否要部署具有隨需或 Spot 容量的節點群組時，應考慮下列條件：

- Spot 執行個體適用於無狀態、容錯、靈活的應用程式。其中包括批次和機器學習訓練工作負載、大數據 ETL (例如 Apache Spark)、佇列處理應用程式，以及無狀態 API 端點。由於 Spot 是可能隨時間變更的備用 Amazon EC2 容量，因此建議您將 Spot 容量用於可接受中斷的工作負載。更具體地說，Spot 容量適用於容錯期間無法使用所需容量的工作負載。
- 我們建議您將隨需模式用於不容錯的應用程式。這包括叢集管理工具 (例如監控和操作工具)、需要 `StatefulSets` 的部署，以及狀態應用程式 (例如資料庫)。
- 為了在使用 Spot 執行個體時最大化應用程式的可用性，建議您將 Spot 受管節點群組設定為使用多個執行個體類型。建議您在使用多個執行個體類型時，套用下列規則：
  - 在受管節點群組中，如果您使用 Cluster Autoscaler，則建議使用具有相同數量 vCPU 和記憶體資源的彈性執行個體類型集。這是為了確保叢集中的節點如預期般擴展。例如，如果您需要四個 vCPU 和八個 GiB 記憶體，請使用 `c3.xlarge`、`c4.xlarge`、`c5.xlarge`、`c5d.xlarge`、`c5a.xlarge`、`c5n.xlarge` 或其他類似的執行個體類型。

- 若要增強應用程式可用性，建議部署多個 Spot 受管節點群組。為此，每個群組應該使用具有相同 vCPU 和記憶體資源的彈性執行個體類型集。例如，如果您需要 4 個 vCPU 和 8 個 GiB 記憶體，建議您使用 `c3.xlarge`、`c4.xlarge`、`c5.xlarge`、`c5d.xlarge`、`c5a.xlarge`、`c5n.xlarge` 或其他類似的執行個體類型建立一個受管節點群組，以及使用 `m3.xlarge`、`m4.xlarge`、`m5.xlarge`、`m5d.xlarge`、`m5a.xlarge`、`m5n.xlarge` 或其他類似的執行個體類型建立另一個受管節點。
- 透過使用自訂啟動範本的 Spot 容量類型部署節點群組時，請使用 API 傳遞多個執行個體類型。請勿透過啟動範本傳遞單一執行個體類型。如需使用啟動範本部署節點群組的詳細資訊，請參閱 the section called “啟動範本”。

建立叢集的受管節點群組

本主題會描述如何啟動已向 Amazon EKS 叢集註冊的節點的 Amazon EKS 受管節點群組。節點加入叢集後，您就可以將 Kubernetes 應用程式部署至其中。

如果這是您第一次啟動 Amazon EKS 受管節點群組，建議您改為按照其中一個 開始使用 指南的步驟進行。這些指南提供建立具有節點的 Amazon EKS 叢集的演練。

> **Important**
>
> - Amazon EKS 節點是標準的 Amazon EC2 執行個體。我們會根據正常 Amazon EC2 價格向您收費。如需詳細資訊，請參閱 Amazon EC2 定價。
> - 您無法在已啟用 AWS Outposts 或 Wavelength AWS 的 AWS 區域中建立受管節點。您可改為建立自我管理的節點。如需詳細資訊，請參閱the section called “Amazon Linux”、the section called “Windows”及the section called “Bottlerocket”。您也可以在 Outpost 上建立自我管理的 Amazon Linux 節點群組。如需詳細資訊，請參閱the section called “節點”。
> - 如果您沒有為 Amazon EKS 最佳化 Linux 或 Bottlerocket 隨附的 `bootstrap.sh` 檔案指定 AMI ID，則受管節點群組會對 `maxPods` 的值強制執行數量上限。對於少於 30 個 vCPU 的執行個體，數量上限為 `110`。對於具有 30 個以上 vCPU 的執行個體，數量上限會跳至 `250`。此強制執行會覆寫其他`maxPods`組態，包括 `maxPodsExpression`。如需如何`maxPods`確定以及如何自訂它的詳細資訊，請參閱 the section called “maxPods的判斷方式”。

- 現有 Amazon EKS 叢集。若要部署叢集，請參閱 the section called “建立 叢集”。

- 供節點使用的現有 IAM 角色。若要建立服務角色，請參閱the section called “節點 IAM 角色”。如果此角色沒有任何 VPC CNI 的政策，則 VPC CNI Pod 需要以下個別角色。
- (選用，但建議) Kubernetes 專用 Amazon VPC CNI 外掛程式的附加元件設定有自己的 IAM 角色，該角色已連接必要的 IAM 政策。如需詳細資訊，請參閱the section called “為 IRSA 設定”。
- 熟悉選擇最佳 Amazon EC2 節點執行個體類型中所列的考量事項。取決於您選擇的執行個體類型，您的叢集和 VPC 可能還有其他先決條件。
- 若要新增 Windows 受管節點群組，您必須先啟用叢集的 Windows 支援。如需詳細資訊，請參閱the section called “啟用 Windows 支援”。

您可以透過以下任一項來建立受管節點群組：

- the section called “eksctl”
- the section called “AWS 管理主控台”

eksctl

使用 eksctl 建立受管節點群組

此程序需要 eksctl 版本 0.215.0 或更新版本。您可使用以下命令檢查您的版本：

```
eksctl version
```

如需有關安裝或更新 eksctl 的指示，請參閱 eksctl 文件中的 Installation 一節。

1. (選用) 如果 AmazonEKS_CNI_Policy 受管 IAM 政策已連接至您的 Amazon EKS 節點 IAM 角色，建議您改為將其指派給您與 Kubernetes aws-node 服務帳戶相關聯的 IAM 角色。如需詳細資訊，請參閱the section called “為 IRSA 設定”。
2. 使用或不使用自訂啟動範本來建立受管節點群組。手動指定啟動範本可讓您更靈活地自訂節點群組。例如，此方式可以允許部署自訂 AMI，或對 Amazon EKS 最佳化 AMI 中的 bootstrap.sh 指令碼提供引數。如需每個可用選項和預設值的完整清單，請輸入以下命令。

   ```
   eksctl create nodegroup --help
   ```

   在以下命令中，將 *my-cluster* 取代為您的叢集名稱，並將 *my-mng* 取代為您的節點群組名稱。節點群組名稱不可超過 63 個字元。它必須以字母或數字開頭，但也可以包含其餘字元的連字號和底線。

> **Important**
>
> 如果在第一次建立受管的節點群組時未使用自訂啟動範本，請勿在之後針對節點群組使用自訂啟動範本。如果沒有指定自訂啟動範本，系統會自動產生啟動範本 (不建議手動修改該範本)。手動修改此自動產生的啟動範本可能會導致錯誤。

沒有啟動範本

`eksctl` 會在您的帳戶中建立預設 Amazon EC2 啟動範本，並使用根據您指定之選項建立的啟動範本來部署節點群組。指定 `--node-type` 的值之前，請參閱 the section called “Amazon EC2 執行個體類型”。

使用允許的關鍵字取代 *`ami-family`*。如需詳細資訊，請參閱 `eksctl` 文件中的 Setting the node AMI Family (設定節點 AMI 系列)。使用 Amazon EC2 金鑰對或公有金鑰的名稱取代 *`my-key`*。此金鑰會在節點啟動後用於將 SSH 套用至節點。

> **Note**
>
> 對於 Windows，此命令不會啟用 SSH。而是將 Amazon EC2 金鑰對與執行個體建立關聯，讓您可以 RDP 至該執行個體中。

如果您還沒有 Amazon EC2 金鑰對，可以在 AWS 管理主控台中建立一個。如需 Linux 資訊，請參閱《Amazon EC2 使用者指南》中的 Amazon EC2 使用者指南。如需 Windows 資訊，請參閱《Amazon EC2 使用者指南》中的 Amazon EC2 金鑰對和 Windows 執行個體。

如果下列條件為真，我們建議封鎖 Pod 對 IMDS 的存取：

- 您計劃將 IAM 角色指派給您的所有 Kubernetes 服務帳戶，以便 Pod 僅具有所需的最低許可。
- 叢集中沒有任何 Pod 因其他原因需要存取 Amazon EC2 執行個體中繼資料服務 (IMDS)，例如擷取目前 AWS 區域。

如需詳細資訊，請參閱限制存取指派給工作節點的執行個體設定檔。

如果要封鎖 Pod 對 IMDS 的存取，請將 `--disable-pod-imds` 選項新增至下列命令。

```
eksctl create nodegroup \
```

```
  --cluster my-cluster \
  --region region-code \
  --name my-mng \
  --node-ami-family ami-family \
  --node-type m5.large \
  --nodes 3 \
  --nodes-min 2 \
  --nodes-max 4 \
  --ssh-access \
  --ssh-public-key my-key
```

您的執行個體可以選擇性地為 Pod 指派更多的 IP 位址、將 IP 位址指派給不同於執行個體之 CIDR 區塊的 Pod，以及在沒有網際網路存取的情況下部署到叢集。如需詳細資訊，請參閱 the section called “增加 IP 位址”、the section called “自訂聯網” 和 the section called “私有叢集” 以取得要新增至之前命令的其他選項。

受管節點群組會根據執行個體類型，計算和套用可在節點群組每個節點上執行的 Pod 數目上限的單一值。如果建立建具有不同執行個體類型的節點群組，則在所有執行個體類型中計算的最小值，會應用為可以在節點群組中的每種執行個體類型上執行的 Pod 數量上限。如需如何計算此值的詳細資訊，請參閱 the section called “maxPods的判斷方式”。

使用啟動範本

啟動範本必須已經存在，且必須符合啟動範本組態基礎知識中指定的要求。如果下列條件為真，我們建議封鎖 Pod 對 IMDS 的存取：

- 您計劃將 IAM 角色指派給您的所有 Kubernetes 服務帳戶，以便 Pod 僅具有所需的最低許可。
- 叢集中沒有任何 Pod 因其他原因需要存取 Amazon EC2 執行個體中繼資料服務 (IMDS)，例如擷取目前 AWS 區域。

如需詳細資訊，請參閱限制存取指派給工作節點的執行個體設定檔。

如果您要封鎖 Pod 對 IMDS 的存取，則請在啟動範本中指定必要的設定。

a. 將以下內容複製到您的裝置。取代範例值，然後執行修改後的命令來建立 `eks-nodegroup.yaml` 檔案。在不使用啟動範本的情況下部署時指定的數個設定會移至啟動範本。如果您未指定 `version`，即使用範本的預設版本。

```
cat >eks-nodegroup.yaml <<EOF
apiVersion: eksctl.io/v1alpha5
```

```
kind: ClusterConfig
metadata:
  name: my-cluster
  region: region-code
managedNodeGroups:
- name: my-mng
  launchTemplate:
    id: lt-id
    version: "1"
EOF
```

如需 eksctl 組態檔案設定的完整清單，請參閱 eksctl 文件中的組態檔案結構描述。您的執行個體可以選擇性地為 Pod 指派更多的 IP 位址、將 IP 位址指派給不同於執行個體之 CIDR 區塊的 Pod，以及在沒有傳出網際網路存取的情況下部署到叢集。如需詳細資訊，請參閱適用於新增至組態檔案之其他選項的 the section called “增加 IP 位址”、the section called “自訂聯網” 和 the section called “私有叢集”。

如果您未在啟動範本中指定 AMI ID，受管節點群組會根據執行個體類型，計算和套用可在節點群組每個節點上執行的 Pod 數目上限的單一值。如果建立建具有不同執行個體類型的節點群組，則在所有執行個體類型中計算的最小值，會應用為可以在節點群組中的每種執行個體類型上執行的 Pod 數量上限。如需如何計算此值的詳細資訊，請參閱 the section called “maxPods的判斷方式”。

如果在啟動範本中指定了 AMI ID，而且正使用自訂聯網或想增加指派給執行個體的 IP 位址數量，請指定可在節點群組的每個節點上執行的 Pod 數量上限。如需詳細資訊，請參閱the section called “maxPods的判斷方式”。

b. 使用下列命令部署節點群組。

```
eksctl create nodegroup --config-file eks-nodegroup.yaml
```

AWS 管理主控台

使用 AWS 管理主控台 建立受管節點群組

1. 等待您的叢集狀態顯示為 ACTIVE。您無法針對尚未進入 ACTIVE 狀態的叢集建立受管節點群組。
2. 開啟 Amazon EKS 主控台。
3. 選擇要在其中建立受管節點群組的叢集名稱。
4. 選取 Compute (運算) 標籤。
5. 選擇 Add node group (新增節點群組)。

6. 在 Configure node group (配置節點群組) 頁面上，相應地填寫參數，然後選擇 Next (下一步)。

- Name (名稱) – 輸入受管節點群組的唯一名稱。節點群組名稱不可超過 63 個字元。它必須以字母或數字開頭，但也可以包含其餘字元的連字號和底線。
- Node IAM role (節點 IAM 角色)：選擇要與節點群組搭配使用的節點執行個體角色。如需詳細資訊，請參閱the section called “節點 IAM 角色”。

> ⚠ Important
>
> - 您無法使用用於建立任何叢集的相同角色。
> - 建議您使用任何自我管理節點群組目前未使用的角色，除非您計劃將其用於新的自我管理節點群組。如需詳細資訊，請參閱the section called “刪除”。

- Use launch template (使用啟動範本)：(選用) 選擇是否要使用現有的啟動範本。選取啟動範本名稱。然後，選取 Launch template version (啟動範本版本)。如果您沒有選取版本，則 Amazon EKS 會使用範本的預設版本。啟動範本允許對節點群組進行更多自訂，如允許您部署自訂 AMI、為 Pod 指派更多的 IP 位址、將 IP 位址指派給不同於執行個體之 CIDR 區塊的 Pod，以及將節點部署至沒有出站網際網路存取的叢集。如需詳細資訊，請參閱the section called “增加 IP 位址”、the section called “自訂聯網”及the section called “私有叢集”。

  啟動範本必須滿足透過啟動範本來自訂受管節點中的要求。如果您不使用自己的啟動範本，Amazon EKS API 會在您的帳戶中建立預設的 Amazon EC2 啟動範本，並使用預設啟動範本部署節點群組。

  如果您為服務帳戶實作 IAM 角色，請直接將必要的許可指派給需要存取 AWS 服務的每個 Pod，而且叢集中沒有任何 Pod 需要存取 IMDS，例如擷取目前 AWS 區域，則您也可以停用在啟動範本中未使用主機聯網之 Pod 的 IMDS 存取權。如需詳細資訊，請參閱限制存取指派給工作節點的執行個體設定檔。
- Kubernetes labels (Kubernetes 標籤) – (選用) 您可以選擇將 Kubernetes 標籤套用至受管節點群組中的節點。
- Kubernetes 污點 – (選用) 您可以選擇將 Kubernetes 污點套用至受管節點群組中的節點。Effect (效果) 功能表中的可用選項是 **`NoSchedule`**、**`NoExecute`** 及 **`PreferNoSchedule`**。如需詳細資訊，請參閱the section called “阻止在特定節點上排程 Pod”。
- Tags (標籤) – (選用) 您可以選擇標記 Amazon EKS 受管節點群組。這些標籤不會傳播到節點群組中的其他資源，例如 Auto Scaling 群組或執行個體。如需詳細資訊，請參閱the section called “標記您的 資源”。

7. 在 Set compute and scaling configuration (設定運算和擴展組態) 頁面上，相應地填寫參數，然後選擇 Next (下一步)。
   - AMI 類型：選取 AMI 類型。如果您在部署 Arm 執行個體，請務必在部署前檢閱 Amazon EKS 最佳化 Arm Amazon Linux AMI 中的考量事項。

     如果您在上一頁中已指定啟動範本，並在啟動範本中指定了 AMI，則無法選取值。系統會顯示範本中的值。範本中指定的 AMI 必須符合指定 AMI 中的要求。
   - Capacity type (容量類型) – 選取容量類型。如需選擇容量類型的詳細資訊，請參閱 the section called “受管節點群組容量類型”。您無法在同一節點群組中混合使用不同的容量類型。如果要同時使用這兩種容量類型，請建立個別的節點群組，每個群組都有自己的容量和執行個體類型。如需佈建和擴展 GPU 加速工作者節點的相關資訊the section called “受管節點群組和自我管理節點”，請參閱 。
   - Instance types (執行個體類型)：依預設會指定一或多個執行個體類型。若要移除預設執行個體類型，請選取執行個體類型的右側的 X。選擇要在受管節點群組中使用的執行個體類型。如需詳細資訊，請參閱the section called “Amazon EC2 執行個體類型”。

     主控台會顯示一組常用的執行個體類型。如果您需要使用未顯示的執行個體類型建立受管節點群組，請使用 `eksctl`、 CLI、 AWS CloudFormation AWS 或 SDK 來建立節點群組。如果您在上一頁指定了啟動範本，則無法選取值，因為必須在啟動範本中指定執行個體類型。系統會顯示啟動範本中的值。如果為 Capacity type (容量類型) 選取了 Spot，則建議您指定多個執行個體類型以增強可用性。
   - Disk size (磁碟大小) – 輸入要用於節點根磁碟區的磁碟大小 (以 GiB 為單位)。

     如果您在上一頁指定了啟動範本，則無法選取值，因為必須在啟動範本中指定該值。
   - Desired size (所需大小) – 指定受管節點群組目前在啟動時應該維護的節點數量。

     > **Note**
     >
     > Amazon EKS 不會自動擴展或縮減您的節點群組。不過，您可以設定 Kubernetes Cluster Autoscaler 為您執行這項操作。如需詳細資訊，請參閱 Cluster Autoscaler on AWS。
   - Minimum size (大小下限) – 指定受管節點群組可縮減至的節點數量下限。
   - Maximum size (大小上限)：指定受管節點群組可擴增至的節點數量上限。
   - Node group update configuration (節點群組更新組態) – (選用) 您可以選取要平行更新的節點數量或百分比。這些節點在更新期間將無法使用。對於 Maximum unavailable (無法使用的上限)，選取下列其中一個選項，然後指定 Value (值)：

- Number (數量)：選取並指定可平行更新節點群組的節點數量。
- Percentage (百分比)：選取並指定可平行更新節點群組的節點百分比。如果您的節點群組中有大量節點，這會很有用。

- 節點自動修復組態 - (選用) 如果您啟用啟用節點自動修復核取方塊，Amazon EKS 將在偵測到問題發生時自動取代節點。如需詳細資訊，請參閱the section called “節點運作狀態和修復”。
- 暖集區組態 – （選用） 如果您啟用啟用暖集區組態核取方塊，Amazon EKS 將在 ASG 上建立暖集區。如需詳細資訊，請參閱the section called “暖集區”。

8. 在 Specify networking (指定聯網) 頁面上，據此填寫參數，然後選擇 Next (下一步)。
   - Subnets (子網路)：選擇要將受管節點啟動至其中的子網路。

> **Important**
>
> 如果您跨越多個由 Amazon EBS 磁碟區提供的可用區域執行狀態應用程式，並且使用 Kubernetes Cluster Autoscaler，您應該設定多個節點群組，每個群組的範圍皆為單一可用區域。另外，您應該啟用 `--balance-similar-node-groups` 功能。

> **Important**
>
> - 如果選擇公有子網路，而且您的叢集只啟用了公有 API 伺服器端點，則子網路的必須將 `MapPublicIPOnLaunch` 設定為 `true`，讓執行個體成功加入叢集。如果子網路是在 2020 年 3 月 26 日或之後使用 `eksctl`或 Amazon EKS vended AWS CloudFormation 範本建立，則此設定已設定為 `true`。如果在 2020 年 3 月 26 日之前使用 `eksctl`或 AWS CloudFormation 範本建立子網路，則需要手動變更設定。如需詳細資訊，請參閱修改子網的公有 IPv4 定址屬性。
> - 如果使用啟動範本並指定多個網路介面，Amazon EC2 便不會自動指派公有 `IPv4` 位址，即使 `MapPublicIpOnLaunch` 已設定為 `true`。對於在此案例中加入叢集的節點，您必須啟用叢集的私有 API 伺服器端點，或使用透過其他方法 (例如 NAT Gateway) 提供的出站網際網存取，啟動私有子網路中的節點。如需詳細資訊，請參閱《Amazon EC2 使用者指南》中的 Amazon EC2 執行個體 IP 定址。

- 設定 SSH 存取節點 (選用)。啟用 SSH 可讓您連接到執行個體，並在發生問題時收集診斷資訊。強烈建議您在建立節點群組時啟用遠端存取。您無法在建立節點群組之後啟用遠端存取。

如果您選擇使用啟動範本，則不會顯示此選項。若要啟用節點的遠端存取，請在啟動範本中指定金鑰對，並確定您在啟動範本中指定之安全群組中的節點已開啟適當的連接埠。如需詳細資訊，請參閱the section called “使用自訂安全群組”。

> **Note**
>
> 對於 Windows，此命令不會啟用 SSH。而是將 Amazon EC2 金鑰對與執行個體建立關聯，讓您可以 RDP 至該執行個體中。

- 對於 SSH key pair (SSH 金鑰對) (選用)，選擇要使用的 Amazon EC2 SSH 金鑰。如需 Linux 資訊，請參閱《Amazon EC2 使用者指南》中的 Amazon EC2 使用者指南。如需 Windows 資訊，請參閱《Amazon EC2 使用者指南》中的 Amazon EC2 金鑰對和 Windows 執行個體。如果您選擇使用啟動範本，則無法選取範本。當使用 Bottlerocket AMI 為節點群組提供 Amazon EC2 SSH 金鑰時，也會啟用管理容器。如需詳細資訊，請參閱 GitHub 上的管理容器。
- 對於 Allow SSH remote access from (允許 SSH 遠端存取來自)，如果要限制對特定執行個體的存取，請選取與這些執行個體相關聯的安全群組。如果沒有選取特定的安全群組，則允許從網際網路 (`0.0.0.0/0`) 上的任何地方存取 SSH。

9. 在 Review and create (檢閱並建立) 頁面上，檢閱您的受管節點群組組態，然後選擇 Create (建立)。

   如果節點無法加入叢集，請參閱「故障診斷」章節中的 the section called “節點無法加入叢集”。

10. 查看節點的狀態，並等待他們到達 `Ready` 狀態。

```
kubectl get nodes --watch
```

11. (僅限 GPU 節點) 若您選擇了 GPU 執行個體類型和 Amazon EKS 最佳化加速 AMI，然後必須套用 Kubernetes 專用 NVIDIA 裝置外掛程式作為叢集上的 DaemonSet。請先以您想要的 NVIDIA/k8s-device-plugin 版本來取代 *`vX.X.X`*，然後再執行下列命令。

```
kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/vX.X.X/
deployments/static/nvidia-device-plugin.yml
```

安裝 Kubernetes 附加元件

現在您已具備含節點之可正常運作的 Amazon EKS 叢集，您即可開始安裝 Kubernetes 附加元件並將應用程式部署到叢集。以下文件主題可協助您擴展叢集的功能。

- 建立叢集的 IAM 主體是唯一可以使用 `kubectl` 或 AWS 管理主控台對 Kubernetes API 伺服器進行呼叫的主體。如果要其他 IAM 主體能夠存取叢集，則需新增這些主體。如需詳細資訊，請參閱the section called “Kubernetes API 存取”及the section called “所需的許可”。
- 如果下列條件為真，我們建議封鎖 Pod 對 IMDS 的存取：
  - 您計劃將 IAM 角色指派給您的所有 Kubernetes 服務帳戶，以便 Pod 僅具有所需的最低許可。
  - 叢集中沒有任何 Pod 因其他原因需要存取 Amazon EC2 執行個體中繼資料服務 (IMDS)，例如擷取目前 AWS 區域。

  如需詳細資訊，請參閱限制存取指派給工作節點的執行個體設定檔。
- 設定 Kubernetes Cluster Autoscaler，以自動調整節點群組中的節點數目。
- 將範例應用程式部署至叢集。
- 使用管理叢集所需的重要工具來組織和監控叢集資源。

使用具有受管節點群組的暖集區，減少開機時間較長之應用程式的延遲

當您的應用程式有很長的初始化或開機時間時，向外擴展事件可能會導致延遲 - 新的節點必須完全開機並加入叢集，才能排程 Pod。此延遲可能會影響流量尖峰或快速擴展事件期間的應用程式可用性。暖集區透過維護已完成開機程序的預先初始化 EC2 執行個體集區來解決此問題。在橫向擴展事件期間，執行個體會直接從暖集區移至叢集，略過耗時的初始化步驟，並大幅減少新容量變成可用所需的時間。如需詳細資訊，請參閱《Amazon EC2 Auto Scaling 使用者指南》中的使用暖集區縮短開機時間的應用程式延遲。 Amazon EC2 Auto Scaling

Amazon EKS 受管節點群組支援 Amazon EC2 Auto Scaling 暖集區。暖集區會與 Auto Scaling 群組一起維護預先初始化的 EC2 執行個體，以便在橫向擴展事件期間快速加入叢集。暖集區中的執行個體已完成開機初始化程序，並且可以保持在 `Stopped`、 `Running`或 `Hibernated` 狀態。

Amazon EKS 使用`AWSServiceRoleForAmazonEKSNodegroup`服務連結角色在整個節點群組生命週期中管理暖集區，以建立、更新和刪除暖集區資源。

運作方式

當您設定暖集區時，Amazon EKS 會建立連接到節點群組 Auto Scaling 群組的 EC2 Auto Scaling 暖集區。執行個體會在暖集區中啟動、完成開機初始化程序，並保持設定狀態 (`Running`、 `Stopped`或 `Hibernated`)，直到需要為止。在橫向擴展事件期間，執行個體會從暖集區移至 Auto Scaling 群組，完成 Amazon EKS 初始化程序以加入叢集，並可用於 Pod 排程。啟用執行個體重複使用時，執行個體可以在縮減事件期間返回暖集區。

> **Important**
>
> 一律使用 `create-nodegroup`或 透過 Amazon EKS API 設定暖集區`update-nodegroup-config`。請勿使用 EC2 Auto Scaling API 手動修改暖集區設定，因為這可能會導致與 資源的 Amazon EKS 管理發生衝突。

考量事項

> **Important**
>
> 在設定暖集區之前，請參閱《Amazon EC2 Auto Scaling 使用者指南》中的 Amazon EC2 Auto Scaling 暖集區中的先決條件和限制。 Amazon EC2 Auto Scaling 並非所有執行個體類型、AMIs或組態都受到支援。

- IAM 許可 – `AWSServiceRoleForAmazonEKSNodegroup`服務連結角色（使用第一個受管節點群組自動建立）包含必要的暖集區管理許可。
- AMI 限制 – 暖集區不支援自訂 AMIs。您必須使用 Amazon EKS 最佳化 AMIs。
- Bottlerocket 限制 – 如果使用 Bottlerocket AMIs，則不支援`Hibernated`集區狀態。僅限使用`Stopped`或`Running`集區狀態。此外， Bottlerocket AMIs 不支援`reuseOnScaleIn`此功能。
- 休眠支援 – 集`Hibernated`區狀態僅支援特定執行個體類型。如需支援的執行個體類型，請參閱《Amazon EC2 使用者指南》中的休眠先決條件。
- 成本影響 – 在不需要時建立暖集區可能會導致不必要的成本。
- 容量規劃 – 根據擴展模式調整暖集區大小，以平衡成本和可用性。從 10-20% 的預期尖峰容量開始。
- VPC 聯網 – 確保 Auto Scaling 群組和暖集區執行個體都有足夠的 IP 地址。

設定暖集區

您可以在建立新的受管節點群組時設定暖集區，或更新現有的受管節點群組以新增暖集區支援。

組態參數

- enabled –（布林值）表示您將暖集區連接至受管節點群組的意圖。啟用暖集區支援時需要。
- maxGroupPreparedCapacity –（整數）跨暖集區和 Auto Scaling 群組合併的執行個體總數上限。

- minSize –（整數）在暖集區中維護的執行個體數目下限。預設：0。
- poolState –（字串）暖集區執行個體的狀態。預設：Stopped。
- reuseOnScaleIn –（布林值）執行個體是否在縮減事件期間返回暖集區，而不是終止它們。預設：false。Bottlerocket AMIs不支援。

使用 AWS CLI

您可以在建立受管節點群組時設定暖集區，或將暖集區新增至現有的節點群組。

使用暖集區建立節點群組

```
aws eks create-nodegroup \
  --cluster-name my-cluster \
  --nodegroup-name my-nodegroup \
  --node-role arn:aws:iam::111122223333:role/AmazonEKSNodeRole \
  --subnets subnet-12345678 subnet-87654321 \
  --region us-east-1 \
  --scaling-config minSize=2,maxSize=10,desiredSize=3 \
  --warm-pool-config
 enabled=true,maxGroupPreparedCapacity=8,minSize=2,poolState=Stopped,reuseOnScaleIn=true
```

將暖集區新增至現有的節點群組

```
aws eks update-nodegroup-config \
  --cluster-name my-cluster \
  --nodegroup-name my-nodegroup \
  --region us-east-1 \
  --warm-pool-config
 enabled=true,maxGroupPreparedCapacity=8,minSize=2,poolState=Stopped,reuseOnScaleIn=true
```

更新組態

隨時使用 更新暖集區設定update-nodegroup-config。現有的暖集區執行個體不會立即受到影響；新設定適用於更新後進入暖集區的執行個體。

```
aws eks update-nodegroup-config \
  --cluster-name my-cluster \
  --nodegroup-name my-nodegroup \
  --region us-east-1 \
```

```
  --warm-pool-config
enabled=true,maxGroupPreparedCapacity=10,minSize=3,poolState=Running,reuseOnScaleIn=true
```

若要停用連接到節點群組的暖集區，請設定 `enabled=false`：

```
aws eks update-nodegroup-config \
  --cluster-name my-cluster \
  --nodegroup-name my-nodegroup \
  --region us-east-1 \
  --warm-pool-config enabled=false
```

其他資源

- 《Amazon EC2 Auto Scaling 使用者指南》中的 Amazon EC2 Auto Scaling 的暖集區 Amazon EC2 Auto Scaling
- 使用受管節點群組簡化節點生命週期

更新叢集的受管節點群組

啟動受管節點群組更新後，Amazon EKS 會自動為您更新節點，完成了解節點更新的每個階段中列出的步驟。如果使用的是 Amazon EKS 最佳化 AMI，則 Amazon EKS 會自動將最新的安全修補程式和作業系統更新套用至您的節點，作為最新 AMI 發行版本的一部分。

有數個案例可用於更新 Amazon EKS 受管節點群組的版本或組態：

- 您已更新 Amazon EKS 叢集的 Kubernetes 版本，並且想要更新您的節點，以使用相同的 Kubernetes 版本。
- 新的 AMI 發行版本可供受管節點群組使用。如需 AMI 版本的詳細資訊，請參閱下列各節：
  - the section called “獲取版本資訊”
  - the section called “Bottlerocket”
  - the section called “獲取版本資訊”
- 您想要調整受管節點群組中執行個體的下限、上限或所需計數。
- 您想要從受管節點群組中的執行個體新增或移除 Kubernetes 標籤。
- 您想要從受管節點群組新增或移除 AWS 標籤。
- 您需要部署具有組態變更的新版啟動範本，例如更新的自訂 AMI。

- 您已部署 版本 `1.9.0`或更新版本的 Amazon VPC CNI 附加元件、啟用用於字首委派的附加元件，並希望節點群組中的新 AWS Nitro System 執行個體支援大幅增加的 Pod 數量。如需詳細資訊，請參閱the section called “增加 IP 位址”。
- 您已為 Windows 節點啟用 IP 字首委派，並希望節點群組中的新 AWS Nitro System 執行個體支援大幅增加的 Pod 數量。如需詳細資訊，請參閱the section called “增加 IP 位址”。

如果有較新的受管節點群組之 Kubernetes 版本的 AMI 發行版本，則可以更新節點群組的版本，以使用較新的 AMI 版本。同樣地，如果叢集執行的 Kubernetes 版本比節點群組新，則可以更新節點群組，以使用符合叢集 Kubernetes 版本的最新 AMI 發行版本。

當受管節點群組中的節點由於擴展操作或更新而終止時，該節點中的 Pod 叢集會先耗盡。如需詳細資訊，請參閱the section called “更新行為詳細資訊”。

更新節點群組版本

您可以透過以下任一項來更新節點群組版本：

- the section called “`eksctl`”
- the section called “AWS 管理主控台”

您更新的版本不能比控制平面的版本還新。

`eksctl`

使用 `eksctl` 來更新受管節點群組

使用下列命令，將受管節點群組更新至目前節點上所部署相同 Kubernetes 版本的最新 AMI 版本。使用您自己的值取代每一個*###*。

```
eksctl upgrade nodegroup \
  --name=node-group-name \
  --cluster=my-cluster \
  --region=region-code
```

> **Note**
>
> 如果要將使用啟動範本部署的節點群組升級至新的啟動範本版本，則請將 `--launch-template-version` *`version-number`* 新增至上述命令。啟動範本必須滿足透過啟動範本來自訂受管節點中所述的要求。如果啟動範本包含自訂 AMI，則 AMI 必須滿足指定 AMI 中的

要求。當您將節點群組升級至較新版本的啟動範本時，每個節點都會回收，以符合指定之啟動範本版本的新組態。

您無法直接將未部署啟動範本的節點群組升級至新的啟動範本版本。相反地，您必須使用啟動範本部署新的節點群組，才能將節點群組更新為新的啟動範本版本。

您可以將節點群組升級至與控制平面的 Kubernetes 版本相同的版本。例如，當您有一個正在執行 Kubernetes `1.36` 的叢集時，您可以使用下列命令，將目前執行 Kubernetes `1.35` 的節點升級到 `1.36` 版。

```
eksctl upgrade nodegroup \
  --name=node-group-name \
  --cluster=my-cluster \
  --region=region-code \
  --kubernetes-version=1.36
```

AWS 管理主控台

使用 AWS 管理主控台 來更新受管節點群組

1. 開啟 Amazon EKS 主控台。
2. 選擇包含要更新之節點群組的叢集。
3. 如果至少有一個節點群組有可用的更新，則頁面頂部會出現一個方塊，通知您可用的更新。如果選取運算索引標籤，您會看到針對有可用更新之節點群組的節點群組資料表中 AMI 發行版本欄位中的立即更新。若要更新節點群組，請選擇 Update now (立即更新)。

   您不會看到使用自訂 AMI 部署之節點群組的通知。如果節點是使用自訂 AMI 進行部署，則請完成以下步驟來部署新的已更新自訂 AMI。

   a. 建立 AMI 的新版本。

   b. 使用新的 AMI ID 建立新的啟動範本版本。

   c. 將節點升級至啟動範本的新版本。
4. 在 Update node group version (更新節點群組版本) 對話方塊中，啟用或停用以下選項：
   - Update node group version (更新節點群組版本)：如果您部署了自訂 AMI，或經 Amazon EKS 優化的 AMI 目前是叢集的最新版本，則無法使用此選項。
   - Change launch template version (變更啟動範本版本)：如果節點群組是在沒有自訂啟動範本的情況下進行部署，則無法使用此選項。您只能更新已使用自訂啟動範本部署之節點群組的啟動範本

版本。選取節點群組要更新至的 Launch template version (啟動範本版本)。如果節點群組設定了自訂 AMI，則您選取的版本也必須指定 AMI。當您升級至較新版本的啟動範本時，每個節點都會回收，以符合指定之啟動範本版本的新組態。

5. 對於 Update strategy (更新策略)，選取下列其中一個選項：
   - Rolling update (滾動更新) – 此選項會遵循叢集的 Pod 中斷預算。如果發生 Pod 中斷預算問題，導致 Amazon EKS 無法正常地消耗正在此節點群組上執行的 Pod，則更新會失敗。
   - 強制更新 – 此選項不會遵循 Pod 中斷預算。無論 Pod 中斷預算問題如何，都會強制節點重新啟動，進行更新。
6. 選擇 Update (更新)。

編輯節點群組組態

您可以修改受管節點群組中的部分組態。

1. 開啟 Amazon EKS 主控台。
2. 選擇包含要編輯之節點群組的叢集。
3. 選取 Compute (運算) 標籤。
4. 選取要編輯的節點群組，然後選擇 Edit (編輯)。
5. (選用) 在 Edit node group (編輯節點群組) 頁面上，執行以下作業：
   a. 編輯 Node group scaling configuration (節點群組擴展組態)。
      - Desired size (所需大小) – 指定受管節點群組目前應該維護的工作者節點數量。
      - Minimum size (大小下限) – 指定受管節點群組可縮減至的節點數量下限。
      - Maximum size (大小上限)：指定受管節點群組可擴增至的節點數量上限。如需節點群組中支援的節點數量上限，請參閱 the section called “Service Quotas”。
   b. (選用) 將 Kubernetes 標籤新增至節點群組中的節點或從中移除這些標籤。這裡顯示的標籤只是您使用 Amazon EKS 所套用的標籤。其他標籤可能存在於這裡未顯示的節點上。
   c. (選用) 將 Kubernetes 污點新增至節點群組中的節點或從中移除這些標籤。已新增的污點可能會有 **`NoSchedule`**、**`NoExecute`** 或 **`PreferNoSchedule`** 效果。如需詳細資訊，請參閱the section called “阻止在特定節點上排程 Pod”。
   d. (選用) 將 Tags (標籤) 新增至節點群組或從中移除標籤。這些標籤僅適用於 Amazon EKS 節點群組。標籤不會傳播到其他資源，例如節點群組中的子網路或 Amazon EC2 執行個體。
   e. (選用) 編輯 Node Group update configuration (節點群組更新組態)。選取任何一個 Number (數字) 或 Percentage (百分比)。

- Number (數量)：選取並指定可平行更新節點群組的節點數量。這些節點在更新期間將無法使用。
- Percentage (百分比)：選取並指定可平行更新節點群組的節點百分比。這些節點在更新期間將無法使用。如果您的節點群組中有許多節點，這會很有用。

f. 完成編輯後，請選擇儲存變更。

> **Important**
>
> 更新節點群組組態後，修改 `NodegroupScalingConfig` 與 Pod 中斷預算 (PDB) 不相符。與更新節點群組程序 (這會在升級階段期間消耗節點且與 PDB 相符) 不同的是，若更新擴展組態，會導致節點透過 Auto Scaling 群組 (ASG) 縮減呼叫立即終止。發生此情況不會考慮 PDB，無論您在縮減的目標大小為何。這意味著，若您減少 Amazon EKS 受管節點群組的 `desiredSize`，Pod 會在節點終止後立即移出，而不與任何 PDB 相符。

回復受管節點群組版本

如果您需要將叢集控制平面復原至先前的 Kubernetes 版本，您必須先復原受管節點群組，以確保符合 Kubernetes 版本扭曲政策。工作者節點無法執行比控制平面更新的版本。

您可以使用具有先前 Kubernetes 版本的 `UpdateNodegroupVersion` API 來復原受管節點群組：

```
aws eks update-nodegroup-version \
  --cluster-name my-cluster \
  --nodegroup-name my-nodegroup \
  --kubernetes-version 1.30 \
  --region us-west-2
```

節點群組轉返會遵守您設定的更新設定 (`maxUnavailable` 或 `maxUnavailablePercentage`) 和更新策略 （ 滾動或強制 ），與升級相同。

如需完整叢集轉返程序的詳細資訊，請參閱 the section called “復原 Kubernetes 版本”。

了解節點更新的每個階段

Amazon EKS 受管工作節點升級策略有四個不同的階段，如下節所述。

設定階段

設定階段具有下列步驟：

1. 它為與節點群組相關聯的 Auto Scaling 群組建立新的 Amazon EC2 啟動範本版本。新的啟動範本版本使用目標 AMI 或自訂啟動範本版本進行更新。
2. 它會更新 Auto Scaling 群組，以使用最新的啟動範本。
3. 它使用節點群組的 `updateConfig` 屬性決定要平行升級的節點數量上限。不可用節點上限配額為 100。預設值為一個節點。如需詳細資訊，請參閱《Amazon EKS API 參考》中的 updateConfig 屬性。

擴充規模階段

升級受管節點群組中的節點時，已升級的節點會在與正在升級的節點相同的可用區域中啟動。為了保證這種置放，我們使用 Amazon EC2 的可用區域重新平衡。如需詳細資訊，請參閱《Amazon EC2 Auto Scaling 使用者指南》中的可用區域容量重新平衡。為了滿足這項需求，我們可在受管節點群組中每個可用區域啟動最多兩個執行個體。

擴充規模階段具有下列步驟：

1. 它按下面的任一大小 (以較大者為准) 增加 Auto Scaling 群組的最大大小和所需大小：
   - Auto Scaling 群組部署的可用區域數量的兩倍。
   - 升級無法使用的上限。

     例如，如果節點群組有五個可用區域，而 `maxUnavailable` 為一個，則升級程序最多能啟動 10 個節點。但是，當 `maxUnavailable` 為 20 (或任何大於 10 的數字)，該程序會啟動 20 個新節點。
2. 擴展 Auto Scaling 群組後，會檢查使用最新配置的節點是否存在於節點群組之中。只有在符合下列條件時，此步驟才會成功：
   - 在節點所在的每個可用區域中，至少會啟動一個新節點。
   - 每個新節點都應該處於 `Ready` 狀態。
   - 新節點應該有 Amazon EKS 套用標籤。

     以下是一般節點群組中工作節點上的 Amazon EKS 套用標籤：
     - `eks.amazonaws.com/nodegroup-image=$amiName`
     - `eks.amazonaws.com/nodegroup=$nodeGroupName`

以下是自訂啟動範本或 AMI 節點群組中工作節點上的 Amazon EKS 套用標籤：

- `eks.amazonaws.com/nodegroup-image=$amiName`
- `eks.amazonaws.com/nodegroup=$nodeGroupName`
- `eks.amazonaws.com/sourceLaunchTemplateId=$launchTemplateId`
- `eks.amazonaws.com/sourceLaunchTemplateVersion=$launchTemplateVersion`

> **Note**
>
> 在未變更擴展組態的情況下啟動更新或升級時，工作流程會使用即時 Auto Scaling 群組值作為起點，而不是節點群組的儲存擴展組態。如需詳細資訊，請參閱the section called “受管節點群組概念”。

3. 它將節點標記為不可排程，以避免排程新的 Pod。它還使用 `node.kubernetes.io/exclude-from-external-load-balancers=true` 標記節點，以便在終止節點之前從負載平衡器中移除舊節點。

以下是在這個階段中會導致 `NodeCreationFailure` 錯誤的已知原因：

可用區域中的容量不足

可用區域可能沒有所請求執行個體類型的容量。建議在建立受管節點群組時設定多個執行個體類型。

您帳戶中的 EC2 執行個體限制

您可能需要增加帳戶可以使用 Service Quotas 同時執行的 Amazon EC2 執行個體的數量。如需詳細資訊，請參閱《Amazon Elastic Compute Cloud Linux 執行個體使用者指南》中的 EC2 Service Quotas。

自訂使用者資料

自訂使用者資料有時會中斷引導程序。這種情況可能會導致 `kubelet` 未在節點上啟動或節點上未取得預期的 Amazon EKS 標籤。如需詳細資訊，請參閱the section called “指定 AMI”。

讓節點運作狀態不佳或未就緒的任何變更

節點磁碟壓力、記憶體壓力和類似情況，可能導致節點無法進入 `Ready` 狀態。

每個節點必須在 15 分鐘內啟動

如果任何節點需要超過 15 分鐘才能完成引導並加入叢集，則會導致升級逾時。這是從需要新節點到新階段加入叢集時，測量的引導新節點的總執行時期。升級受管節點群組時，時間計數器會在 Auto Scaling 群組大小增加時立即啟動。

升級階段

升級階段有有兩種不同的行為方式，具體視更新策略而定。更新策略有兩種：預設和最低。

大多數情況下，我們建議使用預設策略。它會在終止舊節點之前建立新節點，以便在升級階段維持可用容量。在資源或成本受限制的情況下 (例如使用 GPU 等硬體加速器)，這種最低策略非常有用。它會在建立新的節點之前終止舊節點，因此總容量永遠不會超過您設定的數量。

預設更新策略包含下列步驟：

1. 它會增加 Auto Scaling 群組中的節點數量 (所需計數)，進而導致節點群組建立額外的節點。
2. 它隨機選取需要升級的節點，最大不超過為節點群組設定的無法使用上限。
3. 當新節點達到就緒狀態時，它會封鎖舊節點，防止服務控制器傳送新請求至該節點。
4. 其會耗盡節點中的 Pod。如果 Pod 沒有在 15 分鐘內離開節點，並且沒有強制旗標，則升級階段將失敗，並會出現 `PodEvictionFailure` 錯誤。在這種情況下，您可以應用強制旗標，同時使用 `update-nodegroup-version` 請求刪除 Pod。
5. 移出每個 Pod 後，它會等待 60 秒。然後，它會將終止請求傳送至 Auto Scaling 群組，並從作用中節點清單中移除節點。
6. 它重複步驟先前的升級步驟，直到節點群組中沒有使用舊版啟動範本部署的節點為止。

最低更新策略包含下列步驟：

1. 它會在開頭包圍隔離節點群組的所有節點，以便服務控制器不會將任何新請求傳送至這些節點。
2. 它隨機選取需要升級的節點，最大不超過為節點群組設定的無法使用上限。
3. 它會從選取的節點中耗盡 Pod。如果 Pod 沒有在 15 分鐘內離開節點，並且沒有強制旗標，則升級階段將失敗，並會出現 `PodEvictionFailure` 錯誤。在這種情況下，您可以應用強制旗標，同時使用 `update-nodegroup-version` 請求刪除 Pod。
4. 在每個 Pod 被移出並等待 60 秒後，其會向選取的節點的 Auto Scaling 群組傳送終止請求。Auto Scaling 群組會建立新的節點 (與選取的節點的數量相同)，以取代缺少的容量。

5. 它重複步驟先前的升級步驟，直到節點群組中沒有使用舊版啟動範本部署的節點為止。

升級階段期間的 **PodEvictionFailure** 錯誤

以下是在這個階段中會導致 PodEvictionFailure 錯誤的已知原因：

積極 PDB

在 Pod 上定義積極的 PDB，或有多個 PDB 指向同一個 Pod。

可容忍所有污點的部署

一旦移出每個 Pod，節點預期為空，因為節點在先前的步驟中會受到污染。但是，如果部署容忍每一個污點，則節點更有可能是非空白，導致 Pod 移出失敗。

縮減規模階段

縮減規模階段會將 Auto Scaling 群組的大小上限和所需大小遞減一，以便回到更新開始之前的值。

如果升級工作流程判斷 Cluster Autoscaler 在工作流程的縮減規模階段期間擴充節點群組，它會立即結束，而不會將節點群組恢復至原始大小。

> **Note**
>
> ```
> If your node group has a warm pool enabled, warm pool instances are drained
>  before the scale-up operation begins. This is because warm pool instances have
>  not been updated to the new launch template configuration – during the scale-
> up phase, they would be pulled into the Auto Scaling Group instead of launching
>  new instances with the updated configuration, which would break the upgrade
>  process. Draining the warm pool ensures that only new instances with the
>  updated configuration are launched. Once the scale-down operation completes,
>  the warm pool is restored, and the new instances in the warm pool are launched
>  with the updated launch template configuration.
> ```
>
> 如需有關暖集區的詳細資訊，請參閱 the section called “暖集區”。

使用啟動範本自訂受管節點

對於最高層級的自訂，您可以根據此頁面上的步驟，使用自己的啟動範本部署受管節點。使用啟動範本可讓 等功能在部署節點期間提供引導引數（例如，額外的 kubelet 引數）、從與指派給節點的 IP 地

址不同的 CIDR 區塊將 IP 地址指派給 Pod、將您自己的自訂 AMI 部署至節點，或將您自己的自訂 CNI 部署至節點。

如果您在第一次建立受管節點群組期間提供自己的啟動範本，稍後也會得到更出色的靈活性。只要使用自己的啟動範本部署受管節點群組，您就可以使用不同版本的相同啟動範本來反覆更新。將節點群組更新為不同版本的啟動範本時，群組中的所有節點都會回收，以符合指定啟動範本版本的新組態。

一律使用 Amazon EC2 Auto Scaling 群組所使用的啟動範本部署受管節點群組。如果未提供啟動範本，Amazon EKS API 會使用帳戶中的預設值自動建立此啟動範本。然而，我們不建議您修改自動產生的啟動範本。此外，未使用自訂啟動範本的現有節點群組無法直接更新。相反，您必須建立具有自訂啟動範本的新節點群組，才能執行此動作。

啟動範本組態基礎知識

您可以使用 、 AWS 管理主控台 AWS CLI 或 AWS SDK 建立 Amazon EC2 Auto Scaling 啟動範本。如需詳細資訊，請參閱《Amazon EC2 Auto Scaling 使用者指南》中的建立 Auto Scaling 群組的啟動範本。啟動範本中的某些設定與用於受管理節點組態的設定類似。使用啟動範本部署或更新節點群組時，必須在節點群組組態或啟動範本其中一個位置中指定某些設定。請勿同時在兩個位置指定設定。如果設定存在於不應該的地方，則建立或更新節點群組之類的動作會失敗。

下表會列出啟動範本中禁止的設定。該表也會列出受管節點群組組態中需要的類似設定 (如果有)。列出的設定是顯示在主控台中的設定。它們在 CLI 和 SDK AWS 中可能有類似但不同的名稱。

啟動範本：禁止	Amazon EKS 節點群組組態
Network interfaces (網路介面) (Add network interface (新增網路介面)) 下的 Subnet (子網)	Specify networking (指定聯網) 頁面上 Node Group network configuration (節點群組網路組態) 下的 Subnets (子網)
Advanced details (進階詳細資訊) 下的 IAM instance profile (IAM 執行個體描述檔)	Configure Node Group (設定節點群組) 頁面上 Node Group configuration (節點群組組態) 下的 Node IAM Role (節點 IAM 角色)
Advanced details (進階詳細資訊) 下的 Shutdown behavior (關機行為) 和 Stop - Hibernate behavior (停用 – 休眠行為)。為兩種設定在啟動範本中保留預設請勿包含在啟動範本設定中。	無同等。Amazon EKS 必須控制執行個體生命週期，而非 Auto Scaling 群組。

下表會列出受管節點群組組態中禁止的設定。該表也會列出類似的設定 (如果有)，這些設定在啟動範本中是必要設定。列出的設定是顯示在主控台中的設定。它們在 CLI 和 SDK AWS 中可能會有類似的名稱。

Amazon EKS 節點群組組態設定：禁止	啟動範本

(僅當您在啟動模板中指定了自訂 AMI 時) 在 Set compute and scaling configuration (設定運算和擴展組態) 頁面上 Node Group compute configuration (節點群組運算組態) 下的 AMI type (AMI 類型) – 主控台顯示 Specified in launch template (在啟動範本中指定) 以及指定的 AMI ID。
 	
如果未在啟動範本中指定應用程式和作業系統映像 (Amazon Machine Image)，您可在節點群組組態中選取 AMI。
	Launch template contents (啟動範本內容) 下的 Application and OS Images (Amazon Machine Image) (應用程式和作業系統映像 (Amazon Machine Image))：如果您有下列任一要求，則必須指定 ID：
	
	• 使用自訂 AMI。如果您指定的 AMI 不符合指定 AMI 中所列要求，則節點群組部署將會失敗。
	• 想要提供使用者資料將引數提供給 `bootstrap.sh` 檔案，其中包含在 Amazon EKS 最佳化 AMI。您可以啟用執行個體將大量 IP 位址指派給 Pod、將 IP 位址指派給不同於執行個體 CIDR 區塊的 Pod 或部署沒有傳出網際網路存取的私有叢集。如需詳細資訊，請參閱下列主題：
	  • 將更多 IP 位址指派給具有字首的 Amazon EKS 節點
	  • 使用自訂聯網在替代子網路中部署 Pod
	  • 部署網際網路存取受到限制的私有叢集
	  • 指定 AMI

Set compute and scaling configuration (設定運算和擴展組態) 頁面上 Node Group compute configuration (節點群組運算組態) 下的 Disk size (磁碟大小)：主控台顯示 Specified in launch template (在啟動範本中指定)。	Storage (Volumes) (儲存 (磁碟區)) (Add new volume (新增磁碟區)) 下的 Size (大小)。您必須在啟動範本中指定此選項。
Specify Networking (指定聯網) 頁面上 Node Group configuration (節點群組組態) 下的 SSH key pair (SSH 金鑰對) – 主控台會顯示在啟動	Key pair (login) (金鑰對 (登入)) 下的 Key pair name (金鑰對名稱)。

Amazon EKS 節點群組組態設定：禁止	啟動範本
範本中指定的金鑰，或顯示 Not specified in launch template (未在啟動範本中指定)。	
使用啟動範本時，您無法指定允許遠端存取的來源安全群組。	針對執行個體的 Network settings (網路設定) 下的 Security groups (安全群組) 或 Network interfaces (網路介面) 下的 Security groups (安全群組) (Add network interface (新增網路介面))，但不能兩者同時選擇。如需詳細資訊，請參閱the section called “使用自訂安全群組”。

Note

- 如果使用啟動範本部署節點群組，請在啟動範本的 Launch template contents (啟動範本內容) 中指定零或一個 Instance type (執行個體類型)。您也可以在主控台上的 Set compute and scaling configuration (設定運算和擴展組態) 頁面中，為 Instance types (執行個體類型) 指定 0 到 20 個執行個體類型。您還可以使用其他使用 Amazon EKS API 的工具來執行這項操作。如果在啟動範本中指定執行個體類型，並使用該啟動範本來部署節點群組，則無法在主控台中指定任何執行個體類型，或使用其他使用 Amazon EKS API 的工具。如果未在啟動範本、主控台中或使用其他使用 Amazon EKS API 的工具指定執行個體類型，請使用 `t3.medium` 執行個體類型。如果您的節點群組正在使用 Spot 容量類型，則建議使用主控台指定多個執行個體類型。如需詳細資訊，請參閱the section called “受管節點群組容量類型”。
- 如果您部署到節點群組的任何容器使用執行個體中繼資料服務版本 2，請確定在啟動範本中將 Metadata response hop limit (中繼資料回應躍點限制) 設定為 2。如需詳細資訊，請參閱《Amazon EC2 使用者指南》中的執行個體中繼資料與使用者資料。
- 啟動範本不支援允許靈活的執行個體類型選取的 `InstanceRequirements` 功能。

標記 Amazon EC2 執行個體

您可以使用啟動範本的 `TagSpecification` 參數，以指定要將哪些標籤套用至節點群組中的 Amazon EC2 執行個體。IAM 實體呼叫 `CreateNodegroup` 或 `UpdateNodegroupVersion` API 必須擁有 `ec2:RunInstances` 和 `ec2:CreateTags` 許可，而且標籤必須新增至啟動範本。

使用自訂安全群組

您可以使用啟動範本來指定自訂 Amazon EC2 安全群組以套用至節點群組中的執行個體。這可以是在執行個體層級安全群組參數中，也可以是網路介面組態參數的一部分。不過，您無法建立同時指定執行個體層級和網路介面安全群組的啟動範本。請考慮下列適用於搭配受管節點群組使用之自訂安全群組的條件：

- 使用 時 AWS 管理主控台，Amazon EKS 僅允許具有單一網路介面規格的啟動範本。
- 預設情況下，Amazon EKS 將叢集安全群組套用至節點群組中的執行個體，以促進節點與控制平面之間的通訊。如果使用先前提到的任一選項在啟動範本中指定自訂安全群組，則 Amazon EKS 不會新增叢集安全群組。因此，您必須確保安全群組的傳入和傳出規則可啟用與叢集端點的通訊。如果安全群組規則不正確，工作節點便無法加入叢集。如需安全群組規則的詳細資訊，請參閱 the section called “安全群組要求”。
- 如需 SSH 存取節點群組中的執行個體，請包含允許該存取的安全群組。

Amazon EC2 使用者資料

啟動範本包含自訂使用者資料的區段。您可以在此區段中指定節點群組的組態設定，無需手動建立個別自訂 AMI。如需有關 Bottlerocket 可用設定的詳細資訊，請參閱 GitHub 上的使用使用者資料。

您可以在啟動執行個體時使用 `cloud-init` 提供啟動範本中的 Amazon EC2 使用者資料。如需詳細資訊，請參閱 cloud-init 文件。您的使用者資料可用來執行一般組態操作。這包含下列操作：

- 包括使用者或群組
- 安裝套件

與受管節點群組搭配使用的啟動範本中的 Amazon EC2 使用者資料，必須為 Amazon Linux AMI 的 MIME 分段封存格式和 Bottlerocket AMI 的 TOML 格式。這是因為您的使用者資料與節點加入叢集所需的 Amazon EKS 使用者資料合併。請勿在啟動或修改 `kubelet` 的使用者資料中指定任何命令。這會作為 Amazon EKS 合併使用者資料的一部分執行。某些 `kubelet` 參數 (例如在節點上設定標籤) 可以透過受管節點群組 API 直接設定。

> **Note**
>
> 如果有關進階 `kubelet` 自訂 (包括手動啟動或傳入自訂組態參數) 的詳細資訊，請參閱 the section called “指定 AMI”。如果在啟動範本中指定自訂 AMI ID，Amazon EKS 不會合併使用者資料。

下列詳細資訊提供有關使用者資料區段的詳細資訊。

Amazon Linux 2 使用者資料

您可以將多個使用者資料區塊組合在一起成為單一 MIME 分段檔案。例如，您可以將設定 Docker 常駐程式的雲端 Boothook 與安裝自訂套件的使用者資料 Shell 指令碼合併。MIME 分段檔案包含下列元件：

- 內容類型和部分邊界宣告：`Content-Type: multipart/mixed; boundary="==MYBOUNDARY=="`
- MIME 版本宣告：`MIME-Version: 1.0`
- 包含以下元件的一或多個使用者資料區塊：
  - 開啟邊界，表示使用者資料區塊的開始 – `--==MYBOUNDARY==`
  - 區塊的內容類型宣告：`Content-Type: text/cloud-config; charset="us-ascii"`。如需內容類型的詳細資訊，請參閱 cloud-init 文件。
  - 使用者資料的內容 (例如，Shell 命令或 `cloud-init` 指令的清單)。
  - 結束邊界，表示 MIME 分段檔案的結束：`--==MYBOUNDARY==--`

以下是您可以用來建立自己的 MIME 分段檔案的範例。

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="==MYBOUNDARY=="

--==MYBOUNDARY==
Content-Type: text/x-shellscript; charset="us-ascii"

#!/bin/bash
echo "Running custom user data script"

--==MYBOUNDARY==--
```

Amazon Linux 2023 使用者資料

Amazon Linux 2023 (AL2023) 推出了新的節點初始化程序 `nodeadm`，該程序會使用 YAML 組態結構描述。如果您使用自我管理節點群組或具有啟動範本的 AMI，則在建立新節點群組時，需要明確提供額外的叢集中繼資料。以下連結展示了一個包含最低必要參數的範例，其中 `apiServerEndpoint`、`certificateAuthority` 和服務 `cidr` 現在是必需的：

```
---
apiVersion: node.eks.aws/v1alpha1
```

```
kind: NodeConfig
spec:
  cluster:
    name: my-cluster
    apiServerEndpoint: https://example.com
    certificateAuthority: Y2VydGlmaWNhdGVBdXRob3JpdHk=
    cidr: 10.100.0.0/16
```

您通常會在使用者資料中設定此組態，可以依原狀設定，也可以內嵌在 MIME 分段文件中：

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="BOUNDARY"

--BOUNDARY
Content-Type: application/node.eks.aws

---
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig spec: [...]

--BOUNDARY--
```

在 AL2 中，這些參數的中繼資料是透過 Amazon EKS `DescribeCluster` API 呼叫自動發現的。在 AL2023 中，此行為已改變，因為額外的 API 呼叫在大規模節點擴展時有被限流的風險。如果您使用沒有啟動範本的受管節點群組，或使用 Karpenter，則此變更不會對您造成影響。如需有關 `certificateAuthority` 和服務 `cidr` 的更多資訊，請參閱《Amazon EKS API 參考》中的 `DescribeCluster`。

以下是 AL2023 使用者資料的完整範例，其中結合用於自訂節點的 Shell 指令碼 (例如安裝套件或預先快取容器映像) 與所需的 nodeadm 組態。此範例顯示常見的自訂，包括：* 安裝其他系統套件 * 預先快取容器映像，以改善 Pod 啟動時間 * 設定 HTTP 代理組態 * 設定節點標籤的 `kubelet` 旗標

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="BOUNDARY"

--BOUNDARY
Content-Type: text/x-shellscript; charset="us-ascii"

#!/bin/bash
set -o errexit
set -o pipefail
set -o nounset
```

```
# Install additional packages
yum install -y htop jq iptables-services

# Pre-cache commonly used container images
nohup docker pull public.ecr.aws/eks-distro/kubernetes/pause:3.2 &

# Configure HTTP proxy if needed
cat > /etc/profile.d/http-proxy.sh << 'EOF'
export HTTP_PROXY="http://proxy.example.com:3128"
export HTTPS_PROXY="http://proxy.example.com:3128"
export NO_PROXY="localhost,127.0.0.1,169.254.169.254,.internal"
EOF

--BOUNDARY
Content-Type: application/node.eks.aws

apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name: my-cluster
    apiServerEndpoint: https://example.com
    certificateAuthority: Y2VydGlmaWNhdGVBdXRob3JpdHk=
    cidr: 10.100.0.0/16
  kubelet:
    config:
      clusterDNS:
      - 10.100.0.10
    flags:
    - --node-labels=app=my-app,environment=production

--BOUNDARY--
```

Bottlerocket 使用者資料

Bottlerocket 會以 TOML 格式建構使用者資料。您可以提供要與 Amazon EKS 提供的使用者資料合併的使用者資料。例如，您可以提供額外的 `kubelet` 設定。

```
[settings.kubernetes.system-reserved]
cpu = "10m"
memory = "100Mi"
ephemeral-storage= "1Gi"
```

如需有關受支援設定的詳細資訊，請參閱 Bottlerocket 文件。您可以在使用者資料中設定節點標籤和污點。不過，建議您改為在節點群組內設定。在執行這項操作時，Amazon EKS 會套用這些組態。

合併使用者資料時，不會保留格式設定，但內容保持不變。您在使用者資料中提供的組態會覆寫 Amazon EKS 所設定的任何設定。所以，如果您設定 `settings.kubernetes.max-pods` 或 `settings.kubernetes.cluster-dns-ip`，則使用者資料中的值會套用至節點。

Amazon EKS 不支援所有有效的 TOML。以下是已知不支援格式的清單：

- 引號鍵內的引號：`'quoted "value"' = "value"`
- 值中的溢出引號：`str = "I'm a string. \"You can quote me\""`
- 混合浮點數和正整數：`numbers = [ 0.1, 0.2, 0.5, 1, 2, 5 ]`
- 陣列中的混合類型：`contributors = ["foo@example.com", { name = "Baz", email = "baz@example.com" }]`
- 帶引號鍵的括號標題：`[foo."bar.baz"]`

Windows 使用者資料

Windows 使用者資料會使用 PowerShell 命令。建立受管節點群組時，您的自訂使用者資料會與 Amazon EKS 受管使用者資料結合。您的 PowerShell 命令為優先，然後是受管的使用者資料命令，所有這些命令都在一個 `<powershell></powershell>` 標籤中。

> **Important**
>
> 建立 Windows 節點群組時，Amazon EKS 會更新 `aws-auth ConfigMap`，以允許 Linux 型節點加入叢集。服務不會自動設定 Windows AMI 的許可。如果您使用的是 Windows 節點，您將需要透過存取項目 API 或直接更新 `aws-auth ConfigMap` 來管理存取。如需詳細資訊，請參閱the section called “啟用 Windows 支援”。

> **Note**
>
> 如果啟動範本中未指定任何 AMI ID，請勿在使用者資料中使用 Windows Amazon EKS 引導指令碼來設定 Amazon EKS。

範例使用者資料如下。

```
<powershell>
Write-Host "Running custom user data script"
</powershell>
```

指定 AMI

如果有下列其中一項要求，請在啟動範本 `ImageId` 欄位中指定 AMI ID。選取您對其他資訊的要求。

Example提供使用者資料將引數傳遞給 bootstrap.sh 檔案，其中包含 Amazon EKS 最佳化 Linux/Bottlerocket AMI

引導是一個術語，用於描述新增在執行個體啟動時可以執行的命令。例如，引導允許使用額外的 kubelet 引數。您可以使用 `eksctl` 將引數傳遞給 `bootstrap.sh` 而不指定啟動範本。您也可以在啟動範本的使用者資料區段中指定資訊來執行此動作。

eksctl，無需指定啟動範本

使用下列內容，建立名為 *my-nodegroup.yaml* 的檔案。使用您自己的值取代每一個*###*。`--apiserver-endpoint`、`--b64-cluster-ca`，和 `--dns-cluster-ip` 引數為選用。但是，定義引述會允許 `bootstrap.sh` 指令碼避免進行 `describeCluster` 呼叫。這在經常縮減和擴增節點的私有叢集設定或叢集中非常有用。如需 `bootstrap.sh` 指令碼的詳細資訊，請參閱 GitHub 上的 bootstrap.sh 檔案。

- 唯一需要的引數是叢集名稱 (*my-cluster*)。
- 若要擷取 `ami-`*1234567890abcdef0* 的最佳化 AMI ID，您請參閱下列章節：
  - 擷取建議的 Amazon Linux AMI ID
  - 擷取建議的 Bottlerocket AMI ID
  - 擷取建議的 Microsoft Windows AMI ID
- 若要擷取您的叢集的 *certificate-authority*，請執行下列命令。

```
aws eks describe-cluster --query "cluster.certificateAuthority.data" --output text
 --name my-cluster --region region-code
```

- 若要擷取您的叢集的 *api-server-endpoint*，請執行下列命令。

```
aws eks describe-cluster --query "cluster.endpoint" --output text --name my-
cluster --region region-code
```

- `--dns-cluster-ip` 的值是結尾處為 `.10` 的服務 CIDR。若要擷取您的叢集的 *`service-cidr`*，請執行下列命令。例如，如果傳回的值是 `ipv4 10.100.0.0/16`，則您的值是 *`10.100.0.10`*。

```
aws eks describe-cluster --query "cluster.kubernetesNetworkConfig.serviceIpv4Cidr"
 --output text --name my-cluster --region region-code
```

- 此範例使用其中包含 Amazon EKS 最佳化 AMI 的 `bootstrap.sh` 指令碼提供 `kubelet` 引數來設定自訂 `max-pods` 值。節點群組名稱不可超過 63 個字元。它必須以字母或數字開頭，但也可以包含其餘字元的連字號和底線。如需選取 *`my-max-pods-value`* 的說明，請參閱 the section called “maxPods的判斷方式”。如需使用受管節點群組時如何maxPods決定 的詳細資訊，請參閱 the section called “maxPods的判斷方式”。

```
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: my-cluster
  region: region-code

managedNodeGroups:
  - name: my-nodegroup
    ami: ami-1234567890abcdef0
    instanceType: m5.large
    privateNetworking: true
    disableIMDSv1: true
    labels: { x86-al2-specified-mng }
    overrideBootstrapCommand: |
      #!/bin/bash
      /etc/eks/bootstrap.sh my-cluster \
        --b64-cluster-ca certificate-authority \
        --apiserver-endpoint api-server-endpoint \
        --dns-cluster-ip service-cidr.10 \
        --kubelet-extra-args '--max-pods=my-max-pods-value' \
        --use-max-pods false
```

針對每一個可用的 `eksctl config` 檔案選項，請參閱 `eksctl` 文件中的 Config file schema (組態檔案結構描述)。此 `eksctl` 公用程式仍會為您建立啟動範本，並使用在 `config` 檔案提供的資料來填入其使用者資料。

使用下列命令來建立節點群組。

```
eksctl create nodegroup --config-file=my-nodegroup.yaml
```

啟動範本中的使用者資料

在啟動範本的使用者資料區段中指定下列資訊。使用您自己的值取代每一個*###*。--apiserver-endpoint、--b64-cluster-ca，和 --dns-cluster-ip 引數為選用。但是，定義引述會允許 bootstrap.sh 指令碼避免進行 describeCluster 呼叫。這在經常縮減和擴增節點的私有叢集設定或叢集中非常有用。如需 bootstrap.sh 指令碼的詳細資訊，請參閱 GitHub 上的 bootstrap.sh 檔案。

- 唯一需要的引數是叢集名稱 (*my-cluster*)。
- 若要擷取您的叢集的 *certificate-authority*，請執行下列命令。

```
aws eks describe-cluster --query "cluster.certificateAuthority.data" --output text
 --name my-cluster --region region-code
```

- 若要擷取您的叢集的 *api-server-endpoint*，請執行下列命令。

```
aws eks describe-cluster --query "cluster.endpoint" --output text --name my-
cluster --region region-code
```

- --dns-cluster-ip 的值是結尾處為 .10 的服務 CIDR。若要擷取您的叢集的 *service-cidr*，請執行下列命令。例如，如果傳回的值是 ipv4 10.100.0.0/16，則您的值是 *10.100.0.10*。

```
aws eks describe-cluster --query "cluster.kubernetesNetworkConfig.serviceIpv4Cidr"
 --output text --name my-cluster --region region-code
```

- 此範例使用其中包含 Amazon EKS 最佳化 AMI 的 bootstrap.sh 指令碼提供 kubelet 引數來設定自訂 max-pods 值。如需選取 *my-max-pods-value* 的說明，請參閱 the section called “maxPods的判斷方式”。如需使用受管節點群組時如何maxPods決定 的詳細資訊，請參閱 the section called “maxPods的判斷方式”。

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="==MYBOUNDARY=="

--==MYBOUNDARY==
Content-Type: text/x-shellscript; charset="us-ascii"
```

```
#!/bin/bash
set -ex
/etc/eks/bootstrap.sh my-cluster \
  --b64-cluster-ca certificate-authority \
  --apiserver-endpoint api-server-endpoint \
  --dns-cluster-ip service-cidr.10 \
  --kubelet-extra-args '--max-pods=my-max-pods-value' \
  --use-max-pods false

--==MYBOUNDARY==--
```

Example提供使用者資料將引數傳遞給 Start-EKSBootstrap.ps1 檔案，其中包含 Amazon EKS 最佳化 Windows AMI

引導是一個術語，用於描述新增在執行個體啟動時可以執行的命令。您可以使用 eksctl 將引數傳遞給 Start-EKSBootstrap.ps1 而不指定啟動範本。您也可以在啟動範本的使用者資料區段中指定資訊來執行此動作。

如果您想指定自訂 Windows AMI ID，請注意下列考量事項：

- 您必須使用啟動範本，並在使用者資料區段中提供所需引導命令。若要擷取所需的 Windows ID，您可以使用使用最佳化的 Windows AMI 建立節點中的資料表。
- 您需滿足幾個限制和條件。例如，您必須將 eks:kube-proxy-windows新增至 IAM Authenticator AWS 組態映射。如需詳細資訊，請參閱the section called “指定 AMI ID 時的限制和條件”。

在啟動範本的使用者資料區段中指定下列資訊。使用您自己的值取代每一個*###*。-APIServerEndpoint、-Base64ClusterCA，和 -DNSClusterIP 引數為選用。但是，定義引述會允許 Start-EKSBootstrap.ps1 指令碼避免進行 describeCluster 呼叫。

- 唯一需要的引數是叢集名稱 (*my-cluster*)。
- 若要擷取您的叢集的 *certificate-authority*，請執行下列命令。

```
aws eks describe-cluster --query "cluster.certificateAuthority.data" --output text --
name my-cluster --region region-code
```

- 若要擷取您的叢集的 *api-server-endpoint*，請執行下列命令。

```
aws eks describe-cluster --query "cluster.endpoint" --output text --name my-cluster
 --region region-code
```

- `--dns-cluster-ip` 的值是結尾處為 `.10` 的服務 CIDR。若要擷取您的叢集的 *`service-cidr`*，請執行下列命令。例如，如果傳回的值是 `ipv4 10.100.0.0/16`，則您的值是 *`10.100.0.10`*。

```
aws eks describe-cluster --query "cluster.kubernetesNetworkConfig.serviceIpv4Cidr" --
output text --name my-cluster --region region-code
```

- 如需其他引數，請參閱 the section called “引導指令碼組態參數”。

> **Note**
>
> 如果您使用的是自訂服務 CIDR，則需要使用 `-ServiceCIDR` 參數來對其進行指定。否則，叢集中 Pod 的 DNS 解析將會失敗。

```
<powershell>
[string]$EKSBootstrapScriptFile = "$env:ProgramFiles\Amazon\EKS\Start-EKSBootstrap.ps1"
& $EKSBootstrapScriptFile -EKSClusterName my-cluster `
  -Base64ClusterCA certificate-authority `
  -APIServerEndpoint api-server-endpoint `
  -DNSClusterIP service-cidr.10
</powershell>
```

Example由於特定安全、合規或內部政策要求，執行自訂 AMI

如需詳細資訊，請參閱《Amazon EC2 使用者指南》中的 Amazon Machine Images (AMI)。Amazon EKS AMI 建置規範包含用於建置基於 Amazon Linux 的自訂 Amazon EKS AMI 的資源和組態指令碼。如需詳細資訊，請參閱 GitHub 上的 Amazon EKS AMI 建置規範。若要建置使用其他作業系統安裝的自訂 AMI，則請參閱 GitHub 上的 Amazon EKS 樣本自訂 AMI。

在與受管節點群組搭配使用的啟動範本中，您無法使用 AMI ID 的動態參數參考。

> **Important**
>
> - 指定 AMI 時，Amazon EKS 不會針對叢集的控制平面版本驗證 AMI 中內嵌的 Kubernetes 版本。您有責任確保自訂 AMI 的 Kubernetes 版本符合 Kubernetes 版本扭曲政策：
>   - 節點上的`kubelet`版本不得比叢集版本更新

- 節點上的kubelet版本必須等於或最多 3 個次要版本 （適用於 Kubernetes 版本 1.28或更高版本），或最多 2 個次要版本 （適用於 Kubernetes 版本 1.27或更低版本）

  建立具有版本扭曲違規的受管節點群組可能會導致：

- 節點無法加入叢集
- 未定義的行為或 API 不相容
- 叢集不穩定或工作負載失敗

- 指定 AMI 時，Amazon EKS 不會合併任何使用者資料。相反，您必須負責提供所需的 bootstrap 命令，讓節點加入叢集。如果節點無法加入叢集，則 Amazon EKS CreateNodegroup 和 UpdateNodegroupVersion 動作也會失敗。

指定 AMI ID 時的限制和條件

以下是使用受管節點群組指定 AMI ID 所涉及的限制和條件：

- 您必須建立新的節點群組，才能在指定啟動範本的 AMI ID 與不指定 AMI ID 之間切換。
- 有較新的 AMI 版本可用時，您不會在主控台中收到通知。若要將節點群組更新為較新的 AMI 版本，您需要使用更新的 AMI ID 來建立新版本的啟動範本。然後，您需要使用新的啟動範本版本更新節點群組。
- 如果指定 AMI ID，則無法在 API 中設定下列欄位：
  - amiType
  - releaseVersion
  - version
- 如果您指定 AMI ID，則 API 的任何 taints 集合都會以非同步方式套用。若要在節點加入叢集之前套用污點，您必須使用 --register-with-taints 命令列旗標將污點傳遞給使用者資料的 kubelet。如需詳細資訊，請參閱 Kubernetes 文件中的 kubelet。
- 為 Windows 受管節點群組指定自訂 AMI ID 時，請將 eks:kube-proxy-windows新增至您的 AWS IAM Authenticator 組態映射。必須要有這項，DNS 才能正常運作。

  a. 開啟 AWS IAM Authenticator 組態映射以進行編輯。

  ```
  kubectl edit -n kube-system cm aws-auth
  ```

  b. 將此項目新增至與 Windows 節點關聯的每個 rolearn 下的 groups 清單中。您的組態映射看起來應該類似於 aws-auth-cm-windows.yaml。

```
- eks:kube-proxy-windows
```

c. 儲存檔案並結束您的文字編輯器。

- 對於任何使用自訂啟動範本的 AMI，受管節點群組的預設 `HttpPutResponseHopLimit` 已設定為 2。

從您的叢集刪除受管節點群組

本主題會描述可以如何刪除 Amazon EKS 受管節點群組。刪除受管節點群組時，Amazon EKS 會先將 Auto Scaling 群組的最小、最大和所需大小設定為零。這就會讓節點群組縮減規模。

在每個執行個體終止之前，Amazon EKS 會傳送訊號以耗盡該節點。在耗盡過程中，Kubernetes 會對節點上的每個 Pod 執行下列動作： 執行任何設定的`preStop`生命週期掛鉤、將`SIGTERM`訊號傳送至容器，然後等待 `terminationGracePeriodSeconds`正常關閉。如果節點在 5 分鐘後尚未耗盡，Amazon EKS 可讓 Auto Scaling 繼續強制終止執行個體。終止所有執行個體後，會刪除 Auto Scaling 群組。

> **Important**
>
> 如果刪除使用節點 IAM 角色的受管節點群組，且角色未被叢集中任何其他受管節點群組使用，則此角色會從 `aws-auth ConfigMap` 移除。如果叢集中有任何自我管理節點群組使用相同的節點 IAM 角色，則自我管理節點會轉為 `NotReady` 狀態。此外，叢集操作也會中斷。若要僅針對自我管理節點群組新增您正在使用之角色的映射，請參閱 the section called “建立存取項目”，前提是您的叢集的平台版本不低於使用 EKS 存取項目授予 IAM 使用者 Kubernetes 的存取權部分中列出的最低版本。如果您的平台版本早於存取項目所需的最低版本，您可以將項目新增回 `aws-auth ConfigMap`。如需詳細資訊，請在您的終端機中輸入 `eksctl create iamidentitymapping --help`。

您可以透過以下方式刪除受管節點群組：

- the section called “eksctl”
- the section called “AWS 管理主控台”
- the section called “AWS CLI”

eksctl

使用 **eksctl** 刪除受管節點群組

輸入以下命令。使用您自己的值取代每一個 <example value>。

```
eksctl delete nodegroup \
  --cluster <my-cluster> \
  --name <my-mng> \
  --region <region-code>
```

如需更多選項，請參閱 eksctl 文件中的刪除和耗盡節點群組。

AWS 管理主控台

使用 AWS 管理主控台 刪除受管節點群組

1. 開啟 Amazon EKS 主控台。
2. 在叢集頁面上，選擇包含要刪除之節點群組的叢集。
3. 在所選叢集頁面上，選擇運算索引標籤。
4. 在 Node Groups (節點群組) 區段中，選擇要刪除的節點群組。然後選擇刪除。
5. 在刪除節點群組確認對話方塊中，輸入節點群組的名稱。然後選擇刪除。

AWS CLI

使用 CLI AWS 刪除受管節點群組

1. 輸入以下命令。使用您自己的值取代每一個 <example value>。

```
aws eks delete-nodegroup \
  --cluster-name <my-cluster> \
  --nodegroup-name <my-mng> \
  --region <region-code>
```

2. 如果在 CLI 組態中cli_pager=設定 ，請使用鍵盤上的方向鍵來捲動回應輸出。完成後按下 q 鍵。

   如需更多選項，請參閱 CLI delete-nodegroup 命令參考中的 命令。 AWS

藉助自我管理節點來自行維護節點

叢集包含排程 Pod 的一或多個 Amazon EC2 節點。Amazon EKS 節點會在您的 AWS 帳戶中執行並透過叢集 API 伺服器端點連接至您叢集的控制平面。會根據 Amazon EC2 價格收費。如需詳細資訊，請參閱 Amazon EC2 定價。

一個叢集可以包含數個節點群組。每個節點群組包含一個或多個部署在 Amazon EC2 Auto Scaling 群組的節點。群組節點的執行個體類型可能會有所不同，例如透過 Karpenter 使用以屬性為基礎的執行個體類型選項時。節點群組的所有執行個體都必須使用 Amazon EKS 節點 IAM 角色。

Amazon EKS 提供專門的Amazon 機器映像 (AMI)，稱為Amazon EKS 最佳化 AMI。AMI 已設定為搭配 Amazon EKS 一起使用。它們的元件包括 `containerd`、`kubelet` 和 AWS IAM 驗證器。AMI 還包含特殊化啟動程序指令碼，使其可自動探索並連接至您叢集的控制平面。

如果使用 CIDR 區塊限制對叢集公有端點的存取，則建議您也啟用私有端點存取。這樣可以讓節點與叢集通訊。若不啟用私有端點，您指定用於公有存取的 CIDR 區塊必須包含來自 VPC 的輸出來源。如需詳細資訊，請參閱 the section called “叢集端點存取”。

若要將自我管理節點新增至 Amazon EKS 叢集，請參閱以下主題。如果手動啟動自我管理節點，請將以下標籤新增至每個節點，同時確保 `<cluster-name>` 與您的叢集相符。如需詳細資訊，請參閱新增和刪除個別資源的標籤。如果遵循以下指南中的步驟，系統會為您自動新增必要的標籤至節點。

金鑰	值
`kubernetes.io/cluster/<cluster-name>`	`owned`

> **Important**
>
> Amazon EC2 執行個體中繼資料服務 (IMDS) 中的標籤與 EKS 節點不相容。啟用執行個體中繼資料標籤後，不能在標籤值中使用正斜線 ('/')。此限制可能會導致無法啟動執行個體，使用 Karpenter 或叢集自動擴展期等節點管理工具時尤其如此，因為這些服務依賴於包含正斜線的標籤來提供適當的功能。

如需 Kubernetes 對節點普遍觀點的詳細資訊，請參閱 Kubernetes 文件中的節點。

主題

- 建立自我管理的 Amazon Linux 節點
- 建立自我管理的 Bottlerocket 節點
- 建立自我管理的 Microsoft Windows 節點
- 建立自我管理的 Ubuntu Linux 節點
- 更新您的叢集的自我管理節點

建立自我管理的 Amazon Linux 節點

本主題會說明如何啟動已向 Amazon EKS 叢集註冊的 Linux 節點的 Auto Scaling 群組。節點加入叢集後，您就可以將 Kubernetes 應用程式部署至其中。您也可以使用 `eksctl`或 啟動自我管理的 Amazon Linux 節點 AWS 管理主控台。如果您需要在 AWS Outposts 上啟動節點，請參閱 the section called “節點”。

- 現有 Amazon EKS 叢集。若要部署叢集，請參閱 the section called “建立 叢集”。如果您在已啟用 AWS Outposts、 AWS Wavelength 或 AWS Local Zones 的 AWS 區域中有子網路，則這些子網路不得在您建立叢集時傳入。
- 供節點使用的現有 IAM 角色。若要建立服務角色，請參閱the section called “節點 IAM 角色”。如果此角色沒有任何 VPC CNI 的政策，則 VPC CNI Pod 需要以下個別角色。
- (選用，但建議) Kubernetes 專用 Amazon VPC CNI 外掛程式的附加元件設定有自己的 IAM 角色，該角色已連接必要的 IAM 政策。如需詳細資訊，請參閱the section called “為 IRSA 設定”。
- 熟悉選擇最佳 Amazon EC2 節點執行個體類型中所列的考量事項。取決於您選擇的執行個體類型，您的叢集和 VPC 可能還有其他先決條件。

您可以使用下列任意一項啟動自我管理的 Linux 節點：

- the section called “`eksctl`”
- the section called “AWS 管理主控台”

`eksctl`

使用 **`eksctl`** 啟動自我管理的 Linux 節點

1. 安裝安裝在裝置`0.215.0`或 AWS CloudShell 上的`eksctl`命令列工具版本 或更新版本。如需有關安裝或更新 `eksctl` 的指示，請參閱 `eksctl` 文件中的安裝一節。

2. (選用) 如果 AmazonEKS_CNI_Policy 受管 IAM 政策已連接至您的 Amazon EKS 節點 IAM 角色，建議您改為將其指派給您與 Kubernetes aws-node 服務帳戶相關聯的 IAM 角色。如需詳細資訊，請參閱the section called “為 IRSA 設定”。
3. 以下命令會在現有的叢集建立節點群組。將 *al-nodes* 取代為您的節點群組名稱。節點群組名稱不可超過 63 個字元。它必須以字母或數字開頭，但也可以包含其餘字元的連字號和底線。使用您叢集的名稱取代 *my-cluster*。此名稱僅能使用英數字元 (區分大小寫) 和連字號。必須以英數字元開頭，且長度不可超過 100 個字元。名稱在您要建立叢集 AWS 的區域和 AWS 帳戶中必須是唯一的。使用您自己的值取代其餘*###*。依預設，系統會使用與控制平面相同的 Kubernetes 版本來建立節點。

   選擇 --node-type 的值之前，請檢閱選擇最佳 Amazon EC2 節點執行個體類型。

   使用 Amazon EC2 金鑰對或公有金鑰的名稱取代 *my-key*。此金鑰會在節點啟動後用於將 SSH 套用至節點。如果您還沒有 Amazon EC2 金鑰對，可以在 AWS 管理主控台中建立一個。如需詳細資訊，請參閱《Amazon EC2 使用者指南》中的 Amazon EC2 金鑰對。

   使用下列命令來建立您的節點群組。

   > **Important**
   >
   > 如果您想要將節點群組部署至 AWS Outposts、Wavelength 或 Local Zone 子網路，還有其他考量：
   >
   > - 在建立叢集時，不得傳入子網路。
   > - 您必須使用指定子網路和 `volumeType: gp2` 的組態檔案來建立節點群組。如需詳細資訊，請參閱 eksctl 文件中的從組態檔案建立節點群組和組態檔案結構描述。

```
eksctl create nodegroup \
  --cluster my-cluster \
  --name al-nodes \
  --node-type t3.medium \
  --nodes 3 \
  --nodes-min 1 \
  --nodes-max 4 \
  --ssh-access \
  --managed=false \
  --ssh-public-key my-key
```

若要部署節點群組：

- 其可以將比預設組態更大量的 IP 位址指派給 Pod，請參閱 the section called “增加 IP 位址”。
- 其可以從與執行個體不同的 CIDR 區塊將 IPv4 地址指派給 Pod，請參閱 the section called “自訂聯網”。
- 其可以將 IPv6 地址指派給 Pod 和服務，請參閱 the section called “IPv6”。
- 沒有傳入網際網路存取，請參閱 the section called “私有叢集”。

如需所有可用選項和預設值的完整清單，請輸入以下命令。

```
eksctl create nodegroup --help
```

如果節點無法加入叢集，請參閱「故障診斷」章節中的 the section called “節點無法加入叢集”。

範例輸出如下。建立節點時，會有數行輸出。輸出的最後幾行之一類似於以下的範例行。

```
[#]  created 1 nodegroup(s) in cluster "my-cluster"
```

4. (選用) 部署範例應用程式以測試您的叢集和 Linux 節點。
5. 如果下列條件為真，我們建議封鎖 Pod 對 IMDS 的存取：
   - 您計劃將 IAM 角色指派給您的所有 Kubernetes 服務帳戶，以便 Pod 僅具有所需的最低許可。
   - 叢集中沒有任何 Pod 因其他原因需要存取 Amazon EC2 執行個體中繼資料服務 (IMDS)，例如擷取目前 AWS 區域。

   如需詳細資訊，請參閱限制存取指派給工作節點的執行個體設定檔。

AWS 管理主控台

步驟 1：使用 AWS 管理主控台 啟動自我管理的 Linux 節點

1. 下載最新版本的 AWS CloudFormation 範本。

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2025-11-26/
amazon-eks-nodegroup.yaml
```

2. 等待您的叢集狀態顯示為 ACTIVE。如果在叢集處於作用中狀態之前啟動節點，則節點向叢集註冊將會失敗，導致必須再次啟動。
3. 開啟 AWS CloudFormation 主控台。

4. 選擇 Create stack (建立堆疊)，然後選取 With new resources (standard) (使用新資源 (標準))。
5. 針對 Specify template (指定範本)，選取 Upload a template file (上傳範本檔案)，然後選取 Choose file (選擇檔案)。
6. 選取您下載的 `amazon-eks-nodegroup.yaml` 檔案。
7. 選取下一步。
8. 在 Specify stack details (指定堆疊詳細資訊) 頁面上，據此填寫下列參數，然後選擇 Next (下一步)：
   - 堆疊名稱：為您的 AWS CloudFormation 堆疊選擇堆疊名稱。例如，您可以稱它為 *my-cluster-nodes*。此名稱僅能使用英數字元 (區分大小寫) 和連字號。必須以英數字元開頭，且長度不可超過 100 個字元。名稱在您要建立叢集 AWS 的區域和 AWS 帳戶中必須是唯一的。
   - ClusterName：輸入建立 Amazon EKS 叢集時使用的名稱。此名稱必須與叢集名稱相同，否則節點無法加入叢集。
   - ClusterControlPlaneSecurityGroup：從您在建立 VPC 時產生的 AWS CloudFormation 輸出中選擇 SecurityGroups 值。

     下列步驟顯示擷取適用群組的一種操作。

     a. 開啟 Amazon EKS 主控台。

     b. 選擇叢集的名稱。

     c. 選擇 Networking (網路) 索引標籤。

     d. 從 ClusterControlPlaneSecurityGroup 下拉式清單中選取時，請使用 Additional Security Group (其他安全群組) 值作為參考。
   - ApiServerEndpoint：輸入 EKS 叢集的 API 伺服器端點。這可在 EKS 叢集主控台的詳細資訊區段中找到
   - CertificateAuthorityData：輸入 base64 編碼的憑證授權機構資料，也可以在 EKS 叢集主控台的詳細資訊區段中找到。
   - ServiceCidr：輸入用於將 IP 地址配置到叢集內 Kubernetes 服務的 CIDR 範圍。這可在 EKS 叢集主控台的聯網索引標籤中找到。
   - AuthenticationMode：透過檢閱 EKS 叢集主控台中的存取索引標籤，選取 EKS 叢集中使用的身分驗證模式。
   - NodeGroupName：輸入節點群組的名稱。此名稱稍後可用於識別為您節點建立的 Auto Scaling 節點群組。節點群組名稱不可超過 63 個字元。它必須以字母或數字開頭，但也可以包含其餘字元的連字號和底線。
   - NodeAutoScalingGroupMinSize：輸入節點 Auto Scaling 群組可以縮減的最低節點數。

- NodeAutoScalingGroupDesiredCapacity：堆疊建立時，輸入欲擴展的所需節點數量。
- NodeAutoScalingGroupMaxSize：輸入節點 Auto Scaling 群組可以擴增的最大節點數。
- NodeInstanceType：選擇節點的執行個體類型。如需詳細資訊，請參閱the section called “Amazon EC2 執行個體類型”。
- NodeImageIdSSMParam：為變數 Kubernetes 版本預先填入最近 Amazon EKS 最佳化 Amazon Linux 2023 AMI 的 Amazon EC2 Systems Manager 參數。若要使用 Amazon EKS 支援不同的 Kubernetes 次要版本，請將 *1.XX* 取代為不同的支援的版本。建議指定與叢集相同的 Kubernetes 版本。

  您也可以將 *amazon-linux-2023* 取代為不同的 AMI 類型。如需詳細資訊，請參閱the section called “獲取最新 ID”。

  > **Note**
  >
  > Amazon EKS 節點 AMI 以 Amazon Linux 為基礎。您可以在 Amazon Linux 安全中心追蹤 Amazon Linux 2023 的安全或隱私權事件，或訂閱相關聯的 RSS 摘要。安全與隱私權事件包含問題的概觀、哪些套件受到影響，以及如何更新您的執行個體以修正問題。

- NodeImageId：（選用）如果您使用自己的自訂 AMI（而非 Amazon EKS 最佳化 AMI)，請輸入您 AWS 區域的節點 AMI ID。如果您在此指定值，則會覆寫 NodeImageIdSSMParam 欄位中的任何值。
- NodeVolumeSize：為您的節點指定根磁碟區大小 (以 GiB 為單位)。
- NodeVolumeType：為您的節點指定根磁碟區類型。
- KeyName：輸入 Amazon EC2 SSH 金鑰對的名稱，您可以在節點啟動後使用該金鑰對來透過 SSH 連接至節點。如果您還沒有 Amazon EC2 金鑰對，可以在 AWS 管理主控台中建立一個。如需詳細資訊，請參閱《Amazon EC2 使用者指南》中的 Amazon EC2 金鑰對。
- VpcId：輸入您建立的 VPC ID。
- 子網路：選擇您為 VPC 建立的子網路。如果您已依照為您的 Amazon EKS 叢集建立 Amazon VPC 中所述的步驟建立 VPC，則只需指定要啟動節點的 VPC 中的私有子網路。您可以看到哪些子網是私有子網，方法是從叢集的 Networking (聯網) 標籤打開每一個子網連結。

  > **Important**
  >
  > - 如有任何子網路是公有子網路，則必須啟用自動公有 IP 地址指派設定。如果未針對公有子網路啟用設定，則您部署至該公有子網路的任何節點都不會指派公有 IP 地址，也無法與叢集或其他 AWS 服務通訊。如果子網路是在 2020 年 3 月 26 日之前使用任一

Amazon EKS AWS CloudFormation VPC 範本或使用 部署eksctl，則會停用公有子網路的自動公有 IP 地址指派。如需如何啟用子網路的公有 IP 地址指派的相關資訊，請參閱修改您子網路的公有 IPv4 定址屬性。如果節點部署到私有子網路，則可以透過 NAT 閘道與叢集和其他 AWS 服務通訊。

- 如果子網路無法存取網際網路，請確保您已知悉部署網際網路存取受到限制的私有叢集中的考量事項和額外步驟。
- 如果您選取 AWS Outpost、Wavelength 或 Local Zone 子網路，則建立叢集時不得傳入子網路。

9. 請在 Configure stack options (設定堆疊選項) 頁面上選取所需的選項，然後選擇 Next (下一頁)。
10. 選取我確認 AWS CloudFormation 可能會建立 IAM 資源的左側核取方塊，然後選擇建立堆疊。
11. 當堆疊已完成建立時，從主控台將其選取，然後選擇 Outputs (輸出)。如果您使用的是 EKS API或 EKS API and ConfigMap 身分驗證模式，這是最後一個步驟。
12. 如果您使用ConfigMap身分驗證模式，請記錄已建立節點群組的 NodeInstanceRole。

步驟 2：讓節點加入叢集

> **Note**
>
> 只有在 EKS 叢集中使用 Configmap 身分驗證模式時，才需要下列兩個步驟。此外，如果您在沒有傳出網際網路存取的情況下在私有 VPC 內啟動節點，請務必讓節點從 VPC 內加入您的叢集。

1. 檢查以瞭解是否有 aws-auth ConfigMap。

```
kubectl describe configmap -n kube-system aws-auth
```

2. 如果您看到 aws-auth ConfigMap，請視需要更新之。
   a. 開啟 ConfigMap 進行編輯。

```
kubectl edit -n kube-system configmap/aws-auth
```

   b. 視需要新增 mapRoles 個項目。將 rolearn 值設定為您在先前程序中記錄的 NodeInstanceRole 值。

```
[...]
```

```
data:
  mapRoles: |
    - rolearn: <ARN of instance role (not instance profile)>
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes
[...]
```

c. 儲存檔案並結束您的文字編輯器。

3. 如果您收到錯誤，指出 "Error from server (NotFound): configmaps "aws-auth" not found，請套用堆疊 ConfigMap。

a. 下載組態對應。

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/
aws-auth-cm.yaml
```

b. 在 aws-auth-cm.yaml 檔案中，將 rolearn 值設定為您在先前程序中紀錄的 NodeInstanceRole 值。您可以使用文字編輯器來完成此操作，或者透過取代 *my-node-instance-role* 並執行下列命令：

```
sed -i.bak -e 's|<ARN of instance role (not instance profile)>|my-node-instance-
role|' aws-auth-cm.yaml
```

c. 套用組態。此命令可能需要幾分鐘的時間來完成。

```
kubectl apply -f aws-auth-cm.yaml
```

4. 查看節點的狀態，並等待他們到達 Ready 狀態。

```
kubectl get nodes --watch
```

輸入 Ctrl+C 傳回 Shell 提示。

> **Note**
>
> 如果您收到任何授權或資源類型錯誤，請參閱故障診斷主題中的the section called “未經授權或存取遭拒 (kubectl)”。

如果節點無法加入叢集，請參閱「故障診斷」章節中的 the section called “節點無法加入叢集”。

5. (僅限 GPU 節點) 若您選擇了 GPU 執行個體類型和 Amazon EKS 最佳化加速 AMI，則必須套用 Kubernetes 專用 NVIDIA 裝置外掛程式作為叢集上的 DaemonSet。請先以您想要的 NVIDIA/k8s-device-plugin 版本來取代 *vX.X.X*，然後再執行下列命令。

```
kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/vX.X.X/deployments/static/nvidia-device-plugin.yml
```

步驟 3：其他動作

1. (選用) 部署範例應用程式以測試您的叢集和 Linux 節點。
2. (選用) 如果 AmazonEKS_CNI_Policy 受管 IAM 政策 (如果您有 IPv4 叢集) 或 *AmazonEKS_CNI_IPv6_Policy* (您自我建立的政策，前提是如果您有 IPv6 叢集) 連接至 Amazon EKS 節點 IAM 角色，建議改為將其指派給您與 Kubernetes aws-node 服務帳戶相關聯的 IAM 角色。如需詳細資訊，請參閱the section called “為 IRSA 設定”。
3. 如果下列條件為真，我們建議封鎖 Pod 對 IMDS 的存取：
   - 您計劃將 IAM 角色指派給您的所有 Kubernetes 服務帳戶，以便 Pod 僅具有所需的最低許可。
   - 叢集中沒有任何 Pod 因其他原因需要存取 Amazon EC2 執行個體中繼資料服務 (IMDS)，例如擷取目前 AWS 區域。

   如需詳細資訊，請參閱限制存取指派給工作節點的執行個體設定檔。

建立自我管理的 Bottlerocket 節點

> **Note**
>
> 受管節點群組可能會為您的使用案例提供一些優勢。如需詳細資訊，請參閱the section called “受管節點群組”。

本主題會說明如何啟動向 Amazon EKS 叢集註冊的 Bottlerocket 節點的 Auto Scaling 群組。Bottlerocket 是以 Linux 為基礎的開放原始碼作業系統 AWS ，可用於在虛擬機器或裸機主機上執行容器。節點加入叢集後，您就可以將 Kubernetes 應用程式部署至其中。如需 Bottlerocket 的詳細資

訊，請參閱 GitHub 上的搭配 Amazon EKS 使用 Bottlerocket AMI 和 `eksctl` 文件上的自訂 AMI 支援。

如需就地升級的資訊，請參閱 GitHub 上的 Bottlerocket 更新運算子。

> **Important**
>
> - Amazon EKS 節點為標準 Amazon EC2 執行個體，會根據一般 Amazon EC2 執行個體價格向您收取這些節點的費用。如需詳細資訊，請參閱 Amazon EC2 定價。
> - 您可以在 AWS Outposts 上的 Amazon EKS 擴充叢集中啟動 Bottlerocket 節點，但無法在 AWS Outposts 的本機叢集中啟動它們。如需詳細資訊，請參閱 AWS Outposts 上的 Amazon EKS。
> - 您可以使用 x86 或 Arm 處理器部署至 Amazon EC2 執行個體。不過，您無法部署至具有 Inferentia 晶片的執行個體。
> - Bottlerocket 與 AWS CloudFormation 相容。但是，沒有可複製的官方 CloudFormation 範本來部署 Amazon EKS 的 Bottlerocket 節點。
> - Bottlerocket 映像不會隨附 SSH 伺服器或 Shell。您可以使用頻外存取方法來允許 SSH 啟用管理員容器，並傳遞一些引導組態步驟與使用者資料。如需詳細資訊，請參閱 GitHub 上 bottlerocket README.md 中的這些部分：
>   - 探勘
>   - 管理員容器
>   - Kubernetes 設定

此程序需要 `eksctl` 版本 `0.215.0` 或更新版本。您可使用以下命令檢查您的版本：

```
eksctl version
```

如需有關安裝或更新 `eksctl` 的指示，請參閱 `eksctl` 文件中的 Installation 一節。

> **Note**
>
> 此程序只適用於使用 `eksctl` 所建立的叢集。

1. 將以下內容複製到您的裝置。使用您叢集的名稱取代 *`my-cluster`*。此名稱僅能使用英數字元 (區分大小寫) 和連字號。必須以英數字元開頭，且長度不可超過 100 個字元。名稱在您要建立叢集

AWS 的區域和 AWS 帳戶中必須是唯一的。使用您的節點群組名稱取代 *ng-bottlerocket*。節點群組名稱不可超過 63 個字元。它必須以字母或數字開頭，但也可以包含其餘字元的連字號和底線。若要在 Arm 執行個體上部署，請使用 Arm 執行個體類型取代 *m5.large*。使用 Amazon EC2 SSH 金鑰對名稱取代 *my-ec2-keypair-name*，您可以在節點啟動後使用該金鑰對來透過 SSH 連接至節點。如果您還沒有 Amazon EC2 金鑰對，可以在 AWS 管理主控台中建立一個。如需詳細資訊，請參閱《Amazon EC2 使用者指南》中的 Amazon EC2 金鑰對。使用您自己的值取代其餘所有範例值。完成取代後，請執行修改後的命令以建立 `bottlerocket.yaml` 檔案。

如果指定 Arm Amazon EC2 執行個體類型，則請在部署前檢閱 Amazon EKS 最佳化 Arm Amazon Linux AMI 的考量事項。如需了解如何使用自訂 AMI 部署，請參閱 GitHub 上的建置 Bottlerocket 和 `eksctl` 文件中的自訂 AMI 支援。若要部署受管節點群組，請使用啟動範本部署自訂 AMI。如需詳細資訊，請參閱the section called “啟動範本”。

> **Important**
>
> 若要將節點群組部署至 AWS Outposts、 AWS Wavelength 或 AWS Local Zone 子網路，請勿在建立叢集時傳遞 AWS Outposts、 AWS Wavelength 或 AWS Local Zone 子網路。您必須在下列範例中指定子網路。如需詳細資訊，請參閱 `eksctl` 文件中的從組態檔案建立節點群組和組態檔案結構描述。將 *region-code* 取代為您的叢集所在的 AWS 區域。

```
cat >bottlerocket.yaml <<EOF
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: my-cluster
  region: region-code
  version: '1.36'

iam:
  withOIDC: true

nodeGroups:
  - name: ng-bottlerocket
    instanceType: m5.large
    desiredCapacity: 3
    amiFamily: Bottlerocket
    ami: auto-ssm
```

```
    iam:
       attachPolicyARNs:
          - arn:aws: iam::aws:policy/AmazonEKSWorkerNodePolicy
          - arn:aws: iam::aws:policy/AmazonEC2ContainerRegistryReadOnly
          - arn:aws: iam::aws:policy/AmazonSSMManagedInstanceCore
          - arn:aws: iam::aws:policy/AmazonEKS_CNI_Policy
    ssh:
        allow: true
        publicKeyName: my-ec2-keypair-name
EOF
```

2. 使用下列命令部署節點。

```
eksctl create nodegroup --config-file=bottlerocket.yaml
```

範例輸出如下。

建立節點時，會有數行輸出。輸出的最後幾行之一類似於以下的範例行。

```
[#]  created 1 nodegroup(s) in cluster "my-cluster"
```

3. (選用) 使用 Amazon EBS CSI 外掛程式在 Bottlerocket 節點上建立 Kubernetes 持續性磁碟區。預設 Amazon EBS 驅動程式依賴於未包含 Bottlerocket 的檔案系統工具。如需使用驅動程式建立儲存類別的詳細資訊，請參閱 the section called “Amazon EBS”。
4. (選用) 依預設 `kube-proxy` 會將 `nf_conntrack_max` 核心參數設定為預設值，此值可能與開機時 Bottlerocket 原先設定的不同。若要保留 Bottlerocket 的預設設定，請使用以下命令編輯 `kube-proxy` 組態。

```
kubectl edit -n kube-system daemonset kube-proxy
```

新增 `--conntrack-max-per-core` 和 `--conntrack-min` 到 `kube-proxy` 引數，這些引數位於以下範例中。`0` 設定意味著沒有改變。

```
      containers:
      - command:
        - kube-proxy
        - --v=2
        - --config=/var/lib/kube-proxy-config/config
        - --conntrack-max-per-core=0
        - --conntrack-min=0
```

5. (選用) 部署範例應用程式來測試您的 Bottlerocket 節點。
6. 如果下列條件為真，我們建議封鎖 Pod 對 IMDS 的存取：
   - 您計劃將 IAM 角色指派給您的所有 Kubernetes 服務帳戶，以便 Pod 僅具有所需的最低許可。
   - 叢集中沒有任何 Pod 因其他原因需要存取 Amazon EC2 執行個體中繼資料服務 (IMDS)，例如擷取目前 AWS 區域。

   如需詳細資訊，請參閱限制存取指派給工作節點的執行個體設定檔。

建立自我管理的 Microsoft Windows 節點

本主題說明如何啟動已向 Amazon EKS 叢集註冊的 Windows 節點的 Auto Scaling 群組。節點加入叢集後，您就可以將 Kubernetes 應用程式部署至其中。

> **Important**
>
> - Amazon EKS 節點為標準 Amazon EC2 執行個體，會根據一般 Amazon EC2 執行個體價格向您收取這些節點的費用。如需詳細資訊，請參閱 Amazon EC2 定價。
> - 您可以在 AWS Outposts 上的 Amazon EKS 延伸叢集中啟動 Windows 節點，但無法在 AWS Outposts 的本機叢集中啟動它們。如需詳細資訊，請參閱 AWS Outposts 上的 Amazon EKS。

為您的叢集啟用 Windows 支援。我們建議您在啟動 Windows 節點群組之前，先檢閱重要的考量。如需詳細資訊，請參閱the section called “啟用 Windows 支援”。

您可以使用下列任意一項啟動自我管理的 Windows 節點：

- the section called “`eksctl`”
- the section called “AWS 管理主控台”

`eksctl`

使用 **`eksctl`** 啟動自我管理的 Windows 節點

此程序需要您已安裝 `eksctl`，且您的 `eksctl` 版本至少是 `0.215.0`。您可使用以下命令檢查您的版本。

```
eksctl version
```

如需有關安裝或更新 eksctl 的指示，請參閱 eksctl 文件中的 Installation 一節。

> **Note**
>
> 此程序只適用於使用 eksctl 所建立的叢集。

1. (選用) 如果 AmazonEKS_CNI_Policy 受管 IAM 政策 (如果您有 IPv4 叢集) 或 *AmazonEKS_CNI_IPv6_Policy* (您自我建立的政策，前提是如果您有 IPv6 叢集) 連接至 Amazon EKS 節點 IAM 角色，建議改為將其指派給您與 Kubernetes aws-node 服務帳戶相關聯的 IAM 角色。如需詳細資訊，請參閱the section called “為 IRSA 設定”。
2. 此程序假設您有現有的叢集。如果您尚未擁有可新增 Windows 節點群組的 Amazon EKS 叢集和 Amazon Linux 節點群組，我們建議您遵循 the section called “建立叢集 (eksctl)”。該指南提供了建立具有 Amazon Linux 節點的 Amazon EKS 叢集的完整演練。

   使用下列命令來建立您的節點群組。將 *region-code* 取代為您的叢集所在的 AWS 區域。使用您的叢集名稱取代 *my-cluster*。此名稱僅能使用英數字元 (區分大小寫) 和連字號。必須以英數字元開頭，且長度不可超過 100 個字元。名稱在您要建立叢集 AWS 的區域和 AWS 帳戶中必須是唯一的。使用您的節點群組名稱取代 *ng-windows*。節點群組名稱不可超過 63 個字元。它必須以字母或數字開頭，但也可以包含其餘字元的連字號和底線。您可以使用 取代 *2019* 2022 以使用 Windows Server 2022 或使用 2025 Windows Server 2025。使用您自己的值取代範例值的其餘部分。

   > **Important**
   >
   > 若要將節點群組部署至 AWS Outposts、 AWS Wavelength 或 AWS Local Zone 子網路，請勿在建立叢集時傳遞 AWS Outposts、Wavelength 或 Local Zone 子網路。使用組態檔案建立節點群組，指定 AWS Outposts、Wavelength 或 Local Zone 子網路。如需詳細資訊，請參閱 eksctl 文件中的從組態檔案建立節點群組和組態檔案結構描述。

```
eksctl create nodegroup \
    --region region-code \
    --cluster my-cluster \
    --name ng-windows \
```

```
    --node-type t2.large \
    --nodes 3 \
    --nodes-min 1 \
    --nodes-max 4 \
    --managed=false \
    --node-ami-family WindowsServer2019FullContainer
```

> **Note**
>
> - 如果節點無法加入叢集，請參閱故障診斷指南中的 the section called “節點無法加入叢集”。
> - 若要查看 `eksctl` 命令可用的選項，請輸入下列命令。
>
>   ```
>   eksctl command -help
>   ```

範例輸出如下。建立節點時，會有數行輸出。輸出的最後幾行之一類似於以下的範例行。

```
[#]  created 1 nodegroup(s) in cluster "my-cluster"
```

3. (選用) 部署範例應用程式以測試您的叢集和 Windows 節點。
4. 如果下列條件為真，我們建議封鎖 Pod 對 IMDS 的存取：
   - 您計劃將 IAM 角色指派給您的所有 Kubernetes 服務帳戶，以便 Pod 僅具有所需的最低許可。
   - 叢集中沒有任何 Pod 因其他原因需要存取 Amazon EC2 執行個體中繼資料服務 (IMDS)，例如擷取目前 AWS 區域。

   如需詳細資訊，請參閱限制存取指派給工作節點的執行個體設定檔。

AWS 管理主控台

先決條件

- 現有 Amazon EKS 叢集和 Linux 節點群組。如果您沒有這些資源，我們建議您遵循我們 開始使用 中的其中一個指南來建立這些資源。這些指南說明如何建立具有 Linux 節點的 Amazon EKS 叢集。
- 符合 Amazon EKS 叢集要求的現有 VPC 和安全群組。如需詳細資訊，請參閱the section called “VPC 和子網要求”及the section called “安全群組要求”。開始使用 中的指南會建立符合要求的 VPC。或者，您也可以遵循為您的 Amazon EKS 叢集建立 Amazon VPC 手動建立一個。

- 現有的 Amazon EKS 叢集，其使用符合 Amazon EKS 叢集要求的 VPC 和安全群組。如需詳細資訊，請參閱the section called “建立 叢集”。如果您在已啟用 AWS Outposts、 AWS Wavelength 或 AWS Local Zones AWS 的區域中有子網路，則這些子網路不得在您建立叢集時傳入。

步驟 1：使用 AWS 管理主控台 啟動自我管理的 Windows 節點

1. 等待您的叢集狀態顯示為 ACTIVE。如果在叢集處於作用中狀態之前啟動節點，則節點向叢集註冊將會失敗，導致您必須再次啟動。
2. 開啟 AWS CloudFormation 主控台
3. 選擇建立堆疊。
4. 對於 Specify template (指定範本)，選取 Amazon S3 URL。
5. 複製以下 URL 並將其貼到 Amazon S3 URL 中。

```
https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2023-02-09/amazon-eks-windows-nodegroup.yaml
```

6. 選取 Next (下一步) 兩次。
7. 在 Quick create stack (快速建立堆疊) 頁面上，視需要輸入下列參數：
   - 堆疊名稱：為您的 AWS CloudFormation 堆疊選擇堆疊名稱。例如，您可以稱它為 `my-cluster-nodes`。
   - ClusterName：輸入建立 Amazon EKS 叢集時使用的名稱。

     > **Important**
     >
     > 此名稱必須與您在步驟 1：建立 Amazon EKS 叢集中使用的名稱完全相符。否則，您的節點無法加入叢集。

   - ClusterControlPlaneSecurityGroup：從您在建立 VPC 時產生的 AWS CloudFormation 輸出中選擇安全群組。下列步驟顯示擷取適用群組的一種方法。
     a. 開啟 Amazon EKS 主控台。
     b. 選擇叢集的名稱。
     c. 選擇 Networking (網路) 索引標籤。
     d. 從 ClusterControlPlaneSecurityGroup 下拉式清單中選取時，請使用 Additional Security Group (其他安全群組) 值作為參考。

- NodeGroupName：輸入節點群組的名稱。此名稱稍後可用於識別為您節點建立的 Auto Scaling 節點群組。節點群組名稱不可超過 63 個字元。它必須以字母或數字開頭，但也可以包含其餘字元的連字號和底線。
- NodeAutoScalingGroupMinSize：輸入節點 Auto Scaling 群組可以縮減的最低節點數。
- NodeAutoScalingGroupDesiredCapacity：堆疊建立時，輸入欲擴展的所需節點數量。
- NodeAutoScalingGroupMaxSize：輸入節點 Auto Scaling 群組可以擴增的最大節點數。
- NodeInstanceType：選擇節點的執行個體類型。如需詳細資訊，請參閱the section called “Amazon EC2 執行個體類型”。

  > **Note**
  >
  > 最新版本的 Kubernetes 專用 Amazon VPC CNI 外掛程式支援的執行個體類型會在 GitHub 上的 vpc_ip_resource_limit.go 中列出。您可能需要更新 CNI 版本，才能使用最新支援的執行個體類型。如需詳細資訊，請參閱the section called “Amazon VPC CNI”。

- NodeImageIdSSMParam：預先填入目前所建議 Amazon EKS 最佳化 Windows Core AMI ID 的 Amazon EC2 Systems Manager 參數。若要使用完整版本的 Windows，請使用 `Full` 取代 *`Core`*。
- NodeImageId：（選用）如果您使用自己的自訂 AMI（而非 Amazon EKS 最佳化 AMI)，請輸入您 AWS 區域的節點 AMI ID。如果您在此欄位指定值，則會覆寫 NodeImageIdSSMParam 欄位中的任何值。
- NodeVolumeSize：為您的節點指定根磁碟區大小 (以 GiB 為單位)。
- KeyName：輸入 Amazon EC2 SSH 金鑰對的名稱，您可以在節點啟動後使用該金鑰對來透過 SSH 連接至節點。如果您還沒有 Amazon EC2 金鑰對，可以在 AWS 管理主控台中建立一個。如需詳細資訊，請參閱《Amazon EC2 使用者指南》中的 Amazon EC2 金鑰對。

  > **Note**
  >
  > 如果您未在此處提供金鑰對，則無法建立 AWS CloudFormation 堆疊。

- BootstrapArguments：指定要傳遞至節點引導指令碼的任何選用引數，例如使用 `-KubeletExtraArgs` 的額外 `kubelet` 引數。
- DisableIMDSv1：預設情況下，每個節點都支援執行個體中繼資料服務版本 1 (IMDSv1) 和 IMDSv2。您可以停用 IMDSv1。若要防止節點群組中的未來節點和 Pod 使用 IMDSv1，請將 DisableIMDSv1 設定為 true。如需 IMDS 的詳細資訊，請參閱設定執行個體中繼資料服務。

- VpcId：選取您建立的 VPC ID。
- NodeSecurityGroups：選取在建立 VPC 時為您的 Linux 節點群組建立的安全群組。如果 Linux 節點連接了一個以上的安全群組，請指定所有的群組。例如，如果 Linux 節點群組是使用 建立，就會發生這種情況`eksctl`。
- Subnets (子網路)：選擇您建立的子網路。如果您已依照為您的 Amazon EKS 叢集建立 Amazon VPC 中的步驟建立 VPC，則只需指定要啟動節點的 VPC 中的私有子網路。

> **Important**
>
> - 如有任何子網路是公有子網路，則必須啟用自動公有 IP 地址指派設定。如果未針對公有子網路啟用設定，則您部署至該公有子網路的任何節點都不會指派公有 IP 地址，也無法與叢集或其他 AWS 服務通訊。如果子網路是在 2020 年 3 月 26 日之前使用任一 Amazon EKS AWS CloudFormation VPC 範本或使用 部署，`eksctl`則公有子網路的自動公有 IP 地址指派會停用。如需如何啟用子網路的公有 IP 地址指派的相關資訊，請參閱修改您子網路的公有 IPv4 定址屬性。如果節點部署到私有子網路，則可以透過 NAT 閘道與叢集和其他 AWS 服務通訊。
> - 如果子網路無法存取網際網路，則請確保您已知悉部署網際網路存取受到限制的私有叢集中的考量事項和額外步驟。
> - 如果您選取 AWS Outpost、Wavelength 或 Local Zone 子網路，則建立叢集時不得傳入子網路。

8. 確認堆疊可建立 IAM 資源，然後選擇 Create stack (建立堆疊)。
9. 當堆疊已完成建立時，從主控台將其選取，然後選擇 Outputs (輸出)。
10. 為已建立的節點群組記錄 NodeInstanceRole。當您設定 Amazon EKS Windows 節點時會需要此值。

步驟 2：讓節點加入叢集

1. 檢查以瞭解是否有 `aws-auth ConfigMap`。

```
kubectl describe configmap -n kube-system aws-auth
```

2. 如果您看到 `aws-auth ConfigMap`，請視需要更新之。
   a. 開啟 `ConfigMap` 進行編輯。

```
kubectl edit -n kube-system configmap/aws-auth
```

b. 視需要新增 mapRoles 個項目。將 rolearn 值設定為您在先前程序中記錄的 NodeInstanceRole 值。

```
[...]
data:
  mapRoles: |
- rolearn: <ARN of linux instance role (not instance profile)>
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes
    - rolearn: <ARN of windows instance role (not instance profile)>
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes
        - eks:kube-proxy-windows
[...]
```

c. 儲存檔案並結束您的文字編輯器。

3. 如果您收到一個錯誤，說明 "Error from server (NotFound): configmaps "aws-auth" not found，那麼請套用股票 ConfigMap。

a. 下載組態對應。

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/
aws-auth-cm-windows.yaml
```

b. 在 aws-auth-cm-windows.yaml 檔案中，將 rolearn 值設定為您在先前程序中記錄的適用 NodeInstanceRole 值。您可以使用文字編輯器來完成此操作，或者透過取代範例值並執行下列命令：

```
sed -i.bak -e 's|<ARN of linux instance role (not instance profile)>|my-node-
linux-instance-role|' \
    -e 's|<ARN of windows instance role (not instance profile)>|my-node-windows-
instance-role|' aws-auth-cm-windows.yaml
```

> **Important**
>
> - 請勿修改此檔案中的任何其他行。
> - 請勿為 Windows 節點和 Linux 節點使用相同的 IAM 角色。

c. 套用組態。此命令可能需要幾分鐘的時間來完成。

```
kubectl apply -f aws-auth-cm-windows.yaml
```

4. 查看節點的狀態，並等待他們到達 Ready 狀態。

```
kubectl get nodes --watch
```

輸入 Ctrl+C 傳回 Shell 提示。

> **Note**
>
> 如果您收到任何授權或資源類型錯誤，請參閱故障診斷主題中的the section called “未經授權或存取遭拒 (kubectl)”。

如果節點無法加入叢集，請參閱「故障診斷」章節中的 the section called “節點無法加入叢集”。

步驟 3：其他動作

1. (選用) 部署範例應用程式以測試您的叢集和 Windows 節點。
2. (選用) 如果 AmazonEKS_CNI_Policy 受管 IAM 政策 (如果您有 IPv4 叢集) 或 *AmazonEKS_CNI_IPv6_Policy* (您自我建立的政策，前提是如果您有 IPv6 叢集) 連接至 Amazon EKS 節點 IAM 角色，建議改為將其指派給您與 Kubernetes aws-node 服務帳戶相關聯的 IAM 角色。如需詳細資訊，請參閱the section called “為 IRSA 設定”。
3. 如果下列條件為真，我們建議封鎖 Pod 對 IMDS 的存取：
   - 您計劃將 IAM 角色指派給您的所有 Kubernetes 服務帳戶，以便 Pod 僅具有所需的最低許可。
   - 叢集中沒有任何 Pod 因其他原因需要存取 Amazon EC2 執行個體中繼資料服務 (IMDS)，例如擷取目前 AWS 區域。

   如需詳細資訊，請參閱限制存取指派給工作節點的執行個體設定檔。

建立自我管理的 Ubuntu Linux 節點

> **Note**
>
> 受管節點群組可能會為您的使用案例提供一些優勢。如需詳細資訊，請參閱the section called "受管節點群組"。

本主題會說明如何啟動向 Amazon EKS 叢集註冊的 Amazon Elastic Kubernetes Service (EKS) 上的 Ubuntu 或 Amazon Elastic Kubernetes Service (EKS) 上的 Ubuntu Pro 節點的 Auto Scaling 群組。Ubuntu 和 Ubuntu Pro for EKS 是以官方 Ubuntu Minimal LTS 為基礎，包括與 共同開發的自訂 AWS 核心 AWS，並專門為 EKS 建置。Ubuntu Pro 透過支援 EKS 延長支援期間、核心 livepatch、FIPS 合規以及執行無限 Pro 容器的能力來新增額外的安全涵蓋範圍。

節點加入叢集後，您就可以將容器化應用程式部署至其中。如需詳細資訊，請參閱 `eksctl` 文件中的 AWS上的 Ubuntu 和自訂 AMI 支援。

> **Important**
>
> - Amazon EKS 節點為標準 Amazon EC2 執行個體，會根據一般 Amazon EC2 執行個體價格向您收取這些節點的費用。如需詳細資訊，請參閱 Amazon EC2 定價。
> - 您可以在 AWS Outposts 上的 Amazon EKS 擴充叢集中啟動 Ubuntu 節點，但無法在 AWS Outposts 的本機叢集中啟動它們。如需詳細資訊，請參閱 AWS Outposts 上的 Amazon EKS。
> - 您可以使用 x86 或 Arm 處理器部署至 Amazon EC2 執行個體。不過，具有 Inferentia 晶片的執行個體可能需要先安裝 Neuron SDK。

此程序需要 `eksctl` 版本 `0.215.0` 或更新版本。您可使用以下命令檢查您的版本：

```
eksctl version
```

如需有關安裝或更新 `eksctl` 的指示，請參閱 `eksctl` 文件中的 Installation 一節。

> **Note**
>
> 此程序只適用於使用 `eksctl` 所建立的叢集。

1. 將以下內容複製到您的裝置。使用您叢集的名稱取代 `my-cluster`。此名稱僅能使用英數字元 (區分大小寫) 和連字號。必須以字母字元開頭，且長度不可超過 100 個字元。將 `ng-ubuntu` 取代為您的節點群組名稱。節點群組名稱不可超過 63 個字元。它必須以字母或數字開頭，但也可以包含其餘字元的連字號和底線。若要在 Arm 執行個體上部署，請以 Arm 執行個體類型取代 `m5.large`。使用 Amazon EC2 SSH 金鑰對名稱取代 `my-ec2-keypair-name`，您可以在節點啟動後使用該金鑰對來透過 SSH 連接至節點。如果您還沒有 Amazon EC2 金鑰對，可以在 AWS 管理主控台中建立一個。如需詳細資訊，請參閱《Amazon EC2 使用者指南》中的 Amazon EC2 金鑰對。使用您自己的值取代其餘所有範例值。完成取代後，請執行修改後的命令以建立 `ubuntu.yaml` 檔案。

> **Important**
>
> 若要將節點群組部署至 AWS Outposts、 AWS Wavelength 或 AWS Local Zone 子網路，請勿在建立叢集時傳遞 AWS Outposts、 AWS Wavelength 或 AWS Local Zone 子網路。您必須在下列範例中指定子網路。如需詳細資訊，請參閱 `eksctl` 文件中的從組態檔案建立節點群組和組態檔案結構描述。將 *`region-code`* 取代為您的叢集所在的 AWS 區域。

```
cat >ubuntu.yaml <<EOF
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: my-cluster
  region: region-code
  version: '1.36'

iam:
  withOIDC: true

nodeGroups:
  - name: ng-ubuntu
    instanceType: m5.large
    desiredCapacity: 3
    amiFamily: Ubuntu2204
    iam:
       attachPolicyARNs:
          - arn:aws: iam::aws:policy/AmazonEKSWorkerNodePolicy
          - arn:aws: iam::aws:policy/AmazonEC2ContainerRegistryReadOnly
          - arn:aws: iam::aws:policy/AmazonSSMManagedInstanceCore
```

```
            - arn:aws: iam::aws:policy/AmazonEKS_CNI_Policy
    ssh:
        allow: true
        publicKeyName: my-ec2-keypair-name
EOF
```

若要建立 Ubuntu Pro 節點群組，只需將 amiFamily 值變更為 UbuntuPro2204 即可。

2. 使用下列命令部署節點。

```
eksctl create nodegroup --config-file=ubuntu.yaml
```

範例輸出如下。

建立節點時，會有數行輸出。輸出的最後幾行之一類似於以下的範例行。

```
[#]  created 1 nodegroup(s) in cluster "my-cluster"
```

3. (選用) 部署範例應用程式來測試您的 Ubuntu 節點。
4. 如果下列條件為真，我們建議封鎖 Pod 對 IMDS 的存取：
   - 您計劃將 IAM 角色指派給您的所有 Kubernetes 服務帳戶，以便 Pod 僅具有所需的最低許可。
   - 叢集中沒有任何 Pod 因其他原因需要存取 Amazon EC2 執行個體中繼資料服務 (IMDS)，例如擷取目前 AWS 區域。

   如需詳細資訊，請參閱限制存取指派給工作節點的執行個體設定檔。

更新您的叢集的自我管理節點

全新 Amazon EKS 最佳化 AMI 發行時，請考量以新 AMI 取代自我管理節點群組中的節點。同樣地，如果更新 Amazon EKS 叢集的 Kubernetes 版本，請更新節點以使用具相同 Kubernetes 版本的節點。

> **Important**
>
> 本主題涵蓋自我管理節點的節點更新。若使用受管節點群組，則請參閱 the section called “更新”。

有兩種基本方法可以將叢集中的自我管理節點群組更新成使用新的 AMI：

[將應用程式移轉至新的節點群組]

建立新的節點群組，並將 Pod 移轉至該群組。移轉至新節點群組比在現有 AWS CloudFormation 堆疊中更新 AMI ID 更順利。這是因為移轉過程會將舊節點群組汙染為 NoSchedule，並在新堆疊準備好接受現有的 Pod 工作負載之後耗盡節點。

[更新 AWS CloudFormation 節點堆疊]

更新現有節點群組的 AWS CloudFormation 堆疊，以使用新的 AMI。不支援對使用 eksctl 建立的節點群組使用這個方法。

將應用程式移轉至新的節點群組

此主題會說明如何建立新的節點群組，將現有的應用程式從容遷移至新群組，接著從叢集移除舊的節點群組。您可以使用 eksctl 或 AWS 管理主控台遷移至新的節點群組。

- the section called “eksctl”
- the section called “AWS 管理主控台 和 AWS CLI”

eksctl

使用 **eksctl** 將應用程式移轉至新的節點群組

如需有關將 eksctl 用於移轉的詳細資訊，請參閱 eksctl 文件中的未受管的節點群組。

此程序需要 eksctl 版本 0.215.0 或更新版本。您可使用以下命令檢查您的版本：

```
eksctl version
```

如需有關安裝或更新 eksctl 的指示，請參閱 eksctl 文件中的 Installation 一節。

> **Note**
>
> 此程序只適用於使用 eksctl 所建立的叢集和節點群組。

1. 擷取現有節點群組的名稱，使用叢集名稱取代 *my-cluster*。

```
eksctl get nodegroups --cluster=my-cluster
```

範例輸出如下。

```
CLUSTER      NODEGROUP          CREATED               MIN SIZE      MAX SIZE
 DESIRED CAPACITY     INSTANCE TYPE     IMAGE ID
default      standard-nodes   2019-05-01T22:26:58Z  1             4             3
               t3.medium         ami-05a71d034119ffc12
```

2. 使用以下命令啟動具備 `eksctl` 的新節點群組。在命令中，使用您自己的值取代每一個`###`。版本編號不能晚於控制平面的 Kubernetes 版本。也不能比控制平面的 Kubernetes 版本早兩個以上的次要版本。建議使用與控制平面相同的版本。

   如果下列條件為真，我們建議封鎖 Pod 對 IMDS 的存取：

   - 您計劃將 IAM 角色指派給您的所有 Kubernetes 服務帳戶，以便 Pod 僅具有所需的最低許可。
   - 叢集中沒有任何 Pod 因其他原因需要存取 Amazon EC2 執行個體中繼資料服務 (IMDS)，例如擷取目前 AWS 區域。

     如需詳細資訊，請參閱限制存取指派給工作節點的執行個體設定檔。

     若要封鎖 Pod 對 IMDS 的存取，請將 `--disable-pod-imds` 選項新增至下列命令。

     > **Note**
     >
     > 如需更多可用旗標及其描述的資訊，請參閱 https://eksctl.io/。

```
eksctl create nodegroup \
  --cluster my-cluster \
  --version 1.36 \
  --name standard-nodes-new \
  --node-type t3.medium \
  --nodes 3 \
  --nodes-min 1 \
  --nodes-max 4 \
  --managed=false
```

3. 之前的命令完成時，使用下列命令來確認所有節點已達到 `Ready` 狀態：

```
kubectl get nodes
```

4. 使用下列命令來刪除原始的節點群組。在命令中，使用您的叢集和節點群組名稱取代每一個`###`：

```
eksctl delete nodegroup --cluster my-cluster --name standard-nodes-old
```

AWS 管理主控台 和 AWS CLI

使用 AWS 管理主控台 和 AWS CLI 將應用程式遷移至新的節點群組

1. 依照建立自我管理的 Amazon Linux 節點中概述的步驟啟動新的節點群組。
2. 當堆疊已完成建立時，從主控台將其選取，然後選擇 Outputs (輸出)。
3. 為已建立的節點群組記錄 NodeInstanceRole。您需要這樣才能新增 Amazon EKS 節點至叢集。

> **Note**
>
> 如果已將任何額外的 IAM 政策連接至舊節點群組 IAM 角色，則請將相同政策連接至新節點群組 IAM 角色，以便在新群組維持該功能。例如，如果為 Kubernetes Cluster Autoscaler 新增許可，則適用此操作。

4. 更新兩個節點群組的安全群組，使其能夠互相通訊。如需詳細資訊，請參閱the section called “安全群組要求”。
   a. 記錄兩個節點群組的安全群組 ID。這在 AWS CloudFormation 堆疊輸出中顯示為 NodeSecurityGroup 值。

      您可以使用下列 AWS CLI 命令，從堆疊名稱取得安全群組 IDs。在這些命令中， `oldNodes`是舊版節點堆疊的 AWS CloudFormation 堆疊名稱，而 `newNodes`是您要遷移至的堆疊名稱。使用您自己的值取代每一個`###`。

      ```
      oldNodes="old_node_CFN_stack_name"
      newNodes="new_node_CFN_stack_name"

      oldSecGroup=$(aws cloudformation describe-stack-resources --stack-name $oldNodes \
      --query 'StackResources[?
      ResourceType==`AWS::EC2::SecurityGroup`].PhysicalResourceId' \
      --output text)
      newSecGroup=$(aws cloudformation describe-stack-resources --stack-name $newNodes \
      --query 'StackResources[?
      ResourceType==`AWS::EC2::SecurityGroup`].PhysicalResourceId' \
      --output text)
      ```

   b. 新增輸入規則至每個節點安全群組，使其接受彼此的流量。

下列 AWS CLI 命令會將傳入規則新增至每個安全群組，以允許來自其他安全群組的所有通訊協定上的所有流量。此組態可讓每個節點群組中的 Pod 在您將工作負載移轉至新群組時互相通訊。

```
aws ec2 authorize-security-group-ingress --group-id $oldSecGroup \
--source-group $newSecGroup --protocol -1
aws ec2 authorize-security-group-ingress --group-id $newSecGroup \
--source-group $oldSecGroup --protocol -1
```

5. 編輯 aws-auth configmap 以在 RBAC 中對應新的節點執行個體角色。

```
kubectl edit configmap -n kube-system aws-auth
```

為新節點群組新增 mapRoles 項目。

```
apiVersion: v1
data:
  mapRoles: |
    - rolearn: ARN of instance role (not instance profile)
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes>
    - rolearn: arn:aws: iam::111122223333:role/nodes-1-16-NodeInstanceRole-
U11V27W93CX5
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes
```

使用您在上一個步驟中記錄的 NodeInstanceRole 值來取代*###### ARN (########)* 程式碼片段。接著，儲存並關閉檔案以套用更新的 configmap。

6. 注意節點狀態並等待新的節點加入叢集和達到 Ready 狀態。

```
kubectl get nodes --watch
```

7. (選用) 如果您使用的是 Kubernetes Cluster Autoscaler，請將部署縮減為零 (0) 個複本以避免衝突的擴展動作。

```
kubectl scale deployments/cluster-autoscaler --replicas=0 -n kube-system
```

8. 使用以下命令污染每個您想要用 NoSchedule 移除的節點。這樣就不會在您要取代的節點上排程或重新排程新的 Pod。如需詳細資訊，請參閱 Kubernetes 文件中的污點和容差。

```
kubectl taint nodes node_name key=value:NoSchedule
```

如果要將節點升級至新的 Kubernetes 版本，您可以使用以下程式碼片段識別並標示特定 Kubernetes 版本 (在此情況下為 1.34) 的所有節點。版本編號不能晚於控制平面的 Kubernetes 版本。也不能比控制平面的 Kubernetes 版本早兩個以上的次要版本。建議使用與控制平面相同的版本。

```
K8S_VERSION=1.34
nodes=$(kubectl get nodes -o jsonpath="{.items[?(@.status.nodeInfo.kubeletVersion==
\"v$K8S_VERSION\")].metadata.name}")
for node in ${nodes[@]}
do
    echo "Tainting $node"
    kubectl taint nodes $node key=value:NoSchedule
done
```

9. 判斷叢集的 DNS 提供商。

```
kubectl get deployments -l k8s-app=kube-dns -n kube-system
```

範例輸出如下。此叢集針對 DNS 解析度使用 CoreDNS，但您的叢集可能會傳回 kube-dns：

```
NAME      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
coredns   1         1         1            1           31m
```

10. 如果您目前的部署執行少於 2 個複本，請將部署擴增為 2 個複本。如果您先前的命令輸出傳回該項目，請以 kubedns 取代 *coredns*。

```
kubectl scale deployments/coredns --replicas=2 -n kube-system
```

11. 使用以下命令清空您想要從叢集移除的每個節點：

```
kubectl drain node_name --ignore-daemonsets --delete-emptydir-data
```

如果您要將節點升級至新的 Kubernetes 版本，請使用下列程式碼片段識別並耗盡特定 Kubernetes 版本 （ 在本例中為 *1.34*) 的所有節點。

```
K8S_VERSION=1.34
nodes=$(kubectl get nodes -o jsonpath="{.items[?(@.status.nodeInfo.kubeletVersion==
\"v$K8S_VERSION\")].metadata.name}")
for node in ${nodes[@]}
do
    echo "Draining $node"
    kubectl drain $node --ignore-daemonsets --delete-emptydir-data
done
```

12.在舊的節點完成消耗後，請撤銷您先前授權的安全群組傳入規則。然後，刪除 AWS CloudFormation 堆疊以終止執行個體。

> **Note**
>
> 如果您將任何其他 IAM 政策連接至舊節點群組 IAM 角色，例如新增 Kubernetes Cluster Autoscaler 的許可，請先從角色分離這些其他政策，然後才能刪除 AWS CloudFormation 堆疊。

a. 撤銷您先前為節點安全群組建立的傳入規則。在這些命令中， `oldNodes`是舊版節點堆疊的 AWS CloudFormation 堆疊名稱，而 `newNodes`是您要遷移至的堆疊名稱。

```
oldNodes="old_node_CFN_stack_name"
newNodes="new_node_CFN_stack_name"

oldSecGroup=$(aws cloudformation describe-stack-resources --stack-name $oldNodes \
--query 'StackResources[?
ResourceType==`AWS::EC2::SecurityGroup`].PhysicalResourceId' \
--output text)
newSecGroup=$(aws cloudformation describe-stack-resources --stack-name $newNodes \
--query 'StackResources[?
ResourceType==`AWS::EC2::SecurityGroup`].PhysicalResourceId' \
--output text)
aws ec2 revoke-security-group-ingress --group-id $oldSecGroup \
--source-group $newSecGroup --protocol -1
aws ec2 revoke-security-group-ingress --group-id $newSecGroup \
--source-group $oldSecGroup --protocol -1
```

b. 開啟 AWS CloudFormation 主控台。

c. 選取舊的節點堆疊。

d. 選擇 刪除。

e. 在 Delete stack (刪除堆疊) 確認對話方塊中，選擇 Delete stack (刪除堆疊)。

13編輯 aws-auth configmap 以從 RBAC 移除舊的節點執行個體角色。

```
kubectl edit configmap -n kube-system aws-auth
```

刪除舊節點群組的 mapRoles 項目。

```
apiVersion: v1
data:
  mapRoles: |
    - rolearn: arn:aws: iam::111122223333:role/nodes-1-16-NodeInstanceRole-
W70725MZQFF8
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes
    - rolearn: arn:aws: iam::111122223333:role/nodes-1-15-NodeInstanceRole-
U11V27W93CX5
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes>
```

儲存並關閉檔案以套用更新的 configmap。

14(選用) 如果您使用的是 Kubernetes Cluster Autoscaler，請將部署縮減回 1 個複本。

> **Note**
>
> 您也必須適當標記新的 Auto Scaling 群組 (例如 k8s.io/cluster-autoscaler/enabled,k8s.io/cluster-autoscaler/my-cluster)，並更新 Cluster Autoscaler 部署的命令，以指向新標記的 Auto Scaling 群組。如需詳細資訊，請參閱 Cluster Autoscaler on AWS。

```
kubectl scale deployments/cluster-autoscaler --replicas=1 -n kube-system
```

15. (選用) 確認您使用的是最新版本的 Kubernetes 專用 Amazon VPC CNI 外掛程式。您可能需要更新 CNI 版本，才能使用最新支援的執行個體類型。如需詳細資訊，請參閱the section called “Amazon VPC CNI”。
16. 如果您的叢集針對 DNS 解析度使用 kube-dns (請參閱 [migrate-determine-dns-step])，請將 kube-dns 部署縮減至一個複本。

```
kubectl scale deployments/kube-dns --replicas=1 -n kube-system
```

更新 an AWS CloudFormation 節點堆疊

本主題說明如何使用新的 AMI 更新現有的 AWS CloudFormation 自我管理節點堆疊。您可以使用此程序在叢集更新後將節點更新至新版本的 Kubernetes。否則，您可以針對現有 Kubernetes 版本更新至最新的 Amazon EKS 最佳化 AMI。

> **Important**
>
> 本主題涵蓋自我管理節點的節點更新。若要了解透過受管節點群組來簡化節點生命週期的相關詳細資訊，請參閱 the section called “更新”。

最新的預設 Amazon EKS node AWS CloudFormation 範本已設定為在移除舊執行個體之前，在叢集中啟動具有新 AMI 的執行個體，一次一個。此組態可確保您在滾動更新期間，都能掌握叢集中 Auto Scaling 群組所需的作用中執行個體計數。

> **Note**
>
> 不支援對使用 eksctl 建立的節點群組使用這個方法。如果透過 eksctl 建立叢集或節點群組，則請參閱 the section called “移轉”。

1. 判斷叢集的 DNS 供應商。

```
kubectl get deployments -l k8s-app=kube-dns -n kube-system
```

範例輸出如下。此叢集針對 DNS 解析度使用 CoreDNS，但您的叢集可能會傳回 kube-dns。根據您使用的 kubectl 版本，輸出可能會有所不同。

```
NAME      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
coredns   1         1         1            1           31m
```

2. 如果您目前的部署執行少於 2 個複本，請將部署擴增為 2 個複本。如果您先前的命令輸出傳回該項目，請以 kube-dns 取代 *coredns*。

```
kubectl scale deployments/coredns --replicas=2 -n kube-system
```

3. (選用) 如果您使用的是 Kubernetes Cluster Autoscaler，請將部署縮減為零 (0) 個複本以避免衝突的擴展動作。

```
kubectl scale deployments/cluster-autoscaler --replicas=0 -n kube-system
```

4. 確定目前節點群組的執行個體類型和所需執行個體計數。您稍後更新群組的 AWS CloudFormation 範本時，會輸入這些值。
   a. 前往 https://console.aws.amazon.com/ec2/ 開啟 Amazon EC2 主控台。
   b. 在左側導覽窗格中，選擇 Launch Configurations (啟動組態)，並記下現有節點啟動組態的執行個體類型。
   c. 在左側導覽窗格中，選擇 Auto Scaling Groups (Auto Scaling 群組)，並注意現有節點 Auto Scaling 群組的 Desired (所需) 執行個體計數。
5. 開啟 AWS CloudFormation 主控台。
6. 選取您的節點群組堆疊，然後選擇 Update (更新)。
7. 選取 Replace current template (取代目前的範本)，然後選取 Amazon S3 URL。
8. 對於 Amazon S3 URL，請將下列 URL 貼到文字區域，以確保您使用的是最新版本的 node AWS CloudFormation 範本。然後選擇 Next (下一步)：

```
https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2022-12-23/amazon-eks-nodegroup.yaml
```

9. 在 Specify stack details (指定堆疊詳細資訊) 頁面上，填寫下列參數，然後選擇 Next (下一步)：
   - NodeAutoScalingGroupDesiredCapacity：輸入在先前步驟中記錄的所需執行個體計數。或者，在堆疊更新時，輸入欲擴展的所需節點數量。
   - NodeAutoScalingGroupMaxSize：輸入節點 Auto Scaling 群組可以擴增的最大節點數。此值必須至少超過所需容量一個節點。這樣才能夠在更新期間執行節點滾動更新，而不必減少節點計數。
   - NodeInstanceType – 選擇您在上一個步驟中記錄的執行個體類型。或者，為節點選擇不同的執行個體類型。選擇不同的執行個體類型之前，請檢閱選擇最佳 Amazon EC2 節點執行個體類型。

每個 Amazon EC2 執行個體類型都支援最大數量的彈性網路介面 (網路介面)，且每個網路介面支援最大數量的 IP 地址。由於每個工作者節點和 Pod 都會指派自己的 IP 地址，因此請務必選擇執行個體類型，以支援您想要在每個 Amazon EC2 節點上執行的 Pod 數量上限。如需執行個體類型支援的網路介面和 IP 地址數量清單，請參閱每種執行個體類型每個網路介面的 IP 地址。例如，`m5.large` 執行個體類型最多支援工作節點和 Pod 的 30 個 IP 位址。

> **Note**
>
> 最新版本的 Kubernetes 專用 Amazon VPC CNI 外掛程式支援的執行個體類型會顯示在 GitHub 上的 vpc_ip_resource_limit.go。您可能需要更新適用於 Kubernetes 的 Amazon VPC CNI 版本，才能使用最新支援的執行個體類型。如需詳細資訊，請參閱the section called “Amazon VPC CNI”。

> **Important**
>
> 有些執行個體類型可能無法在所有 AWS 區域中使用。

- NodeImageIdSSMParam – 您要更新的 AMI ID 的 Amazon EC2 Systems Manager 參數。以下值對 Kubernetes `1.36` 版使用最新的 Amazon EKS 最佳化 AMI。

```
/aws/service/eks/optimized-ami/1.36/amazon-linux-2/recommended/image_id
```

  您可以使用相同的平台版本取代 *`1.36`*。或者比控制平面上執行的 Kubernetes 版本早一個版本號的版本。建議您將節點保持在與控制平面相同的版本。您亦可使用不同的 AMI 類型來取代 *`amazon-linux-2`*。如需詳細資訊，請參閱the section called “獲取最新 ID”。

  > **Note**
  >
  > 使用 Amazon EC2 Systems Manager 參數可讓您在未來更新節點，而無需查詢和指定 AMI ID。如果您的 AWS CloudFormation 堆疊使用此值，任何堆疊更新一律會針對您指定的 Kubernetes 版本啟動最新的建議 Amazon EKS 最佳化 AMI。即使沒有變更範本中的任何值，情況也會如此。

- NodeImageId：若要使用您自己的自訂 AMI，請輸入要使用的 AMI 的 ID。

> **Important**
>
> 此值會覆寫 NodeImageIdSSMParam 指定的任何值。如果您想要使用 NodeImageIdSSMParam 值，請確定 NodeImageId 的值為空白。

- DisableIMDSv1：在預設情況下，每個節點都支援執行個體中繼資料服務版本 1 (IMDSv1) 和 IMDSv2。不過，您可以停用 IMDSv1。如果不想要任何節點，或節點群組中排定的任何 Pod 使用 IMDSv1，則請選取 true (是)。如需 IMDS 的詳細資訊，請參閱設定執行個體中繼資料服務。如果您已為服務帳戶實作 IAM 角色，請直接將必要的許可指派給需要存取 AWS 服務的所有 Pod。如此一來，叢集中沒有任何 Pod 因為其他原因需要存取 IMDS，例如擷取目前的 AWS 區域。然後，您也可以針對不使用主機聯網的 Pod 停用對 IMDSv2 的存取。如需詳細資訊，請參閱限制存取指派給工作節點的執行個體設定檔。

10. (選用) 在 Options (選項) 頁面上，為堆疊資源加上標籤。選擇下一步。
11. 在 Review (檢閱) 頁面上檢閱您的資訊，確認該堆疊可建立 IAM 資源，然後選擇 Update stack (更新堆疊)。

> **Note**
>
> 叢集中每個節點的更新，都需要幾分鐘的時間。請等待所有節點更新完成再執行後續步驟。

12. 如果您叢集的 DNS 供應商為 `kube-dns`，請將 `kube-dns` 部署縮減至 1 個複本。

```
kubectl scale deployments/kube-dns --replicas=1 -n kube-system
```

13. (選用) 如果您使用的是 Kubernetes Cluster Autoscaler，請將部署縮減回您想要的複本數量。

```
kubectl scale deployments/cluster-autoscaler --replicas=1 -n kube-system
```

14. (選用) 確認您使用的是最新版本的 Kubernetes 專用 Amazon VPC CNI 外掛程式。您可能需要更新適用於 Kubernetes 的 Amazon VPC CNI 版本，才能使用最新支援的執行個體類型。如需詳細資訊，請參閱the section called “Amazon VPC CNI”。

使用 AWS Fargate 簡化運算管理

本主題討論如何使用 Amazon EKS 在 AWS Fargate 上執行 Kubernetes Pod。Fargate 是一種為容器提供隨需、適當大小運算容量的科技。有了 Fargate，就不再需要自行佈建、設定或擴展虛擬機器的群組來執行容器。您也無需選擇伺服器類型、決定何時擴展節點群組，或最佳化叢集壓縮。

您可以控制哪些 Pod 要在 Fargate 上啟動，以及其如何搭配 Fargate 設定檔執行。Fargate 描述檔會定義為 Amazon EKS 叢集的一部分。Amazon EKS 使用 Kubernetes 提供的 AWS 上游可擴展模型建置的控制器，將 Kubernetes 與 Fargate 整合。這些控制器作為 Amazon EKS 受管 Kubernetes 控制平面的一部分執行，且負責將原生 Kubernetes Pod 排程至 Fargate。Fargate 控制器包括新的排程器，除了數個變換與驗證許可控制器之外，也會隨著預設 Kubernetes 排程器執行。當您啟動符合在 Fargate 上執行的條件的 Pod 時，在叢集中執行的 Fargate 控制器便會辨識、更新 Pod 並將 Pod 排程至 Fargate。

本主題會說明在 Fargate 上執行的不同 Pod 元件，並注意搭配使用 Fargate 與 Amazon EKS 的特殊考量。

AWS Fargate 考量事項

以下是在 Amazon EKS 上使用 Fargate 的考慮因素。

- 在 Fargate 上執行的每個 Pod 都有自己的運算界限。這些 Pod 不會與其他 Pod 共用基礎核心、CPU 資源、記憶體資源或彈性網路介面。
- Network Load Balancer 和 Application Load Balancer (ALB) 可以僅與具有 IP 目標的 Fargate 搭配使用。如需詳細資訊，請參閱the section called “建立 Network Load Balancer”及the section called “應用程式負載平衡”。
- Fargate 的公開服務僅能在目標類型 IP 模式下執行，不能在節點 IP 模式下執行。若要檢查在受管節點上和在 Fargate 上執行的服務的連線狀態，建議透過服務名稱來連線。
- Pod 必須與其排程時的 Fargate 設定檔相符，以在 Fargate 上執行。與 Fargate 設定檔不相符的 Pod 可能會卡在 `Pending` 狀態。如果存在相符的 Fargate 設定檔，您可以刪除已建立的待定 Pod，以將其重新排程至 Fargate。
- Fargate 不支援 Daemonsets。如果您的應用程式需要常駐程式，請重新設定該常駐程式，以作為您 Pod 中的附屬容器執行。
- Fargate 不支援具有特殊權限的容器。
- 在 Fargate 上執行的 Pod 不能指定 Pod 資訊清單中的 `HostPort` 或 `HostNetwork`。
- 預設 `nofile` 和 `nproc` 軟性限制為 1024，而 Fargate Pod 的硬性限制為 65535。
- GPU 目前無法在 Fargate 上使用。
- 在 Fargate 上執行的 Pod 僅支援私有子網路（具有 AWS 服務的 NAT 閘道存取權，但不支援直接路由至網際網路閘道），因此叢集的 VPC 必須具有可用的私有子網路。如需沒有傳出網際網路存取權的叢集，請參閱 the section called “私有叢集”。
- 您可以使用透過 Vertical Pod Autoscaler 來調整 Pod 資源為 Fargate Pod 設定 CPU 和記憶體的初始正確大小，然後使用使用 Horizontal Pod Autoscaler 擴展 Pod 部署來擴展這些 Pod。如果

要 Vertical Pod Autoscaler 自動將 Pod 重新部署到具有較大 CPU 和記憶體組合的 Fargate，請將 Vertical Pod Autoscaler 的模式設定為 `Auto` 或 `Recreate` 以確保正確的功能。如需詳細資訊，請參閱 GitHub 上的 Vertical Pod Autoscaler 文件。

- 您的 VPC 必須啟用 DNS 解析和 DNS 主機名稱。如需詳細資訊，請參閱檢視與更新 VPC 的 DNS 支援。
- Amazon EKS Fargate 透過在虛擬機器 (VM) 中隔離每個 Pod，為 Kubernetes 應用程式增加深度防禦功能。此 VM 界限可防止在容器逸出時存取其他 Pod 使用的主機型資源，這是攻擊容器化應用程式並存取容器外資源的常用方法。

  使用 Amazon EKS 並不會變更您在共同責任模式下所需承擔的責任。您應該審慎考慮叢集安全性和控管控制項的組態。隔離應用程式最安全的方法是永遠在個別的叢集中執行。
- Fargate 描述檔支援從 VPC 次要 CIDR 區塊指定子網路。您可能想要指定次要 CIDR 區塊。這是因為子網路僅提供數量有限的 IP 位址。因此，叢集中可建立的 Pod 數量也有限。如果為 Pod 使用不同的子網路，您可增加可用 IP 位址的數量。如需詳細資訊，請參閱為 VPC 新增 IPv4 CIDR 區塊。
- 部署到 Fargate 節點的 Pod 不提供 Amazon EC2 執行個體中繼資料服務 (IMDS)。如果有部署到 Fargate 且需要 IAM 憑證的 Pod，則請使用服務帳戶的 IAM 角色將其指派給 Pod。如果您的 Pod 需要存取 IMDS 提供的其他資訊，則必須將此資訊硬式編碼至 Pod 規格中。這包括部署 Pod 的區域 AWS 或可用區域。
- 您無法將 Fargate Pod 部署至 AWS Outposts、 AWS Wavelength 或 AWS Local Zones。
- Amazon EKS 必須定期修補 Fargate Pod 來確保其安全。我們嘗試以減少影響的方式進行更新，但有時，如果沒有成功移出 Pod，則必須將其刪除。您可採取一些措施以盡可能地減少中斷。如需詳細資訊，請參閱the section called “OS 修補事件”。
- 適用於 Amazon EKS 的 Amazon VPC CNI 外掛程式會安裝在 Fargate 叢集上。您無法將適用於 Amazon EKS 叢集的替代 CNI 外掛程式與 Fargate 節點搭配使用。
- 在 Fargate 上執行的 Pod 會自動掛載 Amazon EFS 檔案系統，而無需執行手動驅動程式安裝步驟。您不能將動態持久性磁碟區佈建與 Fargate 節點搭配使用，但可以使用靜態佈建。
- Amazon EKS 不支援 Fargate Spot。
- 您無法將 Amazon EBS 磁碟區掛載到 Fargate Pod。
- 您可以在 Fargate 節點上執行 Amazon EBS CSI 控制器，但是 Amazon EBS CSI 節點 DaemonSet 僅能在 Amazon EC2 執行個體上執行。
- 版本轉返：Fargate 工作者節點不支援版本轉返。您可以復原使用 Fargate 之叢集的控制平面，但執行與控制平面相同 Kubernetes 版本的 Fargate Pod 會觸發具有 ERROR 狀態的 kubelet 版本扭曲洞見。如果您有 Fargate Pod 的次要版本符合控制平面版本，請在轉返之前刪除這些 Pod。然後復原您的控制平面。當您重新部署時，任何剩餘的 Pod 都會使用復原版本啟動。或者，使用 `--force`略

過洞見檢查，但繼續進行 kubelet 版本扭曲違規可能會導致意外行為。如需詳細資訊，請參閱the section called “復原 Kubernetes 版本”。

- 在 Kubernetes 任務標記為 Completed 或 Failed 之後，任務建立的 Pod 通常會繼續存在。此行為允許您檢視日誌和結果，但使用 Fargate，如果您之後不清理任務，將產生費用。

  若要在任務完成或失敗後自動刪除相關的 Pod，您可以使用存留時間 (TTL) 控制器來指定時間間隔。下列範例顯示在您的任務資訊清單中指定 .spec.ttlSecondsAfterFinished。

```
apiVersion: batch/v1
kind: Job
metadata:
  name: busybox
spec:
  template:
    spec:
      containers:
      - name: busybox
        image: busybox
        command: ["/bin/sh", "-c", "sleep 10"]
      restartPolicy: Never
  ttlSecondsAfterFinished: 60 # <-- TTL controller
```

Fargate 比較表

條件	AWS Fargate
可以部署到 AWS Outposts	否
可以部署到 AWS Local Zone	否
可以執行需要 Windows 的容器	否
可以執行需要 Linux 的容器	是
可以執行需要 Inferentia 晶片的工作負載	否
可以執行需要 GPU 的工作負載	否
可以執行需要 Arm 處理器的工作負載	否

條件	AWS Fargate
可以執行 AWS Bottlerocket	否
Pod 與其他 Pod 共享核心執行時期環境	否：每個 Pod 都有專用核心
Pod 會與其他 Pod 共享 CPU、記憶體、儲存體和網路資源。	否：每個 Pod 都有專用資源，可以獨立調整大小，以最大化資源使用率。
Pod 可以使用的硬體和記憶體比 Pod 規格要求的更多	否：Pod 可以使用較大的 vCPU 和記憶體組態重新部署。
必須部署和管理 Amazon EC2 執行個體	否
必須保護、維護和修補 Amazon EC2 執行個體的作業系統	否
可以在部署節點時提供引導引數，例如額外的 kubelet 引數。	否
可以將 IP 位址指派給來自不同於將 IP 位址指派給節點之 CIDR 區塊的 Pod。	否
可以對節點執行 SSH	否：沒有節點主機作業系統可供 SSH 使用。
可以將自己的自訂 AMI 部署至節點	否
可以將您自己的自訂 CNI 部署至節點	否
必須自行更新節點 AMI	否
必須自行更新節點 Kubernetes 版本	否：您未管理節點。
可以透過 Pod 使用 Amazon EBS 儲存體	否
可以透過 Pod 使用 Amazon EFS 儲存體	是
可以透過 Pod 使用 Amazon FSx for Lustre 儲存體	否
可以針對服務使用 Network Load Balancer	是，使用建立 Network Load Balancer 時

條件	AWS Fargate
Pod 可以在公有子網路中執行	否
可以將不同的 VPC 安全群組指派給個別 Pod	是
可以執行 Kubernetes DaemonSets	否
支援 Pod 資訊清單中的 `HostPort` 和 `HostNetwork`	否
AWS 區域可用性	部分 Amazon EKS 支援的區域
可以在 Amazon EC2 專用主機上執行容器	否
定價	獨立 Fargate 記憶體和 CPU 組態的成本。每個 Pod 都有自己的成本。如需詳細資訊，請參閱 AWS Fargate 定價。

為您的叢集開始使用 AWS Fargate

本主題說明如何開始使用 Amazon EKS 叢集在 AWS Fargate 上執行 Pod。

如果使用 CIDR 區塊限制對叢集公有端點的存取，則建議您也啟用私有端點存取。這樣一來，Fargate Pod 即可與叢集通訊。若不啟用私有端點，您指定用於公有存取的 CIDR 區塊必須包含來自 VPC 的傳出來源。如需詳細資訊，請參閱the section called “叢集端點存取”。

先決條件

現有的叢集。如果還沒有 Amazon EKS 叢集，請參閱 開始使用。

步驟 1：確認現有節點可以與 Fargate Pod 進行通訊

如果正在使用無節點的新叢集，或叢集只有受管節點群組 (請參閱 the section called “受管節點群組”)，您可以跳至 the section called “步驟 2：建立 Fargate Pod 執行角色”。

假設您正在使用的現有叢集已經具有與其相關聯的節點。請確保在這些節點上的 Pod 可以自由地與在 Fargate 上執行的 Pod 進行通訊。在 Fargate 上執行的 Pod 會自動設定為使用其相關聯之叢集的叢集安全群組。請務必確認叢集中的任何現有節點可以傳送與接收叢集安全群組的流量。受管節點群組也

會自動設定為使用叢集安全群組，因此您不需要為此相容性修改或檢查它們 (請參閱 the section called "受管節點群組")。

對於使用 eksctl或 Amazon EKS Managed AWS CloudFormation 範本建立的現有節點群組，您可以將叢集安全群組手動新增至節點。您還可以修改節點群組的 Auto Scaling 群組啟動範本，從而將叢集安全群組連接至執行個體。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的變更執行個體的安全群組。

您可以在叢集的網路區段 AWS 管理主控台 下的 中，檢查叢集的安全群組。或者，您可以使用下列 CLI AWS 命令來執行此操作。使用此命令時，請以您叢集的名稱取代 <my-cluster>。

```
aws eks describe-cluster --name <my-cluster> --query
 cluster.resourcesVpcConfig.clusterSecurityGroupId
```

步驟 2：建立 Fargate Pod 執行角色

當您的叢集在 AWS Fargate 上建立 Pod 時，在 Fargate 基礎設施上執行的元件必須代表您呼叫 AWS APIs。Amazon EKS Pod 執行角色提供進行此類工作的 IAM 許可。若要建立 AWS Fargate Pod 執行角色，請參閱 the section called "Pod 執行 IAM 角色"。

> **Note**
>
> 如果您使用 --fargate 選項以 eksctl 建立叢集，則您的叢集已擁有 Pod 執行角色，您可以在 IAM 主控台中使用模式 eksctl-my-cluster-FargatePodExecutionRole-ABCDEFGHIJKL 找到該角色。同樣地，如果您使用 eksctl 建立 Fargate 設定檔，eksctl 將建立 Pod 執行角色 (如果該角色尚未建立)。

步驟 3：為您的叢集建立 Fargate 設定檔

您必須定義 Fargate 設定檔，指定哪些 Pod 在啟動時使用 Fargate，然後才能排程在叢集 Fargate 上執行的 Pod。如需詳細資訊，請參閱the section called "定義設定檔"。

> **Note**
>
> 若您使用 --fargate 選項以 eksctl 建立叢集，則您叢集的 Fargate 設定檔已建立完成，並包含在 kube-system 和 default 命名空間中所有 Pod 的選取器。使用下列程序來為您想要搭配 Fargate 使用的任何其他命名空間建立 Fargate 描述檔。

您可以使用下列任一工具建立 Fargate 設定檔：

- the section called “eksctl”
- the section called “AWS 管理主控台”

eksctl

此程序需要 eksctl 版本 0.215.0 或更新版本。您可使用以下命令檢查您的版本：

```
eksctl version
```

如需有關安裝或更新 eksctl 的指示，請參閱 eksctl 文件中的 Installation 一節。

若要使用 **eksctl** 建立 Fargate 設定檔

使用以下 eksctl 命令建立您的 Fargate 設定檔，並以您自己的值取代每一個 <example value>。您必須指定一個命名空間。不過，--labels 選項並非必要項目。

```
eksctl create fargateprofile \
    --cluster <my-cluster> \
    --name <my-fargate-profile> \
    --namespace <my-kubernetes-namespace> \
    --labels <key=value>
```

對於 <my-kubernetes-namespace> 和 <key=value> 標籤可以使用某些萬用字元。如需詳細資訊，請參閱the section called “Fargate 設定檔萬用字元”。

AWS 管理主控台

若要使用 AWS 管理主控台 建立 Fargate 設定檔

1. 開啟 Amazon EKS 主控台。
2. 選擇要為其建立 Fargate 設定檔的叢集。
3. 選擇 Compute (運算) 索引標籤。
4. 在 Fargate profiles (Fargate 設定檔) 下，選擇 Add Fargate profile (新增 Fargate 設定檔)。
5. 在 Configure Fargate profile (設定 Fargate 設定檔) 頁面上，執行以下操作：
   a. 對於 Name (名稱)，輸入您 Fargate 描述檔的名稱。名稱必須是唯一的。

b. 對於 Pod 執行角色，請選擇要搭配您的 Fargate 設定檔使用的 Pod 執行角色。只會顯示具有 `eks-fargate-pods.amazonaws.com` 服務委託人的 IAM 角色。如果未列出任何角色，您必須建立一個。如需詳細資訊，請參閱the section called “Pod 執行 IAM 角色”。

c. 視需要修改選取的子網路。

> **Note**
>
> 只有私有子網路支援在 Fargate 上執行的 Pod。

d. 對於 Tags (標籤)，您可以選擇標記 Fargate 設定檔。這些標籤不會傳播至與設定檔相關聯的其他資源，例如 Pod。

e. 選擇下一步。

6. 在設定 Pod 選擇頁面上，執行以下作業：

a. 針對命名空間，輸入與 Pod 相符的命名空間。

- 您可以使用特定的命名空間進行比對，例如 `kube-system` 或 `default`。
- 您可以使用某些萬用字元 (例如 `prod-*`) 來比對多個命名空間 (例如 `prod-deployment` 和 `prod-test`)。如需詳細資訊，請參閱the section called “Fargate 設定檔萬用字元”。

b. (選用) 將 Kubernetes 標籤新增至選取器。將其特別新增至需與特定命名空間中的 Pod 相符的選取器。

- 您可以將標籤 `infrastructure: fargate` 新增至選取器，因此只有在也有 `infrastructure: fargate` Kubernetes 標籤之指定命名空間中的 Pod 與選取器相符。
- 您可以使用某些萬用字元 (例如 `key?: value?`) 來比對多個命名空間 (例如 `keya: valuea` 和 `keyb: valueb`)。如需詳細資訊，請參閱the section called “Fargate 設定檔萬用字元”。

c. 選擇下一步。

7. 在 Review and create (檢閱和建立) 頁面，檢閱您 Fargate 設定檔的資訊並選擇 Create (建立)。

步驟 4：更新 CoreDNS

CoreDNS 預設為在 Amazon EKS 叢集上的 Amazon EC2 基礎設施上執行。若您希望僅在叢集的 Fargate 上執行 Pod，請完成以下步驟。

> **Note**
>
> 如果透過 `--fargate` 選項使用 `eksctl` 建立叢集，您可以直接跳至 the section called “後續步驟”。

1. 使用下列命令來刪除 CoreDNS 的 Fargate 設定檔。使用您的叢集名稱取代 `<my-cluster>`、使用您的帳戶 ID 取代 `<111122223333>`、使用您的 Pod 執行角色名稱取代 `<AmazonEKSFargatePodExecutionRole>`，並使用私有子網路的 ID 取代 `<000000000000000a>`、`<000000000000000b>` 和 `<000000000000000c>`。若沒有 Pod 執行角色，則必須先建立一個 (請參閱 the section called “步驟 2：建立 Fargate Pod 執行角色”)。

   > **Important**
   >
   > 角色 ARN 不能包含 `/` 之外的路徑。例如，如果您的角色名稱是 `development/apps/AmazonEKSFargatePodExecutionRole`，則您必須在指定角色的 ARN 時將其變更為 `AmazonEKSFargatePodExecutionRole`。角色 ARN 的格式必須是 `arn:aws:iam::<111122223333>:role/<AmazonEKSFargatePodExecutionRole>`。

```
aws eks create-fargate-profile \
    --fargate-profile-name coredns \
    --cluster-name <my-cluster> \
    --pod-execution-role-arn arn:aws: iam::<111122223333>:role/
<AmazonEKSFargatePodExecutionRole> \
    --selectors namespace=kube-system,labels={k8s-app=kube-dns} \
    --subnets subnet-<000000000000000a> subnet-<000000000000000b> subnet-
<000000000000000c>
```

2. 觸發 `coredns` 部署的推展。

```
kubectl rollout restart -n kube-system deployment coredns
```

後續步驟

- 您可以透過以下工作流程開始遷移現有的應用程式，以在 Fargate 上執行。

a. the section called “建立 Fargate 設定檔”，與您的應用程式的 Kubernetes 命名空間和 Kubernetes 標籤相符。

b. 刪除並重建新立任何現有 Pod，以便在 Fargate 上對其進行排程。修改 `<namespace>` 和 `<deployment-type>`，以更新您的特定 Pod。

```
kubectl rollout restart -n <namespace> deployment <deployment-type>
```

- 部署 the section called “應用程式負載平衡” 來允許您在 Fargate 執行的 Pod 輸入物件。
- 您可以使用 the section called “Vertical Pod Autoscaler” 為 Fargate Pod 設定 CPU 和記憶體的初始正確大小，然後使用 the section called “Horizontal Pod Autoscaler” 來擴展這些 Pod。若要讓 Vertical Pod Autoscaler 自動將 Pod 重新部署到具有較高 CPU 和記憶體組合的 Fargate，請將 Vertical Pod Autoscaler 的模式設定為 `Auto` 或 `Recreate`。這是為了確認功能可以運作正確。如需詳細資訊，請參閱 GitHub 上的 Vertical Pod Autoscaler 文件。
- 請遵循以下說明設定 AWS Distro for OpenTelemetry (ADOT) 收集器，以便監控應用程式。

定義哪些 Pod 在啟動時會使用 AWS Fargate

您必須定義至少一個 Fargate 設定檔，指定哪些 Pod 在啟動時使用 Fargate，才能排程您叢集中 Fargate 上的 Pod。

管理員可以使用 Fargate 設定檔宣告哪些 Pod 要在 Fargate 上執行。您可以透過設定檔的選取器來執行此操作。您最多可以將五個選取器新增至每個設定檔。每個選取器必須包含一個命名空間。選取器還可以包括標籤。標籤欄位由多個選用鍵/值對組成。與選取器相符的 Pod 會排程在 Fargate 上執行。比對 Pod 時，會使用命名空間和在選取器中指定的標籤。如果命名空間選取器未以標籤定義，Amazon EKS 會嘗試使用設定檔，將以該命名空間執行的所有 Pod 排程至 Fargate。如果要排程的 Pod 符合 Fargate 設定檔中的任一選取器，則該 Pod 會在 Fargate 上排程。

如果 Pod 與多個 Fargate 設定檔相符，您可以將下列 Kubernetes 標籤新增至 Pod 規格，以指定 Pod 使用哪個設定檔：`eks.amazonaws.com/fargate-profile: my-fargate-profile`。Pod 必須與該設定檔中的選取器相符，才能排程至 Fargate。Kubernetes 親和性/反親和性規則不適用，並且對 Amazon EKS Fargate Pod 而言也非必要。

建立 Fargate 設定檔時，您必須指定 Pod 執行角色。此執行角色適用於使用設定檔在 Fargate 基礎設施上執行的 Amazon EKS 元件。其會新增至叢集的 Kubernetes 角色型存取控制 (RBAC)，以進行身分授權。這樣一來，在 Fargate 基礎設施上執行的 `kubelet` 就可以註冊到您的 Amazon EKS 叢集，並作為節點出現在您的叢集中。Pod 執行角色也為 Fargate 基礎結構提供 IAM 許可，以允許對 Amazon ECR 映像儲存庫的讀取存取權。如需詳細資訊，請參閱 the section called “Pod 執行 IAM 角色”。

Fargate 設定檔無法變更。不過，您可以建立新的已更新設定檔來取代現有設定檔，然後刪除原始的設定檔。

> **Note**
>
> 刪除設定檔時，使用 Fargate 設定檔執行的任何 Pod 將停止並進入待定狀態。

如果在叢集中的任何 Fargate 設定檔處於 `DELETING` 狀態，您必須等候該 Fargate 設定檔刪除後，才能在該叢集建立任何其他設定檔。

> **Note**
>
> Fargate 目前不支援 Kubernetes topologySpreadConstraints。

Amazon EKS 和 Fargate 會嘗試在 Fargate 設定檔所定義的每個子網路中散佈 Pod。不過，最終可能會產生不均勻的散佈結果。如果必須產生均勻的散佈結果，請使用兩個 Fargate 設定檔。在您想要部署兩個複本且不希望任何停機時間的情況下，均勻的散佈結果相當重要。我們建議每個設定檔只定義一個子網路。

Fargate 描述檔元件

下列元件包含在 Fargate 描述檔中。

Pod 執行角色

當您的叢集在 AWS Fargate 上建立 Pod 時，在 Fargate 基礎結構上執行的 `kubelet` 必須代表您呼叫 AWS API。例如，它必須透過呼叫從 Amazon ECR 提取容器映像。Amazon EKS Pod 執行角色提供進行此類工作的 IAM 許可。

建立 Fargate 設定檔時，您必須指定要搭配 Pod 使用的 Pod 執行角色。此角色會新增至叢集的 Kubernetes 角色型存取控制 (RBAC)，以進行身分授權。這是為了讓在 Fargate 基礎結構上執行的 `kubelet` 可以註冊到您的 Amazon EKS 叢集並作為節點出現在您的叢集中。如需詳細資訊，請參閱 the section called “Pod 執行 IAM 角色”。

子網路

要在其中啟動 Pod 並使用該設定檔的子網路 ID。目前不會對在 Fargate 上執行的 Pod 指派公有 IP 位址。因此，此參數僅接受私有子網路 (不具網際網路閘道的直接路由)。

選取器

要與 Pod 相符的選取器，以使用此 Fargate 設定檔。您最多可在一個 Fargate 設定檔中指定五個選取器。選取器具有下列元件：

- 命名空間：您必須指定選取器的命名空間。選取器僅與在此命名空間中建立的 Pod 相符。不過，您可以建立多個選取器來定義多個命名空間。
- 標籤：您可以選擇指定符合選取器的 Kubernetes 標籤。選取器僅與擁有在選取器中指定之所有標籤的 Pod 相符。

Fargate 設定檔萬用字元

除了 Kubernetes 允許的字元之外，您還可以在命名空間、標籤索引鍵和標籤值的選取器條件中使用 * 和 ？：

- * 代表無、一個或多個字元。例如，prod* 可以代表 prod 和 prod-metrics。
- ？代表單一字元 (例如，value? 可以代表 valuea)。不過，它不能代表 value 和 value-a，因為 ？只能代表確切一個字元。

這些萬用字元適用於任何位置和組合 (例如，prod*、*dev，以及 frontend*?)。其他萬用字元和模式比對形式 (例如規則運算式) 則不支援。

如果 Pod 規格中的命名空間和標籤有多個相符的設定檔，Fargate 會根據設定檔名稱的英數字元排序選擇設定檔。例如，如果設定檔 A (名稱為 beta-workload) 和設定檔 B (名稱為 prod-workload) 都具有要啟動的 Pod 的相符選取器，則 Fargate 將為 Pod 選擇設定檔 A (beta-workload)。Pod 在 Pod 上具有含設定檔 A 的標籤 (例如，eks.amazonaws.com/fargate-profile=beta-workload)。

如果您想要將現有的 Fargate Pod 移轉至使用萬用字元的新設定檔，有兩種方法：

- 使用相符的選取器建立新的設定檔，然後刪除舊設定檔。標有舊設定檔的 Pod 會重新排程為新的相符設定檔。
- 如果您想要移轉工作負載，但不確定每個 Fargate Pod 上有哪些 Fargate 標籤，則可使用以下方法。建立新的設定檔，其名稱在同一叢集上的設定檔中按英數字元順序排列在第一位。然後，回收需要移轉到新設定檔的 Fargate Pod。

建立 Fargate 設定檔

本節說明如何建立 Fargate 設定檔。您還必須建立用於 Fargate 設定檔的 Pod 執行角色。如需詳細資訊，請參閱 the section called “Pod 執行 IAM 角色”。只有私有子網路支援在 Fargate 上執行的 Pod (具有對 AWS 服務的 NAT 閘道存取，但沒有網際網路閘道的直接路由)。因此，您叢集的 VPC 必須具有可用的私有子網路。

您可以執行下列操作來建立設定檔：

- the section called “`eksctl`”
- the section called “AWS 管理主控台”

`eksctl`

若要使用 **`eksctl`** 建立 Fargate 設定檔

使用下列 `eksctl` 命令建立您的 Fargate 設定檔，並以您自己的值取代每一個範例值。您必須指定一個命名空間。不過，`--labels` 選項並非必要項目。

```
eksctl create fargateprofile \
    --cluster my-cluster \
    --name my-fargate-profile \
    --namespace my-kubernetes-namespace \
    --labels key=value
```

對於 `my-kubernetes-namespace` 和 `key=value` 標籤可以使用某些萬用字元。如需詳細資訊，請參閱 the section called “Fargate 設定檔萬用字元”。

AWS 管理主控台

若要使用 AWS 管理主控台 建立 Fargate 設定檔

1. 開啟 Amazon EKS 主控台。
2. 選擇要為其建立 Fargate 設定檔的叢集。
3. 選擇 Compute (運算) 索引標籤。
4. 在 Fargate profiles (Fargate 設定檔) 下，選擇 Add Fargate profile (新增 Fargate 設定檔)。
5. 在 Configure Fargate profile (設定 Fargate 設定檔) 頁面上，執行以下作業：

a. 對於 Name (名稱)，輸入 Fargate 設定檔的唯一名稱，例如 `my-profile`。

b. 對於 Pod 執行角色，請選擇要搭配您的 Fargate 設定檔使用的 Pod 執行角色。只會顯示具有 `eks-fargate-pods.amazonaws.com` 服務委託人的 IAM 角色。如果未列出任何角色，您必須建立一個。如需詳細資訊，請參閱 the section called “Pod 執行 IAM 角色”。

c. 視需要修改選取的子網路。

> **Note**
>
> 只有私有子網路支援在 Fargate 上執行的 Pod。

d. 對於 Tags (標籤)，您可以選擇標記 Fargate 設定檔。這些標籤不會傳播至與設定檔相關聯的其他資源，例如 Pod。

e. 選擇下一步。

6. 在設定 Pod 選擇頁面上，執行以下作業：

a. 針對命名空間，輸入與 Pod 相符的命名空間。

- 您可以使用特定的命名空間進行比對，例如 `kube-system` 或 `default`。
- 您可以使用某些萬用字元 (例如 `prod-*`) 來比對多個命名空間 (例如 `prod-deployment` 和 `prod-test`)。如需詳細資訊，請參閱 the section called “Fargate 設定檔萬用字元”。

b. (選用) 將 Kubernetes 標籤新增至選取器。將其特別新增至需與特定命名空間中的 Pod 相符的選取器。

- 您可以將標籤 `infrastructure: fargate` 新增至選取器，因此只有在也有 `infrastructure: fargate` Kubernetes 標籤之指定命名空間中的 Pod 與選取器相符。
- 您可以使用某些萬用字元 (例如 `key?: value?`) 來比對多個命名空間 (例如 `keya: valuea` 和 `keyb: valueb`)。如需詳細資訊，請參閱 the section called “Fargate 設定檔萬用字元”。

c. 選擇下一步。

7. 在 Review and create (檢閱和建立) 頁面，檢閱您 Fargate 設定檔的資訊並選擇 Create (建立)。

刪除 Fargate 描述檔

本主題說明如何刪除 Fargate 設定檔。當您刪除 Fargate 描述檔時，使用描述檔排程至 Fargate 的任何 Pod 都會遭到刪除。若這些 Pod 符合其他 Fargate 設定檔，其便會使用該設定檔在 Fargate 上排程。如果其不再符合任何 Fargate 設定檔，則不會排程至 Fargate，且可能仍為待定狀態。

一個叢集中一次只能有一個 Fargate 描述檔可以處於 `DELETING` 狀態。請等候 Fargate 描述檔完成刪除動作，才可以刪除該叢集中的任何其他描述檔。

您可以使用以下任何工具刪除設定檔：

- the section called “eksctl”
- the section called “AWS 管理主控台”
- the section called “AWS CLI”

eksctl

使用 **eksctl** 刪除 Fargate 設定檔

使用下列命令從叢集刪除設定檔。使用您自己的值取代每一個###。

```
eksctl delete fargateprofile  --name my-profile --cluster my-cluster
```

AWS 管理主控台

使用 AWS 管理主控台 刪除 Fargate 設定檔

1. 開啟 Amazon EKS 主控台。
2. 在左側導覽窗格中選擇叢集。在叢集清單中，選擇您要從中刪除 Fargate 設定檔的叢集。
3. 選擇 Compute (運算) 索引標籤。
4. 選擇要刪除的 Fargate 設定檔，然後選擇 Delete (刪除)。
5. 在 Delete Fargate profile (刪除 Fargate 設定檔) 頁面上，輸入設定檔名稱，然後選擇 Delete (刪除)。

AWS CLI

使用 AWS CLI 刪除 Fargate 設定檔

使用下列命令從叢集刪除設定檔。使用您自己的值取代每一個###。

```
aws eks delete-fargate-profile --fargate-profile-name my-profile --cluster-name my-
cluster
```

了解 Fargate Pod 組態詳細資訊

本節會描述部分適用於在 AWS Fargate 上執行 Kubernetes Pod 的專用 Pod 組態詳細資訊。

Pod CPU 和記憶體

透過 Kubernetes，您可定義請求、最低 vCPU 數量，以及配置給 Pod 中每個容器的記憶體資源。Pod 是由 Kubernetes 排程，以確保至少每個 Pod 的請求資源可在運算資源上使用。如需詳細資訊，請參閱 Kubernetes 文件中的管理容器的運算資源。

> **Note**
>
> 由於 Amazon EKS Fargate 的每個節點僅會執行一個 Pod，因此不會發生在資源較少的情況下將 Pod 移出的情況。所有 Amazon EKS Fargate Pod 都是以保證優先順序的方式執行，因此請求的 CPU 和記憶體必須等同於所有容器的限制。如需詳細資訊，請參閱《Kubernetes 說明文件》中的設定 Pod 的服務品質。

當 Pod 在 Fargate 上排程時，在 Pod 規格內的 vCPU 與記憶體保留決定了多少 CPU 和記憶體會佈建給 Pod。

- 任何 Init 容器的最大請求用於判斷 Init 請求 vCPU 和記憶體需求。
- 所有長期執行容器的請求會加總，以判斷長期執行請求 vCPU 和記憶體需求。
- 先前兩個值中較大的值會選擇作為用於您 Pod 的 vCPU 和記憶體請求。
- Fargate 會將 256 MB 新增至所需 Kubernetes 元件之每個 Pod 的記憶體保留 (`kubelet`、`kube-proxy` 和 `containerd`)。

Fargate 會四捨五入至以下最接近 vCPU 和記憶體請求總和的運算組態，以確保 Pod 隨時擁有其執行所需的資源。

若您未指定 vCPU 和記憶體的組合，則會使用最小可用的組合 (0.25 個 vCPU 和 0.5 GB 記憶體)。

下表顯示可用於在 Fargate 上執行之 Pod 的 vCPU 和記憶體組合。

vCPU 數值	記憶體數值
.25 vCPU	0.5 GB、1 GB、2 GB
.5 vCPU	1 GB、2 GB、3 GB、4 GB
1 vCPU	2 GB、3 GB、4 GB、5 GB、6 GB、7 GB、8 GB

vCPU 數值	記憶體數值
2 vCPU	介於 4 GB 與 16 GB 之間，以 1 GB 為單位遞增
4 vCPU	介於 8 GB 與 30 GB 之間，以 1 GB 為單位遞增
8 vCPU	介於 16 GB 與 60 GB 之間，以 4 GB 為單位遞增
16 vCPU	介於 32 GB 與 120 GB 之間，以 8 GB 為單位遞增

為 Kubernetes 元件預訂的額外記憶體，可能會導致 Fargate 任務佈建了超出原先請求數量的 vCPU。例如，對 1 個 vCPU 和 8 GB 記憶體的請求將會新增 256 MB 至其記憶體請求，並且會佈建具有 2 個 vCPU 和 9 GB 記憶體的 Fargate 任務，因為沒有任何具有 1 個 vCPU 和 9 GB 記憶體的可用任務。

在 Fargate 上執行的 Pod 大小與 Kubernetes 使用 `kubectl get nodes` 報告的節點大小之間沒有相關性。報告的節點大小通常大於 Pod 的容量。您可以使用下列命令確認 Pod 容量。以您 Pod 的命名空間來取代*##*，並以您 Pod 的名稱來取代 *pod-name*。

```
kubectl describe pod --namespace default pod-name
```

範例輸出如下。

```
[...]
annotations:
    CapacityProvisioned: 0.25vCPU 0.5GB
[...]
```

`CapacityProvisioned` 註釋代表強制執行的 Pod 容量，並決定在 Fargate 上執行 Pod 的成本。如需運算組態的定價資訊，請參閱 AWS Fargate 定價。

Fargate 儲存

在 Fargate 上執行的 Pod 會自動掛載 Amazon EFS 檔案系統，而無需執行手動驅動程式安裝步驟。您不能將動態持久性磁碟區佈建與 Fargate 節點搭配使用，但可以使用靜態佈建。如需詳細資訊，請參閱 GitHub 上的 Amazon EFS CSI 驅動程式。

佈建時，在 Fargate 上執行的每個 Pod 均會收到預設 20 GiB 的暫時性儲存。此類型的儲存會在 Pod 停止後刪除。在 Fargate 上啟動的新 Pod，預設皆會啟用暫時性儲存磁碟區的加密。暫時性 Pod 儲存會以使用 AWS Fargate 受管金鑰的 AES-256 加密演算法來加密。

> **Note**
>
> 在 Fargate 上執行的 Amazon EKS Pod 預設可用儲存空間少於 20 GiB。這是因為部分空間由 Pod 內部載入的 `kubelet` 和其他 Kubernetes 模組使用。

您可以增加暫時性儲存的總量，最多可達 175 GiB。若要使用 Kubernetes 設定大小，請將 `ephemeral-storage` 資源的請求指定到 Pod 中的每個容器。當 Kubernetes 排程了 Pod，它可以確保每個 Pod 的資源請求總計少於 Fargate 任務的容量。如需詳細資訊，請參閱 Kubernetes 文件中的 Pod 和容器的資源管理。

Amazon EKS Fargate 佈建的暫時性儲存空間比系統使用目的之要求更多。例如，100 GiB 的請求將佈建具有 115 GiB 暫時性儲存的 Fargate 任務。

設定 AWS Fargate OS 修補事件的動作

Amazon EKS 會定期修補 AWS Fargate 節點的 OS 來確保節點的安全。作為修補程序的一部分，我們會回收節點以安裝作業系統修補程式。以對您的服務產生最小影響的方式嘗試更新。但是，如果未成功移出 Pod，有時必須將其刪除。您可以採取以下措施，以最大限度地減少潛在的中斷：

- 設定適當的 Pod 中斷預算 (PDB)，以控制同時關閉的 Pod 數量。
- 建立 Amazon EventBridge 規則，以便在刪除 Pod 之前處理失敗的移出事件。
- 在您收到的通知中發布的移出日期之前，手動重新啟動受影響的 Pod。
- 在 AWS 使用者通知中建立通知組態。

Amazon EKS 與 Kubernetes 社群密切合作，以盡快提供錯誤修復和安全修補程式。所有 Fargate Pod 均從最新的 Kubernetes 修補程式版本開始，該修補程式版本可從 Amazon EKS 取得，用於您叢集的 Kubernetes 版本。若您的 Pod 具有較舊的修補程式版本，Amazon EKS 可能會予以回收，以將其更新到最新版本。這可確保您的 Pod 配備了最新的安全更新。這樣，如果有一個關鍵 Common Vulnerabilities and Exposures (CVE) 問題，您將隨時了解最新資訊，以降低安全風險。

更新 AWS Fargate OS 時，Amazon EKS 會傳送通知給您，其中包括受影響的資源和即將移出 Pod 的日期。如果提供的移出日期不方便，您可以選擇在通知中發布的移出日期之前，手動重新啟動受影響的

Pod。在您收到通知之前建立的任何 Pod 都會被移出。如需如何手動重新啟動 Pod 的進一步指示，請參閱 Kubernetes 文件。

若要限制在回收 Pod 時同時關閉的 Pod 數量，您可以設定 Pod 中斷預算 (PDB)。您可以使用 PDB 根據每個應用程式的要求定義最低可用性，同時仍允許進行更新。PDB 的最低可用性必須小於 100%。如需詳細資訊，請參閱 Kubernetes 文件中的為應用程式指定中斷預算。

Amazon EKS 使用移出 API 以安全地消耗 Pod，同時尊重您為應用程式設定的 PDB。可用區域將 Pod 移出，以最大限度地減少影響。如果移出成功，新 Pod 將會取得最新的修補程式，且無須採取進一步的動作。

當 Pod 的移出失敗時，Amazon EKS 會向您的帳戶傳送一個事件，其中包含有關移出失敗的 Pod 的詳細資訊。您可以在計劃終止時間之前對訊息執行操作。具體時間根據修補程式的緊迫性而有所不同。到了該時間時，Amazon EKS 會再次嘗試移出 pod。但是，這次如果移出失敗，則不會傳送新事件。如果移出再次失敗，則會定期刪除現有的 Pod，以便新 Pod 具有最新的修補程式。

以下是當 Pod 移出失敗時收到的範例事件。其中包含有關叢集、Pod 名稱、Pod 命名空間、Fargate 設定檔和計劃終止時間的詳細資訊。

```
{
    "version": "0",
    "id": "12345678-90ab-cdef-0123-4567890abcde",
    "detail-type": "EKS Fargate Pod Scheduled Termination",
    "source": "aws.eks",
    "account": "111122223333",
    "time": "2021-06-27T12:52:44Z",
    "region": "region-code",
    "resources": [
        "default/my-database-deployment"
    ],
    "detail": {
        "clusterName": "my-cluster",
        "fargateProfileName": "my-fargate-profile",
        "podName": "my-pod-name",
        "podNamespace": "default",
        "evictErrorMessage": "Cannot evict pod as it would violate the pod's disruption
 budget",
        "scheduledTerminationTime": "2021-06-30T12:52:44.832Z[UTC]"
    }
}
```

此外，將多個 PDB 與 Pod 關聯可能會導致移出失敗事件。此事件傳回以下錯誤訊息。

```
"evictErrorMessage": "This pod has multiple PodDisruptionBudget, which the eviction
 subresource does not support",
```

您可以根據此事件建立所需的動作。例如，您可以調整 Pod 中斷預算 (PDB)，以控制如何移出 Pod。更具體地說，假設您從指定可用 Pod 的目標百分比的 PDB 開始。在升級期間強制終止 Pod 之前，您可以將 PDB 調整為不同百分比的 Pod。若要接收此事件，您必須在叢集所屬的 AWS 帳戶和 AWS 區域中建立 Amazon EventBridge 規則。規則必須使用以下自訂模式。如需詳細資訊，請參閱《Amazon EventBridge 使用者指南》中的建立對事件做出反應的 Amazon EventBridge 規則。

```
{
  "source": ["aws.eks"],
  "detail-type": ["EKS Fargate Pod Scheduled Termination"]
}
```

可以為事件設定合適的目標以進行擷取。如需可用目標的完整清單，請參閱《Amazon EventBridge 使用者指南》中的 Amazon EventBridge 目標。您也可以在 AWS 使用者通知中建立通知組態。使用 AWS 管理主控台 建立通知時，請在事件規則下，選擇 Elastic Kubernetes Service (EKS) 作為 AWS 服務名稱，並選擇 EKS Fargate Pod 排程終止 作為事件類型。如需詳細資訊，請參閱《AWS 使用者通知使用者指南》中的 AWS 使用者通知入門。

如需有關 EKS Pod 移出的常見問答集，請參閱 AWS re:Post 中的常見問答集：Fargate Pod 移出通知。

收集 AWS Fargate 應用程式和用量指標

您可以收集 AWS Fargate 的系統指標和 CloudWatch 用量指標。

應用程式指標

對於在 Amazon EKS 和 AWS Fargate 上執行的應用程式，您可以使用 AWS Distro for OpenTelemetry (ADOT)。ADOT 允許您收集系統指標並將其傳送到 CloudWatch Container Insights 儀表板。若要在 Fargate 上執行的應用程式開始使用 ADOT，請參閱 ADOT 文件中的使用 CloudWatch Container Insights 搭配 AWS Distro for OpenTelemetry。

用量指標

您可以使用 CloudWatch 用量指標來了解您帳戶的資源用量。使用這些指標，以 CloudWatch 圖表和儀表板視覺化目前的服務使用狀況。

AWS Fargate 用量指標對應至 AWS 服務配額。您可以設定警示，在您的用量接近服務配額時發出警示。如需 Fargate 的服務配額詳細資訊，請參閱 the section called “Service Quotas”。

AWS Fargate 會在 `AWS/Usage` 命名空間中發佈下列指標。

指標	Description
`ResourceCount`	您的帳戶中正在執行的特定資源總數。資源由與指標相關聯的維度定義。

下列維度用於精簡 AWS Fargate 發佈的用量指標。

維度	Description
`Service`	包含資源 AWS 的服務名稱。對於 AWS Fargate 用量指標，此維度的值為 `Fargate`。
`Type`	正在報告的實體類型。目前， AWS Fargate 用量指標的唯一有效值為 `Resource`。
`Resource`	正在執行的資源類型。 目前， AWS Fargate 會傳回 Fargate 隨需用量的相關資訊。Fargate 隨需用量的資源值為 `OnDemand`。 [注意] ==== Fargate 隨需用量結合了使用 Fargate 的 Amazon EKS Pod、使用 Fargate 啟動類型的 Amazon EKS 任務，以及使用 `FARGATE` 容量提供者的 Amazon ECS 任務。 ====
`Class`	正在追蹤的資源類別。目前， AWS Fargate 不使用類別維度。

建立 CloudWatch 警示來監控 Fargate 資源用量指標

AWS Fargate 提供對應至 Fargate 隨需資源用量 AWS 服務配額的 CloudWatch 用量指標。在 Service Quotas 主控台中，您可以在圖表上一目了然地查看使用情況。您也可以設定警示，在您的用量接近服務配額時發出警示。如需詳細資訊，請參閱the section called “收集指標”。

使用下列步驟，根據其中一個 Fargate 資源用量指標來建立 CloudWatch 警示。

1. 開啟 Service Quotas 主控台，網址為 https://console.aws.amazon.com/servicequotas/。
2. 在左側導覽窗格中，選擇 AWS 服務。
3. 從 AWS 服務清單中，搜尋並選取 AWS Fargate。
4. 在 Service quotas (服務配額) 清單中，選擇您要為其建立警示的 Fargate 用量配額。
5. 在 Amazon CloudWatch alarms (Amazon CloudWatch 警示) 區段中，選擇 Create (建立)。
6. 針對 Alarm threshold (警示閾值)，選擇您想要多少百分比的套用配額值設為警示值。
7. 針對 Alarm name (警示名稱)，輸入警示的名稱，然後選擇 Create (建立)。

為您的叢集啟動 AWS Fargate 記錄

Amazon EKS on Fargate 提供了基於 Fluent Bit 的內建日誌路由器。這表示您沒有明確地執行 Fluent Bit 容器以作為附屬，但 Amazon 會為您執行。您只需設定日誌路由器即可。組態會透過必須符合以下條件的專用 `ConfigMap` 生效：

- 名稱 `aws-logging`
- 在名為 `aws-observability` 的專用命名空間中建立
- 不可超過 5300 個字元。

一旦建立了 `ConfigMap`，Amazon EKS on Fargate 會使用其自動偵測並設定日誌路由器。Fargate 使用 AWS 適用於 Fluent Bit 的 版本，Fluent Bit 是由 管理的上游合規分發 AWS。如需詳細資訊，請參閱 GitHub 上的適用於 Fluent Bit 的AWS。

日誌路由器可讓您在 使用廣泛的服務 AWS 進行日誌分析和儲存。您可以將日誌直接從 Fargate 串流到 Amazon CloudWatch 和 Amazon OpenSearch Service。您也可以透過 Amazon Data Firehose 將日誌串流至 Amazon S3、Amazon Kinesis Data Streams 與合作夥伴工具等目的地。

- 指定您部署 Fargate Pod 的現有 Kubernetes 命名空間的現有 Fargate 設定檔。如需詳細資訊，請參閱 the section called “步驟 3：為您的叢集建立 Fargate 設定檔”。

- 現有 Fargate Pod 執行角色。如需詳細資訊，請參閱 the section called “步驟 2：建立 Fargate Pod 執行角色”。

日誌路由器組態

> **Important**
>
> 若要成功發布日誌，您叢集所在的 VPC 必須具有對日誌目的地的網路存取權。這主要涉及使用者為其 VPC 自訂輸出規則。如需使用 CloudWatch 的範例，請參閱《Amazon CloudWatch Logs 使用者指南》中的搭配介面 VPC 端點使用 CloudWatch Logs。

在下列步驟中，使用您自己的值取代每一個###。

1. 建立名為 aws-observability 的專用 Kubernetes 命名空間。

   a. 將下列內容儲存到電腦上名為 aws-observability-namespace.yaml 的檔案中。name 的值必須為 aws-observability，且需要 aws-observability: enabled 標籤。

   ```
   kind: Namespace
   apiVersion: v1
   metadata:
     name: aws-observability
     labels:
       aws-observability: enabled
   ```

   b. 建立命名空間。

   ```
   kubectl apply -f aws-observability-namespace.yaml
   ```

2. 建立具有 Fluent Conf 資料值的 ConfigMap，將容器日誌寄送至目的地。Fluent Conf 就是 Fluent Bit，這是一種快速且輕量的日誌處理器組態語言，可用於將容器日誌路由至您選擇的日誌目的地。如需詳細資訊，請參閱 Fluent Bit 文件中的組態檔案。

   > **Important**
   >
   > 一般 Fluent Conf 中包含的主要區段是 Service、Input、Filter 和 Output。然而，Fargate 日誌路由器僅接受：
   >
   > - Filter 和 Output 區段。

- Parser 區段。

如果您提供任何其他區段，則這些區段將被拒絕。

Fargate 日誌路由器管理 Service 和 Input 區段。其具有下列 Input 區段，無法修改，也不需要在您的 ConfigMap 中。但是，您可以從中取得洞察，例如記憶體緩衝區限制和套用於日誌的標籤。

```
[INPUT]
    Name tail
    Buffer_Max_Size 66KB
    DB /var/log/flb_kube.db
    Mem_Buf_Limit 45MB
    Path /var/log/containers/*.log
    Read_From_Head On
    Refresh_Interval 10
    Rotate_Wait 30
    Skip_Long_Lines On
    Tag kube.*
```

建立 ConfigMap 時，請考慮下列 Fargate 用來驗證欄位的規則：

- 應該在每個相應的索引鍵下指定 [FILTER]、[OUTPUT] 和 [PARSER]。例如，[FILTER] 必須位於 filters.conf 之下。您可以在 filters.conf 下有一或多個 [FILTER]。[OUTPUT] 和 [PARSER] 區段也應該位於其相應的索引鍵之下。透過指定多個 [OUTPUT] 區段中，您可以同時將日誌路由到不同的目的地。
- Fargate 驗證每個區段所需的索引鍵。Name 和 match 對於每個 [FILTER] 和 [OUTPUT] 都是必要項目。Name 和 format 對於每個 [PARSER] 都是必要項目。索引鍵不區分大小寫。
- ConfigMap 中不允許環境變數 (例如 ${ENV_VAR})。
- 在每個 filters.conf、output.conf 和 parsers.conf 中每個指令或鍵值對的縮排必須是相同的。鍵值對必須比指令縮排更多。
- Fargate 會根據以下支援的篩選條件進行驗證：grep、parser、record_modifier、rewrite_tag、throttle、nest、modify 和 kubernetes。
- Fargate 會根據以下支援的輸出進行驗證：es、firehose、kinesis_firehose、cloudwatch、cloudwatch_logs 和 kinesis。

- 必須在 ConfigMap 中提供至少一個支援的 Output 外掛程式來啟用記錄。啟用記錄時不需要 Filter 和 Parser。

  您也可以使用所需的組態在 Amazon EC2 上執行 Fluent Bit，以對因驗證而產生的任何問題進行故障診斷。使用下列範例之一建立您的 ConfigMap。

  > **Important**
  >
  > Amazon EKS Fargate 記錄不支援 ConfigMap 的動態組態。ConfigMap 的任何變更僅適用於新的 Pod。變更不會套用至現有 Pod。

  使用您所需的日誌目的地範例建立 ConfigMap。

  > **Note**
  >
  > 您也可以將 Amazon Kinesis Data Streams 用於您的日誌目的地。如果您使用 Kinesis Data Streams，請確定 Pod 執行角色已獲 kinesis:PutRecords 許可。如需詳細資訊，請參閱《Fluent Bit：官方手冊》中的 Amazon Kinesis Data Streams 許可。

Example

CloudWatch

使用 CloudWatch 時，您有兩個輸出選項：

- 用 C 編寫的輸出外掛程式
- 用 Golang 編寫的輸出外掛程式

以下範例顯示如何使用 cloudwatch_logs 外掛程式將日誌傳送至 CloudWatch。

a. 將下列內容儲存到名為 aws-logging-cloudwatch-configmap.yaml 的檔案中。將 *region-code* 取代為您的叢集所在的 AWS 區域。[OUTPUT] 下的參數為必要項目。

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: aws-logging
  namespace: aws-observability
data:
```

```
  flb_log_cw: "false"  # Set to true to ship Fluent Bit process logs to
 CloudWatch.
  filters.conf: |
    [FILTER]
        Name parser
        Match *
        Key_name log
        Parser crio
    [FILTER]
        Name kubernetes
        Match kube.*
        Merge_Log On
        Keep_Log Off
        Buffer_Size 0
        Kube_Meta_Cache_TTL 300s
  output.conf: |
    [OUTPUT]
        Name cloudwatch_logs
        Match   kube.*
        region region-code
        log_group_name my-logs
        log_stream_prefix from-fluent-bit-
        log_retention_days 60
        auto_create_group true
  parsers.conf: |
    [PARSER]
        Name crio
        Format Regex
        Regex ^(?<time>[^ ]+) (?<stream>stdout|stderr) (?<logtag>P|F) (?
<log>.*)$
        Time_Key    time
        Time_Format %Y-%m-%dT%H:%M:%S.%L%z
```

b. 將清單檔案套用至叢集。

```
kubectl apply -f aws-logging-cloudwatch-configmap.yaml
```

Amazon OpenSearch Service

如果您想要將日誌傳送到 Amazon OpenSearch Service，您可以使用 es 輸出，這是用 C 語言編寫的外掛程式。下列範例說明如何使用該外掛程式將日誌傳送到 OpenSearch。

a. 將下列內容儲存到名為 `aws-logging-opensearch-configmap.yaml` 的檔案中。使用您自己的值取代每一個`###`。

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: aws-logging
  namespace: aws-observability
data:
  output.conf: |
    [OUTPUT]
      Name  es
      Match *
      Host  search-example-gjxdcilagiprbglqn42jsty66y.region-
code.es.amazonaws.com
      Port  443
      Index example
      Type  example_type
      AWS_Auth On
      AWS_Region region-code
      tls   On
```

b. 將清單檔案套用至叢集。

```
kubectl apply -f aws-logging-opensearch-configmap.yaml
```

Firehose

將日誌傳送到 Firehose 時，您有兩個輸出選項：

- kinesis_firehose：用 C 語言編寫的輸出外掛程式。
- firehose：用 Golang 編寫的輸出外掛程式。

以下範例顯示如何使用 `kinesis_firehose` 外掛程式將日誌傳送到 Firehose。

a. 將下列內容儲存到名為 `aws-logging-firehose-configmap.yaml` 的檔案中。將 *region-code* 取代為您的叢集所在的 AWS 區域。

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: aws-logging
  namespace: aws-observability
data:
  output.conf: |
    [OUTPUT]
```

```
    Name  kinesis_firehose
    Match *
    region region-code
    delivery_stream my-stream-firehose
```

b. 將清單檔案套用至叢集。

```
kubectl apply -f aws-logging-firehose-configmap.yaml
```

3. 為 Fargate Pod 執行角色設定許可，以將日誌傳送至目的地。

a. 將您目的地的 IAM 政策下載到您的電腦。

Example

CloudWatch

將 CloudWatch IAM 政策下載到您的電腦。您也可以在 GitHub 上檢視政策。

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-eks-fluent-
logging-examples/mainline/examples/fargate/cloudwatchlogs/permissions.json
```

Amazon OpenSearch Service

將 OpenSearch IAM 政策下載到您的電腦。您也可以在 GitHub 上檢視政策。

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-eks-
fluent-logging-examples/mainline/examples/fargate/amazon-elasticsearch/
permissions.json
```

確認 OpenSearch Dashboards 的存取控制設定正確。OpenSearch 儀表板中的 `all_access role` 需要具有 Fargate Pod 執行角色和 IAM 角色映射。必須為 `security_manager` 角色進行相同的映射。您可以新增先前的映射，方法是選取 `Menu`、`Security`、`Roles`，然後選取個別的角色。如需詳細資訊，請參閱如何對 CloudWatch Logs 進行故障診斷，以便將其串流至我的 Amazon ES 網域？。

Firehose

將 Firehose IAM 政策下載到您的電腦。您也可以在 GitHub 上檢視政策。

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-eks-fluent-
logging-examples/mainline/examples/fargate/kinesis-firehose/permissions.json
```

b. 從您下載的政策檔案建立 IAM 政策。

```
aws iam create-policy --policy-name eks-fargate-logging-policy --policy-document
 file://permissions.json
```

c. 使用下列命令，將 IAM 政策連接至為 Fargate 設定檔指定的 Pod 執行角色。使用您的帳戶 ID 取代 *111122223333*。使用您的 Pod 執行角色取代 *AmazonEKSFargatePodExecutionRole* (如需詳細資訊，請參閱 the section called “步驟 2：建立 Fargate Pod 執行角色”)。

```
aws iam attach-role-policy \
  --policy-arn arn:aws: iam::111122223333:policy/eks-fargate-logging-policy \
  --role-name AmazonEKSFargatePodExecutionRole
```

Kubernetes 篩選條件支援

Fluent Bit Kubernetes 篩選條件可讓您將 Kubernetes 中繼資料新增至您的日誌檔案。如需有關篩選條件的相關資訊，請參閱 Fluent Bit 說明文件中的 Kubernetes。您可以使用 API 伺服器端點來套用篩選條件。

```
filters.conf: |
    [FILTER]
        Name             kubernetes
        Match            kube.*
        Merge_Log           On
        Buffer_Size         0
        Kube_Meta_Cache_TTL 300s
```

Important

- Kube_URL、Kube_CA_File、Kube_Token_Command 和 Kube_Token_File 是服務擁有的組態參數，您無法指定這些參數。Amazon EKS Fargate 會將這些值填入。
- Kube_Meta_Cache_TTL 是 Fluent Bit 等待直到其可以與 API 伺服器通訊，以取得最新中繼資料所需的時間。若並未指定 Kube_Meta_Cache_TTL，則 Amazon EKS Fargate 會附加預設值 30 分鐘，以減少 API 伺服器上的負載。

將 Fluent Bit 程序日誌傳送至帳戶

您可以選擇性地使用以下 ConfigMap 將 Fluent Bit 程序日誌傳送至 Amazon CloudWatch。將 Fluent Bit 處理日誌傳送至 CloudWatch 需要額外的記錄擷取和儲存費用。將 *region-code* 取代為您的叢集所在的 AWS 區域。

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: aws-logging
  namespace: aws-observability
  labels:
data:
  # Configuration files: server, input, filters and output
  # ======================================================
  flb_log_cw: "true"  # Ships Fluent Bit process logs to CloudWatch.

  output.conf: |
    [OUTPUT]
        Name cloudwatch
        Match kube.*
        region region-code
        log_group_name fluent-bit-cloudwatch
        log_stream_prefix from-fluent-bit-
        auto_create_group true
```

日誌位於與叢集位於相同區域的 CloudWatch AWS 中。日誌群組的名稱是 *my-cluster*-fluent-bit-logs，而 Fluent Bit logstream 的名稱是 fluent-bit-*podname*-*pod-namespace* 。

> **Note**
>
> - 只有當 Fluent Bit 程序成功啟動時，才會傳送程序日誌。若在啟動 Fluent Bit 時失敗，程序日誌就會遺失。您只能將程序日誌傳送至 CloudWatch。
> - 若要對傳送程序日誌至您的帳戶除錯，您可以套用先前的 ConfigMap 來取得程序日誌。Fluent Bit 無法啟動通常是由於您的 ConfigMap 開始時無法被 Fluent Bit 剖析或接受。

若要停止傳送 Fluent Bit 程序日誌

將 Fluent Bit 處理日誌傳送至 CloudWatch 需要額外的記錄擷取和儲存費用。若要排除現有 ConfigMap 設定中的處理日誌，請執行下列步驟。

1. 在啟用 Fargate 記錄後，找出為 Amazon EKS 叢集的 Fluent Bit 程序日誌自動建立的 CloudWatch 日誌群組。它遵循格式 *my-cluster*-fluent-bit-logs。
2. 刪除針對 CloudWatch 日誌群組中每個 Pod 程序日誌所建立的現有 CloudWatch 日誌串流。
3. 編輯 ConfigMap 和設定 flb_log_cw: "false"。
4. 重新啟動叢集中的任何現有 Pod。

測試應用程式

1. 部署範例 Pod。
   a. 將下列內容儲存到電腦上名為 sample-app.yaml 的檔案中。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sample-app
  namespace: same-namespace-as-your-fargate-profile
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:latest
          ports:
            - name: http
              containerPort: 80
```

   b. 將清單檔案套用至叢集。

```
kubectl apply -f sample-app.yaml
```

2. 使用您在 ConfigMap 中設定的目的地檢視 NGINX 日誌。

大小考量因素

建議您為日誌路由器規劃最多 50 MB 的記憶體。如果您希望應用程式以非常高的輸送量產生日誌，則應該規劃最多 100 MB。

疑難排解

若要確認記錄功能是因某些原因 (例如無效的 ConfigMap 以及無效的原因) 啟用或停用，請使用 `kubectl describe pod` *pod-name* 檢查您的 Pod 事件。輸出可能包含可釐清是否已啟用記錄的 Pod 事件，例如下列範例輸出。

```
[...]
Annotations:          CapacityProvisioned: 0.25vCPU 0.5GB
                      Logging: LoggingDisabled: LOGGING_CONFIGMAP_NOT_FOUND
[...]
Events:
  Type     Reason           Age        From
              Message
  ----     ------           ----       ----
              -------
  Warning  LoggingDisabled  <unknown>  fargate-scheduler
               Disabled logging because aws-logging configmap was not found. configmap
 "aws-logging" not found
```

Pod 事件是暫時性的，具體取決於設定的時段。您也可以使用 `kubectl describe pod` *pod-name* 來檢視 Pod 註釋。在 Pod 註釋中，可參閱是否已啟用或停用記錄功能及其原因的相關資訊。

選擇最佳的 Amazon EC2 節點執行個體類型

Amazon EC2 為工作節點提供多種執行個體類型選擇。每個執行個體類型皆提供不同的運算、記憶體、儲存與網路功能。每個執行個體也會依照這些功能分組為執行個體系列。有關清單，請參閱《Amazon EC2 使用者指南》中的可用的執行個體類型。Amazon EKS 發佈了多種 Amazon EC2 AMI 變體以啟用支援。若要確保您選取的執行個體類型與 Amazon EKS 相容，請考慮以下條件。

- 所有 Amazon EKS AMI 目前不支援 `mac` 系列。
- Arm 和非加速的 Amazon EKS AMI 不支援 `g3`、`g4`、`inf` 及 `p` 系列。
- 加速的 Amazon EKS AMI 不支援 `a`、`c`、`hpc`、`m` 及 `t` 系列。
- 對於 ARM 型執行個體，Amazon Linux 2023 (AL2023) 僅支援使用 Graviton2 或更新版本處理器的執行個體類型。AL2023 不支援 `A1` 執行個體。

在 Amazon EKS 支援的執行個體類型之間進行選擇時，請考慮每種類型的以下功能。

節點群組中的執行個體數量

一般來說，較少、較大的執行個體更好，特別是在有很多 Daemonsets. 的情況下。每個執行個體都需要對 API 伺服器進行 API 呼叫，因此擁有的執行個體越多，API 伺服器上的負載就越大。

作業系統

檢閱 Linux、Windows 以及 Bottlerocket 支援的執行個體類型。建立 Windows 執行個體前，請檢閱在 EKS 叢集上部署 Windows 節點。

硬體架構

您需要 x86 還是 Arm？ 部署 Arm 執行個體前，請檢閱 Amazon EKS 最佳化 Arm Amazon Linux AMI。您是否需要建置在 Nitro 系統 (Linux 或 Windows) 上的執行個體，或具有加速功能的執行個體？ 如果您需要加速功能，則只能將 Linux 與 Amazon EKS 搭配使用。

Pod 數目上限

由於每個 Pod 都指派了自己的 IP 位址，因此執行個體類型支援的 IP 位址數量是決定可在執行個體上執行之 Pod 數量的因素。若要了解如何決定執行個體類型的 Pod 數量上限，請參閱 the section called “maxPods的判斷方式”。

AWS Nitro 系統執行個體類型可選擇性地支援比非 Nitro 系統執行個體類型更多的 IP 地址。然而，並非為執行個體指派的所有 IP 位址都可用於 Pod。要為您的執行個體指派大量的 IP 地址，您必須在叢集中安裝並適當設定 Amazon VPC CNI 附加元件 `1.9.0` 或更新版本。如需詳細資訊，請參閱the section called “增加 IP 位址”。要將最大數量的 IP 地址指派給執行個體，您必須在叢集中安裝版本 `1.10.1` 或更新版本的 Amazon VPC CNI 附加元件，並使用 `IPv6` 系列部署叢集。

IP 系列

在將 `IPv4` 系列用於叢集時，您可以使用任何受支援的執行個體類型，這允許您的叢集將私有 `IPv4` 位址指派給您的 Pod 和服務。但是，如果您想在叢集中使用 `IPv6` 系列，則必須使用 AWS Nitro 系統執行個體類型或裸機類型。Windows 執行個體僅支援 `IPv4`。您的叢集必須是執行版本 `1.10.1` 或更新版本 Amazon VPC CNI 附加元件的叢集。如需有關使用 `IPv6` 的詳細資訊，請參閱 the section called “IPv6”。

您正在執行的 Amazon VPC CNI 附加元件版本

最新版的 Kubernetes 專用 Amazon VPC CNI 外掛程式支援這些執行個體類型。您可能需要更新 Amazon VPC CNI 附加元件版本，才能利用最新支援的執行個體類型。如需詳細資訊，請參閱the

section called “Amazon VPC CNI”。最新版本支援與 Amazon EKS 搭配使用的最新功能。舊版不支援所有功能。您可以在 GitHub 上的變更日誌檢視不同版本支援的功能。

您正在其中建立節點的 AWS 區域

並非所有 AWS 區域都提供所有執行個體類型。

您是否使用 Pod 的安全群組

如果您正在使用 Pod 的安全群組，則僅支援特定的執行個體類型。如需詳細資訊，請參閱the section called “Pod 的安全群組”。

maxPods的判斷方式

套用至節點的最終maxPods值取決於以特定優先順序互動的數個元件。了解此順序有助於您在自訂 時避免意外行為maxPods。

優先順序 （ 從高到低 ）：

1. 受管節點群組強制執行 – 當您使用沒有自訂 AMI 的受管節點群組時，Amazon EKS 會在節點的使用者資料maxPods中強制執行 上的上限。對於少於 30 個 vCPUs執行個體，上限為 110。對於 vCPUs執行個體，上限為 250。此值優先於任何其他maxPods組態，包括 maxPodsExpression。
2. kubelet **maxPods**組態 – 如果您maxPods直接在 kubelet 組態中設定 （ 例如，透過具有自訂 AMI 的啟動範本 ），則此值優先於 maxPodsExpression。
3. nodeadm **maxPodsExpression** – 如果您在 maxPodsExpression中使用 NodeConfig，nodeadm 會評估表達式來計算 maxPods。這只有在較高優先順序的來源尚未設定值時才會有效。
4. 預設 ENI 型計算 – 如果未設定其他值，AMI maxPods會根據執行個體類型支援的彈性網路介面和 IP 地址數目來計算。這相當於公式 (number of ENIs × (IPs per ENI − 1)) + 2。每個節點kube-proxy上執行的 Amazon VPC CNI + 2帳戶，不會使用 Pod IP 地址。

> **Important**
>
> 如果您使用受管節點群組並在 maxPodsExpression中設定 NodeConfig，受管節點群組的強制執行會覆寫您的表達式。若要搭配受管節點群組使用自訂maxPods值，您必須在啟動範本中指定自訂 AMI 並maxPods直接設定。如需詳細資訊，請參閱the section called “啟動範本”。

受管節點群組與自我管理節點

使用受管節點群組 （沒有自訂 AMI)，Amazon EKS 會將maxPods值注入節點的引導使用者資料。這表示：

- maxPods 值一律為上限110或250取決於執行個體大小。
- maxPodsExpression 您設定的任何 都會被此注入值覆寫。
- 若要使用不同的maxPods值，請在啟動範本中指定自訂 AMI，然後與 --use-max-pods false一起傳遞--kubelet-extra-args '--max-pods=*my-value*'至bootstrap.sh指令碼。如需範例，請參閱 the section called “啟動範本”。

透過自我管理節點，您可以完全控制引導程序。您可以在 maxPodsExpression中使用 NodeConfig或--max-pods直接傳遞至 bootstrap.sh。

EKS 自動模式的考量事項

EKS 自動模式會將節點上的 Pod 數量限制為以下兩者中較低的值：

- 110 個 Pod 的硬性上限
- number of ENIs × (IPs per ENI - 1)

使用預先建置的最佳化映像建立節點

當您使用受管節點群組或自我管理節點時，您可以使用預先建置的 Amazon EKS 最佳化 Amazon Machine Image (AMI) 或您的自訂 AMI 來部署節點。若您正在執行混合節點，請參閱 the section called “準備作業系統”。如需有關各種 Amazon EKS 最佳化 AMI 類型的詳細資訊，請參閱下列其中一個主題。如需如何建立自己的自訂 AMI 的詳細資訊，請參閱 the section called “建置自訂 EKS 最佳化 Amazon Linux AMI”。

使用 Amazon EKS 自動模式時，EKS 會管理 EC2 執行個體，包括選取和更新 AMI。

主題

- EKS AL2 和 AL2 加速 AMI 移轉功能指南
- 使用最佳化的 Amazon Linux AMI 建立節點
- 使用最佳化的 Bottlerocket AMI 建立節點
- 使用最佳化的 Ubuntu Linux AMI 建立節點
- 使用最佳化的 Windows AMI 建立節點

EKS AL2 和 AL2 加速 AMI 移轉功能指南

> **Warning**
>
> Amazon EKS 已於 2025 年 11 月 26 日停止發佈 EKS 最佳化的 Amazon Linux 2 (AL2) AMIs。適用於 Amazon EKS 的 AL2023 和 Bottlerocket 型 AMIs 適用於所有支援的 Kubernetes 版本，包括 1.33 和更新版本。

AWS 將於 20AL2-optimized和 AL2-accelerated AMIs 的支援。雖然您可以在終止支援 (EOS) 日期 (2025 年 11 月 26 日) 之後繼續使用 EKS AL2 AMI，但 EKS 將不再發布任何新的 Kubernetes 版本或對 AL2 AMI 的更新，包括此日期之後的次要版本、修補程式和錯誤修正。我們建議升級至 Amazon Linux 2023 (AL2023) 或 Bottlerocket AMI：

- AL2023 採用預設安全的方式，具備預配置的安全政策、處於容許模式的 SELinux、預設啟用的僅 IMDSv2 模式、最佳化的開機時間以及增強的套件管理功能，可提升安全性與效能，適用於需要大量自訂 (如直接存取作業系統層或大量節點變更) 的基礎架構。若要進一步了解，請參閱 AL2023 FAQs或檢視我們在 的詳細遷移指引the section called “升級至 AL2023”。
- Bottlerocket 憑藉其專為容器最佳化的設計，實現了增強的安全性、更快的啟動時間和更小的受攻擊面，以提高效率，非常適合具有最少節點自訂的容器原生方法。若要進一步了解，請參閱 Bottlerocket FAQs或檢視詳細的遷移指南，網址為 the section called “Bottlerocket”。

或者，您可以在 EOS 日期the section called “建置自訂 EKS 最佳化 Amazon Linux AMI”之前 (2025 年 11 月 26 日) 。此外，您可以使用 Amazon Linux 2 基本執行個體建置自訂 AMI，直到 Amazon Linux 2 EOS 日期 (2026 年 6 月 30 日) 。

移轉和支援常見問題

如何從 AL2 移轉至 AL2023 AMI？

我們建議制定並實施一個移轉計劃，該計劃包括全面的應用程式工作負載測試和文件化的復原程序，然後按照 EKS 官方文件中從 Amazon Linux 2 升級到 Amazon Linux 2023 的逐步說明進行操作。

我是否可以在 EKS 最佳化 AL2 AMI 的 EKS 終止支援日期 (EOS) 之後建置自訂 AL2 AMI？

雖然我們建議改用官方支援且已發布的 EKS 最佳化 AMI (適用於 AL2023 或 Bottlerocket)，但在 AL2 AMI EOS 日期 (2025 年 11 月 26 日) 之前，您仍可建置自訂 EKS AL2 最佳化與 AL2 加速型 AMI。或者，您可在 Amazon Linux 2 EOS 日期 (2026 年 6 月 30 日) 之前，以 Amazon Linux 2 基礎執行個體

建置自訂 AMI。如需建置自訂 EKS AL2 最佳化與 AL2 加速型 AMI 的逐步指示，請參閱 EKS 官方文件中的建置自訂 Amazon Linux AMI。

EKS Kubernetes 版本支援政策是否適用於 Amazon Linux 發行版本？

不適用。EKS AL2 最佳化和 AL2 加速 AMI 的 EOS 日期獨立於 EKS 對 Kubernetes 版本的標準和延伸支援時間表。即使您使用 EKS 延伸支援，也需要移轉至 AL2023 或 Bottlerocket。

從 cgroupv1 轉向 cgroupv2 對我的移轉有何影響？

Kubernetes 社群已將 `cgroupv1` 支援 (AL2 使用) 置於維護模式，這意味著不會新增新功能，並且僅提供關鍵安全性和重大錯誤修正。要在 Kubernetes 中採用 `cgroupv2`，您需要確保作業系統、核心、容器執行時期和 Kubernetes 元件之間的相容性。這需要預設啟用 `cgroupv2` 的 Linux 發行版本，例如 AL2023、Bottlerocket、Red Hat Enterprise Linux (RHEL) 9+、Ubuntu 22.04+ 或 Debian 11+。這些發行版隨附的核心版本大於等於 5.8，這是 Kubernetes 中 `cgroupv2` 支援的最低要求。要進一步了解，請參閱關於 cgroup v2。

如果我在自訂 AL2 AMI 中需要 Neuron，該怎麼辦？

您無法在基於 AL2 的 AMI 上原生執行完整的 Neuron 驅動應用程式。若要在 AL2 AMI 上利用 AWS Neuron，您必須使用 Neuron 支援的容器與non-AL2 Linux 發行版本 （ 例如 Ubuntu 22.04、Amazon Linux 2023 等 ） 來容器化應用程式，然後在已安裝 Neuron 驅動程式 (`aws-neuronx-dkms`) 的 AL2-based AMI 上部署這些容器。

在 EKS AL2 AMI EOS 日期 (2025 年 11 月 26 日) 之後，是否應切換至裸機 Amazon Linux 2 基礎執行個體？

切換到裸機 Amazon Linux 2 基礎執行個體缺乏官方 EKS AL2 最佳化和 AL2 加速 AMI 提供的特定最佳化、容器執行時期組態和自訂。相反，如果您必須繼續使用基於 AL2 的解決方案，我們建議使用位於 the section called “建置自訂 EKS 最佳化 Amazon Linux AMI” 或 Amazon EKS AMI 建置規格的 EKS AMI 配方，建置自訂 AMI。這確保了與現有工作負載的相容性，並包括 AL2 核心更新，直到 Amazon Linux 2 EOS 日期 (2026 年 6 月 30 日)。

在 EKS AL2 AMI EOS 日期 (2025 年 11 月 26 日) 後，使用 EKS AMI GitHub 儲存庫建置自訂 AL2 AMI 時，哪些支援來自 amzn2-core 和 amzn2extra-docker 等儲存庫的套件？

Amazon EKS AMI 建置規格的 EKS AMI 配方會透過 YUM 從標準 Amazon Linux 2 軟體提取套件，例如 amzn2-core 和 amzn2extra-docker。在 EKS AL2 AMI EOS 日期 (2025 年 11 月 26 日) 之後，這些軟體仍會持續受到支援，直至 Amazon Linux 2 整體 EOS 日期 (2026 年 6 月 30 日)。請注意，此期間

的支援僅限於核心更新，這意味著您需手動管理並套用其他套件更新、安全性修補程式與非核心相依項目，以維持安全性與相容性。

為何在 Amazon EKS 搭配 AL2023 時，使用舊版 JDK8 的 Java 應用程式可能發生記憶體不足 (OOM) 例外與 Pod 重啟？該如何解決？

在搭載 AL2023 的 Amazon EKS 節點上執行時，依賴低於 `jdk8u372` 版本 JDK 8 的 Java 應用程式，可能因 JVM 與 `cgroupv2` 不相容而發生 OOM 例外狀況與 Pod 重啟。此問題的核心原因是，JVM 無法透過 Amazon Linux 2023 預設的 `cgroupv2` 偵測容器記憶體限制。因此，其會依據節點總記憶體而非 Pod 定義的限制來配置堆積記憶體。這是因為 `cgroupv2` 變更了記憶體限制資料的儲存位置，導致舊版 Java 誤讀可用記憶體，並預設使用節點層級資源。可行的解決方案如下：

- 升級 JDK 版本：升級至 `jdk8u372` 或更新版本，或升級至完全 `cgroupv2` 支援的新版 JDK，可解決此問題。如需完全支援 `cgroupv2` 的相容 Java 版本清單，請參閱關於 cgroup v2。
- 建置自訂 AMI：若必須繼續使用基於 AL2 的解決方案，可在 2025 年 11 月 26 日前，透過 the section called “建置自訂 EKS 最佳化 Amazon Linux AMI” 或 Amazon EKS AMI 建置規格建置自訂 AL2 基底 AMI。例如，可在 2025 年 11 月 26 日前建置基於 AL2 的 v1.33 AMI。Amazon EKS 會在 EKS AL2 EOS 日期 (2025 年 11 月 26 日) 前持續提供基於 AL2 的 AMI。在 EOS 日期 (2025 年 11 月 26 日) 之後，您將需要建置自己的 AMI。
- 啟用 cgroupv1：如果您必須繼續使用 `cgroupv1`，可在 EKS AL2023 AMI `cgroupv1` 上啟用。若要啟用，請執行 `sudo grubby --update-kernel=ALL --args="systemd.unified_cgroup_hierarchy=0"` 並重新啟動系統 (例如執行 Amazon Linux 2023 的 EC2 執行個體或節點)。此操作會修改系統開機參數 (例如，在 GRUB 組態中新增核心參數 'systemd.unified_cgroup_hierarchy=0'，指示 systemd 使用舊版 `cgroupv1` 階層)，進而啟用 `cgroupv1`。請注意，當您執行此 grubby 命令時，核心會重新配置為啟用 `cgroupv1` 並停用 `cgroupv2`。節點上的作用中資源管理僅會使用其中一個 cgroup 版本。這與執行 `cgroupv2` 並對 `cgroupv1` API 提供回溯相容性並非同一概念。

> **Warning**
>
> 不建議繼續使用 `cgroupv1`。相反地，建議移轉至 `cgroupv2`。Kubernetes 社群已將 `cgroupv1` 支援 (AL2 使用) 置於維護模式，這意味著不會新增新功能或更新，並且僅提供關鍵安全性和重大錯誤修正。預期未來版本會完全移除 `cgroupv1` 支援，但目前尚未公佈具體移除日期。如果您遇到 的問題`cgroupv1`， AWS 將無法提供支援，並建議您升級至 `cgroupv2`。

相容性和版本

AL2 AMI 支援的 Kubernetes 版本

Kubernetes 1.32 版是 Amazon EKS 將發行 AL2 (Amazon Linux 2) AMI 的最後一個版本。對於支援的、最高達 1.32 的 Kubernetes 版本，EKS 會在 2025 年 11 月 26 日前持續發布 AL2 AMI (AL2_ARM_64、AL2_x86_64) 與 AL2 加速 AMI (AL2_x86_64_GPU)。在此日期後，EKS 將停止為所有 Kubernetes 版本發布 AL2 最佳化與 AL2 加速型 AMI。請注意，EKS AL2 最佳化和 AL2 加速 AMI 的 EOS 日期獨立於 EKS 對 Kubernetes 版本的標準和延伸支援時間表。

AL2、AL2023 與 Bottlerocket AMI 的支援驅動程式與 Linux 核心版本比較

元件	EKS AL2 AMI	EKS AL2023 AMI	EKS Bottlerocket AMI
基本作業系統相容性	RHEL7/CentOS 7	Fedora/CentOS 9	N/A
CUDA 使用者模式驅動程式	12.x	12.x、13.x	12.x、13.x
NVIDIA GPU 驅動程式	R570	R580	R570、R580
AWS Neuron 驅動程式	2.20+	2.20+	2.20+
Linux 核心	5.10	6.1、6.12	6.1、6.12

如需 NVIDIA 驅動程式和 CUDA 相容性的詳細資訊，請參閱 NVIDIA 文件。

AWS Neuron 與 AL2 AMIs相容性

從 AWS Neuron 2.20 版開始，EKS 基於 AL 的 AMI 使用的 Neuron 執行時期 (`aws-neuronx-runtime-lib`) 不再支援 Amazon Linux 2 (AL2)。Neuron 驅動程式 (`aws-neuronx-dkms`) 現在是唯一支援 Amazon Linux 2 的 AWS Neuron 套件。這意味著您無法在基於 AL2 AMI 上原生執行 Neuron 驅動型應用程式。若要在 AL2023 AMI 上設定 Neuron，請參閱 AWS Neuron 設定指南。

Kubernetes 與 AL2 AMI 的相容性

Kubernetes 社群已將 `cgroupv1` 支援 (AL2 所使用) 移至維護模式。這意味著不會新增任何功能，僅提供重大安全性與主要錯誤修復。任何依賴 cgroupv2 的 Kubernetes 功能 (如 MemoryQoS 與增強型

資源隔離) 均無法在 AL2 上使用。此外，Amazon EKS Kubernetes 1.32 版本是最後一個支援 AL2 AMI 的版本。為了維持與最新 Kubernetes 版本的相容性，建議移轉至 AL2023 或 Bottlerocket，這兩者均預設啟用 `cgroupv2`。

AL2 AMI 與 Linux 版本的相容性

在 2026 年 6 月 30 日end-of-support(EOS) 日期之前， 都支援 Amazon Linux 2 (AL2)。 AWS 但隨著 AL2 逐步老化，廣大 Linux 社群對新應用程式與功能的支援已逐漸受限。AL2 AMI 基於 Linux 核心 5.10，而 AL2023 則使用 Linux 核心 6.1。不同於 AL2023，AL2 獲得的廣大 Linux 社群支援有限。這意味著許多上游 Linux 套件與工具需經過回溯移植才能在 AL2 的舊版核心上運作；部分現代 Linux 功能與安全性改進因核心版本過舊而無法使用；許多開源專案已終止或限制對 5.10 等舊版核心的支援。

AL2023 中未包含的已棄用套件

AL2023 中未包含或已變更的常見套件包括：

- Amazon Linux 2 中的部分來源二進制套件在 Amazon Linux 2023 中不再提供
- Amazon Linux 在 AL2023 中支援不同套件版本的方式發生變更 (例如 amazon-linux-extras 系統)
- AL2023 不支援 Extra Packages for Enterprise Linux (EPEL)
- AL2023 中不支援 32 位元應用程式

要進一步了解，請參閱比較 AL2 與 AL2023。

跨 AL2、AL2023 和 Bottlerocket 的 FIPS 驗證比較

Amazon Linux 2 (AL2)、Amazon Linux 2023 (AL2023) 與 Bottlerocket 均支援聯邦資訊處理標準 (FIPS) 合規性。

- AL2 已通過 FIPS 140-2 認證，AL2023 已通過 FIPS 140-3 認證。若要在 AL2023 上啟用 FIPS 模式，需在 Amazon EC2 執行個體上安裝必要套件，並遵循在 AL2023 上啟用 FIPS 模式中的指示執行組態步驟。要進一步了解，請參閱 AL2023 常見問答集。
- Bottlerocket 提供專為 FIPS 設計的變體，這類變體會限制核心與使用者空間元件，僅使用已提交至 FIPS 140-3 密碼編譯模組驗證計劃的密碼編譯模組。

EKS AMI 驅動程式與版本變更日誌

關於所有 EKS AMI 元件及其版本的完整清單，請參閱 GitHub 上的 Amazon EKS AMI 版本備註。

使用最佳化的 Amazon Linux AMI 建立節點

Amazon Elastic Kubernetes Service (Amazon EKS) 提供專門的 Amazon Machine Image (AMIs)，針對執行 Kubernetes 工作者節點進行最佳化。這些 EKS 最佳化 Amazon Linux (AL) AMIs 已預先設定基本元件，例如 `kubelet`、AWS IAM Authenticator 和 `containerd`，以確保叢集內的無縫整合和安全性。本指南詳細說明可用的 AMI 版本，並概述加速運算和 Arm 型架構的特殊選項。

考量事項

- 您可以透過選擇所需版本的標籤，在 Amazon Linux 安全中心追蹤 Amazon Linux 的安全或隱私權事件。您也可以訂閱適用的 RSS 摘要。安全與隱私權事件包含問題的概觀、哪些套件受到影響，以及如何更新您的執行個體以修正問題。
- 部署加速或 Arm AMI 之前，請先檢閱 Amazon EKS 最佳化加速 Amazon Linux AMIs 和 中的資訊the section called “Amazon EKS 最佳化 Arm Amazon Linux AMIs”。
- Amazon EC2 P2 執行個體在 Amazon EKS 上不受支援, 因為它們需要 `NVIDIA` 驅動程式版本 470 或更早版本。
- 在版本 `1.30` 或更新版本的叢集中，任何新建立的受管節點群組將自動預設使用 AL2023 作為節點作業系統。

Amazon EKS 最佳化加速 Amazon Linux AMIs

Amazon EKS 最佳化加速 Amazon Linux (AL) AMIs 是以標準 EKS 最佳化 Amazon Linux AMIs 為基礎建置。它們已設定為 Amazon EKS 節點的選用映像，用以支援以 GPU、Inferentia 和 Trainium 為基礎的工作負載。

如需詳細資訊，請參閱the section called “使用 EKS Linux GPU AMIs”。

Amazon EKS 最佳化 Arm Amazon Linux AMIs

Arm 執行個體可為擴增和 Arm 型應用程式節省大量的成本，例如 Web 伺服器、容器化微型服務、快取機群和分散式資料存放區。將 Arm 節點新增至叢集時，請檢閱下列考量事項。

- 如果叢集是在 2020 年 8 月 17 日之前部署，您必須對重要的叢集附加元件資訊清單進行一次性升級。如此一來，Kubernetes 就可以為叢集中使用中的每個硬體架構提取正確的映像。如需更新叢集附加元件的詳細資訊，請參閱 the section called “步驟 1：準備升級”。如果您在 2020 年 8 月 17 當天或之後部署叢集，則 CoreDNS、`kube-proxy` 和 Kubernetes 專用 Amazon VPC CNI 外掛程式已經具有多架構能力。

- 部署至 Arm 節點的應用程式必須針對 Arm 進行編譯。
- 如果您在現有叢集中部署了 DaemonSet，或者您想將它們部署到您也想在其中部署 Arm 節點的新叢集，請確認您的 DaemonSet 是否可以在您叢集中的所有硬體架構上執行。
- 您可以在相同的叢集中執行 Arm 節點群組和 x86 節點群組。如果您這麼做，請考慮將多架構容器映像部署到容器儲存庫 (例如 Amazon Elastic Container Registry)，然後將節點選取器新增至資訊清單，以便 Kubernetes 知道 Pod 可以部署到哪些硬體架構。如需詳細資訊，請參閱《Amazon ECR 使用者指南》中的推送多架構映像和適用於 Amazon ECR 的多架構容器映像簡介部落格一文。

其他資訊

如需使用 Amazon EKS 最佳化 Amazon Linux AMIs 的詳細資訊，請參閱下列章節：

- 若要將 Amazon Linux 與受管節點群組搭配使用，請參閱 the section called “受管節點群組”。
- 若要啟動自我管理的 Amazon Linux 節點，請參閱 the section called “獲取最新 ID”。
- 如需版本資訊，請參閱the section called “獲取版本資訊”。
- 若要擷取 Amazon EKS 最佳化 Amazon Linux AMIs 的最新 IDs，請參閱 the section called “獲取最新 ID”。
- 如需用於建置 Amazon EKS 最佳化 AMIs開放原始碼指令碼，請參閱 the section called “建置自訂 EKS 最佳化 Amazon Linux AMI”。

從 Amazon Linux 2 升級到 Amazon Linux 2023

> **Warning**
>
> Amazon EKS 已於 2025 年 11 月 26 日停止發佈 EKS 最佳化的 Amazon Linux 2 (AL2) AMIs。適用於 Amazon EKS 的 AL2023 和 Bottlerocket 型 AMIs 適用於所有支援的 Kubernetes 版本，包括 1.33 和更新版本。

AL2023 是以 Linux 為基礎的作業系統，旨在為您的雲端應用程式提供安全、穩定且高效能的環境。它是 Amazon Web Services 推出的下一代 Amazon Linux，適用於所有受支援的 Amazon EKS 版本。

相較於 AL2，AL2023 提供了多項改進。有關完整的比較，請參閱《Amazon Linux 2023 使用者指南》中的比較 AL2 和 Amazon Linux 2023。已從 AL2 新增、升級和移除數個套件。強烈建議在升級前，使用 AL2023 測試您的應用程式。有關 AL2023 中所有套件變更的清單，請參閱《Amazon Linux 2023 版本備註》中的 Amazon Linux 2023 中的套件變更。

除了上述變更，您還應注意以下事項：

- AL2023 引入了新的節點初始化程序 nodeadm，該程序使用 YAML 組態結構描述。如果您使用自我管理節點群組或指定自訂 AMI ID 的啟動範本，您現在需要明確提供額外的叢集中繼資料。建立新節點群組時，您必須提供此中繼資料。nodeadm 最低必要參數範例如下所示，其中cidr現在需要certificateAuthority、 apiServerEndpoint和服務：

  ```
  ---
  apiVersion: node.eks.aws/v1alpha1
  kind: NodeConfig
  spec:
    cluster:
      name: my-cluster
      apiServerEndpoint: https://example.com
      certificateAuthority: Y2VydGlmaWNhdGVBdXRob3JpdHk=
      cidr: 10.100.0.0/16
  ```

  在 AL2 中，這些參數的中繼資料是透過 Amazon EKS DescribeCluster API 呼叫自動發現的。在 AL2023 中，此行為已改變，因為額外的 API 呼叫在大規模節點擴展時有被限流的風險。如果您使用沒有啟動範本的受管節點群組，或使用 Karpenter，則此變更不會對您造成影響。如需有關 certificateAuthority 和服務 cidr 的更多資訊，請參閱《Amazon EKS API 參考》中的 DescribeCluster。

- 對於 AL2023，nodeadm 也更改了使用 NodeConfigSpec 將參數應用到每個節點 kubelet 的格式。在 AL2 中，這是透過 --kubelet-extra-args 參數完成的。這通常用於為節點新增標籤和污點。下面的範例展示了如何將 maxPods 和 --node-labels 套用至節點。

  ```
  ---
  apiVersion: node.eks.aws/v1alpha1
  kind: NodeConfig
  spec:
    cluster:
      name: test-cluster
      apiServerEndpoint: https://example.com
      certificateAuthority: Y2VydGlmaWNhdGVBdXRob3JpdHk=
      cidr: 10.100.0.0/16
    kubelet:
      config:
        maxPods: 110
      flags:
        - --node-labels=karpenter.sh/capacity-type=on-demand,karpenter.sh/nodepool=test
  ```

- AL2023 需要 Amazon VPC CNI 版本 1.16.2 或以上。
- AL2023 預設要求使用 IMDSv2。IMDSv2 有多項有助於提升安全狀態的優點。它使用面向工作階段的身分驗證方法，需要透過一個簡單的 HTTP PUT 請求來建立一個秘密權杖以啟動工作階段。工作階段的權杖有效期可設定為 1 秒到 6 小時之間的任何時間。如需如何從 轉換IMDSv1至 的詳細資訊IMDSv2，請參閱使用執行個體中繼資料服務第 2 版的轉換和取得 IMDSv2 的完整優點，以及在整個 AWS 基礎設施中停用 IMDSv1。若您仍想使用 IMDSv1，可以透過使用執行個體中繼資料選項啟動屬性，手動覆寫設定來實現。

  > **Note**
  >
  > 對於 AL2023 的 IMDSv2，受管節點群組的預設跳數可能有所不同：
  >
  > - 當不使用啟動範本時，預設值設為 1。這意味著容器將無法使用 IMDS 存取節點的憑證。若您的容器需要存取節點的憑證，您仍可透過使用自訂的 Amazon EC2 啟動範本來實現。
  > - 當在啟動範本中使用自訂 AMI 時，預設的 HttpPutResponseHopLimit 設為 2。您可以在啟動範本中手動覆寫 HttpPutResponseHopLimit。
  >
  > 或者，您可使用 Amazon EKS Pod 身分識別來提供憑證，而非使用 IMDSv2。

- AL2023 採用了下一代統一控制組層次結構 (cgroupv2)。cgroupv2 用於實現容器執行時期，並由 systemd 執行。雖然 AL2023 仍然包含可以讓系統使用 cgroupv1 運行的程式碼，但這不是建議建議組態或受支援組態。此組態將在未來的 Amazon Linux 主要版本中完全移除。
- eksctl 需要 eksctl 版本 0.176.0 或以上才能支援 AL2023。

對於既有的受管節點群組，您可根據使用啟動範本的方式執行就地升級或藍綠升級：

- 若您對受管節點群組使用自訂 AMI，可透過交換啟動範本中的 AMI ID 來執行就地升級。在執行此升級策略前，您應確保應用程式和任何使用者資料都能移轉到 AL2023。
- 如果您使用帶有標準啟動範本或未指定 AMI ID 的自訂啟動範本的受管節點群組，則需要使用藍綠策略進行升級。藍綠升級通常更複雜，涉及建立一個全新的節點群組，並在其中指定 AL2023 作為 AMI 類型。然後需要仔細設定新的節點群組，以確保來自 AL2 節點群組的所有自訂資料都與新的作業系統相容。一旦新的節點群組經過您的應用程式測試和驗證，Pod 可以從舊節點群組移轉到新節點群組。移轉完成後，您可刪除舊的節點群組。

若您使用 Karpenter 並希望使用 AL2023，需要修改 EC2NodeClass amiFamily 欄位為 AL2023。預設情況下，Karpenter 啟用了偏離。這意味著一旦 amiFamily 欄位被更改，Karpenter 將在最新 AMI 可用時自動更新您的工作節點。

有關 nodeadm 的其他資訊

使用 EKS 最佳化 Amazon Linux 2023 AMI 或透過官方 amazon-eks-ami GitHub 儲存庫中提供的 Packer 指令碼建置自訂 EKS Amazon Linux 2023 AMI 時，您應該避免在 EC2 使用者資料內或作為自訂 AMI 的一部分明確執行 nodeadm init。

如果您想要在使用者資料中產生動態 NodeConfig，您可以將該組態寫入 中的插入式 yaml 或 json 檔案`/etc/eks/nodeadm.d`。當 Nodeadm init 稍後在開機程序中自動啟動時，這些組態檔案將會合併並套用至您的節點。例如：

```
cat > /etc/eks/nodeadm.d/additional-node-labels.yaml << EOF
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  kubelet:
    flags:
      - --node-labels=foo=bar
EOF
```

EKS 最佳化的 Amazon Linux 2023 AMIs 會透過單獨的系統化服務，以兩個階段自動執行 nodeadm init：Nodeadm-config 會在使用者資料執行之前執行，而 nodeadm-run 會在之後執行。nodeadm-config 服務會在使用者資料執行之前建立容器化和 kubelet 的基準組態。nodeadm-run 服務會執行選取系統協助程式，並在使用者資料執行後完成任何最終組態。如果透過使用者資料或自訂 AMI 再執行一次 nodeadm init 命令，可能會破壞有關執行排序的假設，導致非預期的結果，包括設定ENIs。

擷取 Amazon Linux AMI 版本資訊

Amazon EKS 最佳化 Amazon Linux AMI 是由 Kubernetes 版本和 AMI 發行日期進行版本化，格式如下：

```
k8s_major_version.k8s_minor_version.k8s_patch_version-release_date
```

每個 AMI 發行版本均包含 kubelet、Linux 核心和 containerd 的各種版本。加速的 AMI 亦包括 NVIDIA 驅動程序的各種版本。您可以在 GitHub 的版本中找到此版本資訊。

擷取建議的 Amazon Linux AMI ID

您在部署節點時，可針對預先建置的 Amazon EKS 最佳化 Amazon Machine Image (AMI) 指定 ID。若要擷取符合您所需組態的 AMI ID，請查詢 AWS Systems Manager 參數存放區 API。使用此 API 無

需手動查詢 Amazon EKS 最佳化的 AMI ID。如需詳細資訊，請參閱 GetParameter。您使用的 IAM 主體必須擁有 `ssm:GetParameter` IAM 許可，才能擷取 Amazon EKS 最佳化 AMI 中繼資料。

您可透過以下命令擷取最新建議的 Amazon EKS 最佳化 Amazon Linux AMI 的映像檔 ID，這會使用子參數 `image_id`。視需要對命令進行下列修改，然後執行修改後的命令：

- 使用 Amazon EKS 支援的版本來取代 `<kubernetes-version>`。
- 使用以下其中一個選項來取代 *ami-type*。若要了解 Amazon EC2 執行個體類型的相關詳細資訊，請參閱 Amazon EC2 執行個體類型。
  - 若是 Amazon Linux 2023 (AL2023) x86 型執行個體，則使用 *amazon-linux-2023/x86_64/standard*。
  - 若是 AL2023 ARM 執行個體，例如 AWS Graviton 型執行個體，則使用 *amazon-linux-2023/arm64/standard*。
  - 若是最近核准的 AL2023 NVIDIA x86 型執行個體，則使用 *amazon-linux-2023/x86_64/nvidia*。
  - 若是最近核准的 AL2023 NVIDIA `arm64` 型執行個體，則使用 *amazon-linux-2023/arm64/nvidia*。
  - 若是最近的 AL2023 AWS Neuron 執行個體，則使用 *amazon-linux-2023/x86_64/neuron*。
- `<region-code>` 將 取代為您想要 AMI ID 的 Amazon EKS 支援 AWS 區域。

```
aws ssm get-parameter --name /aws/service/eks/optimized-ami/<kubernetes-version>/<ami-type>/recommended/image_id \
    --region <region-code> --query "Parameter.Value" --output text
```

下面介紹了取代預留位置之後的命令範例。

```
aws ssm get-parameter --name /aws/service/eks/optimized-ami/1.31/amazon-linux-2023/x86_64/standard/recommended/image_id \
    --region us-west-2 --query "Parameter.Value" --output text
```

範例輸出如下。

```
ami-1234567890abcdef0
```

建置自訂 EKS 最佳化 Amazon Linux AMI

> **Warning**
>
> Amazon EKS 已於 2025 年 11 月 26 日停止發佈 EKS 最佳化的 Amazon Linux 2 (AL2) AMIs。適用於 Amazon EKS 的 AL2023 和 Bottlerocket 型 AMIs 適用於所有支援的 Kubernetes 版本，包括 1.33 和更新版本。

Amazon EKS 在 Amazon EKS AMI Build Specification 儲存庫中提供開放原始碼建置指令碼，可用來檢視 的組態`kubelet`、執行時間、適用於 Kubernetes 的 AWS IAM Authenticator，以及從頭開始建置您自己的 AL 型 AMI。

此儲存庫包含適用於 AL2 的專用引導指令碼，以及適用於 AL2023 的 nodeadm 工具，其會在開機時間執行。這些指令碼會設定您的執行個體的憑證資料、控制平面端點、叢集名稱等。指令碼被視為 Amazon EKS 最佳化 AMI 建置的事實來源，因此您可以遵循 GitHub 儲存庫來監控 AMIs的變更。

使用 EKS 最佳化 AMIs 作為基礎建置自訂 AMIs 時，不建議或支援執行作業系統升級 （即 `dnf upgrade`) 或升級 EKS 最佳化 AMIs 中包含的任何 Kubernetes 或 GPU 套件，因為這會破壞元件相容性。如果您確實升級 EKS 最佳化 AMIs 中包含的作業系統或套件，建議您在部署到生產環境之前，先在開發或預備環境中徹底測試。

為 GPU 執行個體建置自訂 AMIs 時，建議您為要執行的每個執行個體類型產生和系列分別建置自訂 AMIs。EKS 最佳化加速 AMIs 會根據基礎執行個體類型產生和系列，在執行時間選擇性地安裝驅動程式和套件。如需詳細資訊，請參閱安裝和執行時間的 EKS AMI 指令碼。

先決條件

- 安裝 AWS CLI
- 安裝 HashiCorp Packer 1.9.4 版及以上
- 安裝 GNU Make

安裝所需 Amazon 外掛程式。

```
packer plugins install github.com/hashicorp/amazon
```

確認 Packer 已安裝。

```
packer --version
```

複製或分支官方 Amazon EKS AMI 儲存庫。

```
git clone https://github.com/awslabs/amazon-eks-ami.git
cd amazon-eks-ami
```

若要檢視預設值和其他選項，請執行下列命令。要進一步了解可用於自訂 AMI 的組態，請參閱 Amazon Linux 2023 頁面上的範本變數。

```
make help
```

建立自訂 AMI

以下是各種自訂 AMI 的範例命令。

EKS NVIDIA AL2 AMI

```
make k8s=1.32 os_distro=al2 \
  enable_accelerator=nvidia \
  nvidia_driver_major_version=580 \
  enable_efa=true
```

EKS NVIDIA AL2023 AMI

```
make k8s=1.36 os_distro=al2023 \
  enable_accelerator=nvidia \
  nvidia_driver_major_version=580 \
  enable_efa=true
```

EKS NVIDIA AL2023 AMI 搭配 NVIDIA 驅動程式 595 版

> **Important**
>
> G7 EC2 執行個體類型需要 NVIDIA 驅動程式 595 版或更新版本。EKS 最佳化加速 AMIs 目前包含不支援 G7 執行個體的 NVIDIA 驅動程式 580 版。您必須使用 NVIDIA 驅動程式 595 版建置自訂 AMI，才能將 G7 執行個體與 Amazon EKS 搭配使用。

如果您使用 Karpenter 進行節點佈建和自動擴展，建議您從使用自動 AMI 選擇的 NodePools 中排除g7執行個體系列。如需如何設定 Karpenter NodeClass 以使用g7執行個體的自訂 AMI，請參閱 Karpenter 文件amiSelectorTerms中的 。

```
make k8s=1.36 os_distro=al2023 \
  enable_accelerator=nvidia \
  nvidia_driver_major_version=595 \
  enable_efa=true
```

STIG 合規 Neuron AL2023 AMI：

```
make k8s=1.31 os_distro=al2023 \
  enable_accelerator=neuron \
  enable_fips=true \
  source_ami_id=ami-0abcd1234efgh5678 \
  kms_key_id=alias/aws-stig
```

在您執行這些命令之後，Packer 會執行下列動作：

- 啟動暫時 Amazon EC2 執行個體。
- 安裝 Kubernetes 元件、驅動程式和組態。
- 在 AWS 帳戶中建立 AMI。

預期的輸出應類似以下：

```
==> Wait completed after 8 minutes 42 seconds

==> Builds finished. The artifacts of successful builds are:
--> amazon-ebs: AMIs were created:
us-west-2: ami-0e139a4b1a7a9a3e9

--> amazon-ebs: AMIs were created:
us-west-2: ami-0e139a4b1a7a9a3e9

--> amazon-ebs: AMIs were created:
us-west-2: ami-0e139a4b1a7a9a3e9
```

使用最佳化的 Bottlerocket AMI 建立節點

Bottlerocket 是由 贊助和支援的開放原始碼 Linux 發行版本 AWS。Bottlerocket 專為託管容器工作負載而打造。利用 Bottlerocket，您可以自動化容器基礎結構的更新，進而改善容器化部署的可用性並降低營運成本。Bottlerocket 僅包含執行容器的基本軟體，可改善資源用量、減少安全威脅，並降低管理開銷。Bottlerocket AMI 包含 `containerd`、 `kubelet`和 AWS IAM Authenticator。除了受管節點群組和自我管理節點外，Karpenter 亦支援 Bottlerocket。

優點

將 Bottlerocket 與 Amazon EKS 叢集搭配使用具有以下優勢：

- 更長的正常運行時間、更低的營運成本和更低的管理複雜性 – 與其他 Linux 發行版相比，Bottlerocket 具有更低的資源佔用、更短的啟動時間，並且不易受到安全威脅。Bottlerocket 更小的佔用空間使用更少的儲存、運算和網路資源，有助於降低成本。
- 透過自動作業系統更新提高安全性 – 對 Bottlerocket 的更新作為一個單元套用，必要時可以復原。這消除了損毀或失敗更新可能使系統處於無法使用狀態的風險。使用 Bottlerocket，只要安全更新可用，就會以干擾性最小的方式自動套用，並在發生故障時復原。
- 進階支援 – 在 Amazon EC2 上 AWS 提供的 Bottlerocket 組建涵蓋在相同的 AWS 支援計畫中，該計畫也涵蓋 Amazon EC2、Amazon EKS 和 Amazon ECR 等 AWS 服務。

考量事項

將 Bottlerocket 用於 AMI 類型時，請考慮下列事項：

- Bottlerocket 支援具有 `x86_64` 與 `arm64` 處理器的 Amazon EC2 執行個體。
- Bottlerocket 支援具有 GPUs 的 Amazon EC2 執行個體。如需詳細資訊，請參閱the section called “使用 EKS Linux GPU AMIs”。
- Bottlerocket 映像不會包含 SSH 伺服器或 Shell。您可以採用頻外存取方法來允許 SSH。這些方法啟用管理員容器，並傳遞一些引導組態步驟與使用者資料。如需詳細資訊，請參閱 GitHub 上 Bottlerocket OS 中的下列章節：
  - 探勘
  - 管理員容器
  - Kubernetes 設定
- Bottlerocket 使用不同的容器類型：

- 在預設情況下，已啟用控制容器。此容器會執行 AWS Systems Manager 代理程式，可以讓您在 Amazon EC2 Bottlerocket 執行個體上執行命令或啟動 Shell 工作階段。如需詳細資訊，請參閱 Systems Manager 使用者指南中的設定 Session Manager。 AWS
- 如果在建立節點群組時提供 SSH 金鑰，則啟用管理容器。建議只在開發和測試案例中使用管理容器，不要在生產環境中使用此種容器。如需詳細資訊，請參閱 GitHub 上的管理容器。

其他資訊

如需使用 Amazon EKS 最佳化 Bottlerocket AMI 的詳細資訊，請參閱下列區段：

- 如需 Bottlerocket 的更多資訊，請參閱 Bottlerocket 文件。
- 有關版本資訊資源，請參閱 the section called “獲取版本資訊”。
- 若要將 Bottlerocket 與受管節點群組搭配使用，請參閱 the section called “受管節點群組”。
- 若要啟動自我管理的 Bottlerocket 節點，請參閱 the section called “Bottlerocket”。
- 若要擷取 Amazon EKS 最佳化 Bottlerocket AMI 的最新 ID，請參閱 the section called “獲取最新 ID”。
- 如需合規性支援的詳細資訊，請參閱 the section called “合規性支援”。
- 若要使用特定 NVIDIA 驅動程式建置自訂 Bottlerocket 變體，請參閱 the section called “建置自訂 Bottlerocket AMI”。

擷取 Bottlerocket AMI 版本資訊

每個 Bottlerocket AMI 發行版本均包含 kubelet、Bottlerocket 核心和 containerd 的各種版本。加速的 AMI 變體亦包括 NVIDIA 驅動程序的各種版本。您可在 Bottlerocket 文件的作業系統主題中找到此版本資訊。從該頁面，導覽至適用的版本資訊子主題。

Bottlerocket 文件的更新速度有時可能會落後於 GitHub 上提供的版本。您可在 GitHub 的版本中找到最新版本的變更清單。

擷取建議的 Bottlerocket AMI ID

您在部署節點時，可針對預先建置的 Amazon EKS 最佳化 Amazon Machine Image (AMI) 指定 ID。如需要擷取與您的所需組態相符的 AMI ID，請查詢 AWS Systems Manager 參數存放區 API。使用此 API 無需手動查詢 Amazon EKS 最佳化的 AMI ID。如需詳細資訊，請參閱 GetParameter。您使用的 IAM 主體必須擁有 `ssm:GetParameter` IAM 許可，才能擷取 Amazon EKS 最佳化 AMI 中繼資料。

您可以使用下列 AWS CLI 命令，擷取最新推薦的 Amazon EKS 最佳化 Bottlerocket AMI 的映像 ID，這會使用子參數 `image_id`。視需要對命令進行下列修改，然後執行修改後的命令：

- 使用支援的平台版本來取代 *kubernetes-version*。
- 使用以下其中一個選項來取代 *-flavor*。
  - 若是未啟用 GPU 的變體，則移除 *-flavor*。
  - 若是啟用 GPU 的變體，則使用 *-nvidia*。
  - 若是啟用 FIPS 的變體，則使用 *-fips*。
- 使用以下其中一個選項來取代*##*。
  - 若是 x86 型執行個體，則使用 *x86_64*。
  - 若是 ARM 執行個體，則使用 *arm64*。
- 使用您需要取得 AMI ID 的 Amazon EKS 支援的 AWS 區域來取代 *region-code*。

```
aws ssm get-parameter --name /aws/service/bottlerocket/aws-k8s-kubernetes-version-
flavor/architecture/latest/image_id \
    --region region-code --query "Parameter.Value" --output text
```

下面介紹了取代預留位置之後的命令範例。

```
aws ssm get-parameter --name /aws/service/bottlerocket/aws-k8s-1.31/x86_64/latest/
image_id \
    --region us-west-2 --query "Parameter.Value" --output text
```

範例輸出如下。

```
ami-1234567890abcdef0
```

透過 Bottlerocket 滿足合規要求

Bottlerocket 符合各種組織定義的建議：

- 為 Bottlerocket 定義了 CIS 基準。在預設組態中，Bottlerocket 映像具有 CIS 第 1 級組態設定檔所需的大部分控制項。您可以實作 CIS 第 2 級組態設定檔所需的控制項。如需詳細資訊，請參閱 AWS 部落格上的針對 CIS 基準驗證 Amazon EKS 最佳化 Bottlerocket AMI。

- 功能集最佳化和受攻擊面減少表示 Bottlerocket 執行個體只需較少的組態即可滿足 PCI DSS 需求。Bottlerocket 的 CIS 基準是強化指引的極佳資源，可支援您在 PCI DSS 需求 2.2 下對安全組態標準的需求。您也可以根據 PCI DSS 要求 10.2 利用 Fluent Bit 來支援作業系統層級稽核記錄的需求。 會定期 AWS 發佈新的 （修補） Bottlerocket 執行個體，以協助您符合 PCI DSS 要求 6.2 （適用於 v3.2.1) 和要求 6.3.3 （適用於 v4.0)。
- Bottlerocket 是符合 HIPAA 資格的功能，授權與 Amazon EC2 和 Amazon EKS 的受管制工作負載搭配使用。如需詳細資訊，請參閱 HIPAA 資格服務參照。
- 可使用預先設定為利用 FIPS 140-3 驗證加密模組的 Bottlerocket AMI。這包括 Amazon Linux 2023 核心加密 API 密碼編譯模組和 AWS-LC 密碼編譯模組。如需詳細資訊，請參閱the section called “Bottlerocket FIPS AMI”。

使用 Bottlerocket FIPS AMI 讓您的工作節點符合 FIPS 就緒狀態

美國聯邦資訊處理標準 (FIPS) 第 140-3 號公報中，美國和加拿大政府的安全標準具體說明密碼模組的安全要求，以保護敏感資訊。Bottlerocket 透過提供具有 FIPS 核心的 AMI，讓您更容易遵循 FIPS 規範。

這些 AMI 已預先設定為使用 FIPS 140-3 驗證的密碼編譯模組。這包括 Amazon Linux 2023 核心加密 API 密碼編譯模組和 Go 密碼編譯模組。

使用 Bottlerocket FIPS AMI 可讓您的工作節點成為「FIPS 就緒」，但不會自動成為「FIPS 相容」。如需詳細資訊，請參閱聯邦資訊處理標準 (FIPS) 140-3。

考量事項

- 如果您的叢集使用隔離子網路，則 Amazon ECR FIPS 端點可能無法存取。這可能導致節點引導失敗。請確保您的網路組態允許存取必要的 FIPS 端點。如需詳細資訊，請參閱 AWS PrivateLink 指南中的透過資源 VPC 端點存取資源。
- 如果您的叢集搭配 PrivateLink 使用子網路，請確定您已設定 Amazon ECR FIPS PrivateLink 端點。Amazon ECR 透過 PrivateLink 支援 FIPS 端點。如需詳細資訊，請參閱 Amazon ECR 介面 VPC 端點。

使用 Bottlerocket FIPS AMI 建立受管節點群組

Bottlerocket FIPS AMI 提供四種支援工作負載的變體：

- `BOTTLEROCKET_x86_64_FIPS`

- BOTTLEROCKET_ARM_64_FIPS
- BOTTLEROCKET_x86_64_NVIDIA_FIPS
- BOTTLEROCKET_ARM_64_NVIDIA_FIPS

要使用 Bottlerocket FIPS AMI 建立受管節點群組，請在建立程序中選擇適用的 AMI 類型。如需詳細資訊，請參閱the section called “建立”。

有關選擇啟用 FIPS 的變體的更多資訊，請參閱 the section called “獲取最新 ID”。

停用不支援 AWS 區域的 FIPS 端點

Bottlerocket FIPS AMIs 直接在美國支援，包括 AWS GovCloud (US) 區域。對於可使用 AMIs 但無法直接支援的 AWS 區域，您仍然可以透過使用啟動範本建立受管節點群組來使用 AMIs。

Bottlerocket FIPS AMI 在啟動程序期間依賴 Amazon ECR FIPS 端點，而這些端點在美國以外地區通常不可用。若要在沒有 Amazon ECR FIPS 端點可用的 AWS 區域中，將 AMI 用於其 FIPS 核心，請執行下列步驟來停用 FIPS 端點：

1. 建立一個包含以下內容的新組態檔案，或將該內容整合到您現有的組態檔中。

```
[default]
use_fips_endpoint=false
```

1. 將檔案內容編碼為 Base64 格式。
2. 在您的啟動範本的 UserData 中，使用 TOML 格式加入以下編碼字串：

```
[settings.aws]
config = "<your-base64-encoded-string>"
```

對於其他設定，請參閱 GitHub 上 Bottlerocket 的設定描述。

以下是啟動範本中 UserData 的範例：

```
[settings]
motd = "Hello from eksctl!"
[settings.aws]
config = "W2RlZmF1bHRdCnVzZV9maXBzX2VuZHBvaW50PWZhbHNlCg==" # Base64-encoded string.
```

```
[settings.kubernetes]
api-server = "<api-server-endpoint>"
cluster-certificate = "<cluster-certificate-authority>"
cluster-name = "<cluster-name>"
...<other-settings>
```

有關使用使用者資料建立啟動範本的更多資訊，請參閱 the section called “啟動範本”。

為 Amazon EKS 建置自訂 Bottlerocket AMI 變體

當您在 Amazon Elastic Kubernetes Service (Amazon EKS) 上執行 GPU 工作負載時，您可以選擇 Bottlerocket 變體。變體必須符合您的 Kubernetes 版本和加速器類型。Bottlerocket 為常見組態提供經過驗證的變體。不過，您的組織可能因下列任何原因而需要不同的變體：

- 較新的 NVIDIA 驅動程式分支
- 符合法規的固定驅動程式版本
- 用於監控的其他套件
- 安全團隊所需的強化基準

由於 GitHub 網站上的 Bottlerocket 是完全開放原始碼，因此您可以建立符合這些需求的自訂變體。本主題說明如何複製現有的變體，並將 NVIDIA 驅動程式從 R580 分支交換為 R595 (595.71.05 版）。然後，您可以建置映像並將其註冊為私有 AMI。

> **Important**
>
> G7 EC2 執行個體類型需要 NVIDIA 驅動程式 595 版或更新版本。EKS Bottlerocket NVIDIA AMIs 目前包含不支援 G7 執行個體的 NVIDIA 驅動程式 580 版。
> 如需使用 NVIDIA 驅動程式 595 版建置變體的說明，請參閱 GitHub 網站上的 Bottlerocket 儲存庫。本主題會逐步解說從步驟 1 開始的完整程序。

建置系統的運作方式

當您執行 時`cargo make -e BUILDSYS_VARIANT=aws-k8s-1.36-nvidia`，會發生三件事：

- 擷取相依性。Twoliter (Bottlerocket 組建協調器） 會從 讀取`Twoliter.toml`和提取三個開放容器計畫 (OCI) 成品`public.ecr.aws/bottlerocket`：
  - bottlerocket-sdk — 具有完整跨編譯工具鏈 (GCC、Rust、Go、RPM 巨集） 的容器映像。

- bottlerocket-kernel-kit — 適用於核心、核心模組（包括 NVIDIA kmod 套件）和韌體的預先建置 RPMs。
- bottlerocket-core-kit：預先建置的使用者空間 RPMs：kubelet、 containerd、NVIDIA 裝置外掛程式和容器工具組、設定外掛程式和系統服務。

  `Twoliter.toml` 釘選版本並`Twoliter.lock`鎖定其摘要。若要變更其中一項，請執行 `./tools/twoliter/twoliter update`以重新解決。

- 建置 變體。Twoliter 會在 SDK 容器內啟動 Docker 組建。它會編譯變體的設定預設木箱、從套件解析 RPM 相依性樹狀結構，並將所有項目組合成磁碟映像。
- 輸出。Twoliter 會將最終`.img.lz4`檔案寫入 `build/images/`。建置會產生決定性輸出：無論主機為何，相同的`Twoliter.toml`接腳和變體`Cargo.toml`一律會產生相同的映像。

儲存庫配置

下列目錄與變體工作相關：

```
bottlerocket/
### Cargo.toml                       # workspace: lists every variant
### Twoliter.toml                    # pins SDK + kit versions
### Twoliter.lock                    # locked digests for the above
### Licenses.toml                    # you create this (NVIDIA license acknowledgement)
### Infra.toml                       # you create this (AMI publish regions)
#
### variants/
#   ### aws-k8s-1.36-nvidia/         # example variant you'll copy
#   #   ### Cargo.toml               #   package list + kernel params
#   #   ### amispec.toml            #   symlink # ../shared/amispec-split.toml
#   ### shared/                      # shared AMI spec templates
#
### sources/
#   ### Cargo.toml                   # workspace: lists every settings-defaults crate
#   ### shared-defaults/             # the actual defaults (symlink targets)
#   ### settings-defaults/
#       ### aws-k8s-1.36-nvidia/
#           ### Cargo.toml
#           ### defaults.d/          # 30+ symlinks into shared-defaults/
#
### packages/
    ### settings-defaults/
    #   ### settings-defaults.spec   # RPM: declares which variants exist
```

```
    ### settings-plugins/
        ### settings-plugins.spec    # RPM: maps variants to settings plugins
```

檢閱下列有關儲存庫結構的備註：

- 變體主要是中繼資料。外部套件提供核心、驅動程式和使用者空間。
- Settings-defaults 檔案是符號連結，而不是副本。使用 `cp -R`（而不是在 macOS `cp -r`上）來保留它們。
- 新增變體需要編輯五個位置：兩個工作區`Cargo.toml`檔案、兩個`.spec`檔案和 `README.md`。

先決條件

若要完成演練，您需要：

- 具有啟動 EC2 執行個體和註冊 AMIs許可 AWS 的帳戶
- 具有至少 8 個核心、16 GiB 記憶體和 150 GB 磁碟的 EC2 執行個體（或同等 Linux x86_64 主機）
- Ubuntu 24.04 LTS（或 Fedora；不支援 macOS 做為建置主機）
- Docker 20.10 或更新版本
- Rust（穩定工具鏈，透過 rustup 安裝）
- 貨物製造（最新版本）
- 熟悉 Git、Rust 的 Cargo 和 RPM 封裝概念

> **Note**
>
> 當您完成本演練時，請終止 EC2 執行個體並取消註冊任何不再需要的 AMIs，以避免持續收費。如需清除的指示，請參閱 the section called “清除”。

步驟 1：準備建置主機

啟動 EC2 執行個體，例如 `c7i.8xlarge`(32 vCPU、64 GiB 記憶體）。使用 150 GB gp3 根磁碟區，並連接 SSM 存取的 `AmazonSSMManagedInstanceCore`受管政策。

使用 AWS Systems Manager (SSM) Session Manager 連線至執行個體：

```
aws ssm start-session --target <instance-id>
```

```
cd ~
```

安裝所需的作業系統套件：

```
apt-get update
apt-get install -y build-essential openssl libssl-dev pkg-config lz4 \
                   git ca-certificates curl gnupg
```

> **Note**
>
> 官方BUILDING.md參考 `liblz4-tool`。在最近的 Ubuntu 版本中，套件名為 `lz4`。

使用下列命令安裝 Docker：

```
install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg \
  -o /etc/apt/keyrings/docker.asc
chmod a+r /etc/apt/keyrings/docker.asc
echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] \
  https://download.docker.com/linux/ubuntu noble stable" \
  > /etc/apt/sources.list.d/docker.list
apt-get update
apt-get install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin
systemctl enable --now docker
```

使用下列命令安裝 Rust 和 Cargo-make：

```
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh -s -- -y
. "$HOME/.cargo/env"
cargo install cargo-make
```

步驟 2：複製儲存庫

複製 GitHub 網站上的 Bottlerocket 儲存庫並導覽至 目錄：

```
cd ~/bottlerocket
```

若要建立可重現的組建，請查看已標記的版本（例如 `git checkout v1.62.1`)。若要使用最新的套件，請保留在`develop`分支上。

步驟 3：驗證套件版本

開啟Twoliter.toml並確認套件版本。對於 R595 支援，您需要 bottlerocket-kernel-kit 6.2.2 版或更新版本：

```
[[kit]]
name = "bottlerocket-kernel-kit"
version = "6.2.2"
vendor = "bottlerocket"
```

如果版本較舊，請更新它並重新產生鎖定：

```
./tools/twoliter/twoliter update
```

步驟 4：解決貨物製造路徑問題

兩升CARGO_HOME設定為 ~/bottlerocket/.cargo，可防止內部貨運程序尋找您全域安裝的貨運製造。創建 symlink：

```
mkdir -p ~/bottlerocket/.cargo/bin
ln -sf /root/.cargo/bin/cargo-make ~/bottlerocket/.cargo/bin/cargo-make
```

如果沒有此步驟，建置會失敗並顯示 error: no such command: make。

步驟 5：尋找可用的驅動程式分支

NVIDIA kmod 套件會在 內運送bottlerocket-kernel-kit。如需可用驅動程式套件的相關資訊，請參閱 GitHub 網站上的核心套件目錄，或 GitHub 網站上的版本備註。

對於核心 6.18（變aws-k8s-1.36體使用）：

```
kmod-6.18-nvidia-r580   # driver 580.159.03 (current default)
kmod-6.18-nvidia-r595   # driver 595.71.05
```

對於核心 6.12（由 aws-k8s-1.33、1.34、 使用1.35)：

```
kmod-6.12-nvidia-r580
kmod-6.12-nvidia-r595
```

> **Note**
>
> 每個 kmod 套件都會運送子套件 (`-tesla`、`-open-gpu`、`-grid`、`-fabricmanager`、`-imex`)。官方 Bottlerocket NVIDIA 變體參考 `-tesla`，透過 RPM 相依性提取所有必要的子套件。在開機時，Bottlerocket 會根據執行個體類型自動選取適當的驅動程式樣式。

步驟 6：建立新的變體

複製現有的變體。使用 `cp -R`保留符號連結：

```
cp -R variants/aws-k8s-1.36-nvidia variants/aws-k8s-1.36-nvidia-595
cp -R sources/settings-defaults/aws-k8s-1.36-nvidia \
      sources/settings-defaults/aws-k8s-1.36-nvidia-595
```

編輯 `variants/aws-k8s-1.36-nvidia-595/Cargo.toml`：

```
- name = "aws-k8s-1_36-nvidia"
+ name = "aws-k8s-1_36-nvidia-595"

- "kmod-6.18-nvidia-r580-tesla",
+ "kmod-6.18-nvidia-r595-tesla",
```

編輯 `sources/settings-defaults/aws-k8s-1.36-nvidia-595/Cargo.toml`：

```
- name = "settings-defaults-aws-k8s-1_36-nvidia"
+ name = "settings-defaults-aws-k8s-1_36-nvidia-595"
```

步驟 7：註冊變體

在五個檔案中註冊新的變體：

1. **`Cargo.toml`** — 新增工作區成員：

```
 "variants/aws-k8s-1.36-nvidia",
+    "variants/aws-k8s-1.36-nvidia-595",
 "variants/aws-k8s-1.36-nvidia-fips",
```

2. **`sources/Cargo.toml`** — 新增 settings-defaults 成員：

```
 "settings-defaults/aws-k8s-1.36-nvidia",
```

```
+    "settings-defaults/aws-k8s-1.36-nvidia-595",
```

3. **packages/settings-defaults/settings-defaults.spec** — 在建置迴圈和%files區段中新增%package區塊、項目：

```
%package aws-k8s-1.36-nvidia-595
Summary: Settings defaults for the aws-k8s 1.36 nvidia-595 variant
Requires: %{_cross_os}variant(aws-k8s-1.36-nvidia-595)
Provides: %{_cross_os}settings-defaults(any)
Provides: %{_cross_os}settings-defaults(aws-k8s-1.36-nvidia-595)
Conflicts: %{_cross_os}settings-defaults(any)

%description aws-k8s-1.36-nvidia-595
%{summary}.
```

將 新增至兩個for defaults in迴圈：

```
    aws-k8s-1.36-nvidia \
+   aws-k8s-1.36-nvidia-595 \
    metal-dev \
```

新增 %files區段：

```
%files aws-k8s-1.36-nvidia-595
%{_cross_defaultsdir}/aws-k8s-1.36-nvidia-595.toml
%{_cross_tmpfilesdir}/storewolf-defaults-aws-k8s-1.36-nvidia-595.conf
```

4. **packages/settings-plugins/settings-plugins.spec** — 在 下新增一Provides:行%package aws-k8s-nvidia：

```
 Provides: %{_cross_os}settings-plugin(aws-k8s-1.36-nvidia)
+Provides: %{_cross_os}settings-plugin(aws-k8s-1.36-nvidia-595)
 Conflicts: %{_cross_os}settings-plugin(any)
```

步驟 8：重新整理 lockfile

新增工作區成員會使 失效sources/Cargo.lock。重新整理它：

```
cd ~/bottlerocket/sources
cargo update --workspace
```

> **Important**
>
> 請勿使用 `cargo generate-lockfile`。它會重寫整個 lockfile 並碰撞傳輸相依性，這會導致cargo-deny重複版本錯誤。

步驟 9：建立 NVIDIA 授權檔案

NVIDIA 會限制驅動程式來源的轉散發。您必須先新增明確的授權認可，才能建置：

```
cat > ~/bottlerocket/Licenses.toml <<'EOF'
[nvidia]
spdx-id = "LicensesRef-NVIDIA-Customer-Use"
licenses = [
  { path = "LICENSE", license-url = "https://www.nvidia.com/en-us/drivers/nvidia-
license/" }
]
EOF
```

步驟 10：建置和發佈 AMI

`Infra.toml` 使用目標區域建立 ：

```
cat > ~/bottlerocket/Infra.toml <<'EOF'
[aws]
regions = ["us-west-2", "us-east-1", "us-east-2"]
EOF
```

建置映像：

```
cd ~/bottlerocket
cargo make \
  -e BUILDSYS_VARIANT=aws-k8s-1.36-nvidia-595 \
  -e BUILDSYS_UPSTREAM_SOURCE_FALLBACK=true \
  -e BUILDSYS_UPSTREAM_LICENSE_FETCH=true \
  -e BUILDSYS_JOBS=32
```

第一個建置會提取 SDK 和套件映像（總計約 2 GB)。後續建置在 32 核心主機上需要 3-5 分鐘。

發佈 AMI：

```
cargo make \
```

```
  -e BUILDSYS_VARIANT=aws-k8s-1.36-nvidia-595 \
  -e PUBLISH_REGIONS=us-west-2,us-east-1,us-east-2 \
  ami
```

建置會將 AMI IDs寫入 `build/images/x86_64-aws-k8s-1.36-nvidia-595/latest/*-amis.json`。在 EKS 受管節點群組、Karpenter `EC2NodeClass`或啟動範本中使用這些節點群組。

您可以建置的其他組合

此演練會交換 NVIDIA 驅動程式分支，但您可以針對上游套件中提供的任何套件使用相同的方法。下列範例顯示您可以組裝哪些項目，而無需偽造套件：

您想要的內容	變體中的變更內容 **`Cargo.toml`**
不同的 NVIDIA 驅動程式分支	`kmod-6.18-nvidia-r580-tesla` → `kmod-6.18-nvidia-r595-tesla`
不同的核心版本	`kernel-6.18` → `kernel-6.12`（相應地調整 kmod）
完全移除 NVIDIA	從 刪除三`nvidia-*`行 `included-packages`
新增 EFA 支援	`kmod-6.18-efa` 新增至 `included-packages`
切換容器執行時間版本	`containerd-2.2` → `containerd-2.1`

如需可用套件的相關資訊，請參閱 GitHub 網站上的 kernel-kit 和 GitHub 網站上的 core-kit。

> **Important**
>
> Bottlerocket 根檔案系統不可變。您無法在執行時間安裝套件。套件中的每個套件都是專門針對 Bottlerocket 進行交叉編譯，標準上游 RPMs無法運作。如果您需要尚未在套件中的軟體，請考慮使用 GitHub 網站上的引導容器或 GitHub 網站上的主機容器作為執行期替代方案。

清除

如果您不再需要建置主機，請終止 EC2 執行個體以避免持續收費。AMIs 會獨立保留在您的帳戶中；如果您不再需要，請透過 EC2 主控台或 CLI 取消註冊這些 AMI。

摘要

本主題說明如何使用不同的 NVIDIA 驅動程式分支建立自訂 Bottlerocket 變體。此程序涉及複製現有的變體、變更一個套件參考、在工作區和 RPM 規格中註冊新的變體，以及執行建置。相同的方法適用於任何自訂：交換核心版本、新增套件，或為新的 Kubernetes 版本建立變體。

使用最佳化的 Ubuntu Linux AMI 建立節點

Canonical 已與 Amazon EKS 合作，建立了可供您用於叢集的節點 AMI。

Canonical 提供專為特定用途打造的 Kubernetes 節點作業系統映像。此基本款 Ubuntu 映像已針對 Amazon EKS 最佳化，包含了與 AWS 共同開發的自訂 AWS 核心。如需詳細資訊，請參閱 Amazon Elastic Kubernetes Service (EKS) 上的 Ubuntu 和 the section called “Ubuntu Linux”。如需有關支援的資訊，請參閱 AWS Premium Support 常見問答集的第三方軟體章節。

使用最佳化的 Windows AMI 建立節點

Windows Amazon EKS 最佳化 AMIs 建置於 Windows Server 2019、Windows Server 2022 和 Windows Server 2025 之上。這些 AMI 已設定為 Amazon EKS 節點的基礎映像。預設情況下，AMI 會包括以下組件：

- kubelet
- kube-proxy
- AWS 適用於 Kubernetes 的 IAM Authenticator
- csi-proxy
- containerd

> **Note**
>
> 您可以使用 Microsoft 安全性更新指南，追蹤 Windows Server 的安全性或隱私權事件。

適用於 Amazon EKS 提供針對有下列變化之 Windows 容器最佳化的 AMI：

- Amazon EKS 最佳化 Windows Server 2019 Core AMI
- Amazon EKS 最佳化 Windows Server 2019 Full AMI
- Amazon EKS 最佳化 Windows Server 2022 Core AMI

- Amazon EKS 最佳化 Windows Server 2022 完整 AMI
- Amazon EKS 最佳化 Windows Server 2025 Core AMI
- Amazon EKS 最佳化 Windows Server 2025 Full AMI

Important

- Amazon EKS 最佳化 Windows Server 20H2 Core AMI 正在逐漸棄用。不再發佈此 AMI 的新版本。
- 為了確保您預設擁有最新的安全性更新，Amazon EKS 會維護最近 4 個月的最佳化 Windows AMI。每個新的 AMI 自首次發布起將提供 4 個月。此期限過後，較舊的 AMI 將被設為私有且無法再存取。我們建議使用最新的 AMI，以避免安全性漏洞並失去對已達支援生命週期終點的舊版 AMI 的存取權。雖然我們無法保證我們可以提供已設為私有AMIs 存取權，但您可以透過向 AWS Support 提交票證來請求存取權。

版本發佈日曆

下表列出 Amazon EKS 上 Windows 版本的發佈和終止支援日期。如果終止日期為空白，則是因為版本仍然受到支援。

Windows 版本	Amazon EKS 發佈	Amazon EKS 終止支援
Windows Server 2025 Core	01/27/2026	
Windows Server 2025 完整	01/27/2026	
Windows Server 2022 Core	10/17/2022	
Windows Server 2022 Full	10/17/2022	
Windows Server 20H2 Core	8/12/2021	8/9/2022
Windows Server 2004 Core	8/19/2020	12/14/2021
Windows Server 2019 Core	10/7/2019	
Windows Server 2019 Full	10/7/2019	

Windows 版本	Amazon EKS 發佈	Amazon EKS 終止支援
Windows Server 1909 Core	10/7/2019	12/8/2020

引導指令碼組態參數

建立 Windows 節點時，節點上會有允許設定不同參數的指令碼。根據設定，可以在節點上類似於以下位置的地方找到該指令碼：`C:\Program Files\Amazon\EKS\Start-EKSBootstrap.ps1`。您可將自訂參數值指定為引導指令碼的引數，以指定自訂參數值。例如，您可以更新啟動範本中的使用者資料。如需詳細資訊，請參閱the section called “Amazon EC2 使用者資料”。

該指令碼包括下列命令列參數：

- `-EKSClusterName`：指定此工作節點要加入的 Amazon EKS 叢集名稱。
- `-KubeletExtraArgs`：指定 `kubelet` 額外的引數 (選用)。
- `-KubeProxyExtraArgs`：指定 `kube-proxy` 額外的引數 (選用)。
- `-APIServerEndpoint`：指定 Amazon EKS 叢集 API 伺服器端點 (可選)。僅在搭配 `-Base64ClusterCA` 使用時有效。略過呼叫 `Get-EKSCluster`。
- `-Base64ClusterCA`：指定 Base64 編碼的叢集 CA 內容 (可選)。僅在搭配 `-APIServerEndpoint` 使用時有效。略過呼叫 `Get-EKSCluster`。
- `-DNSClusterIP`：覆寫要用於叢集內 DNS 查詢的 IP 地址 (選用)。根據主要介面的 IP 地址，預設為 `10.100.0.10` 或 `172.20.0.10`。
- `-ServiceCIDR` – 覆寫叢集服務從中尋址的 Kubernetes 服務 IP 位址範圍。根據主要介面的 IP 地址，預設為 `172.20.0.0/16` 或 `10.100.0.0/16`。
- `-ExcludedSnatCIDRs`：要從來源網路地址轉譯 (SNAT) 排除的 `IPv4` CIDR 清單。這表示 VPC 可定址的 Pod 私有 IP 不會轉譯為傳出流量執行個體 ENI 主要 `IPv4` 位址的 IP 位址。依預設，會新增用於 Amazon EKS Windows 節點之 VPC 的 `IPv4` CIDR。將 CIDR 指定給此參數也會另外排除指定的 CIDR。如需詳細資訊，請參閱the section called “傳出流量”。

除了命令列參數之外，您還可以指定一些環境變數參數。指定命令列參數時，它的優先順序高於個別的環境變數。環境變數應定義為機器 (或系統) 範圍，因為引導指令碼只會讀取機器範圍的變數。

此指令碼會考慮下列環境變數：

- `SERVICE_IPV4_CIDR`：請參閱 `ServiceCIDR` 命令列參數了解定義。

- `EXCLUDED_SNAT_CIDRS`：應為逗號分隔的字串。請參閱 `ExcludedSnatCIDRs` 命令列參數了解定義。

gMSA 身分驗證支援

Amazon EKS Windows Pod 允許不同類型的群組受管服務帳戶 (gMSA) 身分驗證。

- Amazon EKS 支援 Active Directory 網域身分進行身分驗證。如需加入網域的 gMSA 的詳細資訊，請參閱 AWS 部落格上的 Amazon EKS Windowspods 上的 Windows 身分驗證。
- Amazon EKS 提供的外掛程式可讓未加入網域的 Windows 節點擷取具有可攜式使用者身分的 gMSA 憑證。如需無網域 gMSA 的詳細資訊，請參閱 AWS 部落格上的適用於 Amazon EKS Windowspods 的無網域 Windows 身分驗證。

快取容器映像

Amazon EKS Windows 最佳化 AMI 具有針對 `containerd` 執行時期執行期快取的特定容器映像。使用 Amazon 受管建置元件建置自訂 AMI 時，會快取容器映像。如需詳細資訊，請參閱the section called “使用 Amazon 受管建置元件”。

下列的快取容器映像適用於 `containerd` 執行期：

- `amazonaws.com/eks/pause-windows`
- `mcr.microsoft.com/windows/nanoserver`
- `mcr.microsoft.com/windows/servercore`

其他資訊

如需使用 Amazon EKS 最佳化 Windows AMI 的詳細資訊，請參閱下列區段：

- 有關在 Amazon EKS 最佳化加速 Windows AMI 上執行工作負載的詳細資訊，請參 the section called “設定 Windows GPU AMI”。
- 若要將 Windows 與受管節點群組搭配使用，請參閱 the section called “受管節點群組”。
- 若要啟動自我管理的 Windows 節點，請參閱 the section called “Windows”。
- 如需版本資訊，請參閱the section called “獲取版本資訊”。
- 若要擷取 Amazon EKS 最佳化 Windows AMI 的最新 ID，請參閱 the section called “獲取最新 ID”。

- 若要使用 Amazon EC2 Image Builder 來建立自訂 Amazon EKS 最佳化 Windows AMI，請參閱 the section called “自訂建置”。
- 如需最佳實務，請參閱《EKS 最佳實務指南》中的 Amazon EKS 最佳化 Windows AMI 管理。

藉助 `eksctl` 建立自我管理 Windows Server 2022 節點

您可以使用下列 `test-windows-2022.yaml` 作為建立自我管理 Windows Server 2022 節點的參考。使用您自己的值取代每一個`###`。

> **Note**
>
> 您必須使用 `eksctl` 0.116.0 版或更新版本，才能執行自我管理 Windows Server 2022 節點。

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: windows-2022-cluster
  region: region-code
  version: '1.36'

nodeGroups:
  - name: windows-ng
    instanceType: m5.2xlarge
    amiFamily: WindowsServer2022FullContainer
    volumeSize: 100
    minSize: 2
    maxSize: 3
  - name: linux-ng
    amiFamily: AmazonLinux2
    minSize: 2
    maxSize: 3
```

然後，可以使用以下命令建立節點群組。

```
eksctl create cluster -f test-windows-2022.yaml
```

擷取 Windows AMI 版本資訊

本主題列出 Amazon EKS 最佳化 Windows AMI 版本及其 kubelet、containerd 和 csi-proxy 的對應版本。

可透過程式設計方式擷取每個變體的 Amazon EKS 最佳化 AMI 中繼資料 (包括 AMI ID)。如需詳細資訊，請參閱the section called “獲取最新 ID”。

AMI 是由 Kubernetes 版本和 AMI 發行日期進行版本化，格式如下：

```
k8s_major_version.k8s_minor_version-release_date
```

> **Note**
>
> Amazon EKS 受管節點群組支援 2022 年 11 月及後來的 Windows AMI 版本。

如需接收有關此特定文件頁面所有來源檔案變更的通知，您可透過 RSS 閱讀器來訂閱以下 URL：

```
https://github.com/awsdocs/amazon-eks-user-guide/commits/mainline/latest/ug/nodes/eks-ami-versions-windows.adoc.atom
```

Amazon EKS 最佳化 Windows Server 2025 Core AMI

下表列出 Amazon EKS 最佳化 Windows Server 2025 Core AMI 的目前和先前版本。

Example

Kubernetes version 1.36

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.36-2026.08.15	1.36.2	2.2.3	1.2.1	
1.36-2026.07.18	1.36.2	2.2.3	1.2.1	
1.36-2026.06.18	1.36.2	2.2.3	1.2.1	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.36-2026.05.19	1.36.1	2.2.3	1.2.1	

Kubernetes version 1.35

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.35-2026.08.15	1.35.6	2.1.7	1.2.1	
1.35-2026.07.18	1.35.6	2.1.7	1.2.1	
1.35-2026.06.18	1.35.6	2.1.7	1.2.1	
1.35-2026.05.19	1.35.5	2.1.7	1.2.1	
1.35-2026.04.18	1.35.3	2.1.6	1.2.1	
1.35-2026.02.16	1.35.0	2.1.6	1.2.1	
1.35-2026-01-22	1.35.0	2.1.6	1.2.1	

Amazon EKS 最佳化 Windows Server 2025 Full AMI

下表列出 Amazon EKS 最佳化 Windows Server 2025 Full AMI 的目前和先前版本。

Example

Kubernetes version 1.36

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.36-2026.08.15	1.36.2	2.2.3	1.2.1	
1.36-2026.07.18	1.36.2	2.2.3	1.2.1	
1.36-2026.06.18	1.36.2	2.2.3	1.2.1	
1.36-2026.05.19	1.36.1	2.2.3	1.2.1	

Kubernetes version 1.35

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.35-2026.08.15	1.35.6	2.1.7	1.2.1	
1.35-2026.07.21	1.35.6	2.1.7	1.2.1	
1.35-2026.06.18	1.35.6	2.1.7	1.2.1	
1.35-2026.05.19	1.35.5	2.1.7	1.2.1	
1.35-2026.04.18	1.35.3	2.1.6	1.2.1	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.35-2026.02.16	1.35.0	2.1.6	1.2.1	
1.35-2026-01-22	1.35.0	2.1.6	1.2.1	

Amazon EKS 最佳化 Windows Server 2022 Core AMI

下表列出 Amazon EKS 最佳化 Windows Server 2022 Core AMI 的目前版本和先前版本。

Example

Kubernetes version 1.36

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.36-2026.08.15	1.36.2	2.2.3	1.2.1	
1.36-2026.07.18	1.36.2	2.2.3	1.2.1	
1.36-2026.06.18	1.36.2	2.2.3	1.2.1	
1.36-2026.05.19	1.36.1	2.2.3	1.2.1	

Kubernetes version 1.35

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.35-2026.08.15	1.35.6	2.1.7	1.2.1	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.35-2026.07.21	1.35.6	2.1.7	1.2.1	
1.35-2026.06.18	1.35.6	2.1.7	1.2.1	
1.35-2026.05.19	1.35.5	2.1.7	1.2.1	
1.35-2026.04.18	1.35.3	2.1.6	1.2.1	
1.35-2026.02.16	1.35.0	2.1.6	1.2.1	
1.35-2026-01-22	1.35.0	2.1.6	1.2.1	

Kubernetes version 1.34

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.34-2026.08.15	1.34.9	2.1.7	1.2.1	
1.34-2026.07.18	1.34.9	2.1.7	1.2.1	
1.34-2026.06.18	1.34.9	2.1.6	1.2.1	
1.34-2026.05.19	1.34.8	2.1.6	1.2.1	
1.34-2026.04.18	1.34.6	2.1.6	1.2.1	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.34-2026.02.13	1.34.3	2.1.6	1.2.1	
1.34-2026.01.22	1.34.2	2.1.6	1.2.1	已將 containerd 升級至 2.1.6。
1.34-2025.12.15	1.34.2	2.1.4	1.2.1	
1.34-2025.11.14	1.34.1	2.1.4	1.2.1	
1.34-2025.10.18	1.34.1	2.1.4	1.2.1	
1.34-2025.09.13	1.34.0	2.1.4	1.2.1	

Kubernetes version 1.33

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.33-2026.08.15	1.33.13	1.7.30	1.2.1	
1.33-2026.07.18	1.33.13	1.7.30	1.2.1	
1.33-2026.06.18	1.33.13	1.7.30	1.2.1	
1.33-2026.05.19	1.33.11	1.7.30	1.2.1	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.33-2026.04.18	1.33.10	1.7.30	1.2.1	
1.33-2026.02.13	1.33.7	1.7.30	1.2.1	
1.33-2026.01.22	1.33.5	1.7.30	1.2.1	已將 containerd 升級至 1.7.30。
1.33-2025.12.15	1.33.5	1.7.28	1.2.1	
1.33-2025.11.14	1.33.5	1.7.28	1.2.1	
1.33-2025.10.18	1.33.5	1.7.28	1.2.1	
1.33-2025.09.13	1.33.4	1.7.28	1.2.1	將 GMSA 外掛程式日誌變更為 Windows Events
1.33-2025.08.18	1.33.3	1.7.27	1.2.1	
1.33-2025.07.16	1.33.1	1.7.27	1.2.1	
1.33-2025.06.13	1.33.1	1.7.27	1.2.1	
1.33-2025.05.17	1.33.1	1.7.27	1.2.1	

Kubernetes version 1.32

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.32-2026.08.15	1.32.13	1.7.30	1.2.1	
1.32-2026.07.18	1.32.13	1.7.30	1.2.1	
1.32-2026.06.18	1.32.13	1.7.30	1.2.1	
1.32-2026.05.19	1.32.13	1.7.30	1.2.1	
1.32-2026.04.18	1.32.13	1.7.30	1.2.1	
1.32-2026.02.13	1.32.11	1.7.30	1.2.1	
1.32-2026.01.22	1.32.9	1.7.30	1.2.1	已將 containerd 升級至 1.7.30。
1.32-2025.12.15	1.32.9	1.7.28	1.2.1	
1.32-2025.11.14	1.32.9	1.7.28	1.2.1	
1.32-2025.10.18	1.32.9	1.7.28	1.2.1	
1.32-2025.09.13	1.32.8	1.7.28	1.2.1	將 GMSA 外掛程式日誌變更為 Windows Events

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.32-2025.08.18	1.32.7	1.7.27	1.2.1	
1.32-2025.07.16	1.32.5	1.7.27	1.2.1	
1.32-2025.06.13	1.32.5	1.7.27	1.2.1	已將 containerd 升級至 1.7.27。
1.32-2025.05.17	1.32.5	1.7.20	1.2.1	
1.32-2025.04.14	1.32.1	1.7.20	1.1.3	
1.32-2025.03.14	1.32.1	1.7.20	1.1.3	
1.32-2025.02.18	1.32.1	1.7.20	1.1.3	
1.32-2025.01.15	1.32.0	1.7.20	1.1.3	

Kubernetes version 1.31

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.31-2026.08.15	1.31.14	1.7.30	1.2.1	
1.31-2026.07.18	1.31.14	1.7.30	1.2.1	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.31-2026.06.18	1.31.14	1.7.30	1.2.1	
1.31-2026.05.19	1.31.14	1.7.30	1.2.1	
1.31-2026.04.18	1.31.14	1.7.30	1.2.1	
1.31-2026.02.13	1.31.14	1.7.30	1.2.1	
1.31-2026.01.22	1.31.13	1.7.30	1.2.1	已將 containerd 升級至 1.7.30。
1.31-2025.12.15	1.31.13	1.7.28	1.2.1	
1.31-2025.11.14	1.31.13	1.7.28	1.2.1	
1.31-2025.10.18	1.31.13	1.7.28	1.2.1	
1.31-2025.09.13	1.31.12	1.7.28	1.2.1	將 GMSA 外掛程式日誌變更為 Windows Events
1.31-2025.08.18	1.31.11	1.7.27	1.2.1	
1.31-2025.07.16	1.31.9	1.7.27	1.2.1	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.31-2025.06.13	1.31.9	1.7.27	1.2.1	已將 containerd 升級至 1.7.27。
1.31-2025.05.17	1.31.9	1.7.20	1.2.1	
1.31-2025.04.14	1.31.5	1.7.20	1.1.3	
1.31-2025.03.14	1.31.5	1.7.20	1.1.3	
1.31-2025.02.15	1.31.5	1.7.20	1.1.3	
1.31-2025.01.15	1.31.4	1.7.20	1.1.3	
1.31-2025.01.01	1.31.4	1.7.20	1.1.3	包含 CVE-2024-9042 的修補程式。
1.31-2024.12.13	1.31.3	1.7.20	1.1.3	
1.31-2024.11.12	1.31.1	1.7.20	1.1.3	
1.31-2024.10.08	1.31.1	1.7.20	1.1.3	
1.31-2024.10.01	1.31.1	1.7.20	1.1.3	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.31-2024.09.10	1.31.0	1.7.20	1.1.3	

Kubernetes version 1.30

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.30-2026.07.18	1.30.14	1.7.30	1.2.1	
1.30-2026.06.18	1.30.14	1.7.30	1.2.1	
1.30-2026.05.19	1.30.14	1.7.30	1.2.1	
1.30-2026.04.18	1.30.14	1.7.30	1.2.1	
1.30-2026.02.13	1.30.14	1.7.30	1.2.1	
1.30-2026.01.22	1.30.14	1.7.30	1.2.1	已將 containerd 升級至 1.7.30。
1.30-2025.12.15	1.30.14	1.7.28	1.2.1	
1.30-2025.11.21	1.30.14	1.7.28	1.2.1	
1.30-2025.10.18	1.30.14	1.7.28	1.2.1	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.30-2025.09.13	1.30.14	1.7.28	1.2.1	將 GMSA 外掛程式日誌變更為 Windows Events
1.30-2025.08.18	1.30.14	1.7.27	1.2.1	
1.30-2025.07.16	1.30.13	1.7.27	1.2.1	
1.30-2025.06.13	1.30.13	1.7.27	1.2.1	已將 containerd 升級至 1.7.27。
1.30-2025.05.17	1.30.13	1.7.20	1.2.1	
1.30-2025.04.14	1.30.9	1.7.20	1.1.3	
1.30-2025.03.14	1.30.9	1.7.20	1.1.3	已將 containerd 升級至 1.7.20。
1.30-2025.02.15	1.30.9	1.7.14	1.1.3	
1.30-2025.01.15	1.30.8	1.7.14	1.1.3	
1.30-2025.01.01	1.30.8	1.7.14	1.1.3	包含 CVE-2024-9042 的修補程式。

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.30-2024.12.11	1.30.7	1.7.14	1.1.3	
1.30-2024.11.12	1.30.4	1.7.14	1.1.3	
1.30-2024.10.08	1.30.4	1.7.14	1.1.3	
1.30-2024.09.10	1.30.2	1.7.14	1.1.3	
1.30-2024.08.13	1.30.2	1.7.14	1.1.3	
1.30-2024.07.10	1.30.2	1.7.14	1.1.2	包含 CVE-2024-5321 的修補程式。
1.30-2024.06.17	1.30.0	1.7.14	1.1.2	已將 containerd 升級至 1.7.14。
1.30-2024.05.15	1.30.0	1.6.28	1.1.2	

Kubernetes version 1.29

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.29-2026.02.13	1.29.15	1.7.30	1.2.1	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.29-2026.01.22	1.29.15	1.7.30	1.2.1	已將 containerd 升級至 1.7.30。
1.29-2025.12.15	1.29.15	1.7.28	1.2.1	
1.29-2025.11.14	1.29.15	1.7.28	1.2.1	
1.29-2025.10.18	1.29.15	1.7.28	1.2.1	
1.29-2025.09.13	1.29.15	1.7.28	1.2.1	將 GMSA 外掛程式日誌變更為 Windows Events
1.29-2025.08.18	1.29.15	1.7.27	1.2.1	
1.29-2025.07.16	1.29.15	1.7.27	1.2.1	
1.29-2025.06.13	1.29.15	1.7.27	1.2.1	已將 containerd 升級至 1.7.27。
1.29-2025.05.17	1.29.15	1.7.20	1.2.1	
1.29-2025.04.14	1.29.13	1.7.20	1.1.3	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.29-2025.03.14	1.29.13	1.7.20	1.1.3	已將 containerd 升級至 1.7.20。
1.29-2025.02.15	1.29.13	1.7.14	1.1.3	
1.29-2025.01.15	1.29.12	1.7.14	1.1.3	
1.29-2025.01.01	1.29.12	1.7.14	1.1.3	包含 CVE-2024-9042 的修補程式。
1.29-2024.12.11	1.29.10	1.7.14	1.1.3	
1.29-2024.11.12	1.29.8	1.7.14	1.1.3	
1.29-2024.10.08	1.29.8	1.7.14	1.1.3	
1.29-2024.09.10	1.29.6	1.7.14	1.1.3	
1.29-2024.08.13	1.29.6	1.7.14	1.1.3	
1.29-2024.07.10	1.29.6	1.7.11	1.1.2	包含 CVE-2024-5321 的修補程式。

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.29-2024.06.17	1.29.3	1.7.11	1.1.2	
1.29-2024.05.15	1.29.3	1.7.11	1.1.2	已將 containerd 升級至 1.7.11。已將 kubelet 升級至 1.29.3。
1.29-2024.04.09	1.29.0	1.6.28	1.1.2	已將 containerd 升級至 1.6.28。使用 golang 1.22.1 重建 CNI 和 csi-proxy 。
1.29-2024.03.12	1.29.0	1.6.25	1.1.2	
1.29-2024.02.13	1.29.0	1.6.25	1.1.2	
1.29-2024.02.06	1.29.0	1.6.25	1.1.2	修正 kubelet 垃圾回收程序錯誤刪除暫停映像的錯誤。

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.29-2024.01.11	1.29.0	1.6.18	1.1.2	排除 Windows Server 2022 Core AMI 上的獨立 Windows 更新 KB5034439。此 KB 僅適用於具有獨立 WinRE 分割區的 Windows 安裝，而我們的任何 Amazon EKS 最佳化 Windows AMI 均未包含此分割區。

Kubernetes version 1.28

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.28-2025.11.14	1.28.15	1.7.28	1.2.1	
1.28-2025.10.18	1.28.15	1.7.28	1.2.1	
1.28-2025.09.13	1.28.15	1.7.28	1.2.1	將 GMSA 外掛程式日誌變更為 Windows Events
1.28-2025.08.18	1.28.15	1.7.27	1.2.1	
1.28-2025.07.16	1.28.15	1.7.27	1.2.1	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.28-2025.06.13	1.28.15	1.7.27	1.2.1	已將 containerd 升級至 1.7.27。
1.28-2025.05.17	1.28.15	1.7.20	1.2.1	
1.28-2025.04.14	1.28.15	1.7.20	1.1.3	
1.28-2025.03.14	1.28.15	1.7.20	1.1.3	已將 containerd 升級至 1.7.20。
1.28-2025.02.15	1.28.15	1.7.14	1.1.3	
1.28-2025.01.15	1.28.15	1.7.14	1.1.3	
1.28-2025-01-01	1.28.15	1.7.14	1.1.3	包含 CVE-2024-9042 的修補程式。
1.28-2024.12.11	1.28.15	1.7.14	1.1.3	
1.28-2024.11.12	1.28.13	1.7.14	1.1.3	
1.28-2024.10.08	1.28.13	1.7.14	1.1.3	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.28-2024.09.10	1.28.11	1.7.14	1.1.3	
1.28-2024.08.13	1.28.11	1.7.14	1.1.3	
1.28-2024.07.10	1.28.11	1.7.11	1.1.2	包含 CVE-2024-5321 的修補程式。
1.28-2024.06.17	1.28.8	1.7.11	1.1.2	已將 containerd 升級至 1.7.11。
1.28-2024.05.14	1.28.8	1.6.28	1.1.2	已將 containerd 升級至 1.6.28。已將 kubelet 升級至 1.28.8。
1.28-2024.04.09	1.28.5	1.6.25	1.1.2	已將 containerd 升級至 1.6.25。使用 golang 1.22.1 重建 CNI 和 csi-proxy 。
1.28-2024.03.12	1.28.5	1.6.18	1.1.2	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.28-2024.02.13	1.28.5	1.6.18	1.1.2	
1.28-2024.01.11	1.28.5	1.6.18	1.1.2	排除 Windows Server 2022 Core AMI 上的獨立 Windows 更新 KB5034439。此 KB 僅適用於具有獨立 WinRE 分割區的 Windows 安裝，而我們的任何 Amazon EKS 最佳化 Windows AMI 均未包含此分割區。
1.28-2023.12.12	1.28.3	1.6.18	1.1.2	
1.28-2023.11.14	1.28.3	1.6.18	1.1.2	包含 CVE-2023-5528 的修補程式。

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.28-2023.10.19	1.28.2	1.6.18	1.1.2	已將 containerd 升級至 1.6.18。已新增引導指令碼環境變數 (SERVICE_IPV4_CIDR 和 EXCLUDED_SNAT_CIDRS)。
1.28-2023-09.27	1.28.2	1.6.6	1.1.2	已修正 kubelet 的 安全性公告。
1.28-2023.09.12	1.28.1	1.6.6	1.1.2	

Amazon EKS 最佳化 Windows Server 2022 Full AMI

下表列出 Amazon EKS 最佳化 Windows Server 2022 Full AMI 的目前版本和先前版本。

Example

Kubernetes version 1.36

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.36-2026.08.15	1.36.2	2.2.3	1.2.1	
1.36-2026.07.18	1.36.2	2.2.3	1.2.1	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.36-2026.06.18	1.36.2	2.2.3	1.2.1	
1.36-2026.05.19	1.36.1	2.2.3	1.2.1	

Kubernetes version 1.35

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.35-2026.08.15	1.35.6	2.1.7	1.2.1	
1.35-2026.07.18	1.35.6	2.1.7	1.2.1	
1.35-2026.06.18	1.35.6	2.1.7	1.2.1	
1.35-2026.05.19	1.35.5	2.1.7	1.2.1	
1.35-2026.04.18	1.35.3	2.1.6	1.2.1	
1.35-2026.02.16	1.35.0	2.1.6	1.2.1	
1.35-2026-01-22	1.35.0	2.1.6	1.2.1	

Kubernetes version 1.34

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.34-2026.08.15	1.34.9	2.1.7	1.2.1	
1.34-2026.07.18	1.34.9	2.1.7	1.2.1	
1.34-2026.06.18	1.34.9	2.1.6	1.2.1	
1.34-2026.05.19	1.34.8	2.1.6	1.2.1	
1.34-2026.04.18	1.34.6	2.1.6	1.2.1	
1.34-2026.02.13	1.34.3	2.1.6	1.2.1	
1.34-2026.01.22	1.34.2	2.1.6	1.2.1	已將 containerd 升級至 2.1.6。
1.34-2025.12.15	1.34.2	2.1.4	1.2.1	
1.34-2025.11.14	1.34.1	2.1.4	1.2.1	
1.34-2025.10.18	1.34.1	2.1.4	1.2.1	
1.34-2025.09.13	1.34.0	2.1.4	1.2.1	

Kubernetes version 1.33

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.33-2026.08.15	1.33.13	1.7.30	1.2.1	
1.33-2026.07.21	1.33.13	1.7.30	1.2.1	
1.33-2026.06.18	1.33.13	1.7.30	1.2.1	
1.33-2026.05.19	1.33.11	1.7.30	1.2.1	
1.33-2026.04.18	1.33.10	1.7.30	1.2.1	
1.33-2026.02.13	1.33.7	1.7.30	1.2.1	
1.33-2026.01.22	1.33.5	1.7.30	1.2.1	已將 containerd 升級至 1.7.30。
1.33-2025.12.15	1.33.5	1.7.28	1.2.1	
1.33-2025.11.14	1.33.5	1.7.28	1.2.1	
1.33-2025.10.18	1.33.5	1.7.28	1.2.1	
1.33-2025.09.13	1.33.4	1.7.28	1.2.1	將 GMSA 外掛程式日誌變更為 Windows Events

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.33-2025.08.18	1.33.3	1.7.27	1.2.1	
1.33-2025.07.16	1.33.1	1.7.27	1.2.1	
1.33-2025.06.13	1.33.1	1.7.27	1.2.1	
1.33-2025.05.17	1.33.1	1.7.27	1.2.1	

Kubernetes version 1.32

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.32-2026.08.15	1.32.13	1.7.30	1.2.1	
1.32-2026.07.18	1.32.13	1.7.30	1.2.1	
1.32-2026.06.18	1.32.13	1.7.30	1.2.1	
1.32-2026.05.19	1.32.13	1.7.30	1.2.1	
1.32-2026.04.18	1.32.13	1.7.30	1.2.1	
1.32-2026.02.13	1.32.11	1.7.30	1.2.1	
1.32-2026.01.22	1.32.9	1.7.30	1.2.1	已將 container

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
				d 升級至 1.7.30。
1.32-2025.12.15	1.32.9	1.7.28	1.2.1	
1.32-2025.11.14	1.32.9	1.7.28	1.2.1	
1.32-2025.10.18	1.32.9	1.7.28	1.2.1	
1.32-2025.09.13	1.32.8	1.7.28	1.2.1	將 GMSA 外掛程式日誌變更為 Windows Events
1.32-2025.08.18	1.32.7	1.7.27	1.2.1	
1.32-2025.07.16	1.32.5	1.7.27	1.2.1	
1.32-2025.06.13	1.32.5	1.7.27	1.2.1	已將 containerd 升級至 1.7.27
1.32-2025.05.17	1.32.5	1.7.20	1.2.1	
1.32-2025.04.14	1.32.1	1.7.20	1.1.3	
1.32-2025.03.14	1.32.1	1.7.20	1.1.3	
1.32-2025.02.18	1.32.1	1.7.20	1.1.3	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.32-2025.01.01	1.32.0	1.7.20	1.1.3	

Kubernetes version 1.31

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.31-2026.08.15	1.31.14	1.7.30	1.2.1	
1.31-2026.07.18	1.31.14	1.7.30	1.2.1	
1.31-2026.06.18	1.31.14	1.7.30	1.2.1	
1.31-2026.05.19	1.31.14	1.7.30	1.2.1	
1.31-2026.04.18	1.31.14	1.7.30	1.2.1	
1.31-2026.02.13	1.31.14	1.7.30	1.2.1	
1.31-2026.01.22	1.31.13	1.7.30	1.2.1	已將 containerd 升級至 1.7.30。
1.31-2025.12.15	1.31.13	1.7.28	1.2.1	
1.31-2025.11.14	1.31.13	1.7.28	1.2.1	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.31-2025.10.18	1.31.13	1.7.28	1.2.1	
1.31-2025.09.13	1.31.12	1.7.28	1.2.1	將 GMSA 外掛程式日誌變更為 Windows Events
1.31-2025.08.18	1.31.11	1.7.27	1.2.1	
1.31-2025.07.16	1.31.9	1.7.27	1.2.1	
1.31-2025.06.13	1.31.9	1.7.27	1.2.1	已將 containerd 升級至 1.7.27。
1.31-2025.05.17	1.31.9	1.7.20	1.2.1	
1.31-2025.04.14	1.31.5	1.7.20	1.1.3	
1.31-2025.03.14	1.31.5	1.7.20	1.1.3	
1.31-2025.02.15	1.31.5	1.7.20	1.1.3	
1.31-2025.01.15	1.31.4	1.7.20	1.1.3	
1.31-2025.01.01	1.31.4	1.7.20	1.1.3	包含 CVE-2024-9042 的修補程式。

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.31-2024.12.13	1.31.3	1.7.20	1.1.3	
1.31-2024.11.12	1.31.1	1.7.20	1.1.3	
1.31-2024.10.08	1.31.1	1.7.20	1.1.3	
1.31-2024.10.01	1.31.1	1.7.20	1.1.3	
1.31-2024.09.10	1.31.0	1.7.20	1.1.3	

Kubernetes version 1.30

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.30-2026.07.18	1.30.14	1.7.30	1.2.1	
1.30-2026.06.18	1.30.14	1.7.30	1.2.1	
1.30-2026.05.19	1.30.14	1.7.30	1.2.1	
1.30-2026.04.18	1.30.14	1.7.30	1.2.1	
1.30-2026.02.13	1.30.14	1.7.30	1.2.1	
1.30-2026.01.22	1.30.14	1.7.30	1.2.1	已將 container

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
				d 升級至 1.7.30。
1.30-2025.12.15	1.30.14	1.7.28	1.2.1	
1.30-2025.11.21	1.30.14	1.7.28	1.2.1	
1.30-2025.10.18	1.30.14	1.7.28	1.2.1	
1.30-2025.09.13	1.30.14	1.7.28	1.2.1	將 GMSA 外掛程式日誌變更為 Windows Events
1.30-2025.08.18	1.30.14	1.7.27	1.2.1	
1.30-2025.07.16	1.30.13	1.7.27	1.2.1	
1.30-2025.06.13	1.30.13	1.7.27	1.2.1	已將 containerd 升級至 1.7.27。
1.30-2025.05.17	1.30.13	1.7.20	1.2.1	
1.30-2025.04.14	1.30.9	1.7.20	1.1.3	
1.30-2025.03.14	1.30.9	1.7.20	1.1.3	已將 containerd 升級至 1.7.20。

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.30-2025.02.15	1.30.9	1.7.14	1.1.3	
1.30-2025.01.15	1.30.8	1.7.14	1.1.3	
1.30-2025.01.01	1.30.8	1.7.14	1.1.3	包含 CVE-2024-9042 的修補程式。
1.30-2024.12.11	1.30.7	1.7.14	1.1.3	
1.30-2024.11.12	1.30.4	1.7.14	1.1.3	
1.30-2024.10.08	1.30.4	1.7.14	1.1.3	
1.30-2024.09.10	1.30.2	1.7.14	1.1.3	
1.30-2024.08.13	1.30.2	1.7.14	1.1.3	
1.30-2024.07.10	1.30.2	1.7.14	1.1.2	包含 CVE-2024-5321 的修補程式。
1.30-2024.06.17	1.30.0	1.7.14	1.1.2	已將 containerd 升級至 1.7.14。

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.30-2024.05.15	1.30.0	1.6.28	1.1.2	

Kubernetes version 1.29

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.29-2026.02.13	1.29.15	1.7.30	1.2.1	
1.29-2026.01.22	1.29.15	1.7.30	1.2.1	已將 containerd 升級至 1.7.30。
1.29-2025.12.15	1.29.15	1.7.28	1.2.1	
1.29-2025.11.14	1.29.15	1.7.28	1.2.1	
1.29-2025.10.18	1.29.15	1.7.28	1.2.1	
1.29-2025.09.13	1.29.15	1.7.28	1.2.1	將 GMSA 外掛程式日誌變更為 Windows Events
1.29-2025.08.18	1.29.15	1.7.27	1.2.1	
1.29-2025.07.16	1.29.15	1.7.27	1.2.1	
1.29-2025.06.13	1.29.15	1.7.27	1.2.1	已將 container

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
				d 升級至 1.7.27。
1.29-2025.05.17	1.29.15	1.7.20	1.2.1	
1.29-2025.04.14	1.29.13	1.7.20	1.1.3	
1.29-2025.03.14	1.29.13	1.7.20	1.1.3	已將 containerd 升級至 1.7.20。
1.29-2025.02.15	1.29.13	1.7.14	1.1.3	
1.29-2025.01.15	1.29.12	1.7.14	1.1.3	
1.29-2025.01.01	1.29.12	1.7.14	1.1.3	包含 CVE-2024-9042 的修補程式。
1.29-2024.12.11	1.29.10	1.7.14	1.1.3	
1.29-2024.11.12	1.29.8	1.7.14	1.1.3	
1.29-2024.10.08	1.29.8	1.7.14	1.1.3	
1.29-2024.09.10	1.29.6	1.7.14	1.1.3	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.29-2024.08.13	1.29.6	1.7.14	1.1.3	
1.29-2024.07.10	1.29.6	1.7.11	1.1.2	包含 CVE-2024-5321 的修補程式。
1.29-2024.06.17	1.29.3	1.7.11	1.1.2	
1.29-2024.05.15	1.29.3	1.7.11	1.1.2	已將 containerd 升級至 1.7.11。已將 kubelet 升級至 1.29.3。
1.29-2024.04.09	1.29.0	1.6.28	1.1.2	已將 containerd 升級至 1.6.28。使用 golang 1.22.1 重建 CNI 和 csi-proxy 。
1.29-2024.03.12	1.29.0	1.6.25	1.1.2	
1.29-2024.02.13	1.29.0	1.6.25	1.1.2	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.29-2024.02.06	1.29.0	1.6.25	1.1.2	修正 kubelet 垃圾回收程序錯誤刪除暫停映像的錯誤。
1.29-2024.01.09	1.29.0	1.6.18	1.1.2	

Kubernetes version 1.28

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.28-2025.11.14	1.28.15	1.7.28	1.2.1	
1.28-2025.10.18	1.28.15	1.7.28	1.2.1	
1.28-2025.09.13	1.28.15	1.7.28	1.2.1	將 GMSA 外掛程式日誌變更為 Windows Events
1.28-2025.08.18	1.28.15	1.7.27	1.2.1	
1.28-2025.07.16	1.28.15	1.7.27	1.2.1	
1.28-2025.06.13	1.28.15	1.7.27	1.2.1	已將 containerd 升級至 1.7.27。
1.28-2025.05.17	1.28.15	1.7.20	1.2.1	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.28-2025.04.14	1.28.15	1.7.20	1.1.3	
1.28-2025.03.14	1.28.15	1.7.20	1.1.3	已將 containerd 升級至 1.7.20。
1.28-2025.02.15	1.28.15	1.7.14	1.1.3	
1.28-2025.01.15	1.28.15	1.7.14	1.1.3	
1.28-2025.01.01	1.28.15	1.7.14	1.1.3	包含 CVE-2024-9042 的修補程式。
1.28-2024.12.11	1.28.15	1.7.14	1.1.3	
1.28-2024.11.12	1.28.13	1.7.14	1.1.3	
1.28-2024.10.08	1.28.13	1.7.14	1.1.3	
1.28-2024.09.10	1.28.11	1.7.14	1.1.3	
1.28-2024.08.13	1.28.11	1.7.14	1.1.3	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.28-2024.07.10	1.28.11	1.7.11	1.1.2	包含 CVE-2024-5321 的修補程式。
1.28-2024.06.17	1.28.8	1.7.11	1.1.2	已將 containerd 升級至 1.7.11。
1.28-2024.05.14	1.28.8	1.6.28	1.1.2	已將 containerd 升級至 1.6.28。已將 kubelet 升級至 1.28.8。
1.28-2024.04.09	1.28.5	1.6.25	1.1.2	已將 containerd 升級至 1.6.25。使用 golang 1.22.1 重建 CNI 和 csi-proxy 。
1.28-2024.03.12	1.28.5	1.6.18	1.1.2	
1.28-2024.02.13	1.28.5	1.6.18	1.1.2	
1.28-2024.01.09	1.28.5	1.6.18	1.1.2	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.28-2023.12.12	1.28.3	1.6.18	1.1.2	
1.28-2023.11.14	1.28.3	1.6.18	1.1.2	包含 CVE-2023-5528 的修補程式。
1.28-2023.10.19	1.28.2	1.6.18	1.1.2	已將 containerd 升級至 1.6.18。已新增引導指令碼環境變數 (SERVICE_IPV4_CIDR 和 EXCLUDED_SNAT_CIDRS)。
1.28-2023-09.27	1.28.2	1.6.6	1.1.2	已修正 kubelet 的 安全性公告。
1.28-2023.09.12	1.28.1	1.6.6	1.1.2	

Amazon EKS 最佳化 Windows Server 2019 Core AMI

下表列出 Amazon EKS 最佳化 Windows Server 2019 Core AMI 的目前版本和先前版本。

Example

Kubernetes version 1.36

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.36-2026.08.15	1.36.2	2.2.3	1.2.1	
1.36-2026.07.18	1.36.2	2.2.3	1.2.1	
1.36-2026.06.18	1.36.2	2.2.3	1.2.1	
1.36-2026.05.19	1.36.1	2.2.3	1.2.1	

Kubernetes version 1.35

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.35-2026.08.15	1.35.6	2.1.7	1.2.1	
1.35-2026.07.18	1.35.6	2.1.7	1.2.1	
1.35-2026.06.18	1.35.6	2.1.7	1.2.1	
1.35-2026.05.19	1.35.5	2.1.7	1.2.1	
1.35-2026.04.18	1.35.3	2.1.6	1.2.1	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.35-2026.02.16	1.35.0	2.1.6	1.2.1	
1.35-2026-01-22	1.35.0	2.1.6	1.2.1	

Kubernetes version 1.34

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.34-2026.08.15	1.34.9	2.1.7	1.2.1	
1.34-2026.07.18	1.34.9	2.1.7	1.2.1	
1.34-2026.06.18	1.34.9	2.1.6	1.2.1	
1.34-2026.05.19	1.34.8	2.1.6	1.2.1	
1.34-2026.04.18	1.34.6	2.1.6	1.2.1	
1.34-2026.02.13	1.34.3	2.1.6	1.2.1	
1.34-2026.01.22	1.34.2	2.1.6	1.2.1	已將 containerd 升級至 2.1.6。
1.34-2025.12.15	1.34.2	2.1.4	1.2.1	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.34-2025.11.14	1.34.1	2.1.4	1.2.1	
1.34-2025.10.18	1.34.1	2.1.4	1.2.1	
1.34-2025.09.13	1.34.0	2.1.4	1.2.1	

Kubernetes version 1.33

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.33-2026.08.15	1.33.13	1.7.30	1.2.1	
1.33-2026.07.18	1.33.13	1.7.30	1.2.1	
1.33-2026.06.18	1.33.13	1.7.30	1.2.1	
1.33-2026.05.19	1.33.11	1.7.30	1.2.1	
1.33-2026.04.18	1.33.10	1.7.30	1.2.1	
1.33-2026.02.13	1.33.7	1.7.30	1.2.1	
1.33-2026.01.22	1.33.5	1.7.30	1.2.1	已將 containerd 升級至 1.7.30。

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.33-2025.12.15	1.33.5	1.7.28	1.2.1	
1.33-2025.11.14	1.33.5	1.7.28	1.2.1	
1.33-2025.10.18	1.33.5	1.7.28	1.2.1	
1.33-2025.09.13	1.33.4	1.7.28	1.2.1	將 GMSA 外掛程式日誌變更為 Windows Events
1.33-2025.08.18	1.33.3	1.7.27	1.2.1	
1.33-2025.07.16	1.33.1	1.7.27	1.2.1	
1.33-2025.06.13	1.33.1	1.7.27	1.2.1	
1.33-2025.05.17	1.33.1	1.7.27	1.2.1	

Kubernetes version 1.32

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.32-2026.08.15	1.32.13	1.7.30	1.2.1	
1.32-2026.07.18	1.32.13	1.7.30	1.2.1	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.32-2026.06.18	1.32.13	1.7.30	1.2.1	
1.32-2026.05.19	1.32.13	1.7.30	1.2.1	
1.32-2026.04.18	1.32.13	1.7.30	1.2.1	
1.32-2026.02.13	1.32.11	1.7.30	1.2.1	
1.32-2026.01.22	1.32.9	1.7.30	1.2.1	已將 containerd 升級至 1.7.30。
1.32-2025.12.15	1.32.9	1.7.28	1.2.1	
1.32-2025.11.14	1.32.9	1.7.28	1.2.1	
1.32-2025.10.18	1.32.9	1.7.28	1.2.1	
1.32-2025.09.13	1.32.8	1.7.28	1.2.1	將 GMSA 外掛程式日誌變更為 Windows Events
1.32-2025.08.18	1.32.7	1.7.27	1.2.1	
1.32-2025.07.16	1.32.5	1.7.27	1.2.1	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.32-2025.06.13	1.32.5	1.7.27	1.2.1	已將 containerd 升級至 1.7.27。
1.32-2025.05.17	1.32.5	1.7.20	1.2.1	
1.32-2025.04.14	1.32.1	1.7.20	1.1.3	
1.32-2025.03.14	1.32.1	1.7.20	1.1.3	
1.32-2025.02.18	1.32.1	1.7.20	1.1.3	
1.32-2025.01.15	1.32.4	1.7.20	1.1.3	

Kubernetes version 1.31

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.31-2026.08.15	1.31.14	1.7.30	1.2.1	
1.31-2026.07.18	1.31.14	1.7.30	1.2.1	
1.31-2026.06.18	1.31.14	1.7.30	1.2.1	
1.31-2026.05.19	1.31.14	1.7.30	1.2.1	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.31-2026.04.18	1.31.14	1.7.30	1.2.1	
1.31-2026.02.13	1.31.14	1.7.30	1.2.1	
1.31-2026.01.22	1.31.13	1.7.30	1.2.1	已將 containerd 升級至 1.7.30。
1.31-2025.12.15	1.31.13	1.7.28	1.2.1	
1.31-2025.11.14	1.31.13	1.7.28	1.2.1	
1.31-2025.10.18	1.31.13	1.7.28	1.2.1	
1.31-2025.09.13	1.31.12	1.7.28	1.2.1	將 GMSA 外掛程式日誌變更為 Windows Events
1.31-2025.08.18	1.31.11	1.7.27	1.2.1	
1.31-2025.07.16	1.31.9	1.7.27	1.2.1	
1.31-2025.06.13	1.31.9	1.7.27	1.2.1	已將 containerd 升級至 1.7.27。
1.31-2025.05.17	1.31.9	1.7.20	1.2.1	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.31-2025.04.14	1.31.5	1.7.20	1.1.3	
1.31-2025.03.14	1.31.5	1.7.20	1.1.3	
1.31-2025.02.15	1.31.5	1.7.20	1.1.3	
1.31-2025.01.15	1.31.4	1.7.20	1.1.3	
1.31-2025.01.01	1.31.4	1.7.20	1.1.3	包含 CVE-2024-9042 的修補程式。
1.31-2024.12.13	1.31.3	1.7.20	1.1.3	
1.31-2024.11.12	1.31.1	1.7.20	1.1.3	
1.31-2024.10.08	1.31.1	1.7.20	1.1.3	
1.31-2024.10.01	1.31.1	1.7.20	1.1.3	
1.31-2024.09.10	1.31.0	1.7.20	1.1.3	

Kubernetes version 1.30

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.30-2026.07.18	1.30.14	1.7.30	1.2.1	
1.30-2026.06.18	1.30.14	1.7.30	1.2.1	
1.30-2026.05.19	1.30.14	1.7.30	1.2.1	
1.30-2026.04.18	1.30.14	1.7.30	1.2.1	
1.30-2026.02.13	1.30.14	1.7.30	1.2.1	
1.30-2026.01.22	1.30.14	1.7.30	1.2.1	已將 containerd 升級至 1.7.30。
1.30-2025.12.15	1.30.14	1.7.28	1.2.1	
1.30-2025.11.21	1.30.14	1.7.28	1.2.1	
1.30-2025.10.18	1.30.14	1.7.28	1.2.1	
1.30-2025.09.13	1.30.14	1.7.28	1.2.1	將 GMSA 外掛程式日誌變更為 Windows Events
1.30-2025.08.18	1.30.14	1.7.27	1.2.1	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.30-2025.07.16	1.30.13	1.7.27	1.2.1	
1.30-2025.06.13	1.30.13	1.7.27	1.2.1	已將 containerd 升級至 1.7.27。
1.30-2025.05.17	1.30.13	1.7.20	1.2.1	
1.30-2025.04.14	1.30.9	1.7.20	1.1.3	
1.30-2025.03.14	1.30.9	1.7.20	1.1.3	已將 containerd 升級至 1.7.20。
1.30-2025-02-15	1.30.9	1.7.14	1.1.3	
1.30-2025.01.15	1.30.8	1.7.14	1.1.3	
1.30-2025.01.01	1.30.8	1.7.14	1.1.3	包含 CVE-2024-9042 的修補程式。
1.30-2024.12.11	1.30.7	1.7.14	1.1.3	
1.30-2024.11.12	1.30.4	1.7.14	1.1.3	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.30-2024.10.08	1.30.4	1.7.14	1.1.3	
1.30-2024.09.10	1.30.2	1.7.14	1.1.3	
1.30-2024.08.13	1.30.2	1.7.14	1.1.3	
1.30-2024.07.10	1.30.2	1.7.14	1.1.2	包含 CVE-2024-5321 的修補程式。
1.30-2024.06.17	1.30.0	1.7.14	1.1.2	已將 containerd 升級至 1.7.14。
1.30-2024.05.15	1.30.0	1.6.28	1.1.2	

Kubernetes version 1.29

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.29-2026.02.13	1.29.15	1.7.30	1.2.1	
1.29-2026.01.22	1.29.15	1.7.30	1.2.1	已將 containerd 升級至 1.7.30。
1.29-2025.12.15	1.29.15	1.7.28	1.2.1	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.29-2025.11.14	1.29.15	1.7.28	1.2.1	
1.29-2025.10.18	1.29.15	1.7.28	1.2.1	
1.29-2025.09.13	1.29.15	1.7.28	1.2.1	將 GMSA 外掛程式日誌變更為 Windows Events
1.29-2025.08.18	1.29.15	1.7.27	1.2.1	
1.29-2025.07.16	1.29.15	1.7.27	1.2.1	
1.29-2025.06.13	1.29.15	1.7.27	1.2.1	已將 containerd 升級至 1.7.27。
1.29-2025.05.17	1.29.15	1.7.20	1.2.1	
1.29-2025.04.14	1.29.13	1.7.20	1.1.3	
1.29-2025.03.14	1.29.13	1.7.20	1.1.3	已將 containerd 升級至 1.7.20。
1.29-2025.02.15	1.29.13	1.7.14	1.1.3	
1.29-2025.01.15	1.29.12	1.7.14	1.1.3	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.29-2025.01.01	1.29.12	1.7.14	1.1.3	包含 CVE-2024-9042 的修補程式。
1.29-2024.12.11	1.29.10	1.7.14	1.1.3	
1.29-2024.11.12	1.29.8	1.7.14	1.1.3	
1.29-2024.10.08	1.29.8	1.7.14	1.1.3	
1.29-2024.09.10	1.29.6	1.7.14	1.1.3	
1.29-2024.08.13	1.29.6	1.7.14	1.1.3	
1.29-2024.07.10	1.29.6	1.7.11	1.1.2	包含 CVE-2024-5321 的修補程式。
1.29-2024.06.17	1.29.3	1.7.11	1.1.2	
1.29-2024.05.15	1.29.3	1.7.11	1.1.2	已將 containerd 升級至 1.7.11。已將 kubelet 升級至 1.29.3。

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.29-2024.04.09	1.29.0	1.6.28	1.1.2	已將 containerd 升級至 1.6.28。使用 golang 1.22.1 重建 CNI 和 csi-proxy 。
1.29-2024.03.13	1.29.0	1.6.25	1.1.2	
1.29-2024.02.13	1.29.0	1.6.25	1.1.2	
1.29-2024.02.06	1.29.0	1.6.25	1.1.2	修正 kubelet 垃圾回收程序錯誤刪除暫停映像的錯誤。
1.29-2024.01.09	1.29.0	1.6.18	1.1.2	

Kubernetes version 1.28

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.28-2025.11.14	1.28.15	1.7.28	1.2.1	
1.28-2025.10.18	1.28.15	1.7.28	1.2.1	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.28-2025.09.13	1.28.15	1.7.28	1.2.1	將 GMSA 外掛程式日誌變更為 Windows Events
1.28-2025.08.18	1.28.15	1.7.27	1.2.1	
1.28-2025.07.16	1.28.15	1.7.27	1.2.1	
1.28-2025.06.13	1.28.15	1.7.20	1.2.1	已將 containerd 升級至 1.7.27。
1.28-2025.05.17	1.28.15	1.7.20	1.2.1	
1.28-2025.04.14	1.28.15	1.7.20	1.1.3	
1.28-2025.03.14	1.28.15	1.7.20	1.1.3	已將 containerd 升級至 1.7.20。
1.28-2025.02.15	1.28.15	1.7.14	1.1.3	
1.28-2025-01-15	1.28.15	1.7.14	1.1.3	
1.28-2025-01-01	1.28.15	1.7.14	1.1.3	包含 CVE-2024-9042 的修補程式。

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.28-2024.12.11	1.28.15	1.7.14	1.1.3	
1.28-2024.11.12	1.28.13	1.7.14	1.1.3	
1.28-2024.10.08	1.28.13	1.7.14	1.1.3	
1.28-2024.09.10	1.28.11	1.7.14	1.1.3	
1.28-2024.08.13	1.28.11	1.7.14	1.1.3	
1.28-2024.07.10	1.28.11	1.7.11	1.1.2	包含 CVE-2024-5321 的修補程式。
1.28-2024.06.17	1.28.8	1.7.11	1.1.2	已將 containerd 升級至 1.7.11。
1.28-2024.05.14	1.28.8	1.6.28	1.1.2	已將 containerd 升級至 1.6.28。已將 kubelet 升級至 1.28.8。

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.28-2024.04.09	1.28.5	1.6.25	1.1.2	已將 containerd 升級至 1.6.25。使用 golang 1.22.1 重建 CNI 和 csi-proxy 。
1.28-2024.03.13	1.28.5	1.6.18	1.1.2	
1.28-2024.02.13	1.28.5	1.6.18	1.1.2	
1.28-2024.01.09	1.28.5	1.6.18	1.1.2	
1.28-2023.12.12	1.28.3	1.6.18	1.1.2	
1.28-2023.11.14	1.28.3	1.6.18	1.1.2	包含 CVE-2023-5528 的修補程式。

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.28-2023.10.19	1.28.2	1.6.18	1.1.2	已將 containerd 升級至 1.6.18。已新增引導指令碼環境變數 (SERVICE_IPV4_CIDR 和 EXCLUDED_SNAT_CIDRS)。
1.28-2023-09.27	1.28.2	1.6.6	1.1.2	已修正 kubelet 的 安全性公告。
1.28-2023.09.12	1.28.1	1.6.6	1.1.2	

Amazon EKS 最佳化 Windows Server 2019 Full AMI

下表列出 Amazon EKS 最佳化 Windows Server 2019 Full AMI 的目前版本和先前版本。

Example

Kubernetes version 1.36

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.36-2026.08.15	1.36.2	2.2.3	1.2.1	
1.36-2026.07.18	1.36.2	2.2.3	1.2.1	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.36-2026.06.18	1.36.2	2.2.3	1.2.1	
1.36-2026.05.19	1.36.1	2.2.3	1.2.1	

Kubernetes version 1.35

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.35-2026.08.15	1.35.6	2.1.7	1.2.1	
1.35-2026.07.18	1.35.6	2.1.7	1.2.1	
1.35-2026.06.18	1.35.6	2.1.7	1.2.1	
1.35-2026.05.19	1.35.5	2.1.7	1.2.1	
1.35-2026.04.18	1.35.3	2.1.6	1.2.1	
1.35-2026.02.16	1.35.0	2.1.6	1.2.1	
1.35-2026-01-22	1.35.0	2.1.6	1.2.1	

Kubernetes version 1.34

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.34-2026.08.15	1.34.9	2.1.7	1.2.1	
1.34-2026.07.21	1.34.9	2.1.7	1.2.1	
1.34-2026.06.18	1.34.9	2.1.6	1.2.1	
1.34-2026.05.19	1.34.8	2.1.6	1.2.1	
1.34-2026.04.18	1.34.6	2.1.6	1.2.1	
1.34-2026.02.13	1.34.3	2.1.6	1.2.1	
1.34-2026.01.22	1.34.2	2.1.6	1.2.1	已將 containerd 升級至 2.1.6。
1.34-2025.12.15	1.34.2	2.1.4	1.2.1	
1.34-2025.11.14	1.34.1	2.1.4	1.2.1	
1.34-2025.10.18	1.34.1	2.1.4	1.2.1	
1.34-2025.09.13	1.34.0	2.1.4	1.2.1	

Kubernetes version 1.33

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.33-2026.08.15	1.33.13	1.7.30	1.2.1	
1.33-2026.07.18	1.33.13	1.7.30	1.2.1	
1.33-2026.06.18	1.33.13	1.7.30	1.2.1	
1.33-2026.05.19	1.33.11	1.7.30	1.2.1	
1.33-2026.04.18	1.33.10	1.7.30	1.2.1	
1.33-2026.02.13	1.33.7	1.7.30	1.2.1	
1.33-2026.01.22	1.33.5	1.7.30	1.2.1	已將 containerd 升級至 1.7.30。
1.33-2025.12.15	1.33.5	1.7.28	1.2.1	
1.33-2025.11.14	1.33.5	1.7.28	1.2.1	
1.33-2025.10.18	1.33.5	1.7.28	1.2.1	
1.33-2025.09.13	1.33.4	1.7.28	1.2.1	將 GMSA 外掛程式日誌變更為 Windows Events

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.33-2025.08.18	1.33.3	1.7.27	1.2.1	
1.33-2025.07.16	1.33.1	1.7.27	1.2.1	
1.33-2025.06.13	1.33.1	1.7.27	1.2.1	
1.33-2025.05.17	1.33.1	1.7.27	1.2.1	

Kubernetes version 1.32

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.32-2026.08.15	1.32.13	1.7.30	1.2.1	
1.32-2026.07.18	1.32.13	1.7.30	1.2.1	
1.32-2026.06.18	1.32.13	1.7.30	1.2.1	
1.32-2026.05.19	1.32.13	1.7.30	1.2.1	
1.32-2026.04.18	1.32.13	1.7.30	1.2.1	
1.32-2026.02.13	1.32.11	1.7.30	1.2.1	
1.32-2026.01.22	1.32.9	1.7.30	1.2.1	已將 container

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
				d 升級至 1.7.30。
1.32-2025.12.15	1.32.9	1.7.28	1.2.1	
1.32-2025.11.14	1.32.9	1.7.28	1.2.1	
1.32-2025.10.18	1.32.9	1.7.28	1.2.1	
1.32-2025.09.13	1.32.8	1.7.28	1.2.1	將 GMSA 外掛程式日誌變更為 Windows Events
1.32-2025.08.18	1.32.7	1.7.27	1.2.1	
1.32-2025.07.16	1.32.5	1.7.27	1.2.1	
1.32-2025.06.13	1.32.5	1.7.27	1.2.1	已將 containerd 升級至 1.7.27。
1.32-2025.05.17	1.32.5	1.7.20	1.2.1	
1.32-2025.04.14	1.32.1	1.7.20	1.1.3	
1.32-2025.03.14	1.32.1	1.7.20	1.1.3	
1.32-2025.02.18	1.32.1	1.7.20	1.1.3	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.32-2025.01.15	1.32.0	1.7.20	1.1.3	

Kubernetes version 1.31

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.31-2026.08.15	1.31.14	1.7.30	1.2.1	
1.31-2026.07.18	1.31.14	1.7.30	1.2.1	
1.31-2026.06.18	1.31.14	1.7.30	1.2.1	
1.31-2026.05.19	1.31.14	1.7.30	1.2.1	
1.31-2026.04.18	1.31.14	1.7.30	1.2.1	
1.31-2026.02.13	1.31.14	1.7.30	1.2.1	
1.31-2026.01.22	1.31.13	1.7.30	1.2.1	已將 containerd 升級至 1.7.30。
1.31-2025.12.15	1.31.13	1.7.28	1.2.1	
1.31-2025.11.14	1.31.13	1.7.28	1.2.1	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.31-2025.10.18	1.31.13	1.7.28	1.2.1	
1.31-2025.09.13	1.31.12	1.7.28	1.2.1	將 GMSA 外掛程式日誌變更為 Windows Events
1.31-2025.08.18	1.31.11	1.7.27	1.2.1	
1.31-2025.07.16	1.31.9	1.7.27	1.2.1	
1.31-2025.06.13	1.31.9	1.7.27	1.2.1	已將 containerd 升級至 1.7.27。
1.31-2025.05.17	1.31.9	1.7.20	1.2.1	
1.31-2025.04.14	1.31.5	1.7.20	1.1.3	
1.31-2025.03.14	1.31.5	1.7.20	1.1.3	
1.31-2025.02.15	1.31.5	1.7.20	1.1.3	
1.31-2025.01.15	1.31.4	1.7.20	1.1.3	
1.31-2025.01.01	1.31.4	1.7.20	1.1.3	包含 CVE-2024-9042 的修補程式。

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.31-2024.12.13	1.31.3	1.7.20	1.1.3	
1.31-2024.11.12	1.31.1	1.7.20	1.1.3	
1.31-2024.10.08	1.31.1	1.7.20	1.1.3	
1.31-2024.10.01	1.31.1	1.7.20	1.1.3	
1.31-2024.09.10	1.31.0	1.7.20	1.1.3	

Kubernetes version 1.30

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.30-2026.07.18	1.30.14	1.7.30	1.2.1	
1.30-2026.06.18	1.30.14	1.7.30	1.2.1	
1.30-2026.05.19	1.30.14	1.7.30	1.2.1	
1.30-2026.04.18	1.30.14	1.7.30	1.2.1	
1.30-2026.02.13	1.30.14	1.7.30	1.2.1	
1.30-2026.01.22	1.30.14	1.7.30	1.2.1	已將 container

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
				d 升級至 1.7.30。
1.30-2025.12.15	1.30.14	1.7.28	1.2.1	
1.30-2025.11.21	1.30.14	1.7.28	1.2.1	
1.30-2025.10.18	1.30.14	1.7.28	1.2.1	
1.30-2025.09.13	1.30.14	1.7.28	1.2.1	將 GMSA 外掛程式日誌變更為 Windows Events
1.30-2025.08.18	1.30.14	1.7.27	1.2.1	
1.30-2025.07.16	1.30.13	1.7.27	1.2.1	
1.30-2025.06.13	1.30.13	1.7.27	1.2.1	已將 containerd 升級至 1.7.27。
1.30-2025.05.17	1.30.13	1.7.20	1.2.1	
1.30-2025.04.14	1.30.9	1.7.20	1.1.3	
1.30-2025.03.14	1.30.9	1.7.20	1.1.3	已將 containerd 升級至 1.7.20。

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.30-2025.02.15	1.30.9	1.7.14	1.1.3	
1.30-2025.01.15	1.30.8	1.7.14	1.1.3	
1.30-2025.01.01	1.30.8	1.7.14	1.1.3	包含 CVE-2024-9042 的修補程式。
1.30-2024.12.11	1.30.7	1.7.14	1.1.3	
1.30-2024.11.12	1.30.4	1.7.14	1.1.3	
1.30-2024.10.08	1.30.4	1.7.14	1.1.3	
1.30-2024.09.10	1.30.2	1.7.14	1.1.3	
1.30-2024.08.13	1.30.2	1.7.14	1.1.3	
1.30-2024.07.10	1.30.2	1.7.14	1.1.2	包含 CVE-2024-5321 的修補程式。
1.30-2024.06.17	1.30.0	1.7.14	1.1.2	已將 containerd 升級至 1.7.14。

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.30-2024.05.15	1.30.0	1.6.28	1.1.2	

Kubernetes version 1.29

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.29-2026.02.13	1.29.15	1.7.30	1.2.1	
1.29-2026.01.22	1.29.15	1.7.30	1.2.1	已將 containerd 升級至 1.7.30。
1.29-2025.12.15	1.29.15	1.7.28	1.2.1	
1.29-2025.11.14	1.29.15	1.7.28	1.2.1	
1.29-2025.10.18	1.29.15	1.7.28	1.2.1	
1.29-2025.09.13	1.29.15	1.7.28	1.2.1	將 GMSA 外掛程式日誌變更為 Windows Events
1.29-2025.08.18	1.29.15	1.7.27	1.2.1	
1.29-2025.07.16	1.29.15	1.7.27	1.2.1	
1.29-2025.06.13	1.29.15	1.7.27	1.2.1	已將 container

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
				d 升級至 1.7.27。
1.29-2025.05.17	1.29.15	1.7.20	1.2.1	
1.29-2025.04.14	1.29.13	1.7.20	1.1.3	
1.29-2025.03.14	1.29.13	1.7.20	1.1.3	已將 containerd 升級至 1.7.20。
1.29-2025.02.15	1.29.13	1.7.14	1.1.3	
1.29-2025.01.15	1.29.12	1.7.14	1.1.3	
1.29-2025.01.01	1.29.12	1.7.14	1.1.3	包含 CVE-2024-9042 的修補程式。
1.29-2024.12.11	1.29.10	1.7.14	1.1.3	
1.29-2024.11.12	1.29.8	1.7.14	1.1.3	
1.29-2024.10.08	1.29.8	1.7.14	1.1.3	
1.29-2024.09.10	1.29.6	1.7.14	1.1.3	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.29-2024.08.13	1.29.6	1.7.14	1.1.3	
1.29-2024.07.10	1.29.6	1.7.11	1.1.2	包含 CVE-2024-5321 的修補程式。
1.29-2024.06.17	1.29.3	1.7.11	1.1.2	
1.29-2024.05.15	1.29.3	1.7.11	1.1.2	已將 containerd 升級至 1.7.11。已將 kubelet 升級至 1.29.3。
1.29-2024.04.09	1.29.0	1.6.28	1.1.2	已將 containerd 升級至 1.6.28。使用 golang 1.22.1 重建 CNI 和 csi-proxy 。
1.29-2024.03.13	1.29.0	1.6.25	1.1.2	
1.29-2024.02.13	1.29.0	1.6.25	1.1.2	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.29-2024.02.06	1.29.0	1.6.25	1.1.2	修正 kubelet 垃圾回收程序錯誤刪除暫停映像的錯誤。
1.29-2024.01.09	1.29.0	1.6.18	1.1.2	

Kubernetes version 1.28

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.28-2025.11.14	1.28.15	1.7.28	1.2.1	
1.28-2025.10.18	1.28.15	1.7.28	1.2.1	
1.28-2025.09.13	1.28.15	1.7.28	1.2.1	將 GMSA 外掛程式日誌變更為 Windows Events
1.28-2025.08.18	1.28.15	1.7.27	1.2.1	
1.28-2025.07.16	1.28.15	1.7.27	1.2.1	
1.28-2025.06.13	1.28.15	1.7.27	1.2.1	已將 containerd 升級至 1.7.27。
1.28-2025.05.17	1.28.15	1.7.20	1.2.1	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.28-2025.04.14	1.28.15	1.7.20	1.1.3	
1.28-2025.03.14	1.28.15	1.7.20	1.1.3	已將 containerd 升級至 1.7.20。
1.28-2025.02.15	1.28.15	1.7.14	1.1.3	
1.28-2025-01-15	1.28.15	1.7.14	1.1.3	
1.28-2025-01-01	1.28.15	1.7.14	1.1.3	包含 CVE-2024-9042 的修補程式。
1.28-2024.12.11	1.28.15	1.7.14	1.1.3	
1.28-2024.11.12	1.28.13	1.7.14	1.1.3	
1.28-2024.10.08	1.28.13	1.7.14	1.1.3	
1.28-2024.09.10	1.28.11	1.7.14	1.1.3	
1.28-2024.08.13	1.28.11	1.7.14	1.1.3	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.28-2024.07.10	1.28.11	1.7.11	1.1.2	包含 CVE-2024-5321 的修補程式。
1.28-2024.06.17	1.28.8	1.7.11	1.1.2	已將 containerd 升級至 1.7.11。
1.28-2024.05.14	1.28.8	1.6.28	1.1.2	已將 containerd 升級至 1.6.28。已將 kubelet 升級至 1.28.8。
1.28-2024.04.09	1.28.5	1.6.25	1.1.2	已將 containerd 升級至 1.6.25。使用 golang 1.22.1 重建 CNI 和 csi-proxy 。
1.28-2024.03.13	1.28.5	1.6.18	1.1.2	
1.28-2024.02.13	1.28.5	1.6.18	1.1.2	
1.28-2024.01.09	1.28.5	1.6.18	1.1.2	

AMI 版本	kubelet 版本	containerd 版本	csi-proxy 版本	版本備註
1.28-2023.12.12	1.28.3	1.6.18	1.1.2	
1.28-2023.11.14	1.28.3	1.6.18	1.1.2	包含 CVE-2023-5528 的修補程式。
1.28-2023.10.19	1.28.2	1.6.18	1.1.2	已將 containerd 升級至 1.6.18。已新增引導指令碼環境變數 (SERVICE_IPV4_CIDR 和 EXCLUDED_SNAT_CIDRS)。
1.28-2023-09.27	1.28.2	1.6.6	1.1.2	已修正 kubelet 的 安全性公告。
1.28-2023.09.12	1.28.1	1.6.6	1.1.2	

擷取建議的 Microsoft Windows AMI ID

您在部署節點時，可針對預先建置的 Amazon EKS 最佳化 Amazon Machine Image (AMI) 指定 ID。若要擷取符合您所需組態的 AMI ID，請查詢 AWS Systems Manager 參數存放區 API。使用此 API 無需手動查詢 Amazon EKS 最佳化的 AMI ID。如需詳細資訊，請參閱 GetParameter。您使用的 IAM 主體必須擁有 `ssm:GetParameter` IAM 許可，才能擷取 Amazon EKS 最佳化 AMI 中繼資料。

您可透過以下命令擷取最新建議的 Amazon EKS 最佳化 Windows AMI 的映像檔 ID，這會使用子參數 `image_id`。視需要對命令進行下列修改，然後執行修改後的命令：

- 使用以下其中一個選項來取代*####*。
  - 針對 Windows Server *2025* 使用 2025。
  - 若是 Windows Server *2022*，則使用 2022。
  - 若是 Windows Server *2019*，則使用 2019。
- 使用以下其中一個選項來取代 *installation-option*。如需詳細資訊，請參閱 Windows Server 中有哪些 Server Core 安裝選項。
  - 使用 *Core*，實現最少安裝以減少攻擊面。
  - 使用 *Full*，可包括 Windows 桌面體驗。
- 使用支援的平台版本來取代 *kubernetes-version*。
- 將 *region-code* 取代為您想要 AMI ID 的 Amazon EKS 支援 AWS 區域。

```
aws ssm get-parameter --name /aws/service/ami-windows-latest/Windows_Server-release-
English-installation-option-EKS_Optimized-kubernetes-version/image_id \
    --region region-code --query "Parameter.Value" --output text
```

下面介紹了取代預留位置之後的命令範例。

```
aws ssm get-parameter --name /aws/service/ami-windows-latest/Windows_Server-2022-
English-Core-EKS_Optimized-k8s-n-2/image_id \
    --region us-west-2 --query "Parameter.Value" --output text
```

範例輸出如下。

```
ami-1234567890abcdef0
```

使用 Image Builder 建置自訂 Windows AMI

您可以透過下列其中一種選項，使用 EC2 Image Builder 來建立自訂 Amazon EKS 最佳化 Windows AMI：

- 將 Amazon EKS 最佳化 Windows AMI 用作基礎
- 使用 Amazon 受管建置元件

兩種方法都需要您建立自己的 Image Builder 配方。如需詳細資訊，請參閱《Image Builder 使用者指南》中的建立映像配方的新版本。

> **Important**
>
> 下列 `eks` 專用的 Amazon 受管元件包含 `CVE-2024-5321` 的修補程式。
>
> - `1.28.2` 和更高版本
> - `1.29.2` 和更高版本
> - `1.30.1` 和更高版本
> - Kubernetes 1.31 及更高版本的全部版本

將 Amazon EKS 最佳化 Windows AMI 用作基礎

此選項是建立自訂 Windows AMI 的建議方式。相較於 Amazon 受管建置元件，我們提供的 Amazon EKS 最佳化 Windows AMI 會更頻繁地更新。

1. 啟動新的 Image Builder 配方。
   a. 在 https：//https://console.aws.amazon.com/imagebuilder 開啟 EC2 Image Builder 主控台。
   b. 在左側導覽窗格中，選擇映像配方。
   c. 選擇建立映像配方。
2. 在配方詳細資訊區段中，輸入名稱和版本。
3. 在基礎映像區段中指定 Amazon EKS 最佳化 Windows AMI 的 ID。
   a. 選擇輸入自訂 AMI ID。
   b. 擷取所需的 Windows 作業系統版本 AMI ID。如需詳細資訊，請參閱the section called “獲取最新 ID”。
   c. 輸入自訂 AMI ID。如果找不到 AMI ID，請確定 AMI ID AWS 的區域與主控台右上角顯示 AWS 的區域相符。
4. (選用) 若要取得最新的安全性更新，請在建置元件：區段中新增 `update-windows` 元件。
   a. 從依名稱尋找元件搜尋方塊右側的下拉式清單中，選擇 Amazon 受管。
   b. 在依名稱尋找元件搜尋方塊中，輸入 `update-windows`。
   c. 選取 **`update-windows`** 搜尋結果的核取方塊。此元件包含作業系統的最新 Windows 修補程式。
5. 完成具備您所需之組態的剩餘映像配方輸入。如需詳細資訊，請參閱《Image Builder 使用者指南》中的建立映像配方的新版本 (主控台)。
6. 選擇建立配方。

7. 在新的或現有的映像管道中使用新的映像配方。映像管道成功執行後，您的自訂 AMI 將被列為輸出映像並可供使用。如需詳細資訊，請參閱使用 EC2 Image Builder 主控台精靈建立映像管道。

使用 Amazon 受管建置元件

當無法將 Amazon EKS 最佳化 Windows AMI 用作基礎時，您可以改用 Amazon 受管建置元件。在最新支援的 Kubernetes 版本中，此選項可能會延遲。

1. 啟動新的 Image Builder 配方。
   a. 在 https：//https://console.aws.amazon.com/imagebuilder 開啟 EC2 Image Builder 主控台。
   b. 在左側導覽窗格中，選擇映像配方。
   c. 選擇建立映像配方。
2. 在配方詳細資訊區段中，輸入名稱和版本。
3. 在基礎映像區段中，決定您將用來建立自訂 AMI 的選項：
   - 選取受管映像：對於您的映像作業系統 (OS) 選擇 Windows。然後對於映像來源，選擇下列其中一個選項：
     - Quick start (Amazon 受管) – 在映像名稱下拉式清單中，選取 Amazon EKS 支援的 Windows Server 版本。如需詳細資訊，請參閱the section called “Windows”。
     - 我擁有的映像：對於映像名稱，請使用您自己的授權選擇您自己的映像 ARN。您提供的映像尚未安裝 Amazon EKS 元件。
   - 輸入自訂 AMI ID：對於 AMI ID，請使用您自己的授權輸入 AMI 的 ID。您提供的映像尚未安裝 Amazon EKS 元件。
4. 在建置元件 – 視窗區段中，執行下列操作：
   a. 從依名稱尋找元件搜尋方塊右側的下拉式清單中，選擇 Amazon 受管。
   b. 在依名稱尋找元件搜尋方塊中，輸入 `eks`。
   c. 選取 **`eks-optimized-ami-windows`** 搜尋結果的核取方塊，即使傳回的結果可能不是您想要的版本。
   d. 在依名稱尋找元件搜尋方塊中，輸入 `update-windows`。
   e. 選取 update-windows 搜尋結果的核取方塊。此元件包含作業系統的最新 Windows 修補程式。
5. 在選取元件區段中，執行下列操作：
   a. 選擇 **`eks-optimized-ami-windows`** 的版本控制選項。
   b. 選擇指定元件版本。

c. 在元件版本欄位中，輸入 *version.x*，以支援 Kubernetes 的版本取代*##*。輸入版本號碼片段的 *x*，即表示使用最新的元件版本，其亦符合您明確定義的版本片段。請注意控制台輸出，因為它會告知您所需的版本是否可作為受管元件使用。請記住，最新 Kubernetes 版本可能不適用於建置元件。如需有關可用版本的詳細資訊，請參閱 the section called “擷取 eks-optimized-ami-windows 元件版本的相關資訊”。

6. 完成具備您所需之組態的剩餘映像配方輸入。如需詳細資訊，請參閱《Image Builder 使用者指南》中的建立映像配方的新版本 (主控台)。
7. 選擇建立配方。
8. 在新的或現有的映像管道中使用新的映像配方。映像管道成功執行後，您的自訂 AMI 將被列為輸出映像並可供使用。如需詳細資訊，請參閱使用 EC2 Image Builder 主控台精靈建立映像管道。

擷取 **eks-optimized-ami-windows** 元件版本的相關資訊

您可以擷取與每個元件安裝之內容相關的特定資訊。例如，您可以確認安裝的 kubelet 版本。這些元件會在 Amazon EKS 支援的 Windows 作業系統版本上進行功能測試。如需詳細資訊，請參閱the section called “版本發佈日曆”。任何其他未列為支援之 Windows 作業系統版本，或已達到終止支援者可能與元件不相容。

1. 在 https：//https://console.aws.amazon.com/imagebuilder 開啟 EC2 Image Builder 主控台。
2. 在左側導覽窗格中選擇元件。
3. 從依名稱尋找元件搜尋方塊右側的下拉式清單中，將我擁有的變更為 Quick start (Amazon 受管)。
4. 在 Find components by name (依名稱尋找元件) 方塊中，輸入 eks。
5. (選用) 如果您使用的是最新版本，請選擇兩次，將版本欄以遞減順序排序。
6. 選擇具有所需版本的 **eks-optimized-ami-windows** 連結。

結果頁面中的描述會顯示特定資訊。

偵測節點運作狀態問題並啟用自動節點修復

節點運作狀態是指 Kubernetes 節點有效執行工作負載的操作狀態和功能。運作狀態良好的節點會維持預期的網路連線能力、有足夠的運算和儲存資源，而且可以成功執行工作負載而不會中斷。

為了協助在 EKS 叢集中維護正常運作的節點，EKS 提供節點監控代理程式和自動節點修復。這些功能會透過 EKS Auto Mode 運算自動啟用。您也可以搭配 EKS 受管節點群組和 Karpenter 使用自動節點

修復，也可以搭配 AWS Fargate 以外的任何 EKS 運算類型使用 EKS 節點監控代理程式。EKS 節點監控代理程式和自動節點修復在一起使用時最有效，但也可以在 EKS 叢集中個別使用。

> **Important**
>
> 節點監控代理程式與節點自動修復功能僅適用於 Linux。這些功能不適用於 Windows。

節點監控代理程式

EKS 節點監控代理程式會讀取節點日誌，以偵測運作狀態問題。它會剖析日誌，以偵測故障並顯示節點運作狀態的相關資訊。對於偵測到的每個問題類別，代理程式會將專用 `NodeCondition` 套用至工作者節點。如需 EKS 節點監控代理程式偵測到的節點運作狀態問題詳細資訊，請參閱 the section called “節點運作狀態偵測”。

EKS Auto Mode 運算包含節點監控代理程式。對於其他 EKS 運算類型，您可以將節點監控代理程式新增為 EKS 附加元件，也可以使用 Helm 等 Kubernetes 工具進行管理。如需詳細資訊，請參閱the section called “設定節點監控代理程式”。

使用 EKS 節點監控代理程式時，下列類別的節點運作狀態問題會呈現為節點條件。請注意，`Ready`、`DiskPressure`和 `MemoryPressure`是標準 Kubernetes 節點條件，即使沒有 EKS 節點監控代理程式也會呈現。

節點條件	Description
AcceleratedHardwareReady	AcceleratedHardwareReady 指出節點上的加速硬體 (GPU、Neuron) 是否正常運作。
ContainerRuntimeReady	ContainerRuntimeReady 指出容器執行時間（容器等）是否正常運作且能夠執行容器。
DiskPressure	DiskPressure 是標準 Kubernetes 條件，表示節點遇到磁碟壓力（磁碟空間不足或 I/O 高）。
KernelReady	KernelReady 指出核心是否正常運作，而不會發生嚴重錯誤、恐慌或資源耗盡。

節點條件	Description
MemoryPressure	MemoryPressure 是標準 Kubernetes 條件，表示節點正經歷記憶體壓力（可用的記憶體不足）。
NetworkingReady	NetworkingReady 指出節點的網路堆疊是否正常運作（介面、路由、連線能力）。
StorageReady	StorageReady 指出節點的儲存子系統是否正常運作（磁碟、檔案系統、I/O)。
備妥	Ready 是標準 Kubernetes 條件，表示節點運作狀態良好且已準備好接受 Pod。

自動節點修復

EKS 自動節點修復會持續監控節點運作狀態、對偵測到的問題做出反應，並盡可能取代或重新啟動節點。這可在最少手動介入的情況下改善叢集可靠性，並有助於減少應用程式停機時間。

EKS 自動節點修復本身會回應 kubelet `Ready`的條件、任何手動刪除的節點物件，以及無法加入叢集的 EKS 受管節點群組執行個體。在安裝節點監控代理程式的情況下啟用 EKS 自動節點修復時，EKS 自動節點修復會回應其他節點條件：`AcceleratedHardwareReady`、`ContainerRuntimeReady`、`KernelReady`、`NetworkingReady`和 `StorageReady`。

EKS 自動節點修復不會回應標準 Kubernetes `DiskPressure`、`MemoryPressure` 或 `PIDPressure`節點條件。這些條件通常表示應用程式行為、工作負載組態或資源限制的問題，而不是節點層級失敗，因此難以判斷適當的預設修復動作。在這些情況下，工作負載會受到 Kubernetes 節點壓力移出行為的約束。

如需 EKS 自動節點修復的詳細資訊，請參閱 the section called “自動節點修復”。

使用 EKS 節點監控代理程式偵測節點運作狀態問題

本主題詳細說明 EKS 節點監控代理程式偵測到的節點運作狀態問題、這些問題如何呈現為節點條件或事件，以及如何設定節點監控代理程式。

EKS 節點監控代理程式可以搭配或不搭配 EKS 自動節點修復使用。如需 EKS 自動節點修復的詳細資訊，請參閱 the section called “自動節點修復”。

EKS 節點監控代理程式的原始碼會發佈在 aws/eks-node-monitoring-agent 儲存庫的 GitHub 上。

節點運作狀態問題

下表中列出了節點監控代理程式能夠偵測的節點運作狀態問題。有兩種類型的問題：

- 條件 – 必須執行修復動作的終端問題，例如，執行個體取代或重新啟動。啟用自動修復後，Amazon EKS 將執行修復動作，取代節點或重新啟動。如需詳細資訊，請參閱the section called “節點條件”。
- 事件 – 暫時性問題或次佳節點組態。無須執行自動修復動作。如需詳細資訊，請參閱the section called “事件節點”。

AcceleratedHardware 節點運作狀態問題

針對下表中所列嚴重性為「條件」的問題，監控條件為 `AcceleratedHardwareReady`。下表中的事件和條件適用於 NVIDIA 和 Neuron 相關的節點運作狀態問題。

名稱	嚴重性	說明	修復動作
DCGMDiagnosticFailure	條件	DCGM 作用中診斷測試套件的測試案例失敗。	無
DCGMError	條件	DCGM 主機程序的連線遺失或無法建立。	無
DCGMFieldError【代碼】	事件	DCGM 透過欄位識別符偵測到 GPU 降級。	無
DCGMHealthCode【Code】	事件	DCGM 運作狀態檢查以非嚴重方式失敗。	無
DCGMHealthCode【Code】	條件	DCGM 運作狀態檢查嚴重失敗。	無
NeuronDMAError	條件	DMA 引擎遇到無法復原的錯誤。	取代

名稱	嚴重性	說明	修復動作
NeuronHBMUncorrectableError	條件	HBM 遇到無法修正的錯誤，且產生了錯誤結果。	取代
NeuronNCUncorrectableError	條件	偵測到 Neuron Core 無法修正的記憶體錯誤。	取代
NeuronSRAMUncorrectableError	條件	晶片上 SRAM 遇到同位錯誤，且產生了錯誤結果。	取代
NvidiaDeviceCountMismatch	事件	透過 NVML 可見的 GPUs 數量與檔案系統上的 NVIDIA 裝置計數不一致。	無
NvidiaDoubleBitError	條件	GPU 驅動程式產生了雙位元錯誤。	取代
NvidiaNCCLError	事件	NVIDIA Collective Communications 程式庫 () 中發生分段錯誤`libnccl`。	無
NvidiaNVLinkError	條件	GPU 驅動程式報告了 NVLink 錯誤。	取代
NvidiaPCIeError	事件	觸發 PCIe 重播以從傳輸錯誤中復原。	無
NvidiaPageRetirement	事件	GPU 驅動程式已標記記憶體頁面進行淘汰。若在同一位址遇到單一雙位元錯誤或兩個單一位元錯誤，可能會發生此情況。	無

名稱	嚴重性	說明	修復動作
NvidiaPowerError	事件	GPUs 的用電量超過允許的閾值。	無
NvidiaThermalError	事件	GPUs 的熱狀態超過允許的閾值。	無
NvidiaXID【Code】Error	條件	發生嚴重 GPU 錯誤。	取代或重新啟動
NvidiaXID[Code]Warning	事件	發生非關鍵 GPU 錯誤。	無

NVIDIA XID 錯誤代碼

節點監控代理程式會從 GPU 核心日誌偵測 NVIDIA XID 錯誤。XID 錯誤分為兩個類別：

- 眾所周知的 XID 代碼 – 設定節點條件 (`AcceleratedHardwareReady=False`) 並在啟用時觸發自動修復的嚴重錯誤。原因碼格式為 `NvidiaXID[Code]Error`。EKS 節點監控代理程式偵測到的已知 XID 代碼可能不代表需要修復動作的 NVIDIA XID 代碼完整清單。
- 未知 XID 代碼 – 僅記錄為 Kubernetes 事件。這些不會觸發自動修復。原因碼格式為 `NvidiaXID[Code]Warning`。若要調查未知的 XID 錯誤，請使用 檢閱核心日誌`dmesg | grep -i nvrm`。

若要了解 XID 錯誤的詳細資訊，請參閱 NVIDIA GPU 部署與管理文件中的 Xid 錯誤。若要了解個別 XID 訊息的詳細資訊，請參閱 NVIDIA GPU 部署與管理文件中的了解 Xid 訊息。

下表列出已知的 XID 代碼、其意義，以及啟用的預設節點修復動作。

XID 代碼	說明	修復動作
46	GPU 已停止處理 – GPU 因內部逾時而停止處理，且需要重設 GPU 才能復原。	重新啟動
48	雙位元 ECC 錯誤 – GPU 記憶體中發生無法修正的雙位元錯誤，表示潛在的硬體降級。	重新啟動

XID 代碼	說明	修復動作
54	未連接輔助電源 – 輔助電源未連接到 GPU 電路板，通常表示電源連接器未正確就位。	重新啟動
62	內部微型控制器停止 – GPU 的內部微型控制器停止，表示需要重設 GPU 的韌體或硬體故障。	重新啟動
63	GPU 記憶體重新映射事件 – GPU 驅動程式因為偵測到錯誤而重新映射一部分 GPU 記憶體。這通常可以復原。	重新啟動
64	GPU 記憶體重新映射失敗 – GPU 無法重新映射故障的記憶體，表示硬體問題。	取代
74	NVLink 錯誤 – GPUs 之間的高速 NVLink 互連發生錯誤。	取代
79	GPU 已脫離匯流排 – 無法再透過 PCIe 存取 GPU，通常表示硬體故障或電源問題。	取代
95	未包含的記憶體錯誤 – 發生記憶體錯誤，可能已影響其他應用程式或系統記憶體。	重新啟動
109	內容切換逾時 – GPU 內容切換未及時完成，表示 GPU 已停止且需要重設。	重新啟動
110	安全錯誤 – 在 GPU 上偵測到安全錯誤，這需要 GPU 重設，如果持續存在，則需要硬體調查。	重新啟動
119	GSP RPC 逾時 – 與 GPU 系統處理器的通訊逾時，可能是因為韌體問題。	取代
120	GSP 錯誤 – GPU 系統處理器發生錯誤。	取代
136	連結訓練失敗 – GPU 無法在連結訓練期間建立正常運作的 NVLink 連線。	重新啟動
140	ECC 未復原錯誤 – ECC 錯誤逸出遏制並可能已損毀資料。	重新啟動
142	NVENC3 錯誤 – NVENC3 視訊編碼器硬體失敗，沒有可用的自動復原。適用於 GB200。	取代

XID 代碼	說明	修復動作
143	GPU 初始化錯誤 – GPU 無法正確初始化。	重新啟動
151	金鑰輪換錯誤 – 機密運算金鑰輪換失敗。適用於 H100、B100 和 GB200。	取代
155	NVLink 軟體定義錯誤 – 在 NVLink 上報告了軟體定義錯誤。	重新啟動
156	資源淘汰事件 – GPU 由於偵測到的故障而淘汰硬體資源。	重新啟動
158	GPU 嚴重逾時 – GPU 上發生嚴重逾時。	重新啟動

若要檢視與 GPU 運作狀態相關的目前節點條件，請執行下列命令。

```
kubectl get nodes -o custom-
columns='NAME:.metadata.name,ACCELERATOR_READY:.status.conditions[?
(@.type=="AcceleratedHardwareReady")].status,REASON:.status.conditions[?
(@.type=="AcceleratedHardwareReady")].reason'
```

若要在叢集上檢視 XID 相關事件，請執行下列其中一個命令。

```
kubectl get events | grep -i "NvidiaXID"
```

ContainerRuntime 節點運作狀態問題

針對下表中所列嚴重性為「條件」的問題，監控條件為 `ContainerRuntimeReady`。

名稱	嚴重性	說明	修復動作
ContainerRuntimeFailed	事件	容器執行時期未能建立容器，若重複發生，很可能與任何報告的問題相關。	無
DeprecatedContainerdConfiguration	事件	使用已棄用的映像資訊清單第 2 版的容器映像，結構描述 1	無

名稱	嚴重性	說明	修復動作
		最近透過 提取至節點containerd 。	
KubeletFailed	事件	kubelet 進入失敗狀態。	無
LivenessProbeFailures	事件	偵測到活體探查失敗，若重複發生，可能表明應用程式碼問題或逾時值不足。	無
PodStuckTerminating	條件	Pod 終止或終止時凍結時間過長，可能是由於阻止 Pod 狀態進度的 CRI 錯誤所致。	取代
ReadinessProbeFailures	事件	偵測到整備探查失敗，若重複發生，可能表明應用程式碼問題或逾時值不足。	無
【Name】RepeatedRestart	事件	系統化單位經常重新啟動。	無
ServiceFailedToStart	事件	未能啟動系統化單元。	無

核心節點運作狀態問題

針對下表中所列嚴重性為「條件」的問題，監控條件為 KernelReady。

名稱	嚴重性	說明	修復動作
AppBlocked	事件	排程導致任務已遭到長時間封鎖，通常是	無

名稱	嚴重性	說明	修復動作
		由於輸入或輸出遭到封鎖所致。	
AppCrash	事件	節點上的應用程式已當機。	無
ApproachingKernelPidMax	事件	程序數量接近目前`kernel.pid_max` 設定可用的 PIDs 數量上限，之後就無法再啟動任何程序。	無
ApproachingMaxOpenFiles	事件	依據目前的核心設定，開啟的檔案數目正在接近可開啟的最大檔案數目，之後將無法開啟新檔案。	無
ConntrackExceededKernel	事件	連線追蹤超出核心上限，並且不能建立新的連線，這可能會導致封包遺失。	無
ExcessiveZombieProcesses	事件	若程序不能完全回收，則會大量累積，這表明存在應用程式問題，並且可能會導致達到系統程序限制。	無
ForkFailedOutOfPIDs	條件	系統缺少程序 ID 或記憶體導致分支或執行呼叫失敗，這可能是由於殭屍程序或實體記憶體耗盡所致。	取代

名稱	嚴重性	說明	修復動作
KernelBug	事件	Linux 核心本身偵測並報告了核心錯誤，但此錯誤有時可能是由於節點具有高 CPU 或記憶體而導致事件處理延遲所致。	無
LargeEnvironment	事件	此程序的環境變數數量大於預期，可能是由許多將 `enableServiceLinks` 設為 true 的服務所造成，這可能會導致效能問題。	無
RapidCron	事件	此節點上每五分鐘的 Cron 任務執行速度更快，若任務取用大量資源，則可能會影響效能。	無
SoftLockup	事件	CPU 在一定時間內停滯。	無

聯網節點運作狀態問題

針對下表中所列嚴重性為「條件」的問題，監控條件為 `NetworkingReady`。

名稱	嚴重性	說明	修復動作
BandwidthInExceeded	事件	因傳入的彙總頻寬超過執行個體的上限而排入佇列或丟棄的封包數目。	無

名稱	嚴重性	說明	修復動作
BandwidthOutExceeded	事件	因傳出的彙總頻寬超過執行個體的上限而排入佇列或丟棄的封包數目。	無
ConntrackExceeded	事件	連線追蹤超出執行個體上限，並且不能建立新的連線，這可能會導致封包遺失。	無
EFAErrorMetric	事件	EFA 驅動程式指標顯示有效能降低的界面。	無
IPAMDInconsistentState	事件	磁碟上 IPAMD 檢查點的狀態不會反映容器執行時間中的 IPs。	無
IPAMDNoIPs	事件	IPAMD 沒有 IP 地址。	無
IPAMDNotReady	條件	IPAMD 未能連線至 API 伺服器。	取代
IPAMDNotRunning	條件	找不到正在執行的 Amazon VPC CNI 程序。	取代
IPAMDRepeatedlyRestart	事件	IPAMD 服務發生多次重新啟動。	無
InterfaceNotRunning	條件	此介面看起來未在執行中，或者存在網路問題。	取代
InterfaceNotUp	條件	此介面看起來未啟用，或者存在網路問題。	取代

名稱	嚴重性	說明	修復動作
KubeProxyNotReady	事件	Kube-proxy 未能監看或列示資源。	無
LinkLocalExceeded	事件	由於本機代理伺服器服務的流量 PPS 超過網路介面上限而丟棄的封包數目。	無
MACAddressPolicyMisconfigured	事件	systemd-networkd 連結組態的值不正確`MACAddressPolicy` 。	無
MissingDefaultRoutes	事件	缺少預設路由規則。	無
MissingIPRoutes	事件	Pod IPs 缺少路由。	無
MissingIPRules	事件	Pod IPs 缺少規則。	無
MissingLoopbackInterface	條件	此執行個體缺少迴路介面，從而導致服務未能執行，具體取決於本機連線。	取代
NetworkSysctl	事件	此節點的網路`sysctl`設定可能不正確。	無
PPSExceeded	事件	因雙向 PPS 超過執行個體的上限而排入佇列或丟棄的封包數目。	無

名稱	嚴重性	說明	修復動作
PortConflict	事件	如果 Pod 使用 hostPort，則可以撰寫`iptables`規則來覆寫主機的已繫結連接埠，從而可能阻止 API 伺服器存取 `kubelet`。	無
UnexpectedRejectRule	事件	在 中找到非預期的 `REJECT`或 `DROP`規則`iptables`，可能封鎖預期的流量。	無

儲存節點運作狀態問題

針對下表中所列嚴重性為「條件」的問題，監控條件為 `StorageReady`。

名稱	嚴重性	說明	修復動作
EBSInstanceIOPSExceeded	事件	已超過執行個體的 IOPS 上限。	無
EBSInstanceThroughputExceeded	事件	已超過執行個體的最大輸送量。	無
EBSVolumeIOPSExceeded	事件	已超過特定 EBS 磁碟區的 IOPS 上限。	無
EBSVolumeThroughputExceeded	事件	已超過特定 Amazon EBS 磁碟區的輸送量上限。	無
EtcHostsMountFailed	事件	由於使用者資料在`kubelet-container` 操作`/var/lib/`	無

名稱	嚴重性	說明	修復動作
		`kubelet/pods` 期間重新掛載，因此產生的 kubelet 掛載/`etc/hosts` 失敗。	
IODelays	事件	程序中偵測到輸入或輸出延遲，若過多，可能表明輸入輸出佈建不足。	無
KubeletDiskUsageSlow	事件	在嘗試存取 檔案系統時，`kubelet` 報告磁碟用量緩慢。這可能表示磁碟輸入輸出不足或檔案系統問題。	無
XFSSmallAverageClusterSize	事件	XFS 平均叢集大小很小，表示可用空間分段過多。這可以防止在可用節點或可用空間的情況下建立檔案。	無

設定節點監控代理程式

EKS 節點監控代理程式會部署為 DaemonSet。當您將其部署為 EKS 附加元件時，您可以使用下列組態值來自訂安裝。如需預設組態，請參閱 EKS 節點監控代理程式 Helm Chart。

組態選項	說明
`monitoringAgent.resources.requests.cpu`	監控代理程式的 CPU 資源請求。
`monitoringAgent.resources.requests.memory`	監控代理程式的記憶體資源請求。

組態選項	說明
monitoringAgent.resources.limits.cpu	監控代理程式的 CPU 資源限制。
monitoringAgent.resources.limits.memory	監控代理程式的記憶體資源限制。
monitoringAgent.tolerations	在污點節點上排程監控代理程式的容錯。
monitoringAgent.additionalArgs	要傳遞給監控代理程式的其他命令列引數。

> **Note**
>
> 您可以使用 EKS 附加元件或 Helm 安裝`monitoringAgent.additionalArgs`來設定 `hostname-override`和 `verbosity` 。您目前無法使用 EKS 附加元件`8002`或 Helm 安裝，透過其他 args 自訂節點監控代理程式的 `probe-address`() 或 `metrics-address`(`8003`)。

節點監控代理程式包含用於監控 NVIDIA GPU 的 NVIDIA DCGM （資料中心 GPU Manager) 伺服器元件 (`nv-hostengine`) GPUs 。此元件只會在屬於 NVIDIA GPU 執行個體類型的節點上執行，如代理程式 Helm Chart `nodeAffinity`中的 所示。您無法搭配 EKS 節點監控代理程式使用現有的 NVIDIA DCGM 安裝，如果您需要此功能，請針對 EKS 藍圖 GitHub 問題 #2763 提供意見回饋。

當您將 EKS 節點監控代理程式部署為 EKS 附加元件時，您可以使用下列組態值自訂 NVIDIA DCGM 安裝。

組態選項	說明
dcgmAgent.resources.requests.cpu	DCGM 代理程式的 CPU 資源請求。
dcgmAgent.resources.requests.memory	DCGM 代理程式的記憶體資源請求。

組態選項	說明
dcgmAgent.resources.limits.cpu	DCGM 代理程式的 CPU 資源限制。
dcgmAgent.resources.limits.memory	DCGM 代理程式的記憶體資源限制。
dcgmAgent.tolerations	在污點節點上排程 DCGM 代理程式的容錯。

您可以使用下列 AWS CLI 命令來取得 EKS 節點監控代理程式 EKS 附加元件的版本和結構描述的實用資訊。

取得 Kubernetes 版本的最新代理程式附加元件版本。將 取代1.35為您的 Kubernetes 版本。

```
aws eks describe-addon-versions \
  --addon-name eks-node-monitoring-agent \
  --kubernetes-version 1.35 \
  --query='addons[].addonVersions[].addonVersion'
```

取得 EKS 附加元件中支援的代理程式附加元件結構描述。v1.5.1-eksbuild.1 將 取代為您的代理程式版本。

```
aws eks describe-addon-configuration \
  --addon-name eks-node-monitoring-agent \
  --addon-version v1.5.1-eksbuild.1
```

自動修復 EKS 叢集中的節點

本主題詳細說明 EKS 自動節點修復行為，以及如何將其設定為滿足您的需求。EKS 自動節點修復預設為在 EKS Auto 模式中啟用，可與 EKS 受管節點群組和 Karpenter 搭配使用。

預設 EKS 自動節點修復動作摘要如下表所示，適用於 EKS Auto Mode、EKS 受管節點群組和 Karpenter 的行為。使用 EKS Auto Mode 或 Karpenter 時，所有AcceleratedHardwareReady修復動作都是 Replace，只有 EKS 受管節點群組支援Reboot做為修復動作。

如需 EKS 節點監控代理程式偵測到的節點運作狀態問題及其對應節點修復動作的詳細清單，請參閱 the section called “節點運作狀態偵測”。

節點條件	Description	在 之後修復	修復動作（多個）
AcceleratedHardwareReady	AcceleratedHardwareReady 指出節點上的加速硬體 (GPU、Neuron) 是否正常運作。	10m	取代或重新啟動
ContainerRuntimeReady	ContainerRuntimeReady 指出容器執行時間（容器等）是否正常運作且能夠執行容器。	30m	取代
DiskPressure	DiskPressure 是標準 Kubernetes 條件，表示節點遇到磁碟壓力（磁碟空間不足或 I/O 高）。	N/A	無
KernelReady	KernelReady 指出核心是否正常運作，而不會發生嚴重錯誤、恐慌或資源耗盡。	30m	取代
MemoryPressure	MemoryPressure 是標準 Kubernetes 條件，表示節點正經歷記憶體壓力（可用的記憶體不足）。	N/A	無
NetworkingReady	NetworkingReady 指出節點的網路堆疊是否正常運作（介面、路由、連線能力）。	30m	取代
StorageReady	StorageReady 指出節點的儲存子系統是否	30m	取代

節點條件	Description	在 之後修復	修復動作 （多個）
	正常運作 （磁碟、檔案系統、I/O)。		
備妥	Ready 是標準 Kubernetes 條件，表示節點運作狀態良好且已準備好接受 Pod。	30m	取代

在下列情況下，EKS 自動節點修復動作預設為停用。在每個案例中，進行中的節點修復動作都會繼續。如需如何覆寫這些預設設定the section called “設定自動節點修復”，請參閱 。

EKS 受管節點群組

- 節點群組有超過五個節點，且節點群組中超過 20% 的節點運作狀態不佳。
- 透過應用程式復原控制器 (ARC) 觸發叢集的區域轉移。

EKS 自動模式和 Karpenter

- NodePool 中超過 20% 的節點運作狀態不佳。
- 對於獨立 NodeClaims，叢集中的 20% 節點運作狀態不佳。

設定自動節點修復

使用 EKS Auto Mode 時，無法設定自動節點修復，且一律以與 Karpenter 相同的預設設定啟用。

Karpenter

若要搭配 Karpenter 使用自動節點修復，請啟用功能閘道 `NodeRepair=true`。您可以透過 CLI `--feature-gates` 選項或 Karpenter 部署中的`FEATURE_GATES`環境變數來啟用功能閘道。如需詳細資訊，請參閱 Karpenter 文件。

受管節點群組

您可以在建立新的 EKS 受管節點群組或更新現有的 EKS 受管節點群組時啟用自動節點修復。

- Amazon EKS 主控台 – 選取受管節點群組的啟用節點自動修復核取方塊。如需詳細資訊，請參閱the section called “建立”。
- AWS CLI – `--node-repair-config enabled=true`新增至 `eks create-nodegroup`或 `eks update-nodegroup-config`命令。
- eksctl – 設定 `managedNodeGroups.nodeRepairConfig.enabled: true`，請參閱 eksctl GitHub 中的範例。

使用 EKS 受管節點群組時，您可以使用下列設定控制節點自動修復行為。

若要控制節點自動修復何時停止採取動作，請根據節點群組中運作狀態不佳的節點數目來設定閾值。設定絕對計數或百分比，但不能同時設定兩者。

設定	Description
`maxUnhealthyNodeThresholdCount`	節點自動修復停止運作狀態不佳的節點絕對數量。使用此項目來限制修復的範圍。
`maxUnhealthyNodeThresholdPercentage`	節點自動修復停止運作狀態不佳的節點百分比 (0-100)。

若要控制同時修復的節點數量，您可以設定修復平行處理。如同運作狀態不佳的節點閾值，請設定絕對計數或百分比，但不能同時設定兩者。

設定	Description
`maxParallelNodesRepairedCount`	要同時修復的節點數目上限。
`maxParallelNodesRepairedPercentage`	要同時修復的不良節點百分比上限 (0-100)。

使用 `nodeRepairConfigOverrides`，您可以針對特定條件自訂修復行為。當您需要不同的修復動作或不同問題類型的等待時間時，請使用此選項。

每個覆寫都需要下列所有欄位：

欄位	Description
nodeMonitoringCondition	節點監控代理程式報告的節點條件類型。例如：AcceleratedHardwareReady 、NetworkingReady 、StorageReady 、KernelReady 。
nodeUnhealthyReason	狀況不良的特定原因代碼。例如：NvidiaXID31Error 、IPAMDNotRunning 。
minRepairWaitTimeMins	在節點符合修復資格之前，條件必須保留的最短時間，以分鐘為單位。使用此選項可避免針對暫時問題修復節點。
repairAction	符合條件時要採取的動作。有效值： Replace（終止並取代節點）、 Reboot（重新啟動節點） 或 NoAction（無修復動作）。

下列 AWS CLI 範例會建立具有自訂修復設定的節點群組。

```
aws eks create-nodegroup \
  --cluster-name my-cluster \
  --nodegroup-name my-nodegroup \
  --node-role arn:aws:iam::111122223333:role/NodeRole \
  --subnets subnet-0123456789abcdef0 \
  --node-repair-config '{
    "enabled": true,
    "maxUnhealthyNodeThresholdPercentage": 10,
    "maxParallelNodesRepairedCount": 3,
    "nodeRepairConfigOverrides": [
      {
        "nodeMonitoringCondition": "AcceleratedHardwareReady",
        "nodeUnhealthyReason": "NvidiaXID64Error",
        "minRepairWaitTimeMins": 5,
        "repairAction": "Replace"
      },
      {
        "nodeMonitoringCondition": "AcceleratedHardwareReady",
        "nodeUnhealthyReason": "NvidiaXID31Error",
        "minRepairWaitTimeMins": 15,
        "repairAction": "NoAction"
      }
```

```
    ]
  }'
```

此組態會執行下列動作：

- 啟用節點自動修復
- 當超過 10% 的節點運作狀態不佳時，停止修復動作
- 一次最多可修復 3 個節點
- 覆寫 XID 64 錯誤 (GPU 記憶體重新映射失敗），以在 5 分鐘後取代節點。預設值為 10 分鐘後重新啟動。
- 覆寫 XID 31 錯誤 (GPU 記憶體頁面錯誤） 不採取任何動作。預設值為 10 分鐘後重新啟動。

檢視節點的運作狀態

本主題說明可用於監控 Amazon EKS 叢集中節點運作狀態的工具和方法。相關資訊涵蓋節點條件、事件和偵測案例，可協助您識別和診斷節點層級問題。使用此處所述的命令和模式來檢查節點運作狀態資源、解譯狀態條件，並分析節點事件以進行操作故障診斷。

您可以針對所有節點使用 Kubernetes 命令，以取得一些節點運作狀態資訊。此外，如果您透過 Amazon EKS 自動模式或 Amazon EKS 受管附加元件使用節點監控代理程式，您將會取得更多種類的節點訊號，進而協助故障診斷。在可觀測性儀表板中，還會提供節點監控代理程式偵測的運作狀態問題說明。如需詳細資訊，請參閱the section called “節點運作狀態偵測”。

節點條件

節點條件代表需要修復動作的終端問題，例如執行個體替換或重新啟動。

若要取得所有節點的條件：

```
kubectl get nodes -o 'custom-
columns=NAME:.metadata.name,CONDITIONS:.status.conditions[*].type,STATUS:.status.conditions[*].
```

若要取得特定節點的詳細條件

```
kubectl describe node node-name
```

運作狀態良好的節點的範例條件輸出：

```
  - lastHeartbeatTime: "2024-11-21T19:07:40Z"
    lastTransitionTime: "2024-11-08T03:57:40Z"
    message: Monitoring for the Networking system is active
    reason: NetworkingIsReady
    status: "True"
    type: NetworkingReady
```

具有聯網問題之運作狀態不佳節點的範例條件：

```
  - lastHeartbeatTime: "2024-11-21T19:12:29Z"
    lastTransitionTime: "2024-11-08T17:04:17Z"
    message: IPAM-D has failed to connect to API Server which could be an issue with
      IPTable rules or any other network configuration.
    reason: IPAMDNotReady
    status: "False"
    type: NetworkingReady
```

事件節點

節點事件表示暫時性問題或非最佳化的組態。

若要取得節點監控代理程式報告的所有事件

節點監控代理程式可用時，您可以執行下列命令。

```
kubectl get events --field-selector=reportingComponent=eks-node-monitoring-agent
```

輸出範例：

```
LAST SEEN   TYPE      REASON       OBJECT
 MESSAGE
4s          Warning   SoftLockup   node/ip-192-168-71-251.us-west-2.compute.internal
 CPU stuck for 23s
```

若要取得所有節點的事件

```
kubectl get events --field-selector involvedObject.kind=Node
```

若要取得特定節點的事件

```
kubectl get events --field-selector involvedObject.kind=Node,involvedObject.name=node-name
```

若要即時監看事件

```
kubectl get events -w --field-selector involvedObject.kind=Node
```

範例事件輸出：

```
LAST SEEN   TYPE     REASON           OBJECT         MESSAGE
2m          Warning  MemoryPressure   Node/node-1    Node experiencing memory pressure
5m          Normal   NodeReady        Node/node-1    Node became ready
```

常見的故障診斷命令

```
# Get comprehensive node status
kubectl get node node-name -o yaml

# Watch node status changes
kubectl get nodes -w

# Get node metrics
kubectl top node
```

使用 kubectl 及 S3 擷取受管節點的節點日誌

了解如何為已安裝節點監控代理程式的 Amazon EKS 受管節點擷取節點記錄。

先決條件

請確認您已具備以項目：

- 現有已安裝節點監控代理程式的 Amazon EKS 叢集。如需詳細資訊，請參閱the section called “節點運作狀態和修復”。
- 已安裝並設定為可與您叢集通訊的 `kubectl` 命令列工具。
- 安裝並登入 AWS 的 CLI 具有建立 S3 儲存貯體和物件的足夠許可。

- 已安裝最新版本 Python 3
- 已安裝適用於 Python 3、Boto 3 的 AWS SDK。

步驟 1：建立 S3 儲存貯體目的地 (選用)

若您還沒有 S3 儲存貯體來儲存日誌，請先建立一個。使用下列 AWS CLI 命令。儲存貯體預設為 `private` 存取控制清單。以您選擇的唯一儲存貯體名稱取代 *bucket-name*。

```
aws s3api create-bucket --bucket <bucket-name>
```

步驟 2：建立用於 HTTP Put 的預先簽章 S3 URL

Amazon EKS 透過對您指定的 URL 執行 HTTP PUT 操作來回傳節點記錄。在本教學中，我們將產生一個預先簽章的 S3 HTTP PUT URL。

日誌會以 gzip 壓縮的 tar 檔 (副檔名為 `.tar.gz`) 形式傳回。

> **Note**
>
> 您必須使用 AWS API 或 SDK 來建立預先簽章的 S3 上傳 URL，以供 EKS 上傳日誌檔案。您無法使用 CLI 建立預先簽章的 S3 AWS 上傳 URL。

1. 決定您要在儲存貯體中的哪個位置儲存日誌。例如，您可以使用 *2024-11-12/logs1.tar.gz* 做為金鑰。
2. 將下列 Python 程式碼儲存至檔案 https：//*presign-upload.py*。取代 *<bucket-name>* 和 *<key>*。金鑰應以 `.tar.gz` 結尾。

   ```
   import boto3; print(boto3.client('s3').generate_presigned_url(
      ClientMethod='put_object',
      Params={'Bucket': '<bucket-name>', 'Key': '<key>'},
      ExpiresIn=1000
   ))
   ```

3. 使用以下命令執行指令碼

   ```
   python presign-upload.py
   ```

4. 記錄 URL 輸出。在下一個步驟中使用此值做為 *http-put-destination*。

如需詳細資訊，請參閱在適用於 Python 的Boto3 開發套件文件中產生預先簽章的 URL 以上傳檔案。AWS Boto3

步驟 3：建立 NodeDiagnostic 資源

識別您要從中收集日誌的節點名稱。

建立 NodeDiagnostic 資訊清單，使用該節點名稱作為資源名稱，並提供 HTTP PUT URL 目的地。

```
apiVersion: eks.amazonaws.com/v1alpha1
kind: NodeDiagnostic
metadata:
    name: <node-name>
spec:
    logCapture:
        destination: http-put-destination
```

將清單檔案套用至叢集。

```
kubectl apply -f nodediagnostic.yaml
```

您可以透過描述 NodeDiagnostic 資源以檢查集合的狀態：

- 狀態為 Success 或 SuccessWithErrors 表示任務已完成，且日誌已上傳到提供的目的地 (SuccessWithErrors 表示可能缺少部分日誌)
- 如果狀態為「失敗」，請確認上傳 URL 格式正確且未過期。

```
kubectl describe nodediagnostics.eks.amazonaws.com/<node-name>
```

步驟 4：從 S3 下載日誌

請等待大約一分鐘再嘗試下載日誌。然後使用 S3 CLI 下載日誌。

```
# Once NodeDiagnostic shows Success status, download the logs
aws s3 cp s3://<bucket-name>/key ./<path-to-node-logs>.tar.gz
```

步驟 5：清理 NodeDiagnostic 資源

- NodeDiagnostic 資源不會自動刪除。取得日誌成品後，您應自行清除這些項目

```
# Delete the NodeDiagnostic resource
kubectl delete nodediagnostics.eks.amazonaws.com/<node-name>
```

NodeDiagnostic **node** 目的地

從 v1.6.1-eksbuild.1版本的 Node Monitoring Agent 開始，您可以選擇將日誌收集目的地設定為 node。使用此目的地將導致節點上日誌的收集和暫時保留，以供稍後收集。除了此功能之外，在 Node Monitoring Agent 的 GitHub 儲存庫中，您還可以安裝 kubectl 外掛程式，以便輕鬆互動和收集日誌。如需詳細資訊，請參閱 kubectl ekslogs 外掛程式的文件。

使用範例

```
# Collect NodeDiagnostic logs from a single node
kubectl ekslogs <node-name>

# Collect NodeDiagnostic logs from multiple nodes
kubectl ekslogs <node-name-1> <node-name-2> <node-name-3>

# Collect NodeDiagnostic logs from all nodes with a specific label
kubectl ekslogs -l <key>=<value>
```

使用 kubectl 和 S3 擷取受管節點上的網路流量

了解如何在具有節點監控代理程式的 Amazon EKS 受管節點上擷取網路流量。代理程式會在節點上執行 tcpdump、壓縮擷取檔案，並將它們上傳到您的 S3 儲存貯體。

先決條件

請確認您已具備以項目：

- 具有節點監控代理程式的現有 Amazon EKS Auto Mode 叢集。如需詳細資訊，請參閱the section called “節點運作狀態和修復”。
- 已安裝並設定為可與您叢集通訊的 kubectl 命令列工具。
- 安裝並登入 AWS 的 CLI 具有建立 S3 儲存貯體和物件的足夠許可。

- 已安裝最新版本的 Python 3。
- 已安裝適用於 Python 3、Boto 3 的 AWS SDK。
- 已安裝 PyYAML 程式庫 (`pip install pyyaml`)。

步驟 1：建立 S3 儲存貯體目的地 (選用)

如果您還沒有儲存擷取檔案的 S3 儲存貯體，請建立一個。將 *bucket-name* 和 *region* 取代為您的值。

```
aws s3api create-bucket --bucket <bucket-name> \
    --region <region> \
    --create-bucket-configuration LocationConstraint=<region>
```

> **Note**
>
> 除了 以外的所有區域都需要 `--create-bucket-configuration` 參數`us-east-1`。

步驟 2：開始封包擷取

使用節點監控代理程式儲存庫 (`tools/start-capture.py`) 中的`start-capture.py`指令碼來產生預先簽章的 S3 登入資料、建立`NodeDiagnostic`資源，並將其套用至您的叢集。

1. 識別您要從中擷取流量的節點。

   ```
   kubectl get nodes
   ```

2. 將 start-capture.py 指令碼從節點監控代理程式儲存庫儲存至本機電腦，然後執行它。將 *<bucket-name>* 和 *<node-name>* 取代為您的值。

   ```
   python3 start-capture.py --bucket <bucket-name> --node <node-name>
   ```

   常見選項：

   ```
   # Capture for 5 minutes on eth0 with a filter
   python3 start-capture.py --bucket <bucket-name> --node <node-name> \
   ```

```
    --duration 5m --interface eth0 --filter "tcp port 443"

# Preview the YAML without applying
python3 start-capture.py --bucket <bucket-name> --node <node-name> --dry-run
```

指令碼需要使用已安裝 boto3和 的 Python 3pyyaml，並為您的叢集kubectl進行設定。

指令碼會產生類似下列NodeDiagnostic的資源。此範例僅供參考；請注意，upload欄位需要由指令碼以程式設計方式產生的預先簽章 S3 POST 登入資料。

```
apiVersion: eks.amazonaws.com/v1alpha1
kind: NodeDiagnostic
metadata:
  name: <node-name>                    # Required: node instance ID
spec:
  packetCapture:
    duration: "30s"                      # Required: capture duration (max 1h)
    # interface: "eth0"                  # Optional: default is primary ENI. Use
 "any" for all interfaces
    # filter: "tcp port 443"             # Optional: tcpdump filter expression
    # chunkSizeMB: 10                    # Optional: file rotation size in MB (1-100,
 default: 10)
    upload:                               # Required: pre-signed S3 POST credentials
      url: "https://<bucket>.s3.amazonaws.com/"
      fields:
        key: "captures/<node-name>/${filename}"
        # ... other pre-signed POST fields (generated by the script)
```

步驟 3：監控擷取進度

檢查擷取的狀態。

```
kubectl describe nodediagnostic <node-name>
```

狀態會顯示：

- Running 擷取進行時。
- Completed 以及擷取完成和所有檔案上傳Success的原因。
- Completed Failure 如果擷取發生錯誤，則為 原因。

若要查看完整狀態，包括 captureID（用於 S3 路徑識別）：

```
kubectl get nodediagnostic <node-name> -o jsonpath='{.status.captureStatuses}'
```

步驟 4：從 S3 下載擷取檔案

狀態顯示 後Success，請從 S3 下載擷取檔案。

```
aws s3 cp s3://<bucket-name>/captures/ ./captures/ --recursive
```

這些檔案是 gzip 壓縮的 pcap 格式。使用 tcpdump 或 Wireshark 解壓縮和分析：

```
gunzip captures/*.gz
tcpdump -r captures/capture.pcap0000 -n
```

步驟 5：清除

NodeDiagnostic 資源不會自動刪除。在您取得擷取檔案後清除 。在擷取執行時刪除資源會立即停止擷取。

```
kubectl delete nodediagnostic <node-name>
```

組態選項和行為

如需完整的packetCapture規格參考、組態選項和行為詳細資訊，請參閱節點監控代理程式儲存庫中的封包擷取文件。

Amazon EKS 混合節點概觀

透過 Amazon EKS 混合節點，您可以使用內部部署和邊緣基礎設施做為 Amazon EKS 叢集中的節點。 AWS 管理 Amazon EKS 叢集的 AWS託管 Kubernetes 控制平面，並管理在內部部署或邊緣環境中執行的混合節點。這可統一您環境中的 Kubernetes 管理，並將 AWS 內部部署和邊緣應用程式的 Kubernetes 控制平面管理卸載至 。

Amazon EKS 混合節點可與任何內部部署硬體或虛擬機器搭配使用，將 Amazon EKS 的效率、可擴展性和可用性帶入需要執行應用程式的任何環境。您可以搭配 Amazon EKS 混合節點使用各種 Amazon EKS 功能，包括 Amazon EKS 附加元件、Amazon EKS Pod 身分識別、叢集存取項目、叢集洞察和

擴充的 Kubernetes 版本支援。Amazon EKS 混合節點原生整合了 AWS Systems Manager、 AWS IAM Roles Anywhere、Amazon Managed Service for Prometheus 和 Amazon CloudWatch 等 AWS 服務，以進行集中式監控、記錄和身分管理。

使用 Amazon EKS 混合節點時，沒有前期承諾，也沒有最低費用，並且當您混合節點的 vCPU 資源連接到 Amazon EKS 叢集時，會按小時向您收費。如需有關定價的詳細資訊，請參閱 Amazon EKS 定價。

功能

EKS 混合節點具有下列高階功能：

- 受管 Kubernetes 控制平面： AWS 管理 EKS 叢集的 AWS託管 Kubernetes 控制平面，以及管理在內部部署或邊緣環境中執行的混合節點。這可統一您環境中的 Kubernetes 管理，並將 AWS 內部部署和邊緣應用程式的 Kubernetes 控制平面管理卸載至 。透過將 Kubernetes 控制平面移至 AWS，您可以為應用程式節省內部部署容量，並信任 Kubernetes 控制平面隨工作負載擴展。
- 一致的 EKS 體驗：EKS 混合節點支援大多數 EKS 功能，可在內部部署和雲端環境中提供一致的 EKS 體驗，包括 EKS 附加元件、EKS Pod 身分識別、叢集存取項目、叢集洞察、擴充的 Kubernetes 版本支援等。如需 EKS 混合節點支援的 EKS 附加元件的詳細資訊，請參閱 the section called “設定附加元件”。

- 集中式可觀測性和身分管理：EKS 混合節點與 AWS Systems Manager、 AWS IAM Roles Anywhere、Amazon Managed Service for Prometheus 和 Amazon CloudWatch 等 AWS 服務原生整合，以進行集中式監控、記錄和身分管理。
- Burst-to-cloud或新增內部部署容量：單一 EKS 叢集可用於在 AWS 區域、 AWS 本機區域或 AWS Outpost 中執行混合節點和節點至burst-to-cloud，或將內部部署容量新增至 EKS 叢集。如需詳細資訊，請參閱混合模式叢集的考量事項。
- 彈性基礎結構：EKS 混合節點遵循自有基礎結構方法，並且與您用於混合節點的基礎結構無關。您可以在實體或虛擬機器以及 x86 和 ARM 架構上執行混合節點，從而能夠跨不同的基礎結構類型移轉在混合節點上執行的內部部署工作負載。
- 彈性聯網：使用 EKS 混合節點，EKS 控制平面和混合節點之間的通訊會透過您在叢集建立期間傳遞的 VPC 和子網路路由，而該子網路以 EKS 中用於控制平面到節點聯網的現有機制為建構基礎。這對於將內部部署網路連線到 VPC 的偏好方法來說非常靈活 AWS。有多種文件記錄的選項可供使用，包括 AWS Site-to-Site VPN、 AWS Direct Connect 或您自己的 VPN 解決方案，您可以選擇最適合您的使用案例的方法。

限制

- 每個叢集最多支援 15 個遠端節點網路 CIDR 和 15 個 遠端 Pod 網路 CIDR。

考量事項

- EKS 混合節點可與新的或現有的 EKS 叢集搭配使用。
- EKS 混合節點適用於所有 AWS 區域， AWS GovCloud (US) 區域和 AWS 中國區域除外。
- EKS 混合節點必須在您的內部部署環境與 AWS之間建立可靠的連線。EKS 混合節點不適用於中斷連線、中斷、間歇性或受限 (DDIL) 環境。如果您在 DDIL 環境中執行，請考慮 Amazon EKS Anywhere。如需混合節點在網路中斷連線案例中的行為方式的資訊，請參閱 EKS 混合節點的最佳實務。
- 不支援在雲端基礎設施上執行 EKS 混合節點，包括 AWS 區域、 AWS 本機區域、 AWS Outposts 或其他雲端。如果您在 Amazon EC2 執行個體上執行混合節點，則需支付混合節點費用。
- 混合節點的計費從節點加入 EKS 叢集時開始，並在節點從叢集中移除時停止。如果您未使用混合節點，則請務必從 EKS 叢集中將其移除。

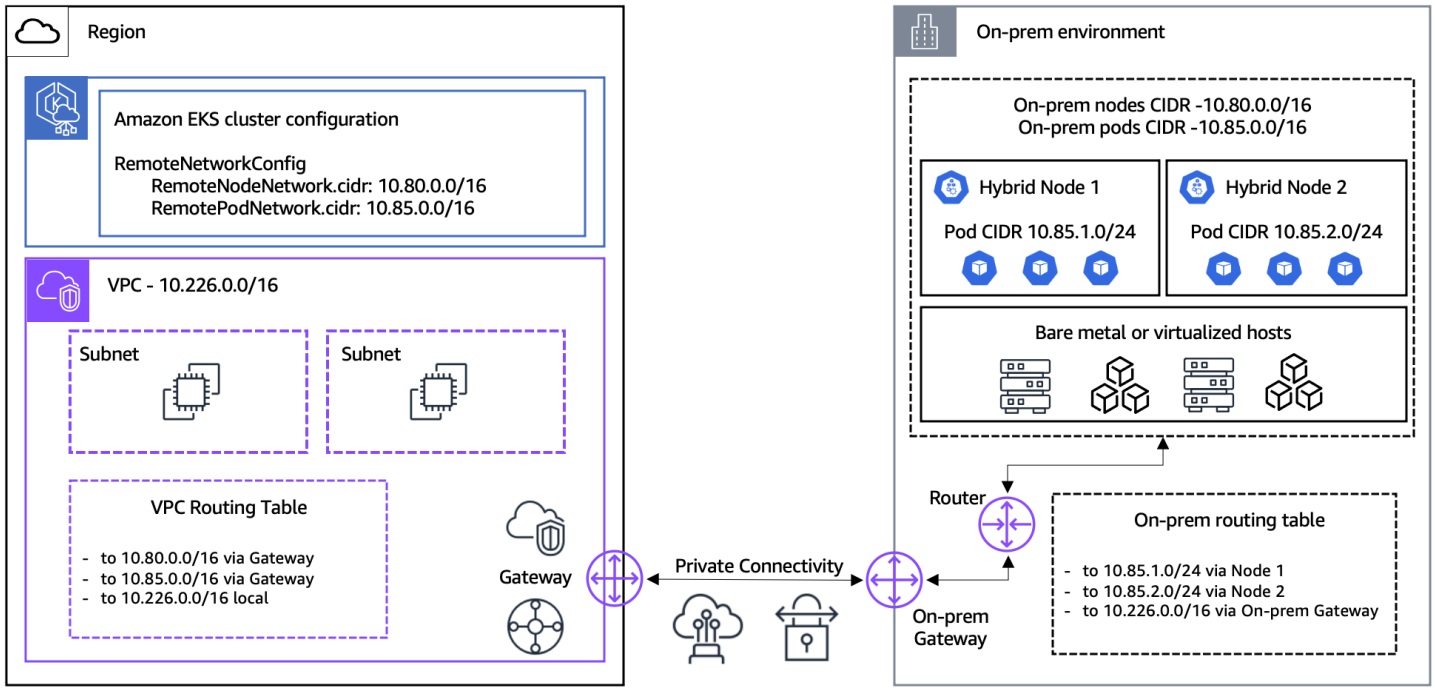
其他資源

- EKS 混合節點工作坊：在示範環境中部署 EKS 混合節點的逐步說明。
- AWS re：Invent： EKS 混合節點： AWS re：Invent 工作階段會向客戶介紹 EKS 混合節點的啟動，顯示他們在其環境中如何使用 EKS 混合節點。
- AWS re:Post：EKS 混合節點的叢集聯網：文章說明各種為 EKS 混合節點設定聯網的方法。
- AWS 部落格：使用 EKS 混合節點跨環境執行 GenAI 推論：部落格文章說明如何使用 EKS 混合節點跨環境執行 GenAI 推論。

混合節點的先決條件設定

若要使用 Amazon EKS 混合節點，您必須擁有從內部部署環境往返的私有連線 AWS、具有支援作業系統的裸機伺服器或虛擬機器，以及已設定的 AWS IAM Roles Anywhere 或 AWS Systems Manager (SSM) 混合啟用。您負責在整個混合節點生命週期中管理這些先決條件。

- 從內部部署環境往返的混合網路連線 AWS
- 實體或虛擬機器形式的基礎結構
- 與混合節點相容的作業系統
- 內部部署 IAM 憑證提供者已設定



混合網路連線

Amazon EKS 控制平面和混合節點之間的通訊會透過您在叢集建立期間傳遞的 VPC 和子網路路由，而該子網路以 Amazon EKS 中用於控制平面到節點聯網的現有機制為建構基礎。有數個文件記錄的選項可讓您將內部部署環境與您的 VPC 連接，包括 AWS Site-to-Site VPN、 AWS Direct Connect 或您自己的 VPN 連接。如需如何使用這些解決方案進行混合網路連線的詳細資訊，請參閱 AWS Site-to-Site VPN 和 AWS Direct Connect 使用者指南。

為了獲得最佳體驗，我們建議您擁有至少 100 Mbps 的可靠網路連線能力，以及最多 200 毫秒的混合節點往返延遲。 AWS 這是適用於大多數使用案例的一般指引，但並非嚴格要求。頻寬和延遲需求可能會因混合節點的數量和工作負載特性而有所不同，例如應用程式映像大小、應用程式彈性、監控和記錄組態，以及存取儲存在其他服務中的資料的應用程式相依性 AWS 。建議您在部署到生產環境之前，先使用您自己的應用程式和環境進行測試，以驗證您的聯網設定是否符合工作負載的需求。

內部部署網路組態

您必須啟用從 Amazon EKS 控制平面到內部部署環境的傳入網路存取，以允許 Amazon EKS 控制平面與在混合節點上執行的 `kubelet` 通訊，並且也可以選擇與在混合節點上執行的 Webhook 通訊。此外，您必須為在其上執行的混合節點和元件啟用傳出網路存取，以便與 Amazon EKS 控制平面通訊。您可以設定此通訊，讓 AWS Direct Connect、 AWS Site-to-Site或您自己的 VPN 連線保持完全私有。

您用於內部部署節點和 Pod 網路的無類別網域間路由 (CIDR) 範圍必須使用 IPv4 RFC-1918 或 CGNAT 地址範圍。您的內部部署路由器必須設定為路由至您的內部部署節點和選用 Pod。如需內部部署網路需求的詳細資訊，請參閱 the section called “內部部署聯網組態”，包括必須在防火牆和內部部署環境中啟用的必要連接埠和通訊協定的完整清單。

EKS 叢集組態

為了將延遲降至最低，建議您在最接近內部部署或邊緣環境的 AWS 區域中建立 Amazon EKS 叢集。您可以在 Amazon EKS 叢集建立期間，透過兩個 API 欄位傳遞內部部署節點和 Pod CIDR：`RemoteNodeNetwork` 和 `RemotePodNetwork`。您可能需要與您的內部部署網路團隊討論，以識別您的內部部署節點和 Pod CIDR。如果您為 CNI 使用覆蓋網路，節點 CIDR 會從內部部署網路配置，而 Pod CIDR 會從您使用的容器網路介面 (CNI) 配置。根據預設，Cilium 和 Calico 會使用覆蓋網路。

您透過 `RemoteNodeNetwork` 和 `RemotePodNetwork` 欄位設定的內部部署節點和 Pod CIDR 可用於設定 Amazon EKS 控制平面，進而將流量透過 VPC 路由至 `kubelet` 以及在混合節點上執行的 Pod。您的內部部署節點和 Pod CIDR 不能彼此重疊，亦不能與您在叢集建立期間傳遞的 VPC CIDR 或 Amazon EKS 叢集的服務 IPv4 組態重疊。此外，Pod CIDR 對每個 EKS 叢集必須是唯一的，以便您的內部部署路由器可以路由流量。

建議您對 Amazon EKS Kubernetes API 伺服器端點使用公有或私有端點存取。如果您選擇「公有和私有」，Amazon EKS Kubernetes API 伺服器端點一律會解析為在 VPC 外部執行的混合節點的公有 IP，以防止您的混合節點加入叢集。當您使用公有端點存取時，Kubernetes API 伺服器端點會解析為公共 IP，而從混合節點到 Amazon EKS 控制平面的通訊將會透過網際網路路由。當您選擇私有端點存取時，Kubernetes API 伺服器端點會解析為私有 IPs，而且從混合節點到 Amazon EKS 控制平面的通訊將透過您的私有連線連結路由，在大多數情況下是 AWS Direct Connect AWS Site-to-Site VPN。

VPC 組態

您必須為您的內部部署節點，使用其路由表中的路由設定在 Amazon EKS 叢集建立期間傳遞的 VPC，以及使用作為目標的虛擬私有閘道 (VGW) 或傳輸閘道 (TGW) 設定 Pod 網路 (選用)。以下所示為範例。使用內部部署網路的值取代 `REMOTE_NODE_CIDR` 和 `REMOTE_POD_CIDR`。

目標	Target	Description
10.226.0.0/16	本機	VPC 內 VPC 路由的本機流量
REMOTE_NODE_CIDR	tgw-abcdef123456	內部部署節點 CIDR，將流量路由到 TGW
REMOTE_POD_CIDR	tgw-abcdef123456	內部部署 Pod CIDR，將流量路由到 TGW

安全群組組態

當您建立叢集時，Amazon EKS 會建立名稱為 `eks-cluster-sg-<cluster-name>-<uniqueID>` 的安全群組。您無法更改此叢集安全群組的傳入規則，但可以限制傳出規則。您必須將其他安全群組新增至叢集，才能啟用在混合節點上執行的 kubelet 和 Webhook (選用)，進而聯絡 Amazon EKS 控制平面。這個其他安全群組所需的傳入規則如下所示。使用內部部署網路的值取代 `REMOTE_NODE_CIDR` 和 `REMOTE_POD_CIDR`。

名稱	安全群組規則 ID	IP 版本	Type	通訊協定	連接埠範圍	來源
內部部署節點傳入	sgr-abcdef123456	IPv4	HTTPS	TCP	443	REMOTE_NODE_CIDR

名稱	安全群組規則 ID	IP 版本	Type	通訊協定	連接埠範圍	來源
內部部署 Pod 傳入	sgr-abcdef654321	IPv4	HTTPS	TCP	443	REMOTE_POD_CIDR

基礎設施

您必須擁有可用作混合節點的裸機伺服器或虛擬機器。混合節點與底層基礎結構無關，並可支援 x86 和 ARM 架構。Amazon EKS 混合節點遵循「自有基礎結構」方法，而您要負責佈建和管理用於混合節點的裸機伺服器或虛擬機器。雖然沒有嚴格的最低資源需求，但我們建議您為混合節點使用至少具有 1 個 vCPU 和 1GiB RAM 的主機。

作業系統

Bottlerocket、Amazon Linux 2023 (AL2023)、Ubuntu 和 RHEL 經過持續驗證，可用作混合節點的節點作業系統。Bottlerocket 僅支援 VMware vSphere 環境中 AWS的 。在 Amazon EC2 外部執行時，AWS 支援計劃不涵蓋 AL2023。 Amazon EC2 AL2023 只能在內部部署虛擬化環境中使用，如需詳細資訊，請參閱 Amazon Linux 2023 使用者指南。 AWS 支援與 Ubuntu 和 RHEL 作業系統整合的混合節點，但不支援作業系統本身。

您負責作業系統佈建和管理。在首次測試混合節點時，最簡單的方式是在已佈建的主機上執行 Amazon EKS 混合節點 CLI (`nodeadm`)。對於生產部署，建議您在黃金作業系統映像中包含 `nodeadm`，並將它設定為作為 systemd 服務執行，以便在主機啟動時自動將主機加入 Amazon EKS 叢集。

內部部署 IAM 憑證提供者

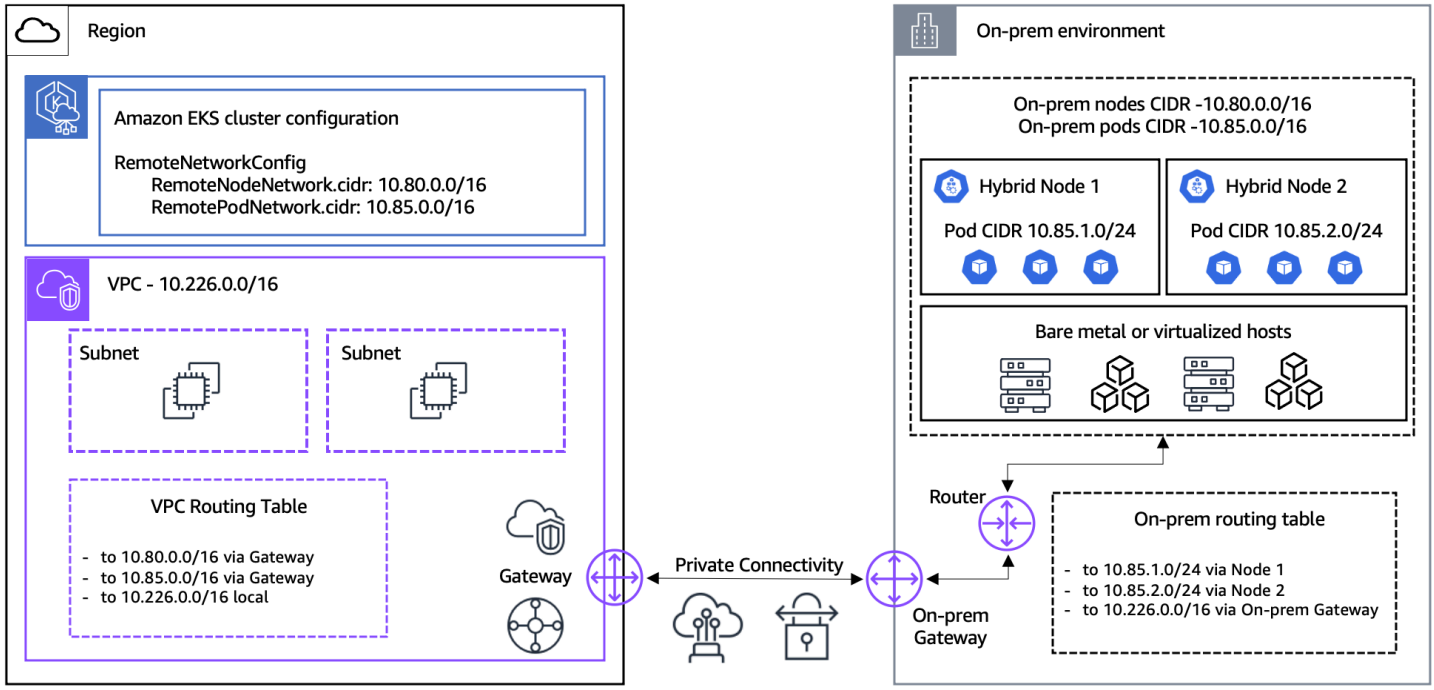
Amazon EKS 混合節點使用由 AWS SSM 混合啟用或 IAM Roles Anywhere AWS 佈建的臨時 IAM 登入資料來驗證 Amazon EKS 叢集。您必須將 AWS SSM 混合啟用或 AWS IAM Roles Anywhere 與 Amazon EKS 混合節點 CLI () 搭配使用`nodeadm`。如果您現有的公有金鑰基礎設施 (PKI) 沒有憑證授權機構 (CA) 和內部部署環境的憑證，建議您使用 AWS SSM 混合啟用。如果您有現有的 PKI 和現場部署憑證，請使用 AWS IAM Roles Anywhere。

與在 Amazon EC2 上執行的節點的 the section called “節點 IAM 角色” 類似，您將建立混合節點 IAM 角色，其中包含將混合節點加入 Amazon EKS 叢集所需的許可。如果您使用的是 AWS IAM Roles Anywhere，請設定信任政策，允許 AWS IAM Roles Anywhere 擔任混合節點 IAM 角色，並使用混合節點 AWS IAM 角色做為可擔任的角色來設定 IAM Roles Anywhere 設定檔。如果您使用的是 AWS SSM，請設定信任政策，允許 AWS SSM 擔任混合節點 IAM 角色，並使用混合節點 IAM 角色建立混

合啟用。如需有關如何建立具有必要許可的混合節點 IAM 角色的資訊，請參閱 the section called “準備憑證”。

準備混合節點的聯網

本主題概述了您在建立 Amazon EKS 叢集和連接混合節點之前必須設定的聯網設定。本指南假設您已符合使用 AWS Site-to-Site VPN、AWS Direct Connect 或您自己的 VPN 解決方案實現混合網路連線的先決條件要求。



內部部署聯網組態

最低網路需求

為了獲得最佳體驗，我們建議您擁有至少 100 Mbps 的可靠網路連線能力，以及最多 200 毫秒的混合節點往返延遲。 AWS 這是適用於大多數使用案例的一般指引，但並非嚴格要求。頻寬和延遲需求可能會因混合節點的數量和工作負載特性而有所不同，例如應用程式映像大小、應用程式彈性、監控和記錄組態，以及存取儲存在其他服務中的資料的應用程式相依性 AWS 。建議您在部署到生產環境之前，先使用您自己的應用程式和環境進行測試，以驗證您的聯網設定是否符合工作負載的需求。

內部部署節點和 Pod CIDR

識別將用於混合節點及其上執行的工作負載的節點和 Pod CIDR。如果您為 CNI 使用覆蓋網路，節點 CIDR 會從內部部署網路配置，而 Pod CIDR 會從您的容器網路介面 (CNI) 配置。當您使用 `RemoteNodeNetwork` 和 `RemotePodNetwork` 欄位建立 EKS 叢集時，您可以將內部部署節點 CIDR

和 Pod CIDR 作為輸入進行傳遞。您的內部部署節點 CIDR 必須在內部部署網路上可路由。如需內部部署 Pod CIDR 可路由性的資訊，請參閱下一節。

內部部署節點和 Pod CIDR 區塊必須符合下列要求：

1. 在下列其中一個 IPv4 RFC-1918 範圍內：10.0.0.0/8、 172.16.0.0/12或 192.168.0.0/16，或在 RFC 6598 定義的 CGNAT 範圍內：100.64.0.0/10。
2. EKS 叢集的 VPC CIDR 或您的 Kubernetes Service IPv4 CIDR 不會彼此重疊。

> **Important**
>
> 如果您在叢集建立期間未明確指定 Kubernetes 服務 IPv4 CIDR，Amazon EKS 會自動選取不會與您設定的遠端節點和 Pod 網路衝突的服務 CIDR。這表示指派的服務 CIDR 可能與標準預設值 (10.100.0.0/16 或) 不同172.20.0.0/16。例如，如果您的遠端網路與 重疊172.20.0.0/16，Amazon EKS 可能會172.16.0.0/16改為指派 。為了避免意外的服務 CIDR 指派，我們建議您在建立叢集時明確指定服務 IPv4 CIDR。如果您需要尋找指派給叢集的服務 CIDR，請執行 `aws eks describe-cluster --name CLUSTER_NAME --query "cluster.kubernetesNetworkConfig.serviceIpv4Cidr"`。

內部部署 Pod 網路路由

使用 EKS 混合節點時，我們通常會建議您將內部部署網路上的內部部署 Pod CIDR 設定為可路由，以便啟用雲端和內部部署環境之間的完整叢集通訊和功能。

可路由的 Pod 網路

如果您能夠將內部部署網路上的 Pod 網路設定為可路由，請遵循下列指引。

1. 設定以下 RemotePodNetwork 欄位：使用內部部署 Pod CIDR 的 EKS 叢集、使用內部部署 Pod CIDR 的 VPC 路由表，以及使用內部部署 Pod CIDR 的 EKS 叢集安全群組。
2. 您可以使用多種技術，讓您的內部部署 Pod CIDR 在內部部署網路上可路由，包括邊界閘道協定 (BGP)、靜態路由或其他自訂路由解決方案。BGP 是建議的解決方案，因為它比需要自訂或手動路由組態的替代解決方案更具可擴展性且更容易管理。 AWS 支援 Cilium 和 Calico 的 BGP 功能來公告 Pod CIDRs，如需詳細資訊，請參閱 the section called “可路由的遠端 Pod CIDR” the section called “設定 CNI”和 。
3. Webhooks 可以在混合節點上執行，因為 EKS 控制平面能夠與指派給 Webhook 的 Pod IP 位址進行通訊。

4. 在雲端節點上執行的工作負載能夠直接與同一 EKS 叢集中混合節點上執行的工作負載進行通訊。
5. AWS 其他服務，例如 AWS Application Load Balancer 和 Amazon Managed Service for Prometheus，能夠與混合節點上執行的工作負載進行通訊，以平衡網路流量和湊集 Pod 指標。

不可路由的 Pod 網路

如果您不能將內部部署網路上的 Pod 網路設定為可路由，請遵循下列指引。

1. Webhook 無法在混合節點上執行，因為 Webhook 需要從 EKS 控制平面連線至指派給 Webhook 的 Pod IP 位址。在此情況下，我們建議您在與混合節點相同的 EKS 叢集中的雲端節點上執行 Webhook，如需詳細資訊，請參閱 the section called “設定 Webhook”。
2. 當對雲端節點使用 VPC CNI 及對混合節點使用 Cilium 或 Calico 時，雲端節點上執行的工作負載將不能直接與混合節點上執行的工作負載進行通訊。
3. 使用服務流量分佈，以將流量保持在其來源區域的本機。如需有關服務流量分佈的詳細資訊，請參閱 the section called “設定服務流量分佈”。
4. 將 CNI 設定為在 Pod 流量離開內部部署主機時，使用輸出偽裝或網路位址轉譯 (NAT)。在 Cilium 中，其預設為啟用。Calico 需要將 `natOutgoing` 設定為 `true`。
5. Application AWS Load Balancer 和 Amazon Managed Service for Prometheus 等 AWS 其他服務無法與在混合節點上執行的工作負載通訊。

混合節點安裝和升級期間所需的存取

在安裝過程中 (即您在主機上安裝混合節點相依性時)，您必須可以存取下列網域。當您建置作業系統映像時，可以執行此程序一次；另外在執行時期，也可在每個主機上執行此程序。這包括初始安裝以及升級混合節點的 Kubernetes 版本時。

某些套件是使用作業系統的預設套件管理工具進行安裝的。對於 AL2023 和 RHEL，`yum` 命令可用於安裝 `containerd`、`ca-certificates`、`iptables` 和 `amazon-ssm-agent`。對於 Ubuntu，`apt` 可用於安裝 `containerd`、`ca-certificates` 和 `iptables`，而 `snap` 可用於安裝 `amazon-ssm-agent`。

元件	URL	通訊協定	站點
EKS 節點成品 (S3)	https://hybrid-assets.eks.amazonaws.com	HTTPS	443

元件	URL	通訊協定	站點
EKS 服務端點	https://eks.*region*.amazonaws.com	HTTPS	443
ECR 服務端點	https://api.ecr.*region*.amazonaws.com	HTTPS	443
EKS ECR 端點	請參閱 the section called “檢視 Amazon 映像登錄檔” 了解區域端點。	HTTPS	443
SSM 二進位檔端點 [1]	https://amazon-ssm-*region*.s3.*region*.amazonaws.com	HTTPS	443
SSM 服務端點 [1]	https://ssm.*region*.amazonaws.com	HTTPS	443
IAM Anywhere 二進位檔端點 [2]	https://rolesanywhere.amazonaws.com	HTTPS	443
IAM Anywhere 服務端點 [2]	https://rolesanywhere.*region*.amazonaws.com	HTTPS	443
作業系統套件管理工具端點	套件儲存庫端點是作業系統特定的，並且可能因地理區域而有所不同。	HTTPS	443

> **Note**
>
> [1] 只有在您為內部部署 AWS IAM 憑證提供者使用 AWS SSM 混合啟用時，才需要存取 SSM 端點。
> [2] 只有在您為內部部署 AWS IAM 憑證提供者使用 AWS IAM Roles Anywhere 時，才需要存取 IAM 端點。

持續叢集操作所需的存取

持續叢集操作需要為您的內部部署防火牆提供下列網路存取。

> **Important**
>
> 視您選擇的 CNI 而定，您需要為 CNI 連接埠設定其他網路存取規則。如需詳細資訊，請參閱 Cilium 文件和 Calico 文件。

Type	通訊協定	Direction	站點	來源	目標	Usage
HTTPS	TCP	傳出	443	遠端節點 CIDR	EKS 叢集 IP [1]	kubelet 到 Kubernetes API 伺服器
HTTPS	TCP	傳出	443	遠端 Pod CIDR	EKS 叢集 IP [1]	Pod 到 Kubernetes API 伺服器
HTTPS	TCP	傳出	443	遠端節點 CIDR	SSM 服務端點	SSM 混合啟用憑證重新整理和 SSM 活動訊號，每 5 分鐘一次
HTTPS	TCP	傳出	443	遠端節點 CIDR	IAM Anywhere 服務端點	IAM Roles Anywhere

Type	通訊協定	Direction	站點	來源	目標	Usage
						憑證重新整理
HTTPS	TCP	傳出	443	遠端 Pod CIDR	STS 區域端點	Pod 到 STS 端點，僅 IRSA 需要
HTTPS	TCP	傳出	443	遠端節點 CIDR	Amazon EKS Auth 服務端點	節點至 Amazon EKS 身分驗證端點，僅 Amazon EKS Pod 身分識別需要
HTTPS	TCP	傳入	10250	EKS 叢集 IP [1]	遠端節點 CIDR	Kubernetes API 伺服器到 kubelet
HTTPS	TCP	傳入	Webhook 連接埠	EKS 叢集 IP [1]	遠端 Pod CIDR	Kubernetes API 伺服器到 Webhook

Type	通訊協定	Direction	站點	來源	目標	Usage
HTTPS	TCP/UDP	傳入，傳出	53	遠端 Pod CIDR	遠端 Pod CIDR	Pod 到 CoreDNS。如果您在雲端中執行至少 1 個 CoreDNS 複本，則必須允許 DNS 流量通往執行 CoreDNS 的 VPC。
使用者定義	使用者定義	傳入，傳出	應用程式連接埠	遠端 Pod CIDR	遠端 Pod CIDR	Pod 到 Pod

> **Note**
>
> [1] EKS 叢集的 IP。請參閱以下有關 Amazon EKS 彈性網路介面的章節。

Amazon EKS 網路介面

Amazon EKS 會將網路介面連接到您在叢集建立期間傳遞的 VPC 中的子網路，以啟用 EKS 控制平面與 VPC 之間的通訊。Amazon EKS 建立的網路界面可在 Amazon EC2 主控台或使用 AWS CLI 建立叢集後找到。在 EKS 叢集上套用變更時，會刪除原始網路介面並建立新的網路介面，例如 Kubernetes 版本升級。您可對叢集建立期間傳遞的子網路使用受限子網路大小以限制 Amazon EKS 網路介面的 IP 範圍，這樣您就能更輕鬆地設定內部部署防火牆，以允許與這組已知的受限 IP 建立傳入/傳出連線。若要控制在其中建立的子網路網路介面，可以限制建立叢集時指定的子網路數量，或在建立叢集後您可以更新子網路。

Amazon EKS 佈建的網路介面具有格式 `Amazon EKS your-cluster-name` 的說明。如需可用來尋找 Amazon AWS EKS 佈建之網路介面 IP 地址的 CLI 命令，請參閱下列範例。使用您在叢集建立期間傳遞的 VPC 的 ID 取代 `VPC_ID`。

```
aws ec2 describe-network-interfaces \
--query 'NetworkInterfaces[?(VpcId == VPC_ID && contains(Description,Amazon
 EKS))].PrivateIpAddress'
```

AWS VPC 和子網路設定

Amazon EKS 的現有 VPC 和子網路要求適用於具有混合節點的叢集。此外，您的 VPC CIDR 無法與內部部署節點和 Pod CIDR 重疊。您必須在 VPC 路由表中為內部部署節點設定路由，也可選擇性地為 Pod CIDR 設定路由。必須設定這些路由，以將流量路由到您用於混合網路連線的閘道，該閘道通常是虛擬私有閘道 (VGW) 或傳輸閘道 (TGW)。如果您使用 TGW 或 VGW 將 VPC 連接到內部部署環境，則必須為您的 VPC 建立 TGW 或 VGW 附件。您的 VPC 必須支援 DNS 主機名稱和 DNS 解析。

下列步驟使用 AWS CLI。您也可以在 中 AWS 管理主控台 或透過 AWS CloudFormation、 AWS CDK 或 Terraform 等其他界面建立這些資源。

步驟 1：建立 VPC

1. 執行下列命令以建立 VPC。將 VPC_CIDR 取代為 RFC 1918（私有）、CGNAT (RFC 6598) 或非 RFC 1918/非 CGNAT（公有）（例如 10.0.0.0/16) 的 IPv4 CIDR 範圍。注意：根據預設，已為 VPC 啟用 DNS 解析 (這是 EKS 要求)。

   ```
   aws ec2 create-vpc --cidr-block VPC_CIDR
   ```

2. 為您的 VPC 啟用 DNS 主機名稱。請注意，根據預設，已為 VPC 啟用 DNS 解析。使用您在上一個步驟中建立的 VPC 的 ID 取代 VPC_ID。

   ```
   aws ec2 modify-vpc-attribute --vpc-id VPC_ID --enable-dns-hostnames
   ```

步驟 2：建立子網路

建立至少 2 個子網路。Amazon EKS 會將這些子網路用於叢集網路介面。如需詳細資訊，請參閱子網路要求和考量事項。

1. 您可以使用下列命令尋找 AWS 區域的可用區域。使用您的區域取代 us-west-2。

   ```
   aws ec2 describe-availability-zones \
        --query 'AvailabilityZones[?(RegionName == us-west-2)].ZoneName'
   ```

2. 建立子網。使用 VPC 的 ID 取代 VPC_ID。使用您子網路的 CIDR 區塊取代 SUBNET_CIDR (例如 10.0.1.0/24)。使用將建立子網路的可用區域取代 AZ (例如 us-west-2a)。您建立的子網路必須至少位於 2 個不同的可用區域。

```
aws ec2 create-subnet \
    --vpc-id VPC_ID \
    --cidr-block SUBNET_CIDR \
    --availability-zone AZ
```

（選用）步驟 3：使用 Amazon VPC Transit Gateway (TGW) 或 AWS Direct Connect 虛擬私有閘道 (VGW) 連接 VPC

如果您使用的是 TGW 或 VGW，則請將 VPC 連接到 TGW 或 VGW。如需詳細資訊，請參閱 Amazon VPC Transit Gateways 中的 Amazon VPC 附件或 AWS Direct Connect 虛擬私有閘道關聯。

轉換閘道

執行下列命令以連接 Transit Gateway。使用 VPC 的 ID 取代 VPC_ID。使用您在上一個步驟中建立的子網路的 ID 取代 SUBNET_ID1 和 SUBNET_ID2。使用您 TGW 的 ID 取代 TGW_ID。

```
aws ec2 create-transit-gateway-vpc-attachment \
    --vpc-id VPC_ID \
    --subnet-ids SUBNET_ID1 SUBNET_ID2 \
    --transit-gateway-id TGW_ID
```

虛擬私有閘道

執行下列命令以連接 Transit Gateway。使用您 VGW 的 ID 取代 VPN_ID。使用 VPC 的 ID 取代 VPC_ID。

```
aws ec2 attach-vpn-gateway \
    --vpn-gateway-id VPN_ID \
    --vpc-id VPC_ID
```

(選用) 步驟 4：建立路由表

您可以修改 VPC 的主要路由表，也可以建立自訂路由表。下列步驟會建立自訂路由表，其中包含通往內部部署節點和 Pod CIDR 的路由。如需詳細資訊，請參閱子網路路由表。使用 VPC 的 ID 取代 VPC_ID。

```
aws ec2 create-route-table --vpc-id VPC_ID
```

步驟 5：建立內部部署節點和 Pod 的路由

在路由表中，為每個內部部署遠端節點建立路由。您可以修改 VPC 的主要路由表，也可以使用您在上一個步驟中建立的自訂路由表。

以下範例顯示如何為您的內部部署節點和 Pod CIDR 建立路由。在範例中，傳輸閘道 (TGW) 可用於連接 VPC 與內部部署環境。如果您有多個內部部署節點和 Pod CIDR，請對每個 CIDR 重複這些步驟。

- 如果您使用的是網際網路閘道或虛擬私有閘道 (VGW)，則請使用 `--gateway-id` 取代 `--transit-gateway-id`。
- 使用您在上一個步驟中建立的路由表的 ID 取代 `RT_ID`。
- 使用您將用於混合節點的 CIDR 範圍取代 `REMOTE_NODE_CIDR`。
- 使用您將用於為混合節點上執行的 Pod 的 CIDR 範圍取代 `REMOTE_POD_CIDR`。Pod CIDR 範圍會對應至容器聯網介面 (CNI) 組態，而此組態最常使用覆蓋網路內部部署。如需詳細資訊，請參閱the section called “設定 CNI”。
- 使用您 TGW 的 ID 取代 `TGW_ID`。

遠端節點網路

```
aws ec2 create-route \
    --route-table-id RT_ID \
    --destination-cidr-block REMOTE_NODE_CIDR \
    --transit-gateway-id TGW_ID
```

遠端 Pod 網路

```
aws ec2 create-route \
    --route-table-id RT_ID \
    --destination-cidr-block REMOTE_POD_CIDR \
```

```
    --transit-gateway-id TGW_ID
```

(選用) 步驟 6：將子網路和路由表建立關聯

如果您在上一個步驟中建立了自訂路由表，請將您在上一個步驟中建立的每個子網路與您的自訂路由表建立關聯。如果您要修改 VPC 主要路由表，子網路會自動與 VPC 的主要路由表建立關聯，並且您可以略過此步驟。

對先前步驟中建立的每個子網路，執行下列命令。使用您在上一個步驟中建立的路由表取代 RT_ID。使用子網路的 ID 取代 SUBNET_ID。

```
aws ec2 associate-route-table --route-table-id RT_ID --subnet-id SUBNET_ID
```

叢集安全群組組態

持續叢集操作需要為 EKS 叢集安全群組提供下列存取。當您使用設定的遠端節點和 Pod 網路建立或更新叢集時，Amazon EKS 會自動為混合節點建立必要的傳入安全群組規則。由於根據預設，安全群組允許所有傳出流量，Amazon EKS 不會自動修改混合節點的叢集安全群組的傳出規則。如果您想要自訂叢集安全群組，您可以根據下表中的規則限制流量。

Type	通訊協定	Direction	站點	來源	目標	Usage
HTTPS	TCP	傳入	443	遠端節點 CIDR	N/A	kubelet 到 Kubernetes API 伺服器
HTTPS	TCP	傳入	443	遠端 Pod CIDR	N/A	Pod，當 CNI 未針對 Pod 流量使用 NAT 時，需要存取 K8s API 伺服器。
HTTPS	TCP	傳出	10250	N/A	遠端節點 CIDR	Kubernetes API 伺服器到 Kubelet

Type	通訊協定	Direction	站點	來源	目標	Usage
HTTPS	TCP	傳出	Webhook 連接埠	N/A	遠端 Pod CIDR	Kubernetes API 伺服器到 Webhook (如果在混合節點上執行 Webhooks on hybrid nodes)

Important

安全群組規則限制：根據預設，Amazon EC2 安全群組最多有 60 條傳入規則。如果您的叢集安全群組接近此限制，則安全群組傳入規則可能不適用。在此情況下，可能需要在缺少的傳入規則中手動進行新增。

CIDR 清除責任：如果您從 EKS 叢集中移除遠端節點或 Pod 網路，EKS 不會自動移除對應的安全群組規則。您負責從安全群組規則中手動移除未使用的遠端節點或 Pod 網路。

如需有關 Amazon EKS 建立的叢集安全群組的詳細資訊，請參閱 the section called “安全群組要求”。

(選用) 手動安全群組組態

如果您需要建立其他安全群組或修改自動建立的規則，您可以使用下列命令作為參考。根據預設，以下命令會建立一個允許所有傳出存取的安全群組。您可以將傳出存取限制為僅包含上述規則。如果您正在考慮限制傳出規則，建議您先徹底測試所有應用程式和 Pod 連線，之後再將變更的規則套用到生產叢集。

- 在第一個命令中，使用您安全群組的名稱取代 `SG_NAME`
- 在第一個命令中，使用您在上一個步驟中建立的 VPC 的 ID 取代 `VPC_ID`
- 在第二個命令中，使用您在第一個命令中建立的安全群組的 ID 取代 `SG_ID`
- 在第二個命令中，使用混合節點和內部部署網路的值取代 `REMOTE_NODE_CIDR` 和 `REMOTE_POD_CIDR`。

```
aws ec2 create-security-group \
    --group-name SG_NAME \
    --description "security group for hybrid nodes" \
    --vpc-id VPC_ID
```

```
aws ec2 authorize-security-group-ingress \
    --group-id SG_ID \
    --ip-permissions '[{"IpProtocol": "tcp", "FromPort": 443, "ToPort": 443,
 "IpRanges": [{"CidrIp": "REMOTE_NODE_CIDR"}, {"CidrIp": "REMOTE_POD_CIDR"}]}]'
```

準備混合節點的作業系統

Bottlerocket、Amazon Linux 2023 (AL2023)、Ubuntu 和 RHEL 經過持續驗證，可用作混合節點的節點作業系統。Bottlerocket 僅支援 VMware vSphere 環境中 AWS的 。在 Amazon EC2 外部執行時，AWS 支援計劃不涵蓋 AL2023。 Amazon EC2 AL2023 只能在內部部署虛擬化環境中使用，如需詳細資訊，請參閱 Amazon Linux 2023 使用者指南。 AWS 支援與 Ubuntu 和 RHEL 作業系統整合的混合節點，但不支援作業系統本身。

您負責作業系統佈建和管理。在首次測試混合節點時，最簡單的方式是在已佈建的主機上執行 Amazon EKS 混合節點 CLI (nodeadm)。對於生產部署，建議您在作業系統映像中包含 nodeadm，並將它設定為作為 systemd 服務執行，以便在主機啟動時自動將主機加入 Amazon EKS 叢集。如果您在 vSphere 上使用 Bottlerocket 作為節點作業系統，則不需要使用 nodeadm，因為 Bottlerocket 已包含混合節點所需的相依性，並且會在主機啟動時自動連接至您設定的叢集。

⚠ SSM 登入資料提供者所需的 nodeadm 版本

如果您使用 AWS Systems Manager (SSM) 作為混合節點的憑證提供者，則必須使用 nodeadm 版本 1.0.19或更新版本進行新的安裝和升級。舊版 nodeadm包含過期的 SSM 簽署金鑰，並在 nodeadm install期間失敗，並nodeadm upgrade出現下列簽章驗證錯誤：

```
"msg":"Command failed","error":"failed to install ssm installer: validating ssm-
setup-cli signature: Signature Verification Error: No matching signature"
```

若要解決此錯誤，請在執行 或 之前下載最新版本的 nodeadm upgrade。 nodeadm nodeadm install

版本相容性

下表所列的作業系統版本代表相容且經過驗證可用作混合節點的節點作業系統。如果您使用的是不包含在此資料表中的其他作業系統變體或版本，則 AWS Support 不會涵蓋混合節點與作業系統變體或版本的相容性。混合節點與底層基礎結構無關，並可支援 x86 和 ARM 架構。

作業系統	版本
Amazon Linux	Amazon Linux 2023 (AL2023)
Bottlerocket	執行 Kubernetes v1.28 及更新版本的 v1.37.0 及更新版本的 VMware 變體
Ubuntu	Ubuntu 20.04、Ubuntu 22.04、Ubuntu 24.04
Red Hat Enterprise Linux	RHEL 8、RHEL 9

作業系統考量事項

一般

- Amazon EKS 混合節點 CLI (nodeadm) 可用於簡化混合節點元件和相依性的安裝和組態。在作業系統映像建置管道期間或每個內部部署主機的執行時期，您可以執行 `nodeadm install` 程序。如需有關 nodeadm 安裝的元件的詳細資訊，請參閱 the section called “混合節點 nodeadm”。
- 如果您在內部部署環境中使用代理連線至網際網路，安裝和升級程序需要額外的作業系統組態，才能將套件管理工具設定為使用代理。如需說明，請參閱 the section called “設定代理”。

Bottlerocket

- 連接 Bottlerocket 節點的步驟和工具與其他作業系統的步驟不同，並會在 the section called “使用 Bottlerocket 連接混合節點” 中單獨介紹，而不會涵蓋在 the section called “連接混合節點” 的步驟中。
- Bottlerocket 的步驟不會使用混合節點 CLI 工具 nodeadm。
- EKS 混合節點僅支援 Bottlerocket 版本 1.37.0 及更新版本的 VMware 變體。Bottlerocket 的 VMware 變體適用於 Kubernetes 版本 1.28 及更新版本。不支援其他 Bottlerocket 變體作為混合節點作業系統。注意： Bottlerocket 的 VMware 變體僅適用於 x86_64 架構。

Containerd

- Containerd 是標準 Kubernetes 容器執行時期，也是混合節點以及所有 Amazon EKS 節點運算類型的相依性。Amazon EKS 混合節點 CLI (nodeadm) 會嘗試在 `nodeadm install` 程序期間安裝 containerd。您可以使用 `--containerd-source` 命令列選項，在 `nodeadm install` 執行時期設定 containerd 安裝。有效選項為 `none`、`distro` 和 `docker`。如果您使用的是 RHEL，則 `distro` 不是有效選項，並且您可以將 nodeadm 設定為從 Docker 的儲存庫安裝 containerd 建置，也可以手動安裝 containerd。使用 AL2023 或 Ubuntu 時，nodeadm 預設為從作業系統分佈中安裝 containerd。如果您不希望 nodeadm 安裝 containerd，則請使用 `--containerd-source none` 選項。

Ubuntu

- 如果您使用的是 Ubuntu 24.04，您可能需要更新您的 containerd 版本，或變更 AppArmor 組態以採用允許 Pod 正確終止的修正，請參閱 Ubuntu #2065423。需要重新啟動才能將變更套用至 AppArmor 設定檔。Ubuntu 24.04 的最新版本在其套件管理工具中具有更新的 containerd 版本，且帶有修正 (containerd 版本 1.7.19+)。

ARM

- 如果您使用的是 ARM 硬體，則需要具有密碼編譯延伸功能的 ARMv8.2 相容處理器 (ARMv8.2+crypto)，才能執行 EKS kube-proxy 附加元件的版本 1.31 及更新版本。Raspberry Pi 5 之前的所有 Raspberry Pi 系統，以及 Cortex-A72 型處理器皆不符合此要求。解決方法是：您可以繼續使用 EKS kube-proxy 附加元件的版本 1.30，直到 2026 年 7 月延長支援結束為止，請參閱 Kubernetes 發布行事曆，或使用來自上游的自訂 kube-proxy 映像。
- kube-proxy 日誌中的下列錯誤訊息會指出這種不相容性：

```
Fatal glibc error: This version of Amazon Linux requires a newer ARM64 processor
 compliant with at least ARM architecture 8.2-a with Cryptographic extensions. On EC2
 this is Graviton 2 or later.
```

建置作業系統映像

Amazon EKS 提供範例 Packer 範本，您可使用這些範本來建立包含 nodeadm 的作業系統映像，並將其設定為在主機啟動時執行。建議執行此程序，以避免在每個主機上個別提取混合節點相依性，並自動化混合節點引導程序。您可以搭配 Ubuntu 22.04、Ubuntu 24.04、RHEL 8 或 RHEL 9 ISO 映像使用範例 Packer 範本，並且可以使用這些格式輸出映像：OVA、Qcow2 或 raw。

先決條件

使用範例 Packer 範本之前，您必須在執行 Packer 的機器上安裝下列內容。

- Packer 版本 1.11.0 或更新版本。如需有關安裝 Packer 的指示，請參閱 Packer 文件中的安裝 Packer。
- 如果建置 OVA，則為 VMware vSphere 外掛程式 1.4.0 或更新版本
- 如果是建置 Qcow2 或原始映像，則為 QEMU 外掛程式版本 1.x

設定環境變數

在執行 Packer 建置之前，請在執行 Packer 的機器上設定下列環境變數。

一般

必須設定下列環境變數，才能運用所有作業系統和輸出格式建置映像。

環境變數	Type	說明
PKR_SSH_PASSWORD	String	Packer 會在佈建時使用 `ssh_username` 和 `ssh_password` 變數，將 SSH 傳送至建立的機器。這需要與在個別作業系統的啟動或使用者資料檔案中建立初始使用者的密碼相符。預設值設定為 "builder" 或 "ubuntu"，視作業系統而定。設定密碼時，請務必在對應的 `ks.cfg` 或 `user-data` 檔案中對其進行變更，以保持相符。
ISO_URL	String	要使用的 ISO 的 URL。可以是從伺服器下載的 Web 連結，也可以是本機檔案的絕對路徑
ISO_CHECKSUM	String	所提供 ISO 的關聯檢查總和。

環境變數	Type	說明
CREDENTIAL_PROVIDER	String	混合節點的憑證提供者。對於 SSM 混合啟用，有效值為 `ssm` (預設)，而對於 IAM Roles Anywhere，有效值為 `iam`
K8S_VERSION	String	混合節點的 Kubernetes 版本 (例如 `1.31`)。如需支援的 Kubernetes 版本，請參閱 Amazon EKS 支援的版本。
NODEADM_ARCH	String	`nodeadm install` 的架構。選取 `amd` 或 `arm`。

RHEL

如果您使用的是 RHEL，則必須設定下列環境變數。

環境變數	Type	說明
RH_USERNAME	String	RHEL 訂閱管理員使用者名稱
RH_PASSWORD	String	RHEL 訂閱管理員密碼
RHEL_VERSION	String	要使用的 Rhel iso 版本。有效值為 8 或 9。

Ubuntu

不需要 Ubuntu 專屬環境變數。

vSphere

如果您要建置 VMware vSphere OVA，則必須設定下列環境變數。

環境變數	Type	說明
VSPHERE_SERVER	String	vSphere 伺服器地址
VSPHERE_USER	String	vSphere 使用者名稱
VSPHERE_PASSWORD	String	vSphere 密碼
VSPHERE_DATACENTER	String	vSphere 資料中心名稱
VSPHERE_CLUSTER	String	vSphere 叢集名稱
VSPHERE_DATASTORE	String	vSphere 資料儲存名稱
VSPHERE_NETWORK	String	vSphere 網路名稱
VSPHERE_OUTPUT_FOLDER	String	範本的 vSphere 輸出資料夾

QEMU

環境變數	Type	說明
PACKER_OUTPUT_FORMAT	String	QEMU 建置器的輸出格式。有效值為 `qcow2` 和 `raw`。

驗證範本

執行建置之前，請設定環境變數，然後使用下列命令驗證範本。如果您對範本使用不同的名稱，請取代 `template.pkr.hcl`。

```
packer validate template.pkr.hcl
```

建置映像

使用下列命令建置映像，並使用 `-only` 旗標來指定映像的目標和作業系統。如果您對範本使用不同的名稱，請取代 `template.pkr.hcl`。

vSphere OVA

> **Note**
>
> 如果您搭配 vSphere 使用 RHEL，您需要將啟動檔案轉換為 OEMDRV 映像，並將其作為 ISO 傳遞以進行引導。如需詳細資訊，請參閱 EKS 混合節點 GitHub 儲存庫中的 Packer Readme。

Ubuntu 22.04 OVA

```
packer build -only=general-build.vsphere-iso.ubuntu22 template.pkr.hcl
```

Ubuntu 24.04 OVA

```
packer build -only=general-build.vsphere-iso.ubuntu24 template.pkr.hcl
```

RHEL 8 OVA

```
packer build -only=general-build.vsphere-iso.rhel8 template.pkr.hcl
```

RHEL 9 OVA

```
packer build -only=general-build.vsphere-iso.rhel9 template.pkr.hcl
```

QEMU

> **Note**
>
> 如果您要為與建置器主機不相符的特定主機 CPU 建置映像，請參閱 QEMU 文件以取得與您主機 CPU 相符的名稱，並在執行下列命令時，使用具有主機 CPU 名稱的 `-cpu` 旗標。

Ubuntu 22.04 Qcow2 / Raw

```
packer build -only=general-build.qemu.ubuntu22 template.pkr.hcl
```

Ubuntu 24.04 Qcow2 / Raw

```
packer build -only=general-build.qemu.ubuntu24 template.pkr.hcl
```

RHEL 8 Qcow2 / Raw

```
packer build -only=general-build.qemu.rhel8 template.pkr.hcl
```

RHEL 9 Qcow2 / Raw

```
packer build -only=general-build.qemu.rhel9 template.pkr.hcl
```

透過使用者資料傳遞 nodeadm 組態

您可以透過 cloud-init 傳遞使用者資料中 nodeadm 的組態，以在主機啟動時設定混合節點並自動將其連接至 EKS 叢集。以下範例說明如何在使用 VMware vSphere 作為混合節點的基礎結構時完成此操作。

1. 依照 GitHub govc 上 govc readme 中的指示安裝 CLI。
2. 在上一節中執行 Packer 建置並佈建範本後，您可以使用下列內容複製範本，進而建立多個不同的節點。對於要建立且將用於混合節點的每個新 VM，您必須複製相應的範本。使用您環境的值取代以下命令中的變數。當您透過 metadata.yaml 檔案插入 VM 的名稱時，以下命令中的 VM_NAME 可用作您的 NODE_NAME。

   ```
   govc vm.clone -vm "/PATH/TO/TEMPLATE" -ds="YOUR_DATASTORE" \
       -on=false -template=false -folder=/FOLDER/TO/SAVE/VM "VM_NAME"
   ```

3. 複製每個新 VM 的範本之後，請為您的 VM 建立 userdata.yaml 和 metadata.yaml。在以下步驟中，您的 VM 可以共用相同的 userdata.yaml 和 metadata.yaml，並且您將以每個 VM 為基礎填入。nodeadm 組態可由 userdata.yaml 的 write_files 區段建立並定義。以下範例使用 AWS SSM 混合啟用做為混合節點的內部部署憑證提供者。如需有關 nodeadm 組態的詳細資訊，請參閱 the section called “混合節點 nodeadm”。

   userdata.yaml：

   ```
   #cloud-config
   users:
     - name: # username for login. Use 'builder' for RHEL or 'ubuntu' for Ubuntu.
       passwd: # password to login. Default is 'builder' for RHEL.
       groups: [adm, cdrom, dip, plugdev, lxd, sudo]
       lock-passwd: false
       sudo: ALL=(ALL) NOPASSWD:ALL
       shell: /bin/bash
   ```

```
write_files:
  - path: /usr/local/bin/nodeConfig.yaml
    permissions: '0644'
    content: |
      apiVersion: node.eks.aws/v1alpha1
      kind: NodeConfig
      spec:
          cluster:
              name: # Cluster Name
              region: # AWS region
          hybrid:
              ssm:
                  activationCode: # Your ssm activation code
                  activationId: # Your ssm activation id

runcmd:
  - /usr/local/bin/nodeadm init -c file:///usr/local/bin/nodeConfig.yaml >> /var/log/
nodeadm-init.log 2>&1
```

metadata.yaml：

為您的環境建立 metadata.yaml。在檔案中保留 "$NODE_NAME" 變數格式，因為會在後續步驟中填入這些值。

```
instance-id: "$NODE_NAME"
local-hostname: "$NODE_NAME"
network:
  version: 2
  ethernets:
    nics:
      match:
        name: ens*
      dhcp4: yes
```

4. 使用下列命令將 userdata.yaml 和 metadata.yaml 檔案新增為 gzip+base64 字串。應該為您建立的每個 VM 執行下列命令。使用您要更新的 VM 的名稱取代 VM_NAME。

```
export NODE_NAME="VM_NAME"
export USER_DATA=$(gzip -c9 <userdata.yaml | base64)

govc vm.change -dc="YOUR_DATASTORE" -vm "$NODE_NAME" -e
 guestinfo.userdata="${USER_DATA}"
```

```
govc vm.change -dc="YOUR_DATASTORE" -vm "$NODE_NAME" -e
 guestinfo.userdata.encoding=gzip+base64

envsubst '$NODE_NAME' < metadata.yaml > metadata.yaml.tmp
export METADATA=$(gzip -c9 <metadata.yaml.tmp | base64)

govc vm.change -dc="YOUR_DATASTORE" -vm "$NODE_NAME" -e
 guestinfo.metadata="${METADATA}"
govc vm.change -dc="YOUR_DATASTORE" -vm "$NODE_NAME" -e
 guestinfo.metadata.encoding=gzip+base64
```

5. 開啟新 VM 的電源，而這應該會自動連接到您設定的 EKS 叢集。

```
govc vm.power -on "${NODE_NAME}"
```

準備混合節點的憑證

Amazon EKS 混合節點使用由 AWS SSM 混合啟用或 IAM Roles Anywhere AWS 佈建的臨時 IAM 登入資料來驗證 Amazon EKS 叢集。您必須將 AWS SSM 混合啟用或 AWS IAM Roles Anywhere 與 Amazon EKS 混合節點 CLI () 搭配使用nodeadm。您不應該同時使用 AWS SSM 混合啟用和 AWS IAM Roles Anywhere。如果您現有的公有金鑰基礎設施 (PKI) 沒有憑證授權機構 (CA) 和內部部署環境的憑證，建議您使用 AWS SSM 混合啟用。如果您有現有的 PKI 和現場部署憑證，請使用 AWS IAM Roles Anywhere。

混合節點 IAM 角色

在將混合節點連接到 Amazon EKS 叢集之前，您必須建立 IAM 角色，以搭配混合節點憑證的 AWS SSM 混合啟用或 AWS IAM Roles Anywhere 使用。建立叢集後，您將搭配 Amazon EKS 存取項目或 aws-auth ConfigMap 項目使用此角色，以將 IAM 角色映射至 Kubernetes 角色型存取控制 (RBAC)。如需建立混合節點 IAM 角色與 Kubernetes RBAC 的關聯的詳細資訊，請參閱 the section called “準備叢集存取”。

混合節點 IAM 角色必須具有下列許可。

- nodeadm 使用 eks:DescribeCluster動作來收集您要連接混合節點之叢集相關資訊的許可。如果您未啟用 eks:DescribeCluster動作，則必須在傳遞給nodeadm init命令的節點組態中傳遞 Kubernetes API 端點、叢集 CA 套件和服務 IPv4 CIDR。
- nodeadm 使用 eks:ListAccessEntries動作列出您要連接混合節點之叢集上存取項目的許可。如果您未啟用 eks:ListAccessEntries動作，則必須在執行 nodeadm init命令時傳遞 --skip cluster-access-validation旗標。

- kubelet 使用 Amazon Elastic Container Registry (Amazon ECR) 中容器映像的許可，如 AmazonEC2ContainerRegistryPullOnly 政策所定義。
- 如果使用 AWS SSM，nodeadm init則允許 使用https://docs.aws.amazon.com/aws-managed-policy/latest/reference/AmazonSSMManagedInstanceCore.html政策中定義的 AWS SSM 混合啟用。
- 如果使用 AWS SSM，則使用 ssm:DeregisterManagedInstance動作和 ssm:DescribeInstanceInformation動作nodeadm uninstall取消註冊執行個體的許可。
- (選用) Amazon EKS Pod 身分識別代理程式使用 eks-auth:AssumeRoleForPodIdentity 動作擷取 Pod 憑證的許可。

設定 AWS SSM 混合啟用

⚠ SSM 登入資料提供者所需的 nodeadm 版本

如果您使用 AWS Systems Manager (SSM) 作為混合節點的憑證提供者，則必須使用 nodeadm 版本 1.0.19或更新版本進行新的安裝和升級。舊版 nodeadm包含過期的 SSM 簽署金鑰，並在 nodeadm install期間失敗，並nodeadm upgrade出現下列簽章驗證錯誤：

```
"msg":"Command failed","error":"failed to install ssm installer: validating ssm-setup-cli signature: Signature Verification Error: No matching signature"
```

若要解決此錯誤，請在執行 或 之前下載最新版本的 nodeadm upgrade。 nodeadm nodeadm install

⚠ Important

Systems Manager 會立即將啟用代碼和 ID 傳回主控台或命令視窗，視您如何建立啟用而定。複製此資訊，並將其存放在安全的地方。如果您離開主控台或關閉命令視窗，您可能會遺失此資訊。如果您遺失此資訊，您必須建立新的啟用。

在設定 AWS SSM 混合啟用之前，您必須建立並設定混合節點 IAM 角色。如需詳細資訊，請參閱the section called “建立混合節點 IAM 角色”。遵循 Systems Manager 使用者指南中的建立混合啟用中的指示，向 Systems Manager 註冊節點，為您的混合節點建立 AWS SSM 混合啟用。 AWS 當您向 Amazon EKS 叢集將主機註冊為混合節點nodeadm時，請使用 啟用碼和 ID。在您為混合節點建立及準備好 Amazon EKS 叢集後，您可以稍後再返回此步驟。

根據預設， AWS SSM 混合啟用會處於作用中狀態 24 小時。或者，您亦可在以時間戳格式建立混合啟用時指定 `--expiration-date`，例如 `2024-08-01T00:00:00`。當您使用 AWS SSM 做為登入資料提供者時，混合節點的節點名稱無法設定，且由 AWS SSM 自動產生。您可以在 Fleet Manager 下的 AWS Systems Manager 主控台中檢視和管理 AWS SSM 受管執行個體。每個 AWS 區域每個帳戶最多可註冊 1，000 個標準混合啟用節點，無需額外費用。不過，登錄超過 1,000 個混合節點需要啟用進階執行個體層。使用 advanced-instances 方案會產生費用，而且 Amazon EKS 混合節點定價中未包含該方案。如需詳細資訊，請參閱 AWS Systems Manager 定價。

請參閱以下範例，了解如何使用混合節點 IAM 角色建立 AWS SSM 混合啟用。當您針對混合節點登入資料使用 AWS SSM 混合啟用時，混合節點的名稱具有 格式，`mi-012345678abcdefgh`且 AWS SSM 佈建的臨時登入資料有效期為 1 小時。使用 AWS SSM 做為登入資料提供者時，您無法變更節點名稱或登入資料持續時間。 AWS SSM 會自動輪換臨時登入資料，輪換不會影響節點或應用程式的狀態。

我們建議您每個 EKS 叢集使用一個 AWS SSM 混合啟用，將混合節點 IAM 角色的 AWS SSM `ssm:DeregisterManagedInstance`許可範圍限定為只能取消註冊與您的 AWS SSM 混合啟用相關聯的執行個體。在此頁面上的範例中，使用具有 EKS 叢集 ARN 的標籤，可用於將 AWS SSM 混合啟用映射至 EKS 叢集。或者，您也可以根據您的許可界限和要求，使用您偏好的標籤和方法來界定 AWS SSM 許可範圍。以下命令中的 `REGISTRATION_LIMIT`選項是用來限制可使用 AWS SSM 混合啟用之機器數量的整數 （ 例如 `10`)

```
aws ssm create-activation \
     --region AWS_REGION \
     --default-instance-name eks-hybrid-nodes \
     --description "Activation for EKS hybrid nodes" \
     --iam-role AmazonEKSHybridNodesRole \
     --tags Key=EKSClusterARN,Value=arn:aws: eks:AWS_REGION:AWS_ACCOUNT_ID:cluster/
CLUSTER_NAME \
     --registration-limit REGISTRATION_LIMIT
```

如需 AWS SSM 混合啟用可用組態設定的詳細資訊，請參閱建立混合啟用以向 Systems Manager 註冊節點的說明。

隨處設定 AWS IAM 角色

請遵循《IAM Roles Anywhere 使用者指南》中 IAM Roles Anywhere 入門上的指示，設定您要用於混合節點 IAM 角色的臨時 IAM 憑證的信任錨和設定檔。建立設定檔時，您可以選擇建立時不新增任何角色。您可以建立此設定檔，返回可建立混合節點 IAM 角色的步驟，然後在建立角色後，將其新增至您的設定檔。或者，您可以在此頁面稍後使用 AWS CloudFormation 步驟來完成混合節點的 IAM Roles Anywhere 設定。

當您將混合節點 IAM 角色新增至設定檔時，請在 IAM Roles Anywhere 主控台的編輯設定檔頁面底部的自訂角色工作階段名稱面板中選取接受自訂角色工作階段名稱。 AWS 這會對應至 `CreateProfile` API 的 acceptRoleSessionName 欄位。這可讓您在引導程序過程中，於傳遞給 `nodeadm` 的組態中，為混合節點提供自訂節點名稱。需要在 `nodeadm init` 過程中傳遞自訂節點名稱。建立設定檔後，您可以更新您的設定檔，以接受自訂角色工作階段名稱。

您可以透過 IAM Roles Anywhere AWS 設定檔的 durationSeconds 欄位，使用 IAM Roles Anywhere AWS 設定憑證有效性持續時間。預設持續時間為 1 小時，最長為 12 小時。混合節點 IAM 角色上的`MaxSessionDuration`設定必須大於 IAM Roles Anywhere AWS 設定檔上的`durationSeconds`設定。如需有關 `MaxSessionDuration` 的詳細資訊，請參閱 UpdateRole API 文件。

從憑證認證機構 (CA) 產生的每個機器憑證和金鑰，必須放置在每個混合節點的 `/etc/iam/pki` 目錄中，其中包含的檔案名稱 `server.pem` 指代憑證，`server.key` 指代金鑰。

建立混合節點 IAM 角色

若要執行本節中的步驟，使用 AWS 主控台或 CLI 的 IAM AWS 主體必須具有下列許可。

- `iam:CreatePolicy`
- `iam:CreateRole`
- `iam:AttachRolePolicy`
- 如果使用 AWS IAM Roles Anywhere
  - `rolesanywhere:CreateTrustAnchor`
  - `rolesanywhere:CreateProfile`
  - `iam:PassRole`

AWS CloudFormation

如果您尚未安裝和設定 AWS CLI。請參閱安裝或更新至最新版本的 AWS CLI。

AWS SSM 混合啟用的步驟

CloudFormation 堆疊會使用上述許可來建立混合節點 IAM 角色。CloudFormation 範本不會建立 AWS SSM 混合啟用。

1. 下載混合節點的 AWS SSM CloudFormation 範本：

```
curl -OL 'https://raw.githubusercontent.com/aws/eks-hybrid/refs/heads/main/example/
hybrid-ssm-cfn.yaml'
```

2. 使用下列選項建立 cfn-ssm-parameters.json：

   a. 使用混合節點 IAM 角色的名稱取代 ROLE_NAME。根據預設，如果您未指定名稱，CloudFormation 範本會使用 AmazonEKSHybridNodesRole 作為其建立的角色的名稱。

   b. TAG_KEY 將 取代為您在建立 AWS SSM 混合啟用時使用的 AWS SSM 資源標籤金鑰。標籤索引鍵和標籤值的組合用於 的條件ssm:DeregisterManagedInstance，只允許混合節點 IAM 角色取消註冊與您的 AWS SSM 混合啟用相關聯的 AWS SSM 受管執行個體。在 CloudFormation 範本中，TAG_KEY 預設為 EKSClusterARN。

   c. TAG_VALUE 將 取代為您建立 AWS SSM 混合啟用時使用的 AWS SSM 資源標籤值。標籤索引鍵和標籤值的組合用於 的條件ssm:DeregisterManagedInstance，只允許混合節點 IAM 角色取消註冊與您的 AWS SSM 混合啟用相關聯的 AWS SSM 受管執行個體。如果您使用 EKSClusterARN 的預設值 TAG_KEY，則請將您的 EKS 叢集 ARN 作為 TAG_VALUE 進行傳遞。EKS 叢集 ARN 的格式為 arn:aws: eks:AWS_REGION:AWS_ACCOUNT_ID:cluster/CLUSTER_NAME。

   ```
   {
     "Parameters": {
       "RoleName": "ROLE_NAME",
       "SSMDeregisterConditionTagKey": "TAG_KEY",
       "SSMDeregisterConditionTagValue": "TAG_VALUE"
     }
   }
   ```

3. 部署 CloudFormation 堆疊。使用 CloudFormation 堆疊的名稱取代 STACK_NAME。

```
aws cloudformation deploy \
    --stack-name STACK_NAME \
    --template-file hybrid-ssm-cfn.yaml \
    --parameter-overrides file://cfn-ssm-parameters.json \
    --capabilities CAPABILITY_NAMED_IAM
```

IAM Roles Anywhere AWS 的步驟

CloudFormation 堆疊會使用您設定的憑證授權機構 AWS (CA) 建立 IAM Roles Anywhere 信任錨點、建立 AWS IAM Roles Anywhere 設定檔，以及使用先前概述的許可建立混合節點 IAM 角色。

1. 設定憑證認證機構 (CA)
   a. 若要使用 AWS Private CA 資源，請開啟 AWS Private Certificate Authority 主控台。請遵循《AWS 私有 CA 使用者指南》中的指示。
   b. 若要使用外部 CA，請遵循 CA 提供的指示。您將在後續步驟中提供憑證內文。
   c. 從公有 CA 核發的憑證不能用作信任錨。
2. 下載混合節點的 AWS IAM Roles Anywhere CloudFormation 範本

```
curl -OL 'https://raw.githubusercontent.com/aws/eks-hybrid/refs/heads/main/example/
hybrid-ira-cfn.yaml'
```

3. 使用下列選項建立 `cfn-iamra-parameters.json`：
   a. 使用混合節點 IAM 角色的名稱取代 `ROLE_NAME`。根據預設，如果您未指定名稱，CloudFormation 範本會使用 `AmazonEKSHybridNodesRole` 作為其建立的角色的名稱。
   b. 使用可唯一識別您主機的每個機器憑證屬性取代 `CERT_ATTRIBUTE`。在您將混合節點連接至叢集時，您使用的憑證屬性必須與您用於 nodeadm 組態的 nodeName 相符。如需更多資訊，請參閱the section called “混合節點 nodeadm”。根據預設，CloudFormation 範本會將 `${aws:PrincipalTag/x509Subject/CN}` 用作 `CERT_ATTRIBUTE`，而其會對應至每個機器憑證的 CN 欄位。或者，您可以將 `$(aws:PrincipalTag/x509SAN/Name/CN}` 作為 `CERT_ATTRIBUTE` 進行傳遞。
   c. 使用您 CA 的憑證內文取代 `CA_CERT_BODY`，且沒有分行符號。`CA_CERT_BODY` 必須是隱私權增強式郵件 (PEM) 格式。如果您的 CA 憑證採用 PEM 格式，則請先移除分行符號以及 BEGIN CERTIFICATE 和 END CERTIFICATE 行，然後再將 CA 憑證內文放入 `cfn-iamra-parameters.json` 檔案中。

```
{
  "Parameters": {
    "RoleName": "ROLE_NAME",
    "CertAttributeTrustPolicy": "CERT_ATTRIBUTE",
    "CABundleCert": "CA_CERT_BODY"
  }
}
```

4. 部署 CloudFormation 範本。使用 CloudFormation 堆疊的名稱取代 `STACK_NAME`。

```
aws cloudformation deploy \
    --stack-name STACK_NAME \
    --template-file hybrid-ira-cfn.yaml \
    --parameter-overrides file://cfn-iamra-parameters.json
```

```
    --capabilities CAPABILITY_NAMED_IAM
```

AWS CLI

如果您尚未安裝和設定 AWS CLI。請參閱安裝或更新至最新版本的 AWS CLI。

建立 EKS 描述叢集政策

1. 使用下列內容建立名為 eks-describe-cluster-policy.json 的檔案：

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "eks:DescribeCluster"
            ],
            "Resource": "*"
        }
    ]
}
```

2. 使用下列命令建立政策：

```
aws iam create-policy \
    --policy-name EKSDescribeClusterPolicy \
    --policy-document file://eks-describe-cluster-policy.json
```

AWS SSM 混合啟用的步驟

1. 使用下列內容建立名為 eks-hybrid-ssm-policy.json 的檔案。此政策會將授與兩個動作 (ssm:DescribeInstanceInformation 和 ssm:DeregisterManagedInstance) 的許可。此政策會根據您在信任政策中指定的資源標籤，將ssm:DeregisterManagedInstance許可限制為與您的 SSM 混合啟用相關聯的 AWS SSM AWS 受管執行個體。
   a. AWS_REGION 將 取代為 AWS SSM 混合啟用 AWS 的區域。
   b. 將 取代AWS_ACCOUNT_ID為 AWS 您的帳戶 ID。
   c. TAG_KEY 將 取代為您在建立 AWS SSM 混合啟用時使用的 AWS SSM 資源標籤金鑰。標籤索引鍵和標籤值的組合用於 的條件ssm:DeregisterManagedInstance，只允許混合節點 IAM 角

色取消註冊與您的 AWS SSM 混合啟用相關聯的 AWS SSM 受管執行個體。在 CloudFormation 範本中，TAG_KEY 預設為 EKSClusterARN。

d. TAG_VALUE 將 取代為您建立 AWS SSM 混合啟用時使用的 AWS SSM 資源標籤值。標籤索引鍵和標籤值的組合用於 的條件ssm:DeregisterManagedInstance，只允許混合節點 IAM 角色取消註冊與您的 AWS SSM 混合啟用相關聯的 AWS SSM 受管執行個體。如果您使用 EKSClusterARN 的預設值 TAG_KEY，則請將您的 EKS 叢集 ARN 作為 TAG_VALUE 進行傳遞。EKS 叢集 ARN 的格式為 arn:aws: eks:AWS_REGION:AWS_ACCOUNT_ID:cluster/CLUSTER_NAME。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "ssm:DescribeInstanceInformation",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "ssm:DeregisterManagedInstance",
            "Resource": "arn:aws:ssm:us-east-1:123456789012:managed-instance/*",
            "Condition": {
                "StringEquals": {
                    "ssm:resourceTag/TAG_KEY": "TAG_VALUE"
                }
            }
        }
    ]
}
```

2. 使用下列命令建立政策

```
aws iam create-policy \
    --policy-name EKSHybridSSMPolicy \
    --policy-document file://eks-hybrid-ssm-policy.json
```

3. 建立名為 eks-hybrid-ssm-trust.json 的檔案。AWS_REGION 將 取代為 AWS SSM 混合啟用 AWS 的區域，並將 AWS_ACCOUNT_ID取代為 AWS 您的帳戶 ID。

```
{
   "Version":"2012-10-17",
```

```
   "Statement":[
      {
         "Sid":"",
         "Effect":"Allow",
         "Principal":{
            "Service":"ssm.amazonaws.com"
         },
         "Action":"sts:AssumeRole",
         "Condition":{
            "StringEquals":{
               "aws:SourceAccount":"123456789012"
            },
            "ArnEquals":{
               "aws:SourceArn":"arn:aws:ssm:us-east-1:123456789012:*"
            }
         }
      }
   ]
}
```

4. 使用下列命令建立角色。

```
aws iam create-role \
    --role-name AmazonEKSHybridNodesRole \
    --assume-role-policy-document file://eks-hybrid-ssm-trust.json
```

5. 連接 EKSHybridSSMPolicy 和您在先前步驟中建立的 EKSDescribeClusterPolicy。將 取代AWS_ACCOUNT_ID為 AWS 您的帳戶 ID。

```
aws iam attach-role-policy \
    --role-name AmazonEKSHybridNodesRole \
    --policy-arn arn:aws: iam::AWS_ACCOUNT_ID:policy/EKSDescribeClusterPolicy
```

```
aws iam attach-role-policy \
    --role-name AmazonEKSHybridNodesRole \
    --policy-arn arn:aws: iam::AWS_ACCOUNT_ID:policy/EKSHybridSSMPolicy
```

6. 連接 AmazonEC2ContainerRegistryPullOnly和 AmazonSSMManagedInstanceCore AWS 受管政策。

```
aws iam attach-role-policy \
    --role-name AmazonEKSHybridNodesRole \
```

```
    --policy-arn arn:aws: iam::aws:policy/AmazonEC2ContainerRegistryPullOnly
```

```
aws iam attach-role-policy \
    --role-name AmazonEKSHybridNodesRole \
    --policy-arn arn:aws: iam::aws:policy/AmazonSSMManagedInstanceCore
```

IAM Roles Anywhere AWS 的步驟

若要使用 AWS IAM Roles Anywhere，您必須在建立混合節點 AWS IAM 角色之前設定 IAM Roles Anywhere 信任錨點。如需說明，請參閱 the section called “隨處設定 AWS IAM 角色”。

1. 建立名為 `eks-hybrid-iamra-trust.json` 的檔案。使用您在 the section called “隨處設定 AWS IAM 角色” 步驟中建立的信任錨的 ARN 取代 `TRUST_ANCHOR  ARN`。此信任政策中的 條件會限制 AWS IAM Roles Anywhere 擔任混合節點 IAM 角色的能力，只有在角色工作階段名稱符合安裝在混合節點上 x509 憑證中的 CN 時，才能交換臨時 IAM 憑證。或者，您也可以使用其他憑證屬性來唯一識別您的節點。您在信任政策中使用的憑證屬性必須對應至您在 nodeadm 組態中設定的 nodeName。如需更多資訊，請參閱the section called “混合節點 nodeadm”。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "rolesanywhere.amazonaws.com"
            },
            "Action": [
                "sts:TagSession",
                "sts:SetSourceIdentity"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:PrincipalTag/x509Subject/CN": "${aws:PrincipalTag/
x509Subject/CN}"
                },
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:rolesanywhere:us-
east-1:123456789012:trust-anchor/TA_ID"
                }
            }
```

```
        },
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "rolesanywhere.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "sts:RoleSessionName": "${aws:PrincipalTag/x509Subject/CN}",
                    "aws:PrincipalTag/x509Subject/CN": "${aws:PrincipalTag/
x509Subject/CN}"
                },
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:rolesanywhere:us-
east-1:123456789012:trust-anchor/TA_ID"
                }
            }
        }
    ]
}
```

2. 使用下列命令建立角色。

```
aws iam create-role \
    --role-name AmazonEKSHybridNodesRole \
    --assume-role-policy-document file://eks-hybrid-iamra-trust.json
```

3. 連接您在先前步驟中建立的 EKSDescribeClusterPolicy。將 取代AWS_ACCOUNT_ID為 AWS 您的帳戶 ID。

```
aws iam attach-role-policy \
    --role-name AmazonEKSHybridNodesRole \
    --policy-arn arn:aws: iam::AWS_ACCOUNT_ID:policy/EKSDescribeClusterPolicy
```

4. 連接 AmazonEC2ContainerRegistryPullOnly AWS 受管政策

```
aws iam attach-role-policy \
    --role-name AmazonEKSHybridNodesRole \
    --policy-arn arn:aws: iam::aws:policy/AmazonEC2ContainerRegistryPullOnly
```

AWS 管理主控台

建立 EKS 描述叢集政策

1. 開啟 Amazon IAM 主控台
2. 在左側導覽窗格中選擇 Policies (政策)。
3. 在政策頁面上，選擇建立政策。
4. 在指定許可頁面，在選取服務面板中，選擇 EKS。
   a. 篩選 DescribeCluster 的動作，然後選取 DescribeCluster Read 動作。
   b. 選擇下一步。
5. 在檢閱和建立頁面上
   a. 輸入政策的政策名稱，例如 `EKSDescribeClusterPolicy`。
   b. 選擇建立政策。

AWS SSM 混合啟用的步驟

1. 開啟 Amazon IAM 主控台
2. 在左側導覽窗格中選擇 Policies (政策)。
3. 在政策頁面上，選擇建立政策。
4. 在指定許可頁面上，在政策編輯器右上角導覽中，選擇 JSON。貼上下列程式碼片段。`AWS_REGION` 將 取代為 AWS SSM 混合啟用 AWS 的區域，並將 取代`AWS_ACCOUNT_ID`為 AWS 您的帳戶 ID。`TAG_VALUE` 使用您在建立 AWS SSM 混合啟用時使用的 AWS SSM 資源標籤金鑰取代 `TAG_KEY`和 。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "ssm:DescribeInstanceInformation",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "ssm:DeregisterManagedInstance",
            "Resource": "arn:aws:ssm:us-east-1:123456789012:managed-instance/*",
            "Condition": {
```

```
                "StringEquals": {
                    "ssm:resourceTag/TAG_KEY": "TAG_VALUE"
                }
            }
        }
    ]
}
```

a. 選擇下一步。

5. 在檢閱和建立頁面上。

a. 輸入政策的政策名稱，例如 EKSHybridSSMPolicy

b. 選擇建立政策。

6. 在左側導覽窗格中，選擇 Roles (角色)。

7. 在 Roles (角色) 頁面上，選擇 Create role (建立角色)。

8. 在 Select trusted entity (選取信任的實體) 頁面上，執行以下作業：

a. 在可信實體類型區段中，選擇自訂信任政策。將下列政策貼到自訂信任政策編輯器。AWS_REGION 將 取代為 AWS SSM 混合啟用 AWS 的區域，並將 AWS_ACCOUNT_ID取代為 AWS 您的帳戶 ID。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"",
         "Effect":"Allow",
         "Principal":{
            "Service":"ssm.amazonaws.com"
         },
         "Action":"sts:AssumeRole",
         "Condition":{
            "StringEquals":{
               "aws:SourceAccount":"123456789012"
            },
            "ArnEquals":{
               "aws:SourceArn":"arn:aws:ssm:us-east-1:123456789012:*"
            }
         }
      }
   ]
```

```
}
```

b. 選擇下一步。

9. 在新增許可頁面上，連接自訂政策或執行下列動作：

a. 在篩選政策方塊中，輸入 EKSDescribeClusterPolicy 或您在上文建立的政策名稱。勾選搜尋結果中您的政策名稱左側的核取方塊。

b. 在篩選政策方塊中，輸入 EKSHybridSSMPolicy 或您在上文建立的政策名稱。勾選搜尋結果中您的政策名稱左側的核取方塊。

c. 在 Filter policies (篩選政策) 方塊中，輸入 AmazonEC2ContainerRegistryPullOnly。勾選搜尋結果中 AmazonEC2ContainerRegistryPullOnly 左側的核取方塊。

d. 在 Filter policies (篩選政策) 方塊中，輸入 AmazonSSMManagedInstanceCore。勾選搜尋結果中 AmazonSSMManagedInstanceCore 左側的核取方塊。

e. 選擇下一步。

10. 在 Name, review, and create (命名、檢閱和建立) 頁面上，執行以下作業：

a. 針對 Role name (角色名稱)，為您的角色輸入唯一名稱 (例如 AmazonEKSHybridNodesRole)。

b. 針對 Description (描述)，請以描述性文字 (例如 Amazon EKS - Hybrid Nodes role) 取代目前的文字。

c. 選擇建立角色。

IAM Roles Anywhere AWS 的步驟

若要使用 AWS IAM Roles Anywhere，您必須在建立混合節點 AWS IAM 角色之前設定 IAM Roles Anywhere 信任錨點。如需說明，請參閱 the section called “隨處設定 AWS IAM 角色”。

1. 開啟 Amazon IAM 主控台

2. 在左側導覽窗格中，選擇 Roles (角色)。

3. 在 Roles (角色) 頁面上，選擇 Create role (建立角色)。

4. 在 Select trusted entity (選取信任的實體) 頁面上，執行以下作業：

a. 在可信實體類型區段中，選擇自訂信任政策。將下列政策貼到自訂信任政策編輯器。使用您在 the section called “隨處設定 AWS IAM 角色” 步驟中建立的信任錨的 ARN 取代 TRUST_ANCHOR ARN。此信任政策中的 條件會限制 AWS IAM Roles Anywhere 擔任混合節點 IAM 角色的能力，只有在角色工作階段名稱符合安裝在混合節點上 x509 憑證中的 CN 時，才能交換臨時 IAM 憑證。或者，您也可以使用其他憑證屬性來唯一識別您的節點。您在信任政策中使用的憑證屬性必

須對應至您在 nodeadm 組態中設定的 nodeName。如需更多資訊，請參閱the section called “混合節點 nodeadm”。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "rolesanywhere.amazonaws.com"
            },
            "Action": [
                "sts:TagSession",
                "sts:SetSourceIdentity"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:PrincipalTag/x509Subject/CN": "${aws:PrincipalTag/
x509Subject/CN}"
                },
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:rolesanywhere:us-
east-1:123456789012:trust-anchor/TA_ID"
                }
            }
        },
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "rolesanywhere.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "sts:RoleSessionName": "${aws:PrincipalTag/x509Subject/CN}",
                    "aws:PrincipalTag/x509Subject/CN": "${aws:PrincipalTag/
x509Subject/CN}"
                },
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:rolesanywhere:us-
east-1:123456789012:trust-anchor/TA_ID"
                }
            }
```

```
        }
    ]
}
```

b. 選擇下一步。

5. 在新增許可頁面上，連接自訂政策或執行下列動作：
   a. 在篩選政策方塊中，輸入 EKSDescribeClusterPolicy 或您在上文建立的政策名稱。勾選搜尋結果中您的政策名稱左側的核取方塊。
   b. 在 Filter policies (篩選政策) 方塊中，輸入 AmazonEC2ContainerRegistryPullOnly。勾選搜尋結果中 AmazonEC2ContainerRegistryPullOnly 左側的核取方塊。
   c. 選擇下一步。
6. 在 Name, review, and create (命名、檢閱和建立) 頁面上，執行以下作業：
   a. 針對 Role name (角色名稱)，為您的角色輸入唯一名稱 (例如 AmazonEKSHybridNodesRole)。
   b. 針對 Description (描述)，請以描述性文字 (例如 Amazon EKS - Hybrid Nodes role) 取代目前的文字。
   c. 選擇建立角色。

建立具有混合節點的 Amazon EKS 叢集

本主題提供可用選項的概觀，並介紹建立已啟用混合節點的 Amazon EKS 叢集時需要考量的事項。EKS 混合節點與包含雲端節點的 Amazon EKS 叢集具有相同的 Kubernetes 版本支援，包括標準支援和延長支援。

如果您未打算使用 EKS 混合節點，請參閱 the section called “建立 叢集” 中的主要 Amazon EKS 建立叢集文件。

先決條件

- the section called “先決條件” 已完成。建立已啟用混合節點的叢集之前，您必須識別出內部部署節點和選用的 Pod CIDR，根據 EKS 要求和混合節點要求建立 VPC 和子網路，以及建立具有適用於內部部署和選用 Pod CIDR 的傳入規則的安全群組。如需有關這些先決條件的詳細資訊，請參閱 the section called “準備聯網”。
- 在您裝置上安裝和設定的最新版本 AWS 命令列界面 (AWS CLI)。若要檢查您目前的版本，請使用 aws --version。yum、apt-get 或 Homebrew for macOS 等套件管理員通常是最新版本 CLI AWS 後面的幾個版本。若要安裝最新版本，請參閱《 AWS 命令列界面使用者指南》中的安裝或更新至最新版本的 AWS CLI 和設定 AWS CLI 的設定。

- 具有建立 IAM 角色和連接政策以及建立和描述 EKS 叢集之許可的 IAM 主體

考量事項

- 您的叢集必須使用 API 或 API_AND_CONFIG_MAP 作為叢集身分驗證模式。
- 您的叢集必須使用 IPv4 位址系列。
- 您的叢集必須使用公有或私有叢集端點連線。您的叢集無法使用「公有和私有」叢集端點連線，因為 Amazon EKS Kubernetes API 伺服器端點會解析為在您的 VPC 外部執行的混合節點的公共 IP。
- 具有混合節點的 EKS 叢集支援 OIDC 身分驗證。
- 您可以新增、變更或移除現有叢集的混合節點組態。如需詳細資訊，請參閱the section called “現有叢集”。
- 如果您在建立啟用混合節點的叢集時未明確指定 Kubernetes 服務 IPv4 CIDR，Amazon EKS 會自動選取與您設定的遠端節點和 Pod 網路不重疊的服務 CIDR。指派的 CIDR 可能與標準預設值 (10.100.0.0/16 或) 不同172.20.0.0/16。例如，如果您的遠端節點或 Pod 網路與這些預設範圍重疊，Amazon EKS 會從私有 IP 範圍中選取替代 CIDR，例如 172.16.0.0/16。我們建議您在叢集建立期間明確指定服務 IPv4 CIDR，以確保可預測的組態。您可以執行 ，在叢集建立之後找到指派的服務 CIDRaws eks describe-cluster --name CLUSTER_NAME --query “cluster.kubernetesNetworkConfig.serviceIpv4Cidr”。

步驟 1：建立叢集 IAM 角色

如果您已有叢集 IAM 角色，或者您要使用 eksctl or AWS CloudFormation 建立叢集，則可以略過此步驟。根據預設，eksctl AWS CloudFormation 範本會為您建立叢集 IAM 角色。

1. 執行下列命令以建立 IAM 信任政策 JSON 檔案。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
```

```
}
```

2. 建立 Amazon EKS 叢集 IAM 角色。如有必要，請在 eks-cluster-role-trust-policy.json 前面加上您在上一步中寫入檔案的電腦的路徑。命令會將您在上一步驟中建立的信任策略與角色相關聯。若要建立 IAM 角色，必須為建立角色的 IAM 主體指派以下 `iam:CreateRole` 動作 (許可)。

```
aws iam create-role \
    --role-name myAmazonEKSClusterRole \
    --assume-role-policy-document file://"eks-cluster-role-trust-policy.json"
```

3. 您可以指派 Amazon EKS 受管政策或建立自己的自訂政策。如需了解在自訂政策中必須使用的最低許可，請參閱 the section called “節點 IAM 角色”。將名為 `AmazonEKSClusterPolicy` 的 Amazon EKS 受管 IAM 政策連接到角色。若要將 IAM 政策連接至 IAM 主體，必須為連接政策的 IAM 實體指派以下 IAM 動作之一 (許可)：`iam:AttachUserPolicy` 或 `iam:AttachRolePolicy`。

```
aws iam attach-role-policy \
    --policy-arn arn:aws: iam::aws:policy/AmazonEKSClusterPolicy \
    --role-name myAmazonEKSClusterRole
```

步驟 2：建立已啟用混合節點的叢集

您可以透過以下方式建立叢集：

- eksctl
- AWS CloudFormation
- AWS CLI
- AWS 管理主控台

建立已啟用混合節點的叢集 - eksctl

您需要安裝最新版本的 `eksctl` 命令列工具。如需有關安裝或更新 `eksctl` 的指示，請參閱 `eksctl` 文件中的安裝一節。

1. 建立 `cluster-config.yaml` 以定義已啟用混合節點的 Amazon EKS IPv4 叢集。請在您的 `cluster-config.yaml` 中執行下列取代。如需設定的完整清單，請參閱 eksctl 文件。

a. 使用您的叢集名稱取代 CLUSTER_NAME。此名稱僅能使用英數字元 (區分大小寫) 和連字號。必須以英數字元開頭，且長度不可超過 100 個字元。名稱在您要建立叢集 AWS 的區域和 AWS 帳戶中必須是唯一的。

b. AWS_REGION 將 取代為您要在其中建立叢集 AWS 的區域。

c. 使用任何 Amazon EKS 支援的版本取代 K8S_VERSION。

d. 使用 ssm 或 ira 取代 CREDS_PROVIDER，具體取決於您在 the section called “準備憑證” 步驟中設定的憑證提供者。

e. CA_BUNDLE_CERT 如果您的登入資料提供者設定為 ira，則取代 ，這會使用 AWS IAM Roles Anywhere 做為登入資料提供者。CA_BUNDLE_CERT 是憑證認證機構 (CA) 憑證內文，取決於您選擇的 CA。憑證必須是隱私權增強式郵件 (PEM) 格式。

f. 使用要連接到 VPC 的虛擬私有閘道或傳輸閘道的 ID 取代 GATEWAY_ID。

g. 使用混合節點的內部部署節點 CIDR 取代 REMOTE_NODE_CIDRS。

h. 對於在混合節點上執行的工作負載，使用內部部署 Pod CIDR 取代為 REMOTE_POD_CIDRS；或者如果您未在混合節點上執行 Webhook，則請從組態中移除該行。如果在 Pod 流量離開您的內部部署主機時，您的 CNI 未使用網路位址轉譯 (NAT) 或偽裝 Pod IP 位址，您必須設定您的 REMOTE_POD_CIDRS。如果您在混合節點上執行 Webhook，您必須設定 REMOTE_POD_CIDRS，如需詳細資訊，請參閱 the section called “設定 Webhook”。

i. 您的內部部署節點和 Pod CIDR 區塊必須符合下列要求：

   i. 在其中一個 IPv4 RFC-1918 範圍內：10.0.0.0/8、 172.16.0.0/12或 192.168.0.0/16，或在 RFC 6598 定義的 CGNAT 範圍內：100.64.0.0/10。

   ii. 您叢集的 VPC CIDR 或您的 Kubernetes Service IPv4 CIDR 不會彼此重疊

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: CLUSTER_NAME
  region: AWS_REGION
  version: "K8S_VERSION"

remoteNetworkConfig:
  iam:
    provider: CREDS_PROVIDER # default SSM, can also be set to IRA
    # caBundleCert: CA_BUNDLE_CERT
  vpcGatewayID: GATEWAY_ID
  remoteNodeNetworks:
  - cidrs: ["REMOTE_NODE_CIDRS"]
```

```
  remotePodNetworks:
  - cidrs: ["REMOTE_POD_CIDRS"]
```

2. 執行以下命令：

```
eksctl create cluster -f cluster-config.yaml
```

叢集佈建需要幾分鐘的時間。建立叢集時，會出現幾行輸出。輸出的最後一行類似於下面的範例行。

```
[#]  EKS cluster "CLUSTER_NAME" in "REGION" region is ready
```

3. 繼續進行 the section called “步驟 3：更新 kubeconfig”。

建立啟用混合節點的叢集 - AWS CloudFormation

CloudFormation 堆疊會使用您指定的 RemoteNodeNetwork 和 RemotePodNetwork 來建立 EKS 叢集 IAM 角色和 EKS 叢集。修改 CloudFormation 範本 如果您需要自訂 CloudFormation 範本中未公開的 EKS 叢集的設定。

1. 下載 CloudFormation 範本。

```
curl -OL 'https://raw.githubusercontent.com/aws/eks-hybrid/refs/heads/main/example/
hybrid-eks-cfn.yaml'
```

2. 建立 cfn-eks-parameters.json，並為每個值指定您的組態。
   a. CLUSTER_NAME：要建立的 EKS 叢集的名稱
   b. CLUSTER_ROLE_NAME：要建立的 EKS 叢集 IAM 角色的名稱。範本中的預設值為 "EKSClusterRole"。
   c. SUBNET1_ID：您在先決條件步驟中建立的第一個子網路的 ID
   d. SUBNET2_ID：您在先決條件步驟中建立的第二個子網路的 ID
   e. SG_ID：您在先決條件步驟中建立的安全群組 ID
   f. REMOTE_NODE_CIDRS：混合節點的內部部署節點 CIDR
   g. REMOTE_POD_CIDRS：在混合節點上執行的工作負載的內部部署 Pod CIDR。如果在 Pod 流量離開您的內部部署主機時，您的 CNI 未使用網路位址轉譯 (NAT) 或偽裝 Pod IP 位址，您必須設定您的 REMOTE_POD_CIDRS。如果您在混合節點上執行 Webhook，您必須設定 REMOTE_POD_CIDRS，如需詳細資訊，請參閱 the section called “設定 Webhook”。

h. 您的內部部署節點和 Pod CIDR 區塊必須符合下列要求：

i. 在其中一個 IPv4 RFC-1918 範圍內：10.0.0.0/8、 172.16.0.0/12或 192.168.0.0/16，或在 RFC 6598 定義的 CGNAT 範圍內：100.64.0.0/10。

ii. 您叢集的 VPC CIDR 或您的 Kubernetes Service IPv4 CIDR 不會彼此重疊。

i. CLUSTER_AUTH：您的叢集的叢集身分驗證模式。有效值為 API 和 API_AND_CONFIG_MAP。範本中的預設值為 API_AND_CONFIG_MAP。

j. CLUSTER_ENDPOINT：您叢集的叢集端點連線。有效值為 "Public" 或 "Private"。範本中的預設值為私有，這表示您只能從 VPC 內連接至 Kubernetes API 端點。

k. K8S_VERSION：要用於您叢集的 Kubernetes 版本。請參閱 Amazon EKS 支援的版本。

```
{
  "Parameters": {
    "ClusterName": "CLUSTER_NAME",
    "ClusterRoleName": "CLUSTER_ROLE_NAME",
    "SubnetId1": "SUBNET1_ID",
    "SubnetId2": "SUBNET2_ID",
    "SecurityGroupId" "SG_ID",
    "RemoteNodeCIDR": "REMOTE_NODE_CIDRS",
    "RemotePodCIDR": "REMOTE_POD_CIDRS",
    "ClusterAuthMode": "CLUSTER_AUTH",
    "ClusterEndpointConnectivity": "CLUSTER_ENDPOINT",
    "K8sVersion": "K8S_VERSION"
  }
 }
```

3. 部署 CloudFormation 堆疊。STACK_NAME 將 取代為 CloudFormation 堆疊的名稱，並將 AWS_REGION取代為建立叢集的 AWS 區域。

```
aws cloudformation deploy \
    --stack-name STACK_NAME \
    --region AWS_REGION \
    --template-file hybrid-eks-cfn.yaml \
    --parameter-overrides file://cfn-eks-parameters.json \
    --capabilities CAPABILITY_NAMED_IAM
```

叢集佈建需要幾分鐘的時間。您可以使用下列命令來檢查堆疊的狀態。STACK_NAME 將 取代為 CloudFormation 堆疊的名稱，並將 AWS_REGION取代為建立叢集的 AWS 區域。

```
aws cloudformation describe-stacks \
```

```
    --stack-name STACK_NAME \
    --region AWS_REGION \
    --query 'Stacks[].StackStatus'
```

4. 繼續進行 the section called “步驟 3：更新 kubeconfig”。

建立啟用混合節點的叢集 - AWS CLI

1. 執行下列命令以建立已啟用混合節點的 EKS 叢集和節點。執行命令之前，請使用您的設定取代下列內容。如需設定的完整清單，請參閱 the section called “建立 叢集” 文件。
   a. CLUSTER_NAME：要建立的 EKS 叢集的名稱
   b. AWS_REGION：建立叢集 AWS 的區域。
   c. K8S_VERSION：要用於您叢集的 Kubernetes 版本。請參閱 Amazon EKS 支援的版本。
   d. ROLE_ARN：您為叢集設定的 Amazon EKS 叢集角色。如需詳細資訊，請參閱 Amazon EKS 叢集 IAM 角色。
   e. SUBNET1_ID：您在先決條件步驟中建立的第一個子網路的 ID
   f. SUBNET2_ID：您在先決條件步驟中建立的第二個子網路的 ID
   g. SG_ID：您在先決條件步驟中建立的安全群組 ID
   h. 您可以使用 API 和 API_AND_CONFIG_MAP 作為叢集存取身分驗證模式。在以下命令中，叢集存取身分驗證模式設定為 API_AND_CONFIG_MAP。
   i. 您可以使用 endpointPublicAccess 和 endpointPrivateAccess 參數來啟用或停用叢集的 Kubernetes API 伺服器端點的公有和私有存取。在以下命令中，endpointPublicAccess 設定為 false，而 endpointPrivateAccess 設定為 true。
   j. REMOTE_NODE_CIDRS：混合節點的內部部署節點 CIDR。
   k. REMOTE_POD_CIDRS (選用)：在混合節點上執行的工作負載的內部部署 Pod CIDR。
   l. 您的內部部署節點和 Pod CIDR 區塊必須符合下列要求：
      i. 在其中一個 IPv4 RFC-1918 範圍內：10.0.0.0/8、 172.16.0.0/12或 192.168.0.0/16，或在 RFC 6598 定義的 CGNAT 範圍內：100.64.0.0/10。
      ii. Amazon EKS 叢集的 VPC CIDR 或您的 Kubernetes Service IPv4 CIDR 不會彼此重疊。

```
aws eks create-cluster \
    --name CLUSTER_NAME \
    --region AWS_REGION \
    --kubernetes-version K8S_VERSION \
    --role-arn ROLE_ARN \
```

```
    --resources-vpc-config
 subnetIds=SUBNET1_ID,SUBNET2_ID,securityGroupIds=SG_ID,endpointPrivateAccess=true,endpo
 \
    --access-config authenticationMode=API_AND_CONFIG_MAP \
    --remote-network-config '{"remoteNodeNetworks":[{"cidrs":
["REMOTE_NODE_CIDRS"]}],"remotePodNetworks":[{"cidrs":["REMOTE_POD_CIDRS"]}]}'
```

2. 佈建叢集需要幾分鐘才能完成。您可以使用下列命令來查詢叢集的狀態。CLUSTER_NAME 將 取代為您建立的叢集名稱，並將 AWS_REGION取代為您建立叢集的 AWS 區域。在傳回的輸出為 ACTIVE 之前，請勿進行下一個步驟。

```
aws eks describe-cluster \
    --name CLUSTER_NAME \
    --region AWS_REGION \
    --query "cluster.status"
```

3. 繼續進行 the section called “步驟 3：更新 kubeconfig”。

建立啟用混合節點的叢集 - AWS 管理主控台

1. 在 Amazon EKS 主控台開啟 Amazon EKS 主控台。
2. 選取 Add cluster (新增叢集)，然後選取 Create (建立)。
3. 在 Configure cluster (設定叢集) 頁面上，輸入下列欄位：
   a. Name (名稱)：叢集的名稱。名稱僅包含英數字元 (區分大小寫)、連字號和底線。必須以英數字元開頭，且長度不可超過 100 個字元。名稱在您要建立叢集 AWS 的區域和 AWS 帳戶中必須是唯一的。
   b. 叢集 IAM 角色 – 選擇您建立的 Amazon EKS 叢集 IAM 角色，以允許 Kubernetes 控制平面代表您管理 AWS 資源。
   c. Kubernetes version (Kubernetes 版本) – 您的叢集使用的 Kubernetes 版本。我們建議選取最新版本，除非您需要較早版本。
   d. 升級政策：選擇延長或標準。
      i. 延長：此選項會在 Kubernetes 版本發布日期後持續為其提供 26 個月的支援。延長支援期間產生額外的每小時成本，該成本會從標準支援期間結束後開始計費。延長支援結束時，您的叢集將會自動升級至下一個版本。
      ii. 延伸：此選項會在 Kubernetes 版本發布日期後持續為其提供 14 個月的支援。無需額外付費。標準支援結束時，您的叢集將會自動升級至下一個版本。

e. 叢集存取 - 選擇允許或不允許叢集管理員存取，然後選取身分驗證模式。已啟用混合節點的叢集支援下列身分驗證模式。

   i. EKS API：叢集將僅從 EKS 存取項目 API 取得已經過身分驗證的 IAM 主體。

   ii. EKS API 和 ConfigMap：叢集將從 EKS 存取項目 API 和 `aws-auth` ConfigMap 取得已經過身分驗證的 IAM 主體。

f. Secrets encryption (秘密加密)：(選用) 選擇使用 KMS 金鑰啟用 Kubernetes 秘密的秘密加密。您也可以在建立叢集後啟用此功能。啟用此功能之前，請確定您已熟悉 the section called “啟用秘密加密” 中的資訊。

g. ARC 區域轉移：如果啟用，EKS 將向 ARC 區域轉移註冊您的叢集，以讓您能夠使用區域轉移將應用程式流量從可用區域轉移。

h. Tags (標籤) – (選用) 將任何標籤新增到您的叢集。如需詳細資訊，請參閱the section called “標記您的 資源”。

i. 完成此頁面後，請選擇下一步。

4. 在 Specify networking (指定網路) 頁面上，選取下列欄位的值：

   a. VPC：選擇符合 the section called “VPC 和子網要求” 和 Amazon EKS 混合節點要求 的現有 VPC。選擇 VPC 之前，建議您先熟悉檢視 VPC、子網路和混合節點的 Amazon EKS 聯網需求中的所有要求和考量事項。建立叢集後，您無法變更要使用的 VPC。如果沒有列出 VPC，則您需要先建立一個。如需詳細資訊，請參閱 the section called “建立 VPC” 和 Amazon EKS 混合節點聯網需求。

   b. Subnets (子網路)：根據預設，前一個欄位指定的 VPC 中的所有可用子網路會預先選取。您必須選取至少兩個。

   c. 安全群組:(選用) 指定一或多個您希望 Amazon EKS 與其建立的網路介面相關聯的安全群組。您指定的安全群組中至少有一個必須具有適用於內部部署節點和 (選用) Pod CIDR 的傳入規則。如需詳細資訊，請參閱 Amazon EKS 混合節點聯網需求。無論您是否選擇任何安全群組，Amazon EKS 都會建立一個安全群組，以支援您的叢集和 VPC 之間的通訊。Amazon EKS 將此安全群組及您選擇的任何群組與其建立的網路介面相關聯。如需有關 Amazon EKS 建立的叢集安全群組的詳細資訊，請參閱 the section called “安全群組要求” 您可以修改 Amazon EKS 建立的叢集安全群組中的規則。

   d. 選擇叢集 IP 位址系列：您必須為已啟用混合節點的叢集選擇 IPv4。

   e. (選用) 選擇設定 Kubernetes 服務 IP 位址範圍，並指定服務 IPv4 範圍。我們建議為啟用混合節點的叢集指定明確的服務 CIDR。如果您未指定，Amazon EKS 會選取 CIDR，以避免與遠端節點和 Pod 網路發生衝突，這可能與標準預設值 (`10.100.0.0/16` 或) 不同`172.20.0.0/16`。

   f. 選擇設定遠端網路以啟用混合節點，然後為混合節點指定您的內部部署節點和 Pod CIDR。

g. 如果在 Pod 流量離開您的內部部署主機時，您的 CNI 未使用網路位址轉譯 (NAT) 或偽裝 Pod IP 位址，您必須設定您的遠端 Pod CIDR。如果您在混合節點上執行 Webhook，則必須設定遠端 Pod CIDR。

h. 您的內部部署節點和 Pod CIDR 區塊必須符合下列要求：

   i. 在其中一個 IPv4 RFC-1918 範圍內：`10.0.0.0/8`、 `172.16.0.0/12`或 `192.168.0.0/16`，或在 RFC 6598 定義的 CGNAT 範圍內：`100.64.0.0/10`。

   ii. 您叢集的 `VPC CIDR` 或您的 Kubernetes Service IPv4 CIDR 不會彼此重疊

i. 針對 Cluster endpoint access (叢集端點存取)，選取一個選項。建立叢集後，您可以變更此選項。對於已啟用混合節點的叢集，您必須選擇公有或私有。在選取非預設選項之前，請務必熟悉這些選項及其含義。如需詳細資訊，請參閱the section called “叢集端點存取”。

j. 完成此頁面後，請選擇下一步。

5. (選用) 在設定可觀測性頁面上，選擇要開啟的指標和控制平面記錄選項。根據預設，系統會關閉每個日誌類型。

   a. 如需有關 Prometheus 指標選項的詳細資訊，請參閱 the section called “Prometheus 指標”。

   b. 如需 EKS 控制記錄選項的詳細資訊，請參閱 the section called “控制平面日誌”。

   c. 完成此頁面後，請選擇下一步。

6. 在 Select add-ons (選取附加元件) 頁面上，選擇您要新增至叢集的附加元件。

   a. 您可以視需要選擇任意數量的 Amazon EKS 附加元件和 AWS Marketplace 附加元件。與混合節點不相容的 Amazon EKS 附加元件會標記為「與混合節點不相容」，並且附加元件具有反親和性規則，可防止其在混合節點上執行。如需詳細資訊，請參閱設定混合節點的附加元件。如果您想要安裝的 AWS Marketplace 附加元件未列出，您可以在搜尋方塊中輸入文字來搜尋可用的 AWS Marketplace 附加元件。您也可以依 category (類別)、vendor (廠商) 或 pricing model (定價模式) 進行搜尋，然後從搜尋結果中選擇附加元件。

   b. 一些附加元件，例如 CoreDNS 和 kube-proxy，是預設安裝的。如果停用任何預設附加元件，可能會影響您執行 Kubernetes 應用程式的能力。

   c. 完成此頁面後，請選擇 `Next`。

7. 在設定選取的附加元件設定頁面上，選取您要安裝的版本。

   a. 建立叢集後，您隨時皆可更新至更新版本。您可以在建立叢集後更新每個附加元件的組態。如需有關設定附加元件的詳細資訊，請參閱the section called “更新附加元件”。如需與混合節點相容的附加元件版本，請參閱 the section called “設定附加元件”。

   b. 完成此頁面後，請選擇下一步。

8. 在 Review and create (檢閱並建立) 頁面上，檢閱您在先前頁面上輸入或選取的資訊。如需變更，請選擇 Edit (編輯)。當您感到滿意時，請選擇建立。在佈建叢集時，Status (狀態) 欄位顯示 CREATING (正在建立)。叢集佈建需要幾分鐘的時間。
9. 繼續進行 the section called “步驟 3：更新 kubeconfig”。

步驟 3：更新 kubeconfig

如果您使用 `eksctl` 建立叢集，則可以略過此步驟。這是因為 `eksctl` 已為您完成此步驟。透過向 `kubectl` 組態檔案新增內容，使 `kubectl` 能夠與您的叢集通訊。如需建立和更新檔案的詳細資訊，請參閱 the section called “透過 kubectl 存取叢集”。

```
aws eks update-kubeconfig --name CLUSTER_NAME --region AWS_REGION
```

範例輸出如下。

```
Added new context arn:aws: eks:AWS_REGION:111122223333:cluster/CLUSTER_NAME to /home/
username/.kube/config
```

透過執行以下命令確認與叢集的通訊。

```
kubectl get svc
```

範例輸出如下。

```
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.100.0.1   <none>        443/TCP   28h
```

步驟 4：叢集設定

下一步，請參閱 the section called “準備叢集存取”，以啟用對混合節點的存取，進而加入叢集。

在現有的 Amazon EKS 叢集上啟用混合節點或修改組態

本主題提供可用選項的概觀，並介紹您建立新增、變更或移除 Amazon EKS 叢集的混合節點組態時需要考量的問題。

若要讓 Amazon EKS 叢集使用混合節點，請在 `RemoteNetworkConfig` 組態中新增內部部署節點的 IP 位址 CIDR 範圍和選用的 Pod 網路。EKS 使用此 CIDR 清單來啟用叢集與內部部署網

路之間的連線。如需更新叢集組態時選項的完整清單，請參閱《Amazon EKS API 參考》中的 UpdateClusterConfig。

您可以對叢集中的 EKS 混合節點聯網組態執行下列任何動作：

- 新增遠端網路組態，以在現有叢集中啟用 EKS 混合節點。
- 在現有叢集中新增、變更或移除遠端節點網路或遠端 Pod 網路。
- 移除所有遠端節點網路 CIDR 範圍，以在現有叢集中停用 EKS 混合節點。

先決條件

- 為混合節點啟用 Amazon EKS 叢集之前，請確保您的環境符合 the section called “先決條件” 中概述的要求以及 the section called “準備聯網”、 the section called “準備作業系統” 和 the section called “準備憑證” 中詳細說明的要求。
- 您的叢集必須使用 IPv4 位址系列。
- 您的叢集必須使用 `API` 或 `API_AND_CONFIG_MAP` 作為叢集身分驗證模式。修改叢集身分驗證模式的程序在 the section called “身分驗證方式” 中說明。
- 建議您對 Amazon EKS Kubernetes API 伺服器端點使用公有或私有端點存取，但不能同時使用兩者。如果您選擇「公有和私有」，Amazon EKS Kubernetes API 伺服器端點一律會解析為在 VPC 外部執行的混合節點的公有 IP，以防止您的混合節點加入叢集。修改叢集網路存取的程序在 the section called “叢集端點存取” 中說明。
- 在您裝置上安裝和設定的最新版本 AWS 命令列界面 (AWS CLI)。若要檢查您目前的版本，請使用 `aws --version`。適用於 macOS 的 yum、apt-get 或 Homebrew 等套件管理員通常是最新版本 CLI AWS 後面的幾個版本。若要安裝最新版本，請參閱《 AWS 命令列界面使用者指南》中的安裝或更新至最新版本的 AWS CLI 和設定 AWS CLI 的設定。
- 具有在您的 Amazon EKS 叢集上呼叫 UpdateClusterConfig 的許可的 IAM 主體。
- 將附加元件更新為與混合節點相容的版本。如需與混合節點相容的附加元件版本，請參閱 the section called “設定附加元件”。
- 如果您執行的附加元件與混合節點不相容，請確保附加元件 DaemonSet 或部署具有下列親和性規則，以防止部署到混合節點。如果尚未存在，請新增下列親和性規則。

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
```

```
        - key: eks.amazonaws.com/compute-type
          operator: NotIn
          values:
          - hybrid
```

考量事項

remoteNetworkConfig JSON 物件在更新期間具有下列行為：

- 您未指定的任何現有組態部分均未變更。如果您未指定 remoteNodeNetworks 或 remotePodNetworks，則該部分將保持不變。
- 如果您要修改 CIDR 的 remoteNodeNetworks 或 remotePodNetworks 清單，則必須在最終組態中指定您想要的完整 CIDR 清單。當您指定對 remoteNodeNetworks 或 remotePodNetworks CIDR 清單的變更時，EKS 會在更新期間取代原始清單。
- 您的內部部署節點和 Pod CIDR 區塊必須符合下列要求：
  1. 在其中一個 IPv4 RFC-1918 範圍內：10.0.0.0/8、172.16.0.0/12 或 192.168.0.0/16，或在 RFC 6598 定義的 CGNAT 範圍內： 100.64.0.0/10
  2. Amazon EKS 叢集的 VPC 的所有 CIDR 或您的 Kubernetes Service IPv4 CIDR 不會彼此重疊。

在現有叢集上啟用混合節點

您可以使用下列方式，在現有叢集中啟用 EKS 混合節點：

- AWS CloudFormation
- AWS CLI
- AWS 管理主控台

在現有叢集中啟用 EKS 混合節點 - AWS CloudFormation

1. 若要在叢集中啟用 EKS 混合節點，請將 RemoteNodeNetwork 和 (選用) RemotePodNetwork 新增至 CloudFormation 範本並更新堆疊。請注意，RemoteNodeNetwork 是最多包含一個 Cidrs 項目的清單，而 Cidrs 是多個 IP CIDR 範圍的清單。

```
RemoteNetworkConfig:
  RemoteNodeNetworks:
    - Cidrs: [RemoteNodeCIDR]
  RemotePodNetworks:
```

```
    - Cidrs: [RemotePodCIDR]
```

2. 繼續進行the section called “準備叢集存取”。

在現有叢集中啟用 EKS 混合節點 - AWS CLI

1. 執行下列命令以為您的 EKS 叢集啟用 EKS 混合節點的 RemoteNetworkConfig。執行命令之前，請使用您的設定取代下列內容。如需設定的完整清單，請參閱《Amazon EKS API 參考》中的UpdateClusterConfig。
   a. CLUSTER_NAME：要更新的 EKS 叢集的名稱。
   b. AWS_REGION執行 EKS 叢集的 ： AWS Region。
   c. REMOTE_NODE_CIDRS：混合節點的內部部署節點 CIDR。
   d. REMOTE_POD_CIDRS (選用)：在混合節點上執行的工作負載的內部部署 Pod CIDR。

```
aws eks update-cluster-config \
    --name CLUSTER_NAME \
    --region AWS_REGION \
    --remote-network-config '{"remoteNodeNetworks":[{"cidrs":
["REMOTE_NODE_CIDRS"]}],"remotePodNetworks":[{"cidrs":["REMOTE_POD_CIDRS"]}]}'
```

2. 更新叢集需要幾分鐘才能完成。您可以使用下列命令來查詢叢集的狀態。CLUSTER_NAME 將 取代為您修改的叢集名稱，並將 AWS_REGION取代為執行叢集的 AWS 區域。在傳回的輸出為 ACTIVE 之前，請勿進行下一個步驟。

```
aws eks describe-cluster \
    --name CLUSTER_NAME \
    --region AWS_REGION \
    --query "cluster.status"
```

3. 繼續進行the section called “準備叢集存取”。

在現有叢集中啟用 EKS 混合節點 - AWS 管理主控台

1. 在 Amazon EKS 主控台開啟 Amazon EKS 主控台。
2. 選擇叢集名稱以顯示您叢集的資訊。
3. 選擇聯網索引標籤，然後選擇管理。
4. 在下拉式清單中，選擇遠端網路。

5. 選擇設定遠端網路以啟用混合節點，然後為混合節點指定您的內部部署節點和 Pod CIDR。
6. 選擇 Save changes (儲存變更) 以完成操作。等待叢集狀態變回作用中。
7. 繼續進行the section called “準備叢集存取”。

在現有叢集中更新混合節點組態

您可以使用下列任何一種方式，在現有混合叢集中修改 `remoteNetworkConfig`：

- AWS CloudFormation
- AWS CLI
- AWS 管理主控台

更新現有叢集 - AWS CloudFormation 中的混合組態

1. 使用新的網路 CIDR 值更新您的 CloudFormation 範本。

```
RemoteNetworkConfig:
  RemoteNodeNetworks:
    - Cidrs: [NEW_REMOTE_NODE_CIDRS]
  RemotePodNetworks:
    - Cidrs: [NEW_REMOTE_POD_CIDRS]
```

> **Note**
>
> 更新 `RemoteNodeNetworks` 或 `RemotePodNetworks` CIDR 清單時，請包含所有 CIDR (新的和現有的)。EKS 會在更新期間取代整個清單。從更新請求省略這些欄位將會保留其現有組態。

2. 使用修改過的範本更新 CloudFormation 堆疊，並等待堆疊更新完成。

更新現有叢集 - AWS CLI 中的混合組態

1. 若要修改遠端網路 CIDR，請執行下列命令。使用您的設定取代這些值：

```
aws eks update-cluster-config
--name CLUSTER_NAME
--region AWS_REGION
```

```
--remote-network-config '{"remoteNodeNetworks":[{"cidrs":
["NEW_REMOTE_NODE_CIDRS"]}],"remotePodNetworks":[{"cidrs":
["NEW_REMOTE_POD_CIDRS"]}]}'
```

> **Note**
>
> 更新 `remoteNodeNetworks` 或 `remotePodNetworks` CIDR 清單時，請包含所有 CIDR (新的和現有的)。EKS 會在更新期間取代整個清單。從更新請求省略這些欄位將會保留其現有組態。

2. 等待叢集狀態變回「作用中」，然後再繼續。

更新現有叢集中的混合組態 - AWS 管理主控台

1. 在 Amazon EKS 主控台開啟 Amazon EKS 主控台。
2. 選擇叢集名稱以顯示您叢集的資訊。
3. 選擇聯網索引標籤，然後選擇管理。
4. 在下拉式清單中，選擇遠端網路。
5. 視需要更新 `Remote node networks` 和 `Remote pod networks - Optional` 下的 CIDR。
6. 選擇儲存變更，然後等待叢集狀態返回作用中。

在現有叢集中停用混合節點

您可以使用下列方式，在現有叢集中停用 EKS 混合節點：

- AWS CloudFormation
- AWS CLI
- AWS 管理主控台

在現有叢集中停用 EKS 混合節點 - AWS CloudFormation

1. 若要停用叢集中的 EKS 混合節點，請在 CloudFormation 範本中將 `RemoteNodeNetworks` 和 `RemotePodNetworks` 設定為空陣列，並更新堆疊。

```
RemoteNetworkConfig:
  RemoteNodeNetworks: []
```

```
  RemotePodNetworks: []
```

在現有叢集中停用 EKS 混合節點 - AWS CLI

1. 執行下列命令以從 EKS 叢集中移除 RemoteNetworkConfig。執行命令之前，請使用您的設定取代下列內容。如需設定的完整清單，請參閱《Amazon EKS API 參考》中的 UpdateClusterConfig。
   a. CLUSTER_NAME：要更新的 EKS 叢集的名稱。
   b. AWS_REGION執行 EKS 叢集的 ： AWS Region。

```
aws eks update-cluster-config \
    --name CLUSTER_NAME \
    --region AWS_REGION \
    --remote-network-config '{"remoteNodeNetworks":[],"remotePodNetworks":[]}'
```

2. 更新叢集需要幾分鐘才能完成。您可以使用下列命令來查詢叢集的狀態。CLUSTER_NAME 將 取代為您修改的叢集名稱，並將 AWS_REGION取代為執行叢集的 AWS 區域。在傳回的輸出為 ACTIVE 之前，請勿進行下一個步驟。

```
aws eks describe-cluster \
    --name CLUSTER_NAME \
    --region AWS_REGION \
    --query "cluster.status"
```

在現有叢集中停用 EKS 混合節點 - AWS 管理主控台

1. 在 Amazon EKS 主控台開啟 Amazon EKS 主控台。
2. 選擇叢集名稱以顯示您叢集的資訊。
3. 選擇聯網索引標籤，然後選擇管理。
4. 在下拉式清單中，選擇遠端網路。
5. 選擇設定遠端網路以啟用混合節點，然後移除 Remote node networks 和 Remote pod networks - Optional 下的所有 CIDR。
6. 選擇 Save changes (儲存變更) 以完成操作。等待叢集狀態變回作用中。

準備混合節點的叢集存取

將混合節點連線至 Amazon EKS 叢集之前，您必須啟用具有 Kubernetes 許可的混合節點 IAM 角色，才能加入叢集。如需如何建立混合節點 IAM 角色的資訊，請參閱 the section called “準備憑證”。Amazon EKS 支援兩種方式將 IAM 主體與 Kubernetes 角色型存取控制 (RBAC)、Amazon EKS 存取項目和 `aws-auth` ConfigMap 建立關聯。如需有關 Amazon EKS 存取管理的詳細資訊，請參閱 the section called “Kubernetes API 存取”。

使用以下程序，以將您的混合節點 IAM 角色與 Kubernetes 許可建立關聯。若要使用 Amazon EKS 存取項目，您的叢集必須已使用 `API` 或 `API_AND_CONFIG_MAP` 身分驗證模式建立。若要使用 `aws-auth` ConfigMap，您的叢集必須已使用 `API_AND_CONFIG_MAP` 身分驗證模式建立。已啟用混合節點的 Amazon EKS 叢集不支援僅限 `CONFIG_MAP` 的身分驗證模式。

使用混合節點的 Amazon EKS 存取項目 IAM 角色

名為 HYBRID_LINUX 的混合節點 Amazon EKS 存取項目類型，可與 IAM 角色搭配使用。使用此存取項目類型時，使用者名稱會自動設定為 system:node:{{SessionName}}。如需建立存取項目的詳細資訊，請參閱 the section called “建立存取項目”。

AWS CLI

1. 您必須在裝置上安裝並設定最新版本的 AWS CLI。若要檢查您目前的版本，請使用 `aws --version`。yum、apt-get 或 Homebrew for macOS 等套件管理員通常是最新版本 CLI AWS 後面的幾個版本。若要安裝最新版本，請參閱《 AWS 命令列界面使用者指南》中的使用 aws 設定安裝 和快速組態。
2. 使用下列命令建立您的存取項目。將 CLUSTER_NAME 取代為您的叢集名稱，並將 HYBRID_NODES_ROLE_ARN 取代為您在 the section called “準備憑證” 的步驟中建立的角色的 ARN。

```
aws eks create-access-entry --cluster-name CLUSTER_NAME \
    --principal-arn HYBRID_NODES_ROLE_ARN \
    --type HYBRID_LINUX
```

AWS 管理主控台

1. 在 Amazon EKS 主控台開啟 Amazon EKS 主控台。
2. 選擇已啟用混合節點的叢集的名稱。
3. 選擇存取索引標籤。

4. 選擇建立存取項目。
5. 針對 IAM 主體，選取您在 the section called “準備憑證” 的步驟中建立的混合節點 IAM 角色。
6. 針對類型，選取混合 Linux。
7. (選用) 您可以使用標籤為存取項目指派標籤。例如，為了更輕鬆地找到具有相同標籤的所有資源而指定標籤。
8. 選擇跳至檢閱和建立。您無法將政策新增至混合 Linux 存取項目或變更其存取範圍。
9. 檢查存取項目的組態。如果有任何內容看起來不正確，請選擇上一步以返回上一步並修正錯誤。如果組態正確，請選擇建立。

針對混合節點 IAM 角色,使用 aws-auth ConfigMap

在下列步驟中，您將使用您在 aws-auth 的步驟中建立的混合節點 IAM 角色的 ARN 來建立或更新 the section called “準備憑證” ConfigMap。

1. 檢查您的叢集是否已有現有的 aws-auth ConfigMap。請注意，如果您使用特定 kubeconfig 檔案，則請使用 --kubeconfig 旗標。

```
kubectl describe configmap -n kube-system aws-auth
```

2. 如果看到 aws-auth ConfigMap，則請視需要進行更新。
   a. 開啟 ConfigMap 進行編輯。

```
kubectl edit -n kube-system configmap/aws-auth
```

   b. 視需要新增 mapRoles 個項目。使用混合節點 IAM 角色的 ARN 取代 HYBRID_NODES_ROLE_ARN。請注意，{{SessionName}} 是在 ConfigMap 中儲存的正確範本格式。請勿將其取代為其他值。

```
data:
  mapRoles: |
  - groups:
    - system:bootstrappers
    - system:nodes
    rolearn: HYBRID_NODES_ROLE_ARN
    username: system:node:{{SessionName}}
```

   c. 儲存檔案並結束您的文字編輯器。

3. 如果您的叢集沒有現有的 aws-auth ConfigMap，則請使用下列命令予以建立。使用混合節點 IAM 角色的 ARN 取代 HYBRID_NODES_ROLE_ARN。請注意，{{SessionName}} 是在 ConfigMap 中儲存的正確範本格式。請勿將其取代為其他值。

```
kubectl apply -f=/dev/stdin <<-EOF
apiVersion: v1
kind: ConfigMap
metadata:
  name: aws-auth
  namespace: kube-system
data:
  mapRoles: |
  - groups:
    - system:bootstrappers
    - system:nodes
    rolearn: HYBRID_NODES_ROLE_ARN
    username: system:node:{{SessionName}}
EOF
```

在混合節點上執行內部部署工作負載

在已啟用混合節點的 EKS 叢集中，您可以使用您在 AWS 雲端中使用的相同 Amazon EKS 叢集、功能和工具，在自己的基礎結構上執行內部部署和邊緣應用程式。

下列各節包含使用混合節點的逐步說明。

主題

- 連接混合節點
- 使用 Bottlerocket 連接混合節點
- 升級叢集的混合節點
- 混合節點的修補程式安全更新
- 移除混合節點

連接混合節點

> **Note**
>
> 下列步驟適用於執行 Bottlerocket 以外的相容作業系統的混合節點。如需連線執行 Bottlerocket 的混合節點的步驟，請參閱 the section called “使用 Bottlerocket 連接混合節點”。

本主題會說明如何將混合節點連接至 Amazon EKS 叢集。混合節點加入叢集後，其即會在 Amazon EKS 主控台和 Kubernetes 相容工具 (例如 kubectl) 中顯示為「未就緒」狀態。完成此頁面上的步驟後，請繼續 the section called “設定 CNI”，以讓您的混合節點準備好執行應用程式。

先決條件

將混合節點連接至 Amazon EKS 叢集之前，請確保您已完成先決條件步驟。

- 您可以從內部部署環境連線至託管 Amazon EKS 叢集 AWS 的區域。如需詳細資訊，請參閱the section called “準備聯網”。
- 您已在內部部署主機上安裝了適用於混合節點的相容作業系統。如需詳細資訊，請參閱the section called “準備作業系統”。
- 您已建立混合節點 IAM 角色，並設定內部部署憑證提供者 (AWS Systems Manager 混合啟用或 AWS IAM Roles Anywhere)。如需詳細資訊，請參閱the section called “準備憑證”。
- 您已建立了已啟用混合節點的 Amazon EKS 叢集。如需詳細資訊，請參閱the section called “建立叢集”。
- 您已將混合節點 IAM 角色與 Kubernetes 角色型存取控制 (RBAC) 許可建立關聯。如需詳細資訊，請參閱the section called “準備叢集存取”。

> **SSM 登入資料提供者所需的 nodeadm 版本**
>
> 如果您使用 AWS Systems Manager (SSM) 作為混合節點的憑證提供者，則必須使用 nodeadm 版本 `1.0.19`或更新版本進行新的安裝和升級。舊版 nodeadm包含過期的 SSM 簽署金鑰，並在 `nodeadm install`期間失敗，並`nodeadm upgrade`出現下列簽章驗證錯誤：
>
> ```
> "msg":"Command failed","error":"failed to install ssm installer: validating ssm-setup-cli signature: Signature Verification Error: No matching signature"
> ```

若要解決此錯誤，請在執行 或 之前下載最新版本的 nodeadm upgrade。 nodeadm nodeadm install

步驟 1：在每個內部部署主機上安裝混合節點 CLI (**nodeadm**)

如果您在預先建置的作業系統映像中包含 Amazon EKS 混合節點 CLI (nodeadm)，則可以略過此步驟。如需 nodeadm 混合節點版本的詳細資訊，請參閱 the section called “混合節點 nodeadm”。

nodeadm 的混合節點版本託管於前端為 Amazon CloudFront 的 Amazon S3 中。若要在每個內部部署主機上安裝 nodeadm，您可以從內部部署主機執行下列命令。

對於 x86_64 主機：

```
curl -OL 'https://hybrid-assets.eks.amazonaws.com/releases/latest/bin/linux/amd64/
nodeadm'
```

對於 ARM 主機

```
curl -OL 'https://hybrid-assets.eks.amazonaws.com/releases/latest/bin/linux/arm64/
nodeadm'
```

在每個主機上，為下載的二進位檔新增可執行檔許可。

```
chmod +x nodeadm
```

步驟 2：使用 **nodeadm** 安裝混合節點相依性

如果您要在預先建置的作業系統映像中安裝混合節點相依性，則可以略過此步驟。nodeadm install 命令可用於安裝混合節點所需的所有相依性。混合節點相依性包括 containerd、kubelet、kubectl 和 AWS SSM 或 AWS IAM Roles Anywhere 元件。如需 nodeadm install 所安裝元件和檔案位置的詳細資訊，請參閱 the section called “混合節點 nodeadm”。如需 nodeadm install 程序必須在內部部署防火牆中允許的網域的詳細資訊，請參閱混合節點的 the section called “準備聯網”。

執行以下命令，以在您的內部部署主機上安裝混合節點相依性。以下命令必須由主機上具有 sudo/root 存取權的使用者執行。

> **Important**
>
> 混合節點 CLI (nodeadm) 必須由主機上具有 sudo/root 存取權的使用者執行。

- 使用 Amazon EKS 叢集的 Kubernetes 次要版本取代 `K8S_VERSION`，例如 `1.31`。如需支援的 Kubernetes 版本的清單，請參閱 Amazon EKS 支援的版本。
- 使用您正在使用的內部部署憑證提供者取代 `CREDS_PROVIDER`。有效值`ssm`適用於 AWS SSM 和 `iam-ra` AWS IAM Roles Anywhere。

```
nodeadm install K8S_VERSION --credential-provider CREDS_PROVIDER
```

步驟 3：將混合節點連接至您的叢集

將混合節點連接至您的叢集之前，請確保您已獲得內部部署防火牆和叢集的安全群組中的必要存取權，允許往返 Amazon EKS 控制平面與混合節點之間的通訊。此步驟中的大多數問題都與防火牆組態、安全群組組態或混合節點 IAM 角色組態相關。

> **Important**
>
> 混合節點 CLI (nodeadm) 必須由主機上具有 sudo/root 存取權的使用者執行。

1. 在每個主機上建立包含部署值的 `nodeConfig.yaml` 檔案。如需可用組態設定的完整說明，請參閱 the section called “混合節點 nodeadm”。如果您的混合節點 IAM 角色沒有 `eks:DescribeCluster` 動作的許可，您必須在 `nodeConfig.yaml` 的叢集區段中傳遞 Kubernetes API 端點、叢集 CA 套件和 Kubernetes Service IPv4 CIDR。
   a. 如果您使用現場部署憑證提供者的 AWS SSM 混合啟用，請使用下列`nodeConfig.yaml`範例。
      i. 使用您叢集的名稱取代 `CLUSTER_NAME`。
      ii. 將 取代`AWS_REGION`為託管叢集的 AWS 區域。例如 `us-west-2`。
      iii. `ACTIVATION_CODE` 以您在建立 AWS SSM 混合啟用時收到的啟用代碼取代 。如需詳細資訊，請參閱the section called “準備憑證”。
      iv. `ACTIVATION_ID` 以您在建立 AWS SSM 混合啟用時收到的啟用 ID 取代 。您可以從 AWS Systems Manager 主控台或 CLI AWS `aws ssm describe-activations`命令擷取此資訊。

```
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name: CLUSTER_NAME
    region: AWS_REGION
  hybrid:
    ssm:
      activationCode: ACTIVATION_CODE
      activationId: ACTIVATION_ID
```

b. 如果您為內部部署憑證提供者使用 AWS IAM Roles Anywhere，請使用下列nodeConfig.yaml範例。

   i. 使用您叢集的名稱取代 CLUSTER_NAME。

   ii. 將 取代AWS_REGION為託管叢集的 AWS 區域。例如 us-west-2。

   iii. 使用您節點的名稱取代 NODE_NAME。如果您已使用 "sts:RoleSessionName": "${aws:PrincipalTag/x509Subject/CN}" 資源條件設定混合節點 IAM 角色的信任政策，則節點名稱必須與主機上的憑證的 CN 相符。您使用的 nodeName 不可超過 64 個字元。

   iv. 使用您在準備混合節點憑證的步驟中設定的信任錨的 ARN 取代 TRUST_ANCHOR_ARN。

   v. 使用您在 the section called “準備憑證” 的步驟中設定的信任錨的 ARN 取代 PROFILE_ARN。

   vi. 使用混合節點 IAM 角色的 ARN 取代 ROLE_ARN。

   vii. 使用磁碟中節點憑證的路徑取代 CERTIFICATE_PATH。如未指定，預設為 /etc/iam/pki/server.pem。

   viii. 使用磁碟中憑證私有金鑰的路徑取代 KEY_PATH。如未指定，預設為 /etc/iam/pki/server.key。

```
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name: CLUSTER_NAME
    region: AWS_REGION
  hybrid:
    iamRolesAnywhere:
      nodeName: NODE_NAME
      trustAnchorArn: TRUST_ANCHOR_ARN
      profileArn: PROFILE_ARN
      roleArn: ROLE_ARN
```

```
    certificatePath: CERTIFICATE_PATH
    privateKeyPath: KEY_PATH
```

2. 使用您的 nodeConfig.yaml 執行 nodeadm init 命令，以將混合節點連接至 Amazon EKS 叢集。

```
nodeadm init -c file://nodeConfig.yaml
```

如果上述命令成功完成，則您的混合節點已加入您的 Amazon EKS 叢集。您可以在 Amazon EKS 主控台中透過導覽至叢集的 Compute (運算) 索引標籤 (確保 IAM 主體具有檢視許可) 或使用 kubectl get nodes 來驗證這一點。

> **Important**
>
> 您的節點將處於 Not Ready 狀態，這在意料之中，因為您的混合節點上並未執行 CNI。如果您的節點未加入叢集，請參閱 the section called “疑難排解”。

步驟 4：設定混合節點的 CNI

要讓您的混合節點準備好執行應用程式，請繼續執行 the section called “設定 CNI” 上的步驟。

使用 Bottlerocket 連接混合節點

本主題會說明如何將執行 Bottlerocket 的混合節點連接至 Amazon EKS 叢集。Bottlerocket 是由贊助和支援的開放原始碼 Linux 發行版本 AWS。Bottlerocket 專為託管容器工作負載而打造。利用 Bottlerocket，您可以自動化容器基礎結構的更新，進而改善容器化部署的可用性並降低營運成本。Bottlerocket 僅包含執行容器的基本軟體，可改善資源用量、減少安全威脅，並降低管理開銷。

EKS 混合節點僅支援 Bottlerocket 版本 1.37.0 及更新版本的 VMware 變體。Bottlerocket 的 VMware 變體適用於 Kubernetes 版本 1.28 及更新版本。這些變體的作業系統映像包括 kubelet、containerd、aws-iam-authenticator 和 EKS 混合節點的其他軟體先決條件。您可以使用 Bottlerocket 設定檔案來設定這些元件，其中該檔案包含 Bottlerocket 引導和管理員容器的 base64 編碼使用者資料。設定這些設定可讓 Bottlerocket 使用您的混合節點憑證提供者，進而驗證叢集的混合節點。混合節點加入叢集後，其即會在 Amazon EKS 主控台和 Kubernetes 相容工具 (例如 kubectl) 中顯示為「Not Ready」狀態。完成此頁面上的步驟後，請繼續 the section called “設定 CNI”，以讓您的混合節點準備好執行應用程式。

先決條件

將混合節點連接至 Amazon EKS 叢集之前，請確保您已完成先決條件步驟。

- 您可以從內部部署環境連線至託管 Amazon EKS 叢集 AWS 的區域。如需詳細資訊，請參閱the section called “準備聯網”。
- 您已建立混合節點 IAM 角色，並設定內部部署憑證提供者 (AWS Systems Manager 混合啟用或 AWS IAM Roles Anywhere)。如需詳細資訊，請參閱the section called “準備憑證”。
- 您已建立了已啟用混合節點的 Amazon EKS 叢集。如需詳細資訊，請參閱the section called “建立叢集”。
- 您已將混合節點 IAM 角色與 Kubernetes 角色型存取控制 (RBAC) 許可建立關聯。如需詳細資訊，請參閱the section called “準備叢集存取”。

步驟 1：建立 Bottlerocket 設定 TOML 檔案

若要為混合節點設定 Bottlerocket，您需要建立具有必要組態的 `settings.toml` 檔案。TOML 檔案的內容會根據您使用的憑證提供者 (SSM 或 IAM Roles Anywhere) 而有所不同。佈建 Bottlerocket 執行個體時，此檔案將會作為使用者資料傳遞。

> **Note**
>
> 以下提供的 TOML 檔案僅代表將 Bottlerocket VMWare 機器初始化為 EKS 叢集上的節點所需的最低設定。Bottlerocket 提供廣泛的設定，可處理多種不同的使用案例，因此對於混合節點初始化以外的其他組態選項，請參閱 Bottlerocket 文件，以取得您使用之 Bottlerocket 版本的所有文件化設定的完整清單（例如，以下是 Bottlerocket 1.51.x 可用的所有設定）。

SSM

如果您使用 AWS Systems Manager 做為登入資料提供者，請使用下列內容建立`settings.toml`檔案：

```
[settings.kubernetes]
cluster-name = "<cluster-name>"
api-server = "<api-server-endpoint>"
cluster-certificate = "<cluster-certificate-authority>"
hostname-override = "<hostname>"
provider-id = "eks-hybrid:///<region>/<cluster-name>/<hostname>"
authentication-mode = "aws"
```

```
cloud-provider = ""
server-tls-bootstrap = true

[settings.network]
hostname = "<hostname>"

[settings.aws]
region = "<region>"

[settings.kubernetes.credential-providers.ecr-credential-provider]
enabled = true
cache-duration = "12h"
image-patterns = [
    "*.dkr.ecr.*.amazonaws.com",
    "*.dkr.ecr.*.amazonaws.com.cn",
    "*.dkr.ecr.*.amazonaws.eu",
    "*.dkr.ecr-fips.*.amazonaws.com",
    "*.dkr.ecr-fips.*.amazonaws.eu",
    "public.ecr.aws"
]

[settings.kubernetes.node-labels]
"eks.amazonaws.com/compute-type" = "hybrid"
"eks.amazonaws.com/hybrid-credential-provider" = "ssm"

[settings.host-containers.admin]
enabled = true
user-data = "<base64-encoded-admin-container-userdata>"

[settings.bootstrap-containers.eks-hybrid-setup]
mode = "always"
user-data = "<base64-encoded-bootstrap-container-userdata>"

[settings.host-containers.control]
enabled = true
```

使用下列值取代預留位置：

- <cluster-name>：Amazon EKS 叢集的名稱。
- <api-server-endpoint>：叢集的 API 伺服器端點。
- <cluster-certificate-authority>：叢集的 base64 編碼 CA 套件。
- <region>：託管叢集 AWS 的區域，例如 "us-east-1"。

- <hostname>：Bottlerocket 執行個體的主機名稱，其也會設定為節點名稱。這可以是您選擇的任何唯一值，不過必須遵循 Kubernetes 物件命名慣例。此外，您使用的主機名稱不可超過 64 個字元。注意：使用 SSM 提供商時，在向 SSM 註冊執行個體之後，此主機名稱和節點名稱將會取代為受管執行個體 ID (例如 mi-* ID)。
- <base64-encoded-admin-container-userdata>：Bottlerocket 管理員容器組態的 base64 編碼內容。啟用管理員容器可讓您使用 SSH 連接至 Bottlerocket 執行個體，以進行系統勘探和偵錯。雖然這並非必要的設定，但我們建議您將其啟用，以便進行故障診斷。如需使用管理員容器進行身分驗證的詳細資訊，請參閱 Bottlerocket 管理員容器文件。管理員容器採用 JSON 格式的 SSH 使用者和金鑰輸入，例如：

```
{
  "user": "<ssh-user>",
  "ssh": {
    "authorized-keys": [
      "<ssh-authorized-key>"
    ]
  }
}
```

- <base64-encoded-bootstrap-container-userdata>：Bottlerocket 引導容器組態的 base64 編碼內容。如需其組態的詳細資訊，請參閱 Bottlerocket 引導容器文件。引導容器負責將執行個體註冊為 AWS SSM 受管執行個體，並將其聯結為 Amazon EKS 叢集上的 Kubernetes 節點。傳遞至引導容器的使用者資料採用命令調用的形式，而其接受您先前建立的 SSM 混合啟用代碼和 ID 作為輸入：

```
eks-hybrid-ssm-setup --activation-id=<activation-id> --activation-code=<activation-
code> --region=<region>
```

IAM Roles Anywhere

如果您使用 AWS IAM Roles Anywhere 做為登入資料提供者，請使用下列內容建立settings.toml檔案：

```
[settings.kubernetes]
cluster-name = "<cluster-name>"
api-server = "<api-server-endpoint>"
cluster-certificate = "<cluster-certificate-authority>"
hostname-override = "<hostname>"
```

```
provider-id = "eks-hybrid:///<region>/<cluster-name>/<hostname>"
authentication-mode = "aws"
cloud-provider = ""
server-tls-bootstrap = true

[settings.network]
hostname = "<hostname>"

[settings.aws]
region = "<region>"
config = "<base64-encoded-aws-config-file>"

[settings.kubernetes.credential-providers.ecr-credential-provider]
enabled = true
cache-duration = "12h"
image-patterns = [
    "*.dkr.ecr.*.amazonaws.com",
    "*.dkr.ecr.*.amazonaws.com.cn",
    "*.dkr.ecr.*.amazonaws.eu",
    "*.dkr.ecr-fips.*.amazonaws.com",
    "*.dkr.ecr-fips.*.amazonaws.eu",
    "public.ecr.aws"
]

[settings.kubernetes.node-labels]
"eks.amazonaws.com/compute-type" = "hybrid"
"eks.amazonaws.com/hybrid-credential-provider" = "iam-ra"

[settings.host-containers.admin]
enabled = true
user-data = "<base64-encoded-admin-container-userdata>"

[settings.bootstrap-containers.eks-hybrid-setup]
mode = "always"
user-data = "<base64-encoded-bootstrap-container-userdata>"
```

使用下列值取代預留位置：

- <cluster-name>：Amazon EKS 叢集的名稱。
- <api-server-endpoint>：叢集的 API 伺服器端點。
- <cluster-certificate-authority>：叢集的 base64 編碼 CA 套件。
- <region>：託管叢集 AWS 的區域，例如 "us-east-1"

- <hostname>：Bottlerocket 執行個體的主機名稱，其也會設定為節點名稱。這可以是您選擇的任何唯一值，不過必須遵循 Kubernetes 物件命名慣例。此外，您使用的主機名稱不可超過 64 個字元。注意：使用 IAM-RA 提供商時，如果您已使用 "sts:RoleSessionName": "${aws:PrincipalTag/x509Subject/CN}" 資源條件設定混合節點 IAM 角色的信任政策，則節點名稱必須與主機上的憑證的 CN 相符。
- <base64-encoded-aws-config-file>：組態檔案的 base64 AWS 編碼內容。檔案的內容應如下所示：

```
[default]
credential_process = aws_signing_helper credential-process --certificate /root/.aws/
node.crt --private-key /root/.aws/node.key --profile-arn <profile-arn> --role-arn
 <role-arn> --trust-anchor-arn <trust-anchor-arn> --role-session-name <role-session-
name>
```

- <base64-encoded-admin-container-userdata>：Bottlerocket 管理員容器組態的 base64 編碼內容。啟用管理員容器可讓您使用 SSH 連接至 Bottlerocket 執行個體，以進行系統勘探和偵錯。雖然這並非必要的設定，但我們建議您將其啟用，以便進行故障診斷。如需使用管理員容器進行身分驗證的詳細資訊，請參閱 Bottlerocket 管理員容器文件。管理員容器採用 JSON 格式的 SSH 使用者和金鑰輸入，例如：

```
{
  "user": "<ssh-user>",
  "ssh": {
    "authorized-keys": [
      "<ssh-authorized-key>"
    ]
  }
}
```

- <base64-encoded-bootstrap-container-userdata>：Bottlerocket 引導容器組態的 base64 編碼內容。如需其組態的詳細資訊，請參閱 Bottlerocket 引導容器文件。引導容器負責在執行個體上建立 IAM Roles Anywhere 主機憑證和憑證私有金鑰檔案。然後，aws_signing_helper 會使用這些憑證來取得臨時憑證，以便與您的 Amazon EKS 叢集進行身分驗證。傳遞至引導容器的使用者資料採用命令調用的形式，而其接受您先前建立的憑證和私有金鑰的內容作為輸入：

```
eks-hybrid-iam-ra-setup --certificate=<certificate> --key=<private-key>
```

步驟 2：使用使用者資料佈建 Bottlerocket vSphere VM

建構 TOML 檔案後，請在 vSphere VM 建立期間將其作為使用者資料進行傳遞。請記住，必須在 VM 第一次開機之前設定使用者資料。因此，您需要在建立執行個體時提供它，或者如果您想要提前建立 VM，則 VM 必須處於關機狀態，直到您為其設定使用者資料為止。例如，如果使用 govc CLI：

第一次建立 VM

```
govc vm.create \
  -on=true \
  -c=2 \
  -m=4096 \
  -net.adapter=<network-adapter> \
  -net=<network-name> \
  -e guestinfo.userdata.encoding="base64" \
  -e guestinfo.userdata="$(base64 -w0 settings.toml)" \
  -template=<template-name> \
  <vm-name>
```

更新現有 VM 的使用者資料

```
govc vm.create \
    -on=false \
    -c=2 \
    -m=4096 \
    -net.adapter=<network-adapter> \
    -net=<network-name> \
    -template=<template-name> \
    <vm-name>

govc vm.change
    -vm <vm-name> \
    -e guestinfo.userdata="$(base64 -w0 settings.toml)" \
    -e guestinfo.userdata.encoding="base64"

govc vm.power -on <vm-name>
```

在上述區段中，`-e guestinfo.userdata.encoding="base64"` 選項會指定使用者資料採用 base64 編碼。`-e guestinfo.userdata` 選項會將 `settings.toml` 檔案的 base64 編碼內容作為

使用者資料傳遞至 Bottlerocket 執行個體。使用特定值取代預留位置，例如 Bottlerocket OVA 範本和聯網詳細資訊。

步驟 3：驗證混合節點連線

Bottlerocket 執行個體啟動後，其會嘗試加入您的 Amazon EKS 叢集。您可以導覽至叢集的「運算」索引標籤，或執行下列命令，以在 Amazon EKS 主控台中驗證連線：

```
kubectl get nodes
```

> **Important**
>
> 您的節點將處於 `Not Ready` 狀態，這在意料之中，因為您的混合節點上並未執行 CNI。如果您的節點未加入叢集，請參閱 the section called “疑難排解”。

步驟 4：設定混合節點的 CNI

要讓您的混合節點準備好執行應用程式，請繼續執行 the section called “設定 CNI” 上的步驟。

升級叢集的混合節點

升級混合節點的指引類似於在 Amazon EC2 中執行的自我管理的 Amazon EKS 節點。我們建議您在目標 Kubernetes 版本上建立新的混合節點、將現有的應用程式從容移轉至新 Kubernetes 版本上的混合節點，接著從叢集移除舊 Kubernetes 版本上的混合節點。開始升級之前，請務必檢閱 Amazon EKS 最佳實務，以了解升級的相關資訊。Amazon EKS 混合節點對包含雲端節點的 Amazon EKS 叢集具有相同的 Kubernetes 版本支援，包括標準支援和延長支援。

Amazon EKS 混合節點遵循與上游 Kubernetes 相同的節點版本差異政策。Amazon EKS 混合節點不能使用比 Amazon EKS 控制平面更新的版本，而混合節點最多可能比 Amazon EKS 控制平面次要版本低三個 Kubernetes 次要版本。

如果您沒有備用容量，可在目標 Kubernetes 版本上建立新的混合節點以進行切換移轉升級策略，則您也可以使用 Amazon EKS 混合節點 CLI (`nodeadm`) 來就地升級混合節點的 Kubernetes 版本。

> **Important**
>
> 如果您要使用 `nodeadm` 就地升級混合節點，則在關閉舊版 Kubernetes 元件以及安裝和啟動新的 Kubernetes 版本元件的過程中，節點可能會出現停機時間。

⚠ SSM 登入資料提供者所需的 nodeadm 版本

如果您使用 AWS Systems Manager (SSM) 作為混合節點的憑證提供者，則必須使用 nodeadm 版本 1.0.19或更新版本進行新的安裝和升級。舊版 nodeadm包含過期的 SSM 簽署金鑰，並在 nodeadm install期間失敗，並nodeadm upgrade出現下列簽章驗證錯誤：

```
"msg":"Command failed","error":"failed to install ssm installer: validating ssm-setup-cli signature: Signature Verification Error: No matching signature"
```

若要解決此錯誤，請在執行 或 之前下載最新版本的 nodeadm upgrade。 nodeadm nodeadm install

先決條件

升級之前，請確認您已完成下列先決條件。

- 混合節點升級的目標 Kubernetes 版本必須等於或小於 Amazon EKS 控制平面版本。
- 如果您遵循切換移轉升級策略，則在目標 Kubernetes 版本上安裝的新混合節點必須符合 the section called “先決條件” 要求。這包括您在 Amazon EKS 叢集建立期間傳遞的遠端節點網路 CIDR 內的 IP 位址。
- 對於切換移轉和就地升級，混合節點必須能夠存取必要網域，以提取混合節點相依性的新版本。
- 您必須在用來與 Amazon EKS Kubernetes API 端點互動的本機電腦或執行個體上安裝 kubectl。
- 您的 CNI 版本必須支援您要升級到的 Kubernetes 版本。如果不能，則請在先升級 CNI 版本，然後再升級您的混合節點。如需詳細資訊，請參閱the section called “設定 CNI”。

切換移轉 (藍綠) 升級

切換移轉升級是指使用目標 Kubernetes 版本在新主機上建立新的混合節點、將現有的應用程式從容移轉至目標 Kubernetes 版本上的新混合節點，以及從叢集移除舊 Kubernetes 版本上的混合節點的過程。此策略也稱為藍綠移轉。

1. 請遵循 the section called “連接混合節點” 步驟，將您的新主機連接為混合節點。執行 nodeadm install 命令時，請使用您的目標 Kubernetes 版本。
2. 啟用目標 Kubernetes 版本上的新混合節點與舊 Kubernetes 版本上的混合節點之間的通訊。此組態可讓您在將工作負載移轉至目標 Kubernetes 版本上的混合節點時，Pod 之間能夠彼此進行通訊。
3. 確認目標 Kubernetes 版本上的混合節點已成功加入叢集，且狀態為就緒。

4. 使用下列命令，將每個您想要移除的節點標記為不可排程。這樣就不會在您要取代的節點上排程或重新排程新的 Pod。如需詳細資訊，請參閱 Kubernetes 文件中的 kubectl cordon。使用舊 Kubernetes 版本上的混合節點的名稱取代 NODE_NAME。

```
kubectl cordon NODE_NAME
```

您可以使用下列程式碼片段來識別及包圍隔離特定 Kubernetes 版本 (在此情況下為 1.28) 的所有節點。

```
K8S_VERSION=1.28
for node in $(kubectl get nodes -o json | jq --arg K8S_VERSION
 "$K8S_VERSION" -r '.items[] | select(.status.nodeInfo.kubeletVersion |
 match("\($K8S_VERSION)")).metadata.name')
do
    echo "Cordoning $node"
    kubectl cordon $node
done
```

5. 如果您目前的部署在混合節點上執行的 CoreDNS 複本少於兩個，請將部署擴增為至少兩個複本。我們建議您在混合節點上執行至少兩個 CoreDNS 複本，以在正常操作期間實現彈性。

```
kubectl scale deployments/coredns --replicas=2 -n kube-system
```

6. 使用下列命令，耗盡清空您想要從叢集移除的舊 Kubernetes 版本上的每個混合節點。如需耗盡節點的詳細資訊，請參閱 Kubernetes 文件中的安全地耗盡節點。使用舊 Kubernetes 版本上的混合節點的名稱取代 NODE_NAME。

```
kubectl drain NODE_NAME --ignore-daemonsets --delete-emptydir-data
```

您可以使用下列程式碼片段來識別及耗盡特定 Kubernetes 版本 (在此情況下為 1.28) 的所有節點。

```
K8S_VERSION=1.28
for node in $(kubectl get nodes -o json | jq --arg K8S_VERSION
 "$K8S_VERSION" -r '.items[] | select(.status.nodeInfo.kubeletVersion |
 match("\($K8S_VERSION)")).metadata.name')
do
    echo "Draining $node"
    kubectl drain $node --ignore-daemonsets --delete-emptydir-data
done
```

7. 您可以使用 nodeadm 從主機停止和移除混合節點成品。您必須與擁有 root/sudo 權限的使用者一同執行 nodeadm。根據預設，如果節點上還有剩餘的 Pod，則 nodeadm uninstall 不會繼續。如需更多資訊，請參閱the section called “混合節點 nodeadm”。

```
nodeadm uninstall
```

8. 若停止並解除安裝混合節點成品，則請從您的叢集中移除節點資源。

```
kubectl delete node node-name
```

   您可以使用下列程式碼片段來識別及刪除特定 Kubernetes 版本 (在此情況下為 1.28) 的所有節點。

```
K8S_VERSION=1.28
for node in $(kubectl get nodes -o json | jq --arg K8S_VERSION
 "$K8S_VERSION" -r '.items[] | select(.status.nodeInfo.kubeletVersion |
 match("\($K8S_VERSION)")).metadata.name')
do
    echo "Deleting $node"
    kubectl delete node $node
done
```

9. 視您選擇的 CNI 而定，在執行上述步驟後，您的混合節點上可能會有成品剩餘。如需詳細資訊，請參閱the section called “設定 CNI”。

就地升級

就地升級程序是指使用 nodeadm upgrade 來升級混合節點的 Kubernetes 版本，而無需使用新的實體或虛擬主機和切換移轉策略。nodeadm upgrade 程序會關閉在混合節點上執行的現有的較舊 Kubernetes 元件、解除安裝現有的較舊 Kubernetes 元件、安裝新的目標 Kubernetes 元件，以及啟動新的目標 Kubernetes 元件。強烈建議您一次升級一個節點，以最大限度地降低對混合節點上執行的應用程式的影響。此程序的持續時間取決於您的網路頻寬和延遲。

1. 使用下列命令，將您要升級的節點標記為不可排程。這樣就不會在您要升級的節點上排程或重新排程新的 Pod。如需詳細資訊，請參閱 Kubernetes 文件中的 kubectl cordon。使用您要升級的混合節點的名稱取代 NODE_NAME

```
kubectl cordon NODE_NAME
```

2. 使用下列命令，耗盡您要升級的節點。如需耗盡節點的詳細資訊，請參閱 Kubernetes 文件中的安全地耗盡節點。使用您要升級的混合節點的名稱取代 NODE_NAME。

```
kubectl drain NODE_NAME --ignore-daemonsets --delete-emptydir-data
```

3. 在您要升級的混合節點上執行 nodeadm upgrade。您必須與擁有 root/sudo 權限的使用者一同執行 nodeadm。節點的名稱會透過 AWS SSM 和 IAM Roles Anywhere AWS 憑證提供者的升級來保留。您無法在升級程序期間變更憑證提供者。如需 nodeConfig.yaml 的組態值，請參閱 the section called “混合節點 nodeadm”。使用您要升級的目標 Kubernetes 版本取代 K8S_VERSION。

```
nodeadm upgrade K8S_VERSION -c file://nodeConfig.yaml
```

4. 若要允許於升級之後在節點上排程 Pod，請輸入下列內容。使用節點的名稱取代 NODE_NAME。

```
kubectl uncordon NODE_NAME
```

5. 觀察混合節點的狀態，等待節點關閉，然後以就緒狀態重新啟動新的 Kubernetes 版本。

```
kubectl get nodes -o wide -w
```

混合節點的修補程式安全更新

本主題會說明針對混合節點上執行的特定套件和相依性，就地執行安全更新修補的程序。基於最佳實務，我們建議您定期更新您的混合節點，以接收 CVE 和安全修補程式。

如需升級 Kubernetes 版本的步驟，請參閱 the section called “升級混合節點”。

可能需要安全修補的軟體範例之一是 containerd。

Containerd

containerd 是 EKS 混合節點的標準 Kubernetes 容器執行時期和核心相依性，可用於管理容器生命週期，包括提取映像和管理容器執行。在混合節點containerd上，您可以透過 nodeadm CLI 或手動安裝 。視節點的作業系統而定，nodeadm 會從作業系統分散式套件或 Docker 套件安裝 containerd。

當 containerd 中的 CVE 發布後，您可透過下列選項將您混合節點上的 containerd 升級至修補程式版本。

步驟 1：檢查修補程式是否已發布至套件管理工具

您可以參考對應的安全公告，檢查 CVE containerd 修補程式是否已發布至每個各自的作業系統套件管理工具：

- Amazon Linux 2023
- RHEL
- Ubuntu 20.04
- Ubuntu 22.04
- Ubuntu 24.04

如果您使用 Docker 儲存庫作為 containerd 的來源，您可以檢查 Docker 安全公告，以識別 Docker 儲存庫中提供的修補版本。

步驟 2：選擇安裝修補程式的方法

您可以透過三種方法來在節點上就地修補和安裝安全升級。您可使用哪種方法取決於套件管理工具中的作業系統是否提供該修補程式：

1. 使用發布至套件管理工具的 nodeadm upgrade 安裝修補程式，請參閱步驟 2 a。
2. 使用套件管理工具直接安裝修補程式，請參閱步驟 2 b。
3. 安裝尚未在套件管理工具中發布的自訂修補程式。請注意，對於 containerd 的自訂修補程式，存在特殊考量，請參閱步驟 2 c。

步驟 2 a：使用 **nodeadm upgrade** 進行修補

確認 containerd CVE 修補程式已發布至作業系統或 Docker 儲存庫 (Apt 或 RPM) 後，您可以使用 nodeadm upgrade 命令升級至 containerd 的最新版本。由於這並非 Kubernetes 版本升級，您必須將目前的 Kubernetes 版本傳遞至 nodeadm 升級命令。

```
nodeadm upgrade K8S_VERSION --config-source file:///root/nodeConfig.yaml
```

步驟 2 b：使用作業系統套件管理工具進行修補

或者，您也可以透過各自的套件管理工具進行更新，並使用它來升級 containerd 套件，如下所示。

Amazon Linux 2023

```
sudo yum update -y
sudo yum install -y containerd
```

RHEL

```
sudo yum install -y yum-utils
sudo yum-config-manager --add-repo https://download.docker.com/linux/rhel/docker-
ce.repo
sudo yum update -y
sudo yum install -y containerd
```

Ubuntu

```
sudo mkdir -p /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/
docker.asc
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc]
 https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "${UBUNTU_CODENAME:-$VERSION_CODENAME}") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt update -y
sudo apt install -y --only-upgrade containerd.io
```

步驟 2 c：套件管理工具中尚未發布的 **Containerd** CVE 修補程式

如果 containerd 的修補程式版本僅可透過套件管理工具以外的其他方式獲得，例如在 GitHub 發行版本中，則您可以從官方 GitHub 網站安裝 containerd。

1. 如果機器已將叢集加入為混合節點，則需要執行 nodeadm uninstall 命令。
2. 安裝官方 containerd 二進位檔。您可以使用 GitHub 上的步驟官方安裝步驟。
3. 執行 nodeadm install 命令，並將 --containerd-source 引數設定為 none，因此可透過 nodeadm 略過 containerd 安裝。對於節點正在執行的任何作業系統，您可以使用 containerd 來源中的 none 值。

   ```
   nodeadm install K8S_VERSION --credential-provider CREDS_PROVIDER --containerd-source
    none
   ```

移除混合節點

本主題會說明如何從 Amazon EKS 叢集中刪除混合節點。您必須使用您選擇的 Kubernetes 相容工具 (例如 kubectl) 刪除混合節點。節點物件從 Amazon EKS 叢集移除時，即會停止收取混合節點的費用。如需混合節點定價的詳細資訊，請參閱 Amazon EKS 定價。

> **Important**
>
> 移除節點會對節點上執行的工作負載產生干擾。刪除混合節點之前，建議您先耗盡節點，以將 Pod 移動至另一個作用中的節點。如需耗盡節點的詳細資訊，請參閱 Kubernetes 文件中的安全地耗盡節點。

從您用來與 Amazon EKS 叢集的 Kubernetes API 端點互動的本機機器或執行個體，執行下列 kubectl 步驟。如果您使用特定 `kubeconfig` 檔案，則請使用 `--kubeconfig` 旗標。

步驟 1：列出您的節點

```
kubectl get nodes
```

步驟 2：耗盡您的節點

如需 `kubectl drain` 命令的詳細資訊，請參閱 Kubernetes 文件中的 kubectl 耗盡。

```
kubectl drain --ignore-daemonsets <node-name>
```

步驟 3：停止並解除安裝混合節點成品

您可以使用 Amazon EKS 混合節點 CLI (`nodeadm`)，以從主機停止和移除混合節點成品。您必須與擁有 root/sudo 權限的使用者一同執行 `nodeadm`。根據預設，如果節點上還有剩餘的 Pod，則 `nodeadm uninstall` 不會繼續。如果您使用 AWS Systems Manager (SSM) 作為憑證提供者，則 `nodeadm uninstall` 命令會將主機取消註冊為 AWS SSM 受管執行個體。如需詳細資訊，請參閱 the section called “混合節點 nodeadm”。

```
nodeadm uninstall
```

步驟 4：從叢集中刪除節點

若停止並解除安裝混合節點成品，則請從您的叢集中移除節點資源。

```
kubectl delete node <node-name>
```

步驟 5：檢查剩餘的成品

視您選擇的 CNI 而定，在執行上述步驟後，您的混合節點上可能會有成品剩餘。如需詳細資訊，請參閱「the section called “設定 CNI”」。

設定混合節點的應用程式聯網、附加元件和 Webhook

為混合節點建立 EKS 叢集後，請設定應用程式聯網的其他功能 (CNI、BGP、傳入、負載平衡、網路政策)、附加元件、Webhook 和代理設定。如需與混合節點相容的 EKS 和社群附加元件的完整清單，請參閱 the section called “設定附加元件”。

EKS 叢集洞察 EKS 包括混合節點設定中錯誤組態的洞察檢查，這些設定可能會使叢集或工作負載的功能受損。如需有關叢集洞察的詳細資訊，請參閱 the section called “叢集洞察”。

以下清單列出了您可與混合節點搭配使用的常見功能和附加元件：

- 容器聯網介面 (CNI)：AWS 支援 Cilium 作為混合節點的 CNI。如需詳細資訊，請參閱 the section called “設定 CNI”。請注意，AWS VPC CNI 無法與混合節點搭配使用。
- CoreDNS 和 **`kube-proxy`** ：CoreDNS 和 `kube-proxy` 會在混合節點加入 EKS 叢集時自動安裝。這些附加元件可以在叢集建立後作為 EKS 附加元件進行管理。
- 傳入及負載平衡：您可以將 AWS Load Balancer 控制器和 Application Load Balancer (ALB) 或 Network Load Balancer (NLB) 與在混合節點上執行的工作負載的目標類型 `ip` 搭配使用。針對在混合節點上執行的工作負載，AWS 支援 Cilium 的內建傳入、閘道和 Kubernetes Service 負載平衡功能。如需詳細資訊，請參閱the section called “設定 Ingress”及the section called “設定 LoadBalancer 服務”。
- 指標：您可以使用 Amazon Managed Service for Prometheus (AMP) 無代理程式湊集器、AWS Ditro for Open Telemetry (ADOT) 和具有混合節點的 Amazon CloudWatch 可觀測性代理程式。若要對混合節點上的 Pod 指標使用 AMP 無代理程式湊集器，您的 Pod 必須可從您用於 EKS 叢集的 VPC 存取。
- 日誌：您可以為已啟用混合節點的叢集啟用 EKS 控制平面記錄。您可以使用 ADOT EKS 附加元件和 Amazon CloudWatch 可觀測性代理程式 EKS 附加元件進行混合節點和 Pod 記錄。
- Pod 身分識別和 IRSA：您可以搭配混合節點上執行的應用程式使用 EKS Pod 身分識別和服務帳戶的 IAM 角色 (IRSA)，以為在混合節點上執行的 Pod 與其他 AWS 服務啟用精細存取。
- Webhook：如果您正在執行 Webhook，請參閱 the section called “設定 Webhook” 以了解相關考量事項和步驟，即如果您無法讓內部部署 Pod 網路可路由，則可選擇在雲端節點上執行 Webhook。

- 代理：如果您在內部部署環境中為離開資料中心或邊緣環境的流量使用代理伺服器，則您可以將混合節點和叢集設定為使用代理伺服器。如需詳細資訊，請參閱 the section called “設定代理”。

主題

- 設定混合節點的 CNI
- 設定混合節點的附加元件
- 設定混合節點的 Webhook
- 設定混合節點的代理
- 設定混合節點的 Cilium BGP
- 設定混合節點的 Kubernetes Ingress
- 為混合節點設定 LoadBalancer 類型的服務
- 設定混合節點的 Kubernetes 網路政策

設定混合節點的 CNI

Cilium 是 AWS Amazon EKS 混合節點支援的容器聯網界面 (CNI)。您必須為混合節點安裝 CNI，才能做好準備以為工作負載提供服務。在 CNI 執行之前，混合節點會顯示為 `Not Ready` 狀態。您可以使用您選擇的工具 (例如 Helm) 來管理 CNI。此頁面的說明涵蓋 Cilium 生命週期管理 (安裝、升級、刪除)。如需如何為傳入、載入平衡和網路政策設定 Cilium，請參閱 the section called “Cilium Ingress 和 Cilium Gateway 概觀”、the section called “服務類型 LoadBalancer” 和 the section called “設定網路政策”。

在 AWS 雲端節點上執行 AWS 時，不支援 Cilium。Amazon VPC CNI 與混合節點不相容，且 VPC CNI 已針對 `eks.amazonaws.com/compute-type: hybrid` 標籤設定反親和性。

此頁面上先前的 Calico 文件已移至 EKS 混合範例儲存庫。

版本相容性

Amazon EKS 支援的每個 Kubernetes 版本`v1.18.x`都支援 Cilium 版本 `v1.17.x`和 EKS 混合節點。

> **Note**
>
> Cilium v1.18.3 核心需求：由於核心需求 (Linux 核心 >= 5.10)，不支援 Cilium v1.18.3：

- Ubuntu 20.04

- Red Hat Enterprise Linux (RHEL) 8

如需系統需求，請參閱 Cilium 系統需求。

如需 Amazon EKS 支援的 Kubernetes 版本，請參閱 Kubernetes 版本支援。EKS 混合節點對包含雲端節點的 Amazon EKS 叢集具有相同的 Kubernetes 版本支援。

支援的功能

AWS 維護以開放原始碼 Cilium 專案為基礎的 EKS 混合節點的 Cilium 組建。若要從 取得 Cilium AWS 的支援，您必須使用 AWS維護的 Cilium 組建和支援的 Cilium 版本。

AWS 為下列 Cilium 功能的預設組態提供技術支援，以便與 EKS 混合節點搭配使用。如果您計劃在 AWS 支援範圍之外使用功能，建議您取得 Cilium 的替代商業支援，或擁有內部專業知識來故障診斷並對 Cilium 專案提供修正。

Cilium 功能	支援者 AWS
Kubernetes 網路一致性	是
核心叢集連線	是
IP 系列	IPv4
生命週期管理	Helm
聯網模式	VXLAN 封裝
IP 位址管理 (IPAM)	Cilium IPAM 叢集範圍
網路政策	Kubernetes 網路政策
邊界閘道協定 (BGP)	Cilium BGP 控制平面
Kubernetes Ingress	Cilium Ingress、Cilium Gateway
Service LoadBalancer IP 配置	Cilium Load Balancer IPAM
Service LoadBalancer IP 位址公告	Cilium BGP 控制平面
kube-proxy 取代	是

Cilium 考量事項

- Helm 儲存庫 - 在 Amazon EKS Cilium/Cilium 的 Amazon Elastic Container Registry Public (Amazon ECR Public) 中 AWS 託管 Cilium Helm Chart。 https://gallery.ecr.aws/eks/cilium/cilium可用的版本包括：
  - Cilium 1.17.9 版： `oci://public.ecr.aws/eks/cilium/cilium:1.17.9-0`
  - Cilium 1.18.3 版： `oci://public.ecr.aws/eks/cilium/cilium:1.18.3-0`

    本主題中的命令使用此儲存庫。請注意，某些 `helm repo` 命令不適用於 Amazon ECR Public 中的 Helm 儲存庫，因此您無法從本機 Helm 儲存庫名稱引用此儲存庫。反之，大多數命令中會使用完整的 URI。
- 根據預設，Cilium 已設定為以覆蓋/通道模式執行，並以 VXLAN 作為封裝法。此模式對基礎實體網路的要求最低。
- 根據預設，Cilium 會將離開叢集的所有 Pod 流量的來源 IP 位址偽裝為節點的 IP 位址。如果您停用偽裝，則您的 Pod CIDR 必須在內部部署網路上可路由。
- 如果您在混合節點上執行 Webhook，您的 Pod CIDR 必須在內部部署網路上可路由。如果您的 Pod CIDR 在內部部署網路上不可路由，那麼建議您在相同叢集的雲端節點上執行 Webhook。如需詳細資訊，請參閱 the section called “設定 Webhook” 和 the section called “準備聯網”。
- AWS 建議使用 Cilium 的內建 BGP 功能，讓您的 Pod CIDRs可在內部部署網路上路由。如需如何使用混合節點設定 Cilium BGP 的詳細資訊，請參閱 the section called “設定 BPA”。
- Cilium 中的預設 IP 位址管理 (IPAM) 稱為叢集範圍，而 Cilium 運算子會基於使用者設定的 Pod CIDR 為每個節點配置 IP 位址。

在混合節點上安裝 Cilium

程序

1. 建立稱為 `cilium-values.yaml` 的 YAML 檔案。下列範例透過為 Cilium 代理程式和運算子設定 `eks.amazonaws.com/compute-type: hybrid` 標籤的親和性，將 Cilium 設定為僅在混合節點上執行。
   - 使用您為 EKS 叢集遠端 Pod 網路設定的相同 Pod CIDR，進而設定 `clusterPoolIpv4PodCIDRList`。例如 `10.100.0.0/24`。Cilium 運算子會從設定的 `clusterPoolIpv4PodCIDRList` IP 空間內配置 IP 位址配量。您的 Pod CIDR 不得與內部部署節點 CIDR、VPC CIDR 或 Kubernetes Service CIDR 重疊。
   - 基於每個節點所需的 Pod 來設定 `clusterPoolIpv4MaskSize`。例如，對於每個節點 128 個 Pod 的 /25 區段大小，其值為 25。

- 在叢集上部署 Cilium 之後，請勿變更 `clusterPoolIpv4PodCIDRList` 或 `clusterPoolIpv4MaskSize`，如需詳細資訊，請參閱展開叢集集區。
- 如果您以 kube-proxy 取代模式執行 Cilium，請在 Helm 值中設定 `kubeProxyReplacement: "true"`，並確保您並未在與 Cilium 相同的節點上執行現有的 kube-proxy 部署。
- 以下範例會停用 Cilium 用於 L7 網路政策和傳入的 Envoy Layer 7 (L7) 代理。如需詳細資訊，請參閱the section called “設定網路政策”及the section called “Cilium Ingress 和 Cilium Gateway 概觀”。
- 以下範例會設定 `loadBalancer.serviceTopology`：如果您為服務設定服務流量分佈，其會正確運作，則此為 `true`。如需詳細資訊，請參閱the section called “設定服務流量分佈”。
- 如需 Cilium 的 Helm 值的完整清單，請參閱 Cilium 文件中的 Helm 參考。

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
        - key: eks.amazonaws.com/compute-type
          operator: In
          values:
          - hybrid
ipam:
  mode: cluster-pool
  operator:
    clusterPoolIPv4MaskSize: 25
    clusterPoolIPv4PodCIDRList:
    - POD_CIDR
loadBalancer:
  serviceTopology: true
operator:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: eks.amazonaws.com/compute-type
            operator: In
            values:
              - hybrid
  unmanagedPodWatcher:
    restart: false
```

```
loadBalancer:
  serviceTopology: true
envoy:
  enabled: false
kubeProxyReplacement: "false"
```

2. 在叢集上安裝 Cilium。
   - CILIUM_VERSION 將 取代為 Cilium 版本 （ 例如 1.17.9-0或 1.18.3-0)。建議為 Cilium 次要版本使用最新的修補程式版本。
   - 確保您的節點符合您所選版本的核心需求。Cilium v1.18.3 需要 Linux 核心 >= 5.10。
   - 如果您使用特定的 kubeconfig 檔案，請搭配 Helm 安裝命令使用 --kubeconfig 旗標。

```
helm install cilium oci://public.ecr.aws/eks/cilium/cilium \
    --version CILIUM_VERSION \
    --namespace kube-system \
    --values cilium-values.yaml
```

3. 使用下列命令來確認您的 Cilium 安裝是否成功。您應該會看到 cilium-operator 部署和每個混合節點上執行的 cilium-agent。此外，您的混合節點現在應該處於狀態 Ready。如需如何設定 Cilium BGP 將 Pod CIDR 公告至內部部署網路的相關資訊，請繼續 the section called “設定 BPA”。

```
kubectl get pods -n kube-system
```

```
NAME                              READY   STATUS    RESTARTS   AGE
cilium-jjjn8                      1/1     Running   0          11m
cilium-operator-d4f4d7fcb-sc5xn   1/1     Running   0          11m
```

```
kubectl get nodes
```

```
NAME                   STATUS   ROLES    AGE   VERSION
mi-04a2cf999b7112233   Ready    <none>   19m   v1.31.0-eks-a737599
```

在混合節點上升級 Cilium

升級 Cilium 部署之前，請仔細檢閱 Cilium 升級文件和升級備註，以了解目標 Cilium 版本中的變更。

1. 請確保您已在命令列環境中安裝了 helm CLI。如需安裝說明，請參閱 Helm 文件。

2. 執行 Cilium 升級預檢檢查。使用目標 Cilium 版本取代 `CILIUM_VERSION`。我們建議您為 Cilium 次要版本執行最新的修補程式版本。您可以在 Cilium 文件的穩定版本部分中找到指定次要 Cilium 版本的最新修補程式版本。

```
helm install cilium-preflight oci://public.ecr.aws/eks/cilium/cilium --version
 CILIUM_VERSION \
  --namespace=kube-system \
  --set preflight.enabled=true \
  --set agent=false \
  --set operator.enabled=false
```

3. 套用 `cilium-preflight.yaml` 之後，請確保 READY Pod 的數量與執行的 Cilium Pod 數量相同。

```
kubectl get ds -n kube-system | sed -n '1p;/cilium/p'
```

```
NAME                      DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE
 SELECTOR   AGE
cilium                    2         2         2       2            2           <none>
          1h20m
cilium-pre-flight-check   2         2         2       2            2           <none>
          7m15s
```

4. 只要就緒 Pod 的數量相等，請確保 Cilium 預檢部署也標記為 READY 1/1。如果顯示 READY 0/1，請參閱 CNP 驗證部分並解決部署問題，然後再繼續升級。

```
kubectl get deployment -n kube-system cilium-pre-flight-check -w
```

```
NAME                      READY   UP-TO-DATE   AVAILABLE   AGE
cilium-pre-flight-check   1/1     1            0           12s
```

5. 刪除預檢

```
helm uninstall cilium-preflight --namespace kube-system
```

6. 在執行 `helm upgrade` 命令之前，請保留 `existing-cilium-values.yaml` 中部署的值，或在執行升級命令時，使用 `--set` 命令列選項進行設定。升級操作會覆寫 Cilium ConfigMap，因此請務必在升級時傳遞您的組態值。

```
helm get values cilium --namespace kube-system -o yaml > existing-cilium-values.yaml
```

7. 在一般叢集操作期間，所有 Cilium 元件都應執行相同的版本。下列步驟說明如何將所有元件從一個穩定版本升級至更新的穩定版本。從一個次要版本升級到另一個次要版本時，建議先升級到現有 Cilium 次要版本的最新修補程式版本。若要將中斷降至最低，請將 upgradeCompatibility 選項設定為您在此叢集中安裝的初始 Cilium 版本。

```
helm upgrade cilium oci://public.ecr.aws/eks/cilium/cilium --version CILIUM_VERSION \
  --namespace kube-system \
  --set upgradeCompatibility=1.X \
  -f existing-cilium-values.yaml
```

8. (選用) 如果您因為問題而需要復原升級，請執行下列命令。

```
helm history cilium --namespace kube-system
helm rollback cilium [REVISION] --namespace kube-system
```

從混合節點中刪除 Cilium

1. 執行下列命令以從叢集中解除安裝所有 Cilium 元件。請注意，解除安裝 CNI 可能會影響節點和 Pod 的運作狀態，因此不應在生產叢集上執行。

```
helm uninstall cilium --namespace kube-system
```

   從叢集移除 CNI 時，依預設不會移除 Cilium 設定的介面和路由，如需詳細資訊，請參閱 GitHub 問題。

2. 若要清除磁碟上的組態檔案和資源，如果您使用標準組態目錄，則可以移除 GitHub 上 Cilium 儲存庫中 cni-uninstall.sh 指令碼顯示的檔案。

3. 若要從叢集中移除 Cilium 自訂資源定義 (CRD)，您可以執行下列命令。

```
kubectl get crds -oname | grep "cilium" | xargs kubectl delete
```

設定混合節點的附加元件

此頁面說明在 Amazon EKS 混合節點上執行 AWS 附加元件和社群附加元件的考量。若要進一步了解 Amazon EKS 附加元件，以及從叢集建立、升級和移除附加元件的程序，請參閱 the section called

"Amazon EKS 附加元件"。除非此頁面另有說明，否則對於具有混合節點的 Amazon EKS 叢集，建立、升級和移除 Amazon EKS 附加元件的程序與在 AWS Cloud 中執行節點的 Amazon EKS 叢集相同。只有此頁面中包含的附加元件已經過驗證，可與 Amazon EKS 混合節點相容。

下列 AWS 附加元件與 Amazon EKS 混合節點相容。

AWS 附加元件	相容的附加元件版本
kube-proxy	v1.25.14-eksbuild.2 及更新版本
CoreDNS	v1.9.3-eksbuild.7 及更新版本
AWS Distro for OpenTelemetry (ADOT)	v0.102.1-eksbuild.2 及更新版本
CloudWatch 可觀測性代理程式	v2.2.1-eksbuild.1 及更新版本
EKS Pod 身分識別代理程式	• v1.3.3-eksbuild.1 及更新版本，Bottlerocket 除外 • v1.3.7-eksbuild.2 及更新版本，適用於 Bottlerocket
節點監控代理程式	v1.2.0-eksbuild.1 及更新版本
CSI 快照控制器	v8.1.0-eksbuild.1 及更新版本
AWS Kubernetes 專用私有 CA 連接器	v1.6.0-eksbuild.1 及更新版本
Amazon FSx CSI 驅動程式	v1.7.0-eksbuild.1 及更高版本
AWS Secrets Store CSI 驅動程式提供者	v2.1.1-eksbuild.1 及更高版本

下列社群附加元件與 Amazon EKS 混合節點相容。若要進一步了解社群附加元件，請參閱 the section called "社群附加元件"。

社群附加元件	相容的附加元件版本
Kubernetes 指標伺服器	v0.7.2-eksbuild.1 及更新版本
cert-manager	v1.17.2-eksbuild.1 及更新版本

社群附加元件	相容的附加元件版本
Prometheus Node Exporter	v1.9.1-eksbuild.2 及更新版本
kube-state-metrics	v2.15.0-eksbuild.4 及更新版本
外部 DNS	v0.19.0-eksbuild.1 及更新版本

除了上表中的 Amazon EKS 附加元件之外，Amazon Managed Service for Prometheus 收集器及適用於應用程式傳入 (HTTP) 和負載平衡 (TCP/UDP) 的 AWS Load Balancer 控制器可與混合節點相容。

有些 AWS 附加元件和社群附加元件與 Amazon EKS 混合節點不相容。這些附加元件的最新版本對套用至混合節點的預設 `eks.amazonaws.com/compute-type: hybrid` 標籤具有反親和性規則。這可以防止它們在叢集中部署時於混合節點上執行。如果您有在 AWS Cloud 中同時執行混合節點和節點的叢集，您可以將叢集中的這些附加元件部署到在 AWS Cloud 中執行的節點。Amazon VPC CNI 與混合節點不相容，並且支援 Cilium 和 Calico 作為 Amazon EKS 混合節點的容器聯網介面 (CNI)。如需詳細資訊，請參閱the section called “設定 CNI”。

AWS 附加元件

以下各節說明在混合節點上執行相容 AWS 附加元件與其他 Amazon EKS 運算類型之間的差異。

kube-proxy 和 CoreDNS

當您使用 AWS API 和 AWS SDKs 建立 EKS 叢集時，EKS 預設會將 kube-proxy 和 CoreDNS 安裝為自我管理附加元件，包括來自 AWS CLI 的 。建立叢集後，您可以使用 Amazon EKS 附加元件覆寫這些附加元件。如需有關 the section called “kube-proxy” 和 the section called “CoreDNS” 的詳細資訊，請參閱 EKS 文件。如果您執行的混合模式叢集同時具有混合節點和 AWS Cloud 中的節點， AWS 建議在混合節點上至少有一個 CoreDNS 複本，並在 AWS Cloud 中的節點上至少有一個 CoreDNS 複本。如需組態步驟，請參閱 the section called “設定 CoreDNS 複本”。

CloudWatch 可觀測性代理程式

CloudWatch 可觀測性代理程式運算子使用 Webhook。如果您在混合節點上執行運算子，您的內部部署 Pod CIDR 必須在內部部署網路上可路由，而且您必須使用遠端 Pod 網路設定 EKS 叢集。如需詳細資訊，請參閱設定混合節點的 Webhook。

節點層級指標不適用於混合節點，因為 CloudWatch Container Insights 取決於節點層級指標的執行個體中繼資料服務 (IMDS) 的可用性。叢集、工作負載、Pod 和容器層級指標可供混合節點使用。

遵循使用 Amazon CloudWatch 可觀測性安裝 CloudWatch 代理程式中所述的步驟安裝附加元件後，必須先更新附加元件資訊清單，然後代理程式才能在混合節點上成功執行。編輯叢集上的 amazoncloudwatchagents 資源，以新增 RUN_WITH_IRSA 環境變數，如下所示。

```
kubectl edit amazoncloudwatchagents -n amazon-cloudwatch cloudwatch-agent
```

```
apiVersion: v1
items:
- apiVersion: cloudwatch.aws.amazon.com/v1alpha1
  kind: AmazonCloudWatchAgent
  metadata:
    ...
    name: cloudwatch-agent
    namespace: amazon-cloudwatch
    ...
  spec:
    ...
    env:
    - name: RUN_WITH_IRSA # <-- Add this
      value: "True" # <-- Add this
    - name: K8S_NODE_NAME
      valueFrom:
        fieldRef:
          fieldPath: spec.nodeName
          ...
```

適用於混合節點的 Amazon Managed Prometheus 受管收集器

Amazon Managed Service for Prometheus (AMP) 受管收集器包含一個抓取器，並且可從 Amazon EKS 叢集中的資源探索和收集指標。AMP 會為您管理抓取器，而無需自行管理任何執行個體、代理程式或抓取器。

您可以使用 AMP 受管收集器，而無需任何其他專屬於混合節點的組態。不過，混合節點上應用程式的指標端點必須可從 VPC 連線，包括從 VPC 路由到遠端 Pod 網路 CIDR 及內部部署防火牆中開啟的連接埠。此外，您的叢集必須擁有私有叢集端點存取權。

請遵循《Amazon Managed Service for Prometheus 使用者指南》中的使用 AWS 受管收集器中的步驟。

AWS Distro for OpenTelemetry (ADOT)

您可以使用 AWS Distro for OpenTelemetry (ADOT) 附加元件，從混合節點上執行的應用程式收集指標、日誌和追蹤資料。ADOT 使用許可 Webhook 來變更和驗證收集器自訂資源請求。如果您在混合節點上執行 ADOT 運算子，您的內部部署 Pod CIDR 必須在內部部署網路上可路由，而且您必須使用遠端 Pod 網路設定 EKS 叢集。如需詳細資訊，請參閱設定混合節點的 Webhook。

請遵循 AWS Distro for OpenTelemetry 文件中的使用 EKS 附加元件的 Distro for OpenTelemetry 入門中的步驟。 AWS OpenTelemetry

AWS Load Balancer控制器

您可以使用AWS Load Balancer控制器和 Application Load Balancer (ALB) 或 Network Load Balancer (NLB) 搭配混合節點上ip工作負載的目標類型。搭配 ALB 或 NLB 使用的 IP 目標必須可路由 AWS。The AWS Load Balancer 控制器也使用 Webhook。如果您在混合節點上執行 AWS Load Balancer控制器運算子，您的內部部署 Pod CIDR 必須在內部部署網路上可路由，而且您必須使用遠端 Pod 網路設定 EKS 叢集。如需詳細資訊，請參閱設定混合節點的 Webhook。

若要安裝 AWS Load Balancer控制器，請遵循 the section called “AWS Application Load Balancer”或中的步驟the section called “AWS Network Load Balancer”。

對於使用 ALB 的傳入，您必須指定以下註釋。如需詳細資訊，請參閱the section called “應用程式負載平衡”。

```
alb.ingress.kubernetes.io/target-type: ip
```

對於使用 NLB 的負載平衡，您必須指定以下註釋。如需詳細資訊，請參閱the section called “網路負載平衡”。

```
service.beta.kubernetes.io/aws-load-balancer-type: "external"
service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: "ip"
```

EKS Pod 身分識別代理程式

> **Note**
>
> 若要在執行 Bottlerocket 的混合節點上成功部署 EKS Pod 身分識別代理程式附加元件，請確保您的 Bottlerocket 版本至少為 v1.39.0。在混合節點環境中，較早版本的 Bottlerocket 不支援 Pod 身分識別代理程式。

原始 Amazon EKS Pod Identity Agent DaemonSet 依賴節點上 EC2 IMDS 的可用性來取得所需的 AWS 登入資料。由於 IMDS 不適用於混合節點，因此從版本 1.3.3-eksbuild.1 開始，Pod 身分識別代理程式附加元件可選擇性地部署會掛載所需憑證的 DaemonSet。執行 Bottlerocket 的混合節點需要不同的方法來掛載憑證，並且版本 1.3.7-eksbuild.2 開始，Pod 身分識別代理程式附加元件可選擇性地部署專門以 Bottlerocket 混合節點為目標的 DaemonSet。下列各節說明啟用選用 DaemonSets 的程序。

Ubuntu/RHEL/AL2023

1. 若要在 Ubuntu/RHEL/Al2023 混合節點上使用 Pod 身分識別代理程式，請在 nodeadm 組態的混合區段中設定 `enableCredentialsFile: true`，如下所示：

   ```
   apiVersion: node.eks.aws/v1alpha1
   kind: NodeConfig
   spec:
       hybrid:
           enableCredentialsFile: true # <-- Add this
   ```

   這將設定 nodeadm 以建立憑證檔案，而該檔案可在 `/eks-hybrid/.aws/credentials` 下的節點上進行設定，且，在其將由 `eks-pod-identity-agent` Pod 使用。此登入資料檔案將包含將定期重新整理的臨時 AWS 登入資料。

2. 更新每個節點上的 nodeadm 組態後，請使用 nodeConfig.yaml 執行下列 `nodeadm init` 命令，以將混合節點加入 Amazon EKS 叢集。如果您的節點先前已加入叢集，仍請再次執行 `nodeadm init` 命令。

   ```
   nodeadm init -c file://nodeConfig.yaml
   ```

3. 使用 CLI 或 ，`eks-pod-identity-agent`在啟用混合節點支援的情況下安裝 AWS AWS 管理主控台。

   a. AWS CLI：從您用來管理叢集的機器中，執行下列命令來安裝 `eks-pod-identity-agent`，並啟用混合節點的支援。使用您叢集的名稱取代 `my-cluster`。

      ```
      aws eks create-addon \
          --cluster-name my-cluster \
          --addon-name eks-pod-identity-agent \
          --configuration-values '{"daemonsets":{"hybrid":{"create": true}}}'
      ```

   b. AWS 管理主控台：如果您透過 AWS 主控台安裝 Pod Identity Agent 附加元件，請將下列項目新增至選用組態，以部署以混合節點為目標的 DaemonSet。

```
{"daemonsets":{"hybrid":{"create": true}}}
```

Bottlerocket

1. 若要在 Bottlerocket 混合節點上使用 Pod 身分識別代理程式，請將 `--enable-credentials-file=true` 旗標新增至用於 Bottlerocket 引導容器使用者資料的命令，如 the section called “使用 Bottlerocket 連接混合節點” 中所述。

   a. 如果您使用的是 SSM 憑證提供者，則您的命令應該如下所示：

   ```
   eks-hybrid-ssm-setup --activation-id=<activation-id> --activation-
   code=<activation-code> --region=<region> --enable-credentials-file=true
   ```

   b. 如果您使用的是 IAM Roles Anywhere 憑證提供者，則您的命令應該如下所示：

   ```
   eks-hybrid-iam-ra-setup --certificate=<certificate> --key=<private-key> --enable-
   credentials-file=true
   ```

   這將設定引導指令碼，以在 `/var/eks-hybrid/.aws/credentials` 下的節點上建立憑證，且其將由 `eks-pod-identity-agent` Pod 使用。此登入資料檔案將包含將定期重新整理的臨時 AWS 登入資料。

2. 使用 CLI 或 ，`eks-pod-identity-agent`在已啟用 Bottlerocket AWS 混合節點的支援下安裝 AWS 管理主控台。

   a. AWS CLI：從您用來管理叢集的機器中，執行下列命令來安裝 `eks-pod-identity-agent` ，並啟用 Bottlerocket 混合節點的支援。使用您叢集的名稱取代 `my-cluster`。

   ```
   aws eks create-addon \
       --cluster-name my-cluster \
       --addon-name eks-pod-identity-agent \
       --configuration-values '{"daemonsets":{"hybrid-bottlerocket":{"create":
    true}}}'
   ```

   b. AWS 管理主控台：如果您透過 AWS 主控台安裝 Pod Identity Agent 附加元件，請將下列項目新增至選用組態，以部署以 Bottlerocket 混合節點為目標的 DaemonSet。

   ```
   {"daemonsets":{"hybrid-bottlerocket":{"create": true}}}
   ```

CSI 快照控制器

從版本 開始`v8.1.0-eksbuild.2`，CSI 快照控制器附加元件會套用混合節點的軟性反親和性規則，偏好控制器在與 Amazon EKS 控制平面相同的 AWS 區域中於 EC2 `deployment`上執行。在`deployment`與 Amazon EKS 控制平面相同的 AWS 區域中聯合放置 可改善延遲。

社群附加元件

以下各節說明在混合節點上執行相容的社群附加元件與其他 Amazon EKS 運算類型之間的差異。

Kubernetes 指標伺服器

控制平面需要連接指標伺服器的 Pod IP (如果啟用 hostNetwork，則為節點 IP)。因此，除非您在 hostNetwork 模式下執行指標伺服器，否則您必須在建立 Amazon EKS 叢集時設定遠端 Pod 網路，而且必須使 Pod IP 位址可路由。使用 CNI 實作邊界閘道協定 (BGP) 是讓您的 Pod IP 位址可路由的常見方式之一。

cert-manager

`cert-manager` 會使用 Webhook。如果您在混合節點上執行 `cert-manager`，您的內部部署 Pod CIDR 必須在內部部署網路上可路由，而且您必須使用遠端 Pod 網路設定 EKS 叢集。如需詳細資訊，請參閱設定混合節點的 Webhook。

設定混合節點的 Webhook

此頁面詳細說明使用混合節點執行 Webhook 的考量事項。Webhook 用於 Kubernetes 應用程式和開放原始碼專案，例如 AWS Load Balancer控制器和 CloudWatch 可觀測性代理程式，以在執行時間執行變動和驗證功能。

可路由的 Pod 網路

如果您能夠在內部部署網路上讓內部部署 Pod CIDR 可路由，則可在混合節點上執行 Webhook。您可以使用多種技術，讓您的內部部署 Pod CIDR 在內部部署網路上可路由，包括邊界閘道協定 (BGP)、靜態路由或其他自訂路由解決方案。BGP 是建議的解決方案，因為它比需要自訂或手動路由組態的替代解決方案更具可擴展性且更容易管理。 AWS 支援 Cilium 和 Calico 的 BGP 功能來公告 Pod CIDRs，如需詳細資訊，請參閱 the section called “可路由的遠端 Pod CIDR” the section called “設定 CNI”和 。

不可路由的 Pod 網路

如果您無法在使內部部署 Pod CIDR 在內部部署網路上可路由並且需要執行 Webhook，則建議您在與混合節點相同的 EKS 叢集中的雲端節點上，執行所有 Webhook。

混合模式叢集的考量事項

混合模式叢集定義為 EKS 叢集，其具有在 AWS Cloud 中執行的混合節點和節點。執行混合模式叢集時，請考量下列建議：

- 在 AWS 雲端的節點上執行 VPC CNI，在混合節點上執行 Cilium 或 Calico。在 AWS 雲端節點上執行 AWS 時，不支援 Cilium 和 Calico。
- 設定 Webhook 在 AWS 雲端的節點上執行。如需如何設定 AWS 和 社群附加元件的 Webhook，the section called “為附加元件設定 Webhook”請參閱 。
- 如果您的應用程式需要在 AWS Cloud 節點上執行的 Pod 直接與在混合節點上執行的 Pod 通訊 (「東西通訊」)，而且您在 AWS Cloud 節點上使用 VPC CNI，在混合節點上使用 Cilium 或 Calico，則您的內部部署 Pod CIDR 必須可在內部部署網路上路由。
- 在 AWS 雲端的節點上執行至少一個 CoreDNS 複本，並在混合節點上執行至少一個 CoreDNS 複本。
- 設定服務流量分佈，以將服務流量保持在其來源區域的本機。如需有關服務流量分佈的詳細資訊，請參閱 the section called “設定服務流量分佈”。
- 如果您將 AWS Application Load Balancer (ALB) 或 Network Load Balancer (NLB) 用於在混合節點上執行的工作負載流量，則與 ALB 或 NLB 搭配使用的 IP 目標必須是可路由的 AWS。
- 指標伺服器附加元件需要從 EKS 控制平面連線至指標伺服器 Pod IP 位址。如果您在混合節點上執行指標伺服器附加元件，則您的內部部署 Pod CIDR 必須在內部部署網路上可路由。
- 若要使用 Amazon Managed Service for Prometheus (AMP) 受管收集器收集混合節點的指標，您的內部部署 Pod CIDR 必須在內部部署網路上可路由。或者，您可以將 AMP 受管收集器用於在 AWS 雲端中執行的 EKS 控制平面指標和資源，以及 AWS Distro for OpenTelemetry (ADOT) 附加元件來收集混合節點的指標。

設定混合模式叢集

若要檢視叢集上執行的變更和驗證 Webhook，您可以在叢集的 EKS 主控台的資源面板中檢視延伸模組模源類型，或者也可以使用下列命令。EKS 還會在叢集可觀測性儀表板中報告 Webhook 指標，如需詳細資訊，請參閱 the section called “可觀測性儀表板”。

```
kubectl get mutatingwebhookconfigurations
```

```
kubectl get validatingwebhookconfigurations
```

設定服務流量分佈

執行混合模式叢集時，建議您使用服務流量分佈，以將服務流量保持在其來源區域的本機。服務流量分佈 (適用於 EKS 中的 Kubernetes 版本 1.31 及更新版本) 是拓撲感知路由的建議解決方案，因為其可預測性更高。透過服務流量分佈，區域中運作狀態良好的端點將會接收該區域的所有流量。利用拓撲感知路由，每個服務都必須符合該區域中的數個條件，以套用自訂路由，否則其會將流量平均路由至所有端點。

如果您使用 Cilium 作為 CNI，則必須執行 CNI，並將 `enable-service-topology` 設定為 `true` 以啟用服務流量分佈。您可以使用 Helm 安裝旗標 `--set loadBalancer.serviceTopology=true` 傳遞此組態，或者，您也可以使用 Cilium CLI 命令 `cilium config set enable-service-topology true` 更新現有的安裝。更新現有安裝的組態後，必須重新啟動在每個節點上執行的 Cilium 代理程式。

下節會顯示如何為 CoreDNS 服務設定服務流量分佈的範例，並且建議您為叢集中的所有服務啟用相同的服務流量分佈，以避免意外的跨環境流量。

設定 CoreDNS 複本

如果您在 AWS 雲端中同時執行混合模式叢集與混合節點和節點，建議您在混合節點上至少有一個 CoreDNS 複本，並在 AWS 雲端中的節點上至少有一個 CoreDNS 複本。為了防止混合模式叢集設定中的延遲和網路問題，您可以將 CoreDNS 服務設定為偏好使用具有服務流量分佈的最接近的 CoreDNS 複本。

服務流量分佈 (適用於 EKS 中的 Kubernetes 版本 1.31 及更新版本) 是拓撲感知路由的建議解決方案，因為其可預測性更高。在服務流量分佈中，區域中運作狀態良好的端點將會接收該區域的所有流量。在拓撲感知路由中，每個服務都必須符合該區域中的數個條件，以套用自訂路由，否則其會將流量平均路由至所有端點。下列步驟可設定服務流量分佈。

如果您使用 Cilium 作為 CNI，則必須執行 CNI，並將 `enable-service-topology` 設定為 `true` 以啟用服務流量分佈。您可以使用 Helm 安裝旗標 `--set loadBalancer.serviceTopology=true` 傳遞此組態，或者，您也可以使用 Cilium CLI 命令 `cilium config set enable-service-topology true` 更新現有的安裝。更新現有安裝的組態後，必須重新啟動在每個節點上執行的 Cilium 代理程式。

1. 為每個混合節點新增一個拓撲區域標籤，例如 `topology.kubernetes.io/zone: onprem`。或者，您可以在 `nodeadm` 組態中指定標籤，進而在 `nodeadm init` 階段設定標籤，請參閱 the section called “用於自訂 kubelet 的節點組態 (選用)”。請注意，在 AWS Cloud 中執行的節點會自動取得套用至它們的拓撲區域標籤，其對應至節點的可用區域 (AZ)。

```
kubectl label node hybrid-node-name topology.kubernetes.io/zone=zone
```

2. 使用拓撲區域金鑰將 podAntiAffinity 新增至 CoreDNS 部署。或者，您可以在安裝過程中使用 EKS 附加元件設定 CoreDNS 部署。

```
kubectl edit deployment coredns -n kube-system
```

```
spec:
  template:
    spec:
      affinity:
       ...
        podAntiAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
          - podAffinityTerm:
              labelSelector:
                matchExpressions:
                - key: k8s-app
                  operator: In
                  values:
                  - kube-dns
              topologyKey: kubernetes.io/hostname
            weight: 100
          - podAffinityTerm:
              labelSelector:
                matchExpressions:
                - key: k8s-app
                  operator: In
                  values:
                  - kube-dns
              topologyKey: topology.kubernetes.io/zone
            weight: 50
      ...
```

3. 將設定 trafficDistribution: PreferClose 新增至 kube-dns 服務組態，以啟用服務流量分佈。

```
kubectl patch svc kube-dns -n kube-system --type=merge -p '{
  "spec": {
    "trafficDistribution": "PreferClose"
```

```
  }
}'
```

4. 您可以檢視 kube-dns 服務的端點配量，進而確認已啟用服務流量分佈。您的端點配量必須顯示拓撲區域標籤的 hints，進而確認已啟用服務流量分佈。如果您沒有看到每個端點地址的 hints，則不會啟用服務流量分佈。

```
kubectl get endpointslice -A | grep "kube-dns"
```

```
kubectl get endpointslice kube-dns-<id>  -n kube-system -o yaml
```

```
addressType: IPv4
apiVersion: discovery.k8s.io/v1
endpoints:
- addresses:
  - <your-hybrid-node-pod-ip>
  hints:
    forZones:
    - name: onprem
  nodeName: <your-hybrid-node-name>
  zone: onprem
- addresses:
  - <your-cloud-node-pod-ip>
  hints:
    forZones:
    - name: us-west-2a
  nodeName: <your-cloud-node-name>
  zone: us-west-2a
```

為附加元件設定 Webhook

下列附加元件使用 Webhook 並支援與混合節點搭配使用。

- AWS Load Balancer控制器
- CloudWatch 可觀測性代理程式
- AWS Distro for OpenTelemetry (ADOT)
- cert-manager

請參閱下列各節，以設定這些附加元件所使用的 Webhook 在 AWS 雲端節點上執行。

AWS Load Balancer控制器

若要在混合模式叢集設定中使用 AWS Load Balancer控制器，您必須在 AWS 雲端的節點上執行控制器。為此，請將下列內容新增至 Helm 值組態，或使用 EKS 附加元件組態指定值。

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
        - key: eks.amazonaws.com/compute-type
          operator: NotIn
          values:
          - hybrid
```

CloudWatch 可觀測性代理程式

CloudWatch 可觀測性代理程式附加元件具有使用 Webhook 的 Kubernetes Operator。若要在混合模式叢集設定中對 AWS Cloud 中的節點執行運算子，請編輯 CloudWatch Observability Agent Operator 組態。您無法在安裝過程中使用 Helm 和 EKS 附加元件設定運算子親和性 (請參閱 Container-roadmap 問題 #2431)。

```
kubectl edit -n amazon-cloudwatch deployment amazon-cloudwatch-observability-
controller-manager
```

```
spec:
  ...
  template:
    ...
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: eks.amazonaws.com/compute-type
                operator: NotIn
                values:
                - hybrid
```

AWS Distro for OpenTelemetry (ADOT)

AWS Distro for OpenTelemetry (ADOT) 附加元件具有使用 Webhook 的 Kubernetes Operator。若要在混合模式叢集設定中於 AWS 雲端節點上執行運算子，請將下列項目新增至 Helm 值組態，或使用 EKS 附加元件組態指定值。

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
        - key: eks.amazonaws.com/compute-type
          operator: NotIn
          values:
          - hybrid
```

如果您的 Pod CIDR 在內部部署網路上不可路由，則 ADOT 收集器必須在混合節點上執行，以從混合節點和在其上執行的工作負載中抓取指標。為此，請編輯自訂資源定義 (CRD)。

```
kubectl -n opentelemetry-operator-system edit opentelemetrycollectors.opentelemetry.io
 adot-col-prom-metrics
```

```
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: eks.amazonaws.com/compute-type
            operator: In
            values:
            - hybrid
```

您可以將下列 `relabel_configs` 新增至 ADOT 收集器 CRD 組態中的每個 `scrape_configs`，從而將 ADOT 收集器設定為僅從混合節點和混合節點上執行的資源抓取指標。

```
relabel_configs:
  - action: keep
    regex: hybrid
    source_labels:
```

```
    - __meta_kubernetes_node_label_eks_amazonaws_com_compute_type
```

ADOT 附加元件需要cert-manager為 ADOT 運算子 Webhook 所使用的 TLS 憑證安裝 。 cert-manager也會執行 Webhook，而且您可以使用下列 Helm 值組態，將其設定為在 AWS Cloud 中的節點上執行。

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
        - key: eks.amazonaws.com/compute-type
          operator: NotIn
          values:
          - hybrid
webhook:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: eks.amazonaws.com/compute-type
            operator: NotIn
            values:
            - hybrid
cainjector:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: eks.amazonaws.com/compute-type
            operator: NotIn
            values:
            - hybrid
startupapicheck:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: eks.amazonaws.com/compute-type
            operator: NotIn
```

```
            values:
            - hybrid
```

cert-manager

附加cert-manager元件會執行 Webhook，您可以使用下列 Helm 值組態，將其設定為在 AWS Cloud 中的節點上執行。

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
        - key: eks.amazonaws.com/compute-type
          operator: NotIn
          values:
          - hybrid
webhook:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: eks.amazonaws.com/compute-type
            operator: NotIn
            values:
            - hybrid
cainjector:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: eks.amazonaws.com/compute-type
            operator: NotIn
            values:
            - hybrid
startupapicheck:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
```

```
          - key: eks.amazonaws.com/compute-type
            operator: NotIn
            values:
            - hybrid
```

設定混合節點的代理

如果您在內部部署環境中為離開資料中心或邊緣環境的流量使用代理伺服器，則您需要將節點和叢集分別設定為使用代理伺服器。

叢集

在叢集上，您需要設定 kube-proxy，以使用您的代理伺服器。在建立 Amazon EKS 叢集後，您必須設定 kube-proxy。

節點

在節點上，您必須設定作業系統、containerd、kubelet 和 Amazon SSM 代理程式，以使用您的代理伺服器。您可以在作業系統映像的建置程序期間或在每個混合節點上執行 nodeadm init 之前，進行這些變更。

節點層級組態

您必須在作業系統映像中或在每個混合節點上執行 nodeadm init 之前，套用下列組態。

containerd 代理組態

containerd 是 Kubernetes 的預設容器管理執行時期。如果您使用代理進行網際網路存取，則必須設定 containerd，如此才能提取 Kubernetes 和 Amazon EKS 所需的容器映像。

使用下列內容，在 /etc/systemd/system/containerd.service.d 目錄中稱為 http-proxy.conf 的每個混合節點上，建立檔案。將 proxy-domain 和 port 取代為您環境的值。

```
[Service]
Environment="HTTP_PROXY=http://proxy-domain:port"
Environment="HTTPS_PROXY=http://proxy-domain:port"
Environment="NO_PROXY=localhost"
```

來自使用者資料的 **containerd** 組態

需要為此檔案建立 containerd.service.d 目錄。您需要重新載入 systemd，才能在不重新啟動的情況下取得組態檔案。在 AL2023 中，當您的指令碼執行時，服務可能已在執行中，因此您還需要將其重新啟動。

```
mkdir -p /etc/systemd/system/containerd.service.d
echo '[Service]' > /etc/systemd/system/containerd.service.d/http-proxy.conf
echo 'Environment="HTTP_PROXY=http://proxy-domain:port"' >> /etc/systemd/system/
containerd.service.d/http-proxy.conf
echo 'Environment="HTTPS_PROXY=http://proxy-domain:port"' >> /etc/systemd/system/
containerd.service.d/http-proxy.conf
echo 'Environment="NO_PROXY=localhost"' >> /etc/systemd/system/containerd.service.d/
http-proxy.conf
systemctl daemon-reload
systemctl restart containerd
```

kubelet 代理組態

kubelet 是在每個 Kubernetes 節點上執行的 Kubernetes 節點代理程式，且需負責管理其上執行的節點和 Pod。如果您在內部部署環境中使用代理，則必須設定 kubelet，如此才能與您 Amazon EKS 叢集的公有或私有端點通訊。

使用下列內容，在 /etc/systemd/system/kubelet.service.d/ 目錄中稱為 http-proxy.conf 的每個混合節點上，建立檔案。將 proxy-domain 和 port 取代為您環境的值。

```
[Service]
Environment="HTTP_PROXY=http://proxy-domain:port"
Environment="HTTPS_PROXY=http://proxy-domain:port"
Environment="NO_PROXY=localhost"
```

來自使用者資料的 **kubelet** 組態

必須為此檔案建立 kubelet.service.d 目錄。您需要重新載入 systemd，才能在不重新啟動的情況下取得組態檔案。在 AL2023 中，當您的指令碼執行時，服務可能已在執行中，因此您還需要將其重新啟動。

```
mkdir -p /etc/systemd/system/kubelet.service.d
echo '[Service]' > /etc/systemd/system/kubelet.service.d/http-proxy.conf
echo 'Environment="HTTP_PROXY=http://proxy-domain:port"' >> /etc/systemd/system/
kubelet.service.d/http-proxy.conf
```

```
echo 'Environment="HTTPS_PROXY=http://proxy-domain:port"' >> /etc/systemd/system/
kubelet.service.d/http-proxy.conf
echo 'Environment="NO_PROXY=localhost"' >> /etc/systemd/system/kubelet.service.d/http-
proxy.conf
systemctl daemon-reload
systemctl restart kubelet
```

ssm 代理組態

ssm 是可用於初始化混合節點的憑證提供者之一。ssm 負責使用 AWS 來進行驗證以及產生 kubelet 使用的臨時憑證。如果您在內部部署環境中使用代理，並使用 ssm 作為節點上的憑證提供者，則必須設定 ssm，以便其可以與 Amazon SSM 服務端點通訊。

根據作業系統，在下列路徑中稱為 http-proxy.conf 的每個混合節點上，建立檔案

- Ubuntu - /etc/systemd/system/snap.amazon-ssm-agent.amazon-ssm-agent.service.d/http-proxy.conf
- Amazon Linux 2023 和 Red Hat Enterprise Linux - /etc/systemd/system/amazon-ssm-agent.service.d/http-proxy.conf

使用下列內容填入檔案。將 proxy-domain 和 port 取代為您環境的值。

```
[Service]
Environment="HTTP_PROXY=http://proxy-domain:port"
Environment="HTTPS_PROXY=http://proxy-domain:port"
Environment="NO_PROXY=localhost"
```

來自使用者資料的 **ssm** 組態

必須為此檔案建立 ssm systemd 服務檔案目錄。目錄路徑取決於節點上使用的作業系統。

- Ubuntu - /etc/systemd/system/snap.amazon-ssm-agent.amazon-ssm-agent.service.d
- Amazon Linux 2023 和 Red Hat Enterprise Linux - /etc/systemd/system/amazon-ssm-agent.service.d

根據節點上使用的作業系統，取代以下重新啟動命令中的 systemd 服務名稱

- Ubuntu - snap.amazon-ssm-agent.amazon-ssm-agent

- Amazon Linux 2023 和 Red Hat Enterprise Linux - `amazon-ssm-agent`

```
mkdir -p systemd-service-file-directory
echo '[Service]' > [.replaceable]#systemd-service-file-directory/http-proxy.conf
echo 'Environment="HTTP_PROXY=http://[.replaceable]#proxy-domain:port"' >> systemd-
service-file-directory/http-proxy.conf
echo 'Environment="HTTPS_PROXY=http://[.replaceable]#proxy-domain:port"' >>
 [.replaceable]#systemd-service-file-directory/http-proxy.conf
echo 'Environment="NO_PROXY=localhost"' >> [.replaceable]#systemd-service-file-
directory/http-proxy.conf
systemctl daemon-reload
systemctl restart [.replaceable]#systemd-service-name
```

作業系統代理組態

如果您使用代理進行網際網路存取，則必須將作業系統設定為能夠從作業系統的套件管理工具提取混合節點相依性。

Ubuntu

1. 透過以下命令，設定 snap 以使用您的代理：

   ```
   sudo snap set system proxy.https=http://proxy-domain:port
   sudo snap set system proxy.http=http://proxy-domain:port
   ```

2. 若要啟用 apt 的代理，請在 /etc/apt/ 目錄中建立稱為 apt.conf 的檔案。將代理網域和連接埠取代為您環境的值。

   ```
   Acquire::http::Proxy "http://proxy-domain:port";
   Acquire::https::Proxy "http://proxy-domain:port";
   ```

Amazon Linux 2023

1. 設定 dnf 以使用您的代理。為您的環境建立具有代理網域和連接埠值的檔案 /etc/dnf/dnf.conf。

   ```
   proxy=http://proxy-domain:port
   ```

Red Hat Enterprise Linux

1. 設定 yum 以使用您的代理。為您的環境建立具有代理網域和連接埠值的檔案 /etc/yum.conf。

```
proxy=http://proxy-domain:port
```

IAM Roles Anywhere 代理組態

IAM Roles Anywhere 憑證提供者服務負責在搭配 enableCredentialsFile 旗標使用 IAM Roles Anywhere 時重新整理憑證 (請參閱 the section called “EKS Pod 身分識別代理程式”)。如果您在內部部署環境中使用代理，則必須設定服務，以便其可以與 IAM Roles Anywhere 端點通訊。

使用下列內容，在 /etc/systemd/system/aws_signing_helper_update.service.d/ 目錄中，建立稱為 http-proxy.conf 的檔案。將 proxy-domain 和 port 取代為您環境的值。

```
[Service]
Environment="HTTP_PROXY=http://proxy-domain:port"
Environment="HTTPS_PROXY=http://proxy-domain:port"
Environment="NO_PROXY=localhost"
```

整個叢集的組態

建立 Amazon EKS 叢集之後以及在每個混合節點上執行 nodeadm init 之前，必須套用本節中的組態。

kube-proxy 代理組態

當您的混合節點加入叢集時，Amazon EKS 會自動在每個混合節點上安裝 kube-proxy 作為 DaemonSet。kube-proxy 會在由 Amazon EKS 叢集上的 Pod 支援的 服務之間啟用路由。若要設定每個主機，kube-proxy 需要對您的 Amazon EKS 叢集端點進行 DNS 解析。

1. 使用下列命令編輯 kube-proxy DaemonSet

```
kubectl -n kube-system edit ds kube-proxy
```

   這將會在設定的編輯器上開啟 kube-proxy DaemonSet 定義。
2. 新增 HTTP_PROXY 和 HTTPS_PROXY 的環境變數。請注意，NODE_NAME 環境變數應該已存在於您的組態中。將 proxy-domain 和 port 取代為您環境的值。

```
containers:
  - command:
    - kube-proxy
    - --v=2
    - --config=/var/lib/kube-proxy-config/config - --hostname-override=$(NODE_NAME)
    env:
    - name: HTTP_PROXY
      value: http://proxy-domain:port
    - name: HTTPS_PROXY
      value: http://proxy-domain:port
    - name: NODE_NAME
      valueFrom:
        fieldRef:
          apiVersion: v1
          fieldPath: spec.nodeName
```

設定混合節點的 Cilium BGP

本主題會說明如何設定 Amazon EKS 混合節點的 Cilium 邊界閘道協定 (BGP)。Cilium 的 BGP 功能稱為 Cilium BGP 控制平面，且可用於向內部部署網路公告 Pod 和服務地址。如需讓 Pod CIDR 在內部部署網路上可路由的替代方法，請參閱 the section called “可路由的遠端 Pod CIDR”。

設定 Cilium BGP

先決條件

- 遵循 the section called “設定 CNI” 中的指示安裝 Cilium。

程序

1. 若要搭配 Cilium 使用 BGP 向內部部署網路公告 Pod 或服務地址，Cilium 必須與 `bgpControlPlane.enabled: true` 一起安裝。如果您要為現有的 Cilium 部署啟用 BGP，則必須重新啟動 Cilium 運算子才能套用 BGP 組態 (如果先前尚未啟用 BGP)。您可以在 Helm 值中將 `operator.rollOutPods` 設定為 `true`，以在 Helm 安裝/升級程序中重新啟動 Cilium 運算子。

```
helm upgrade cilium oci://public.ecr.aws/eks/cilium/cilium \
  --namespace kube-system \
  --reuse-values \
  --set operator.rollOutPods=true \
```

```
  --set bgpControlPlane.enabled=true
```

2. 確認 Cilium 運算子和代理程式已重新啟動並正在執行。

```
kubectl -n kube-system get pods --selector=app.kubernetes.io/part-of=cilium
```

```
NAME                               READY   STATUS    RESTARTS   AGE
cilium-grwlc                       1/1     Running   0          4m12s
cilium-operator-68f7766967-5nnbl   1/1     Running   0          4m20s
cilium-operator-68f7766967-7spfz   1/1     Running   0          4m20s
cilium-pnxcv                       1/1     Running   0          6m29s
cilium-r7qkj                       1/1     Running   0          4m12s
cilium-wxhfn                       1/1     Running   0          4m1s
cilium-z7hlb                       1/1     Running   0          6m30s
```

3. 建立稱為 `cilium-bgp-cluster.yaml` 的檔案，其中具有 `CiliumBGPClusterConfig` 定義。您可能需要向您的網路管理員取得下列資訊。
   - 針對執行 Cilium 的節點，使用 ASN 設定 `localASN`。
   - 針對內部部署路由器，使用 ASN 設定 `peerASN`。
   - 使用執行 Cilium 的每個節點將與之對等的內部部署路由器 IP，設定 `peerAddress`。

```
apiVersion: cilium.io/v2alpha1
kind: CiliumBGPClusterConfig
metadata:
  name: cilium-bgp
spec:
  nodeSelector:
    matchExpressions:
    - key: eks.amazonaws.com/compute-type
      operator: In
      values:
      - hybrid
  bgpInstances:
  - name: "rack0"
    localASN: NODES_ASN
    peers:
    - name: "onprem-router"
      peerASN: ONPREM_ROUTER_ASN
      peerAddress: ONPREM_ROUTER_IP
      peerConfigRef:
        name: "cilium-peer"
```

4. 將 Cilium BGP 叢集組態套用至您的叢集。

```
kubectl apply -f cilium-bgp-cluster.yaml
```

5. 使用定義 BGP 對等組態的 CiliumBGPPeerConfig 資源建立名為 cilium-bgp-peer.yaml 的檔案。多個對等可以共用相同的組態，並提供常見 CiliumBGPPeerConfig 資源的參考。如需組態選項的完整清單，請參閱 Cilium 文件中的 BGP 對等組態。

   下列 Cilium 對等設定的值必須符合您要對等互連的內部部署路由器的值。

   - 設定 holdTimeSeconds，其可決定在宣告工作階段關閉之前 BGP 對等等待保持連線或更新訊息的時長。預設為 90 秒。
   - 設定 keepAliveTimeSeconds，其可決定 BGP 對等是否仍可連線，以及 BGP 工作階段是否在作用中。預設為 30 秒。
   - 設定 restartTimeSeconds，其可決定 Cilium 的 BGP 控制平面在重新啟動後預期重新建立 BGP 工作階段的時間。預設值為 120 秒。

```
apiVersion: cilium.io/v2alpha1
kind: CiliumBGPPeerConfig
metadata:
  name: cilium-peer
spec:
  timers:
    holdTimeSeconds: 90
    keepAliveTimeSeconds: 30
  gracefulRestart:
    enabled: true
    restartTimeSeconds: 120
  families:
    - afi: ipv4
      safi: unicast
      advertisements:
        matchLabels:
          advertise: "bgp"
```

6. 將 Cilium BGP 對等組態套用至您的叢集。

```
kubectl apply -f cilium-bgp-peer.yaml
```

7. 使用 CiliumBGPAdvertisement 資源建立名為 cilium-bgp-advertisement-pods.yaml 的檔案，以向內部部署網路公告 Pod CIDR。

- `CiliumBGPAdvertisement` 資源可用於定義與其相關聯的廣告類型和屬性。以下範例會將 Cilium 設定為僅公告 Pod CIDR。如需設定 Cilium 以公告服務地址的詳細資訊，請參閱 the section called “服務類型 LoadBalancer” 和 the section called “Cilium 叢集內負載平衡” 中的範例。
- 執行 Cilium 代理程式的每個混合節點皆與上游已啟用 BGP 的路由器對等。當 Cilium 的 `advertisementType` 設定為 `PodCIDR` (如以下範例所示) 時，每個節點都會公告其擁有的 Pod CIDR 範圍。如需詳細資訊，請參閱 Cilium 文件中的 BGP 公告組態。

```
apiVersion: cilium.io/v2alpha1
kind: CiliumBGPAdvertisement
metadata:
  name: bgp-advertisement-pods
  labels:
    advertise: bgp
spec:
  advertisements:
    - advertisementType: "PodCIDR"
```

8. 將 Cilium BGP 公告組態套用至您的叢集。

```
kubectl apply -f cilium-bgp-advertisement-pods.yaml
```

9. 您可以使用 `cilium bgp peers` 命令，確認 BGP 對等互連與 Cilium CLI 搭配運作。您應該會在環境的輸出中看到正確的值，且工作階段狀態為 `established`。如需有關故障診斷的詳細資訊，請參閱 Cilium 文件中的故障診斷和操作指南。

   在以下範例中，有五個執行 Cilium 代理程式的混合節點，且每個節點將會公告其擁有的 Pod CIDR 範圍。

```
cilium bgp peers
```

```
Node                   Local AS    Peer AS              Peer Address        Session
 State   Uptime     Family         Received   Advertised
mi-026d6a261e355fba7   NODES_ASN
                   ONPREM_ROUTER_ASN
                   ONPREM_ROUTER_IP    established     1h18m58s   ipv4/unicast   1
      2
mi-082f73826a163626e   NODES_ASN
                   ONPREM_ROUTER_ASN
```

```
                  ONPREM_ROUTER_IP    established     1h19m12s   ipv4/unicast   1
      2
mi-09183e8a3d755abf6   NODES_ASN
                  ONPREM_ROUTER_ASN
                  ONPREM_ROUTER_IP    established     1h18m47s   ipv4/unicast   1
      2
mi-0d78d815980ed202d   NODES_ASN
                  ONPREM_ROUTER_ASN
                  ONPREM_ROUTER_IP    established     1h19m12s   ipv4/unicast   1
      2
mi-0daa253999fe92daa   NODES_ASN
                  ONPREM_ROUTER_ASN
                  ONPREM_ROUTER_IP    established     1h18m58s   ipv4/unicast   1
      2
```

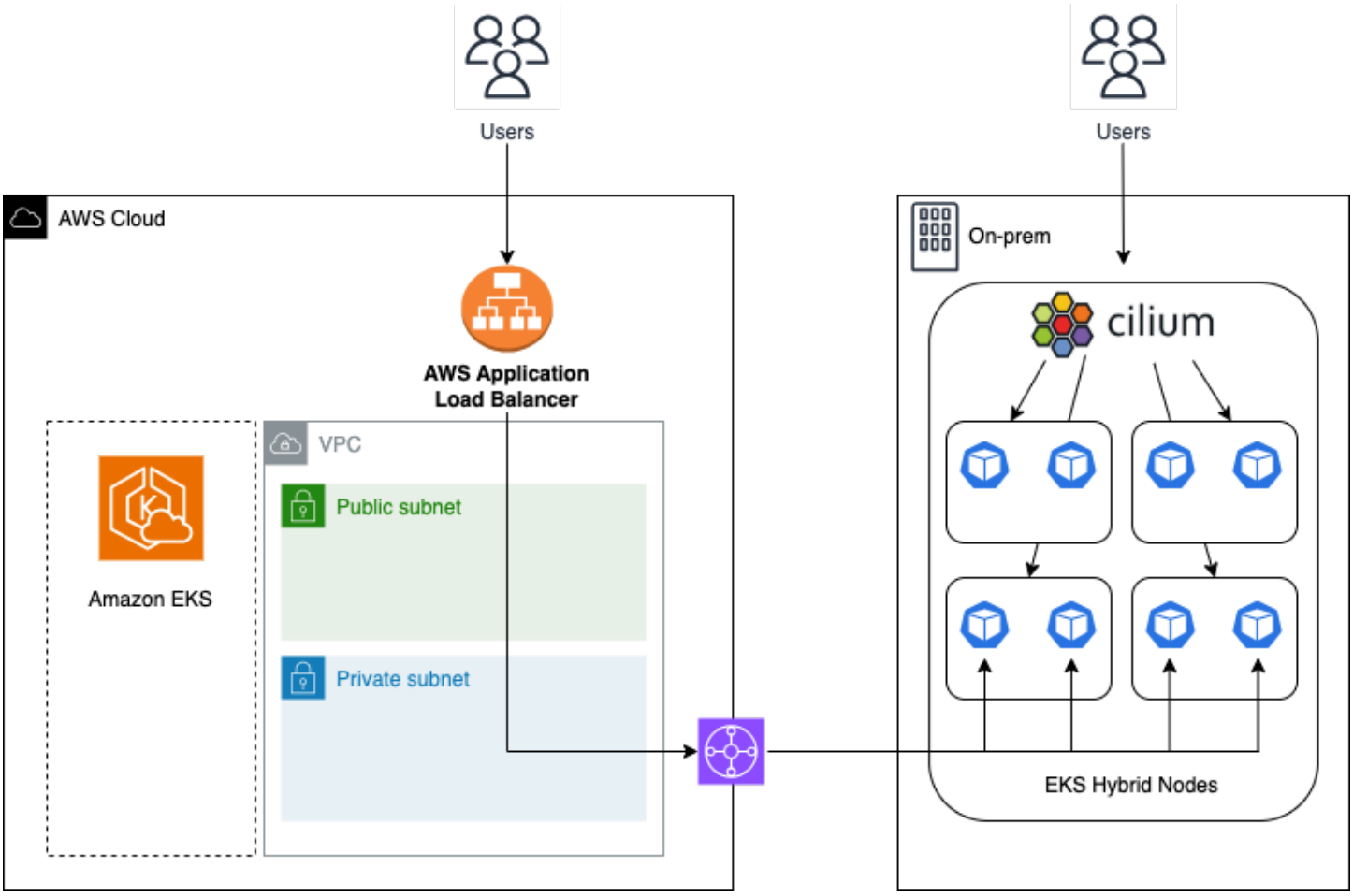
```
cilium bgp routes
```

```
Node                   VRouter       Prefix           NextHop   Age         Attrs
mi-026d6a261e355fba7   NODES_ASN     10.86.2.0/26     0.0.0.0   1h16m46s   [{Origin:
 i} {Nexthop: 0.0.0.0}]
mi-082f73826a163626e   NODES_ASN     10.86.2.192/26   0.0.0.0   1h16m46s   [{Origin:
 i} {Nexthop: 0.0.0.0}]
mi-09183e8a3d755abf6   NODES_ASN     10.86.2.64/26    0.0.0.0   1h16m46s   [{Origin:
 i} {Nexthop: 0.0.0.0}]
mi-0d78d815980ed202d   NODES_ASN     10.86.2.128/26   0.0.0.0   1h16m46s   [{Origin:
 i} {Nexthop: 0.0.0.0}]
mi-0daa253999fe92daa   NODES_ASN     10.86.3.0/26     0.0.0.0   1h16m46s   [{Origin:
 i} {Nexthop: 0.0.0.0}]
```

設定混合節點的 Kubernetes Ingress

本主題會說明如何為在 Amazon EKS 混合節點上執行的工作負載設定 Kubernetes Ingress。Kubernetes Ingress 會將叢集外部的 HTTP 和 HTTPS 路由公開至叢集內的服務。若要利用傳入資源，需要 Kubernetes Ingress 控制器來設定可提供網路流量的聯網基礎結構和元件。

AWS 對於在 EKS 混合節點上執行的工作負載，support AWS Application Load Balancer (ALB) 和適用於 Kubernetes Ingress 的 Cilium。使用 ALB 還是 Cilium for Ingress 的決定取決於應用程式流量的來源。如果應用程式流量來自 AWS 區域， AWS 建議使用 AWS ALB 和 AWS Load Balancer控制器。如果應用程式流量來自本機內部部署或邊緣環境， AWS 建議使用 Cilium 的內建輸入功能，可在您的環境中搭配或不搭配負載平衡器基礎設施使用。



AWS Application Load Balancer

對於在混合節點上執行的工作負載，您可以搭配目標類型 `ip` 使用 AWS Load Balancer 控制器和 Application Load Balancer (ALB)。使用目標類型 `ip` 時，ALB 可繞過服務層網路路徑，將流量直接轉送至 Pod。若要讓 ALB 達到混合節點上的 Pod IP 目標，您的內部部署 Pod CIDR 必須在內部部署網路上可路由。此外， AWS Load Balancer控制器使用 Webhook，並且需要從 EKS 控制平面直接通訊。如需詳細資訊，請參閱the section called “設定 Webhook”。

考量事項

- 如需 AWS Application Load Balancer和 AWS Load Balancer控制器的詳細資訊the section called “使用 Helm 安裝”，請參閱the section called “應用程式負載平衡”和。
- 如需如何在 AWS Application Load Balancer負載平衡器和網路負載平衡器之間選擇的相關資訊，請參閱負載平衡器的最佳實務。 AWS Load Balancer
- 如需可以使用應用程式AWS Load Balancer為輸入資源設定的註釋清單，請參閱負載平衡器控制器輸入註釋。 AWS Application Load Balancer

先決條件

- 遵循 the section called “設定 CNI” 中的指示安裝 Cilium。
- 遵循 the section called “設定 BPA” 中的指示啟用 Cilium BGP 控制平面。如果您不想使用 BGP，則必須使用替代方法，以讓內部部署 Pod CIDR 在內部部署網路上可路由。如果您未讓內部部署 Pod CIDR 可路由，則 ALB 將無法註冊或聯絡您的 Pod IP 目標。
- 已在命令列環境中安裝 Helm，如需詳細資訊，請參閱安裝 Helm 說明。
- 已在命令列環境中安裝 eksctl，如需詳細資訊，請參閱安裝 eksctl 說明。

程序

1. 下載 AWS Load Balancer控制器的 IAM 政策，允許其代表您呼叫 AWS APIs。

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-
controller/refs/heads/main/docs/install/iam_policy.json
```

2. 使用上一個步驟中下載的政策，建立 IAM 政策。

```
aws iam create-policy \
    --policy-name AWSLoadBalancerControllerIAMPolicy \
    --policy-document file://iam_policy.json
```

3. 將叢集名稱 (CLUSTER_NAME)、 AWS 區域 (AWS_REGION) 和 AWS 帳戶 ID (AWS_ACCOUNT_ID) 的值取代為您的設定，然後執行下列命令。

```
eksctl create iamserviceaccount \
    --cluster=CLUSTER_NAME \
    --namespace=kube-system \
    --name=aws-load-balancer-controller \
    --attach-policy-arn=arn:aws:iam::AWS_ACCOUNT_ID:policy/
AWSLoadBalancerControllerIAMPolicy \
    --override-existing-serviceaccounts \
    --region AWS_REGION \
    --approve
```

4. 新增 eks-charts Helm Chart 儲存庫並更新您的本機 Helm 儲存庫，以確保您擁有最近的圖表。

```
helm repo add eks https://aws.github.io/eks-charts
```

```
helm repo update eks
```

5. 安裝 AWS Load Balancer控制器。將叢集名稱 (CLUSTER_NAME)、 AWS 區域 (AWS_REGION)、VPC ID (VPC_ID) 和 AWS Load Balancer控制器 Helm Chart 版本 (AWS_LBC_HELM_VERSION) 的值取代為您的設定，並執行下列命令。如果您在 AWS 雲端中同時執行混合模式叢集與混合節點和節點，則可以遵循 中的指示，在雲端節點上執行 AWS Load Balancer 控制器the section called “AWS Load Balancer控制器”。
   - 您可以透過執行 `helm search repo eks/aws-load-balancer-controller --versions` 來尋找最新版本的 Helm Chart。

```
helm install aws-load-balancer-controller eks/aws-load-balancer-controller \
  -n kube-system \
  --version AWS_LBC_HELM_VERSION \
  --set clusterName=CLUSTER_NAME \
  --set region=AWS_REGION \
  --set vpcId=VPC_ID \
  --set serviceAccount.create=false \
  --set serviceAccount.name=aws-load-balancer-controller
```

6. 確認已成功安裝 AWS Load Balancer控制器。

```
kubectl get -n kube-system deployment aws-load-balancer-controller
```

```
NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
aws-load-balancer-controller   2/2     2            2           84s
```

7. 建立範例應用程式。以下範例會使用 Istio Bookinfo 範例微服務應用程式。

```
kubectl apply -f https://raw.githubusercontent.com/istio/istio/refs/heads/master/
samples/bookinfo/platform/kube/bookinfo.yaml
```

8. 使用下列內容建立名為 `my-ingress-alb.yaml` 的檔案。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress
  namespace: default
  annotations:
    alb.ingress.kubernetes.io/load-balancer-name: "my-ingress-alb"
```

```
    alb.ingress.kubernetes.io/target-type: "ip"
    alb.ingress.kubernetes.io/scheme: "internet-facing"
    alb.ingress.kubernetes.io/healthcheck-path: "/details/1"
spec:
  ingressClassName: alb
  rules:
  - http:
      paths:
      - backend:
          service:
            name: details
            port:
              number: 9080
        path: /details
        pathType: Prefix
```

9. 將傳入組態套用至叢集。

```
kubectl apply -f my-ingress-alb.yaml
```

10為您的傳入資源佈建 ALB 可能需要幾分鐘的時間。佈建 ALB 後，您的傳入資源將會為其指派一個與 ALB 部署的 DNS 名稱對應的地址。地址的格式為 <alb-name>-<random-string>.<region>.elb.amazonaws.com。

```
kubectl get ingress my-ingress
```

```
NAME         CLASS   HOSTS   ADDRESS
    PORTS   AGE
my-ingress   alb     *       my-ingress-alb-<random-
string>.<region>.elb.amazonaws.com   80      23m
```

11.使用 ALB 的地址存取服務。

```
curl -s http//my-ingress-alb-<random-string>.<region>.elb.amazonaws.com:80/details/1
 | jq
```

```
{
  "id": 1,
  "author": "William Shakespeare",
  "year": 1595,
  "type": "paperback",
```

```
  "pages": 200,
  "publisher": "PublisherA",
  "language": "English",
  "ISBN-10": "1234567890",
  "ISBN-13": "123-1234567890"
  "details": "This is the details page"
}
```

Cilium Ingress 和 Cilium Gateway 概觀

Cilium 的傳入功能內建於 Cilium 的架構中，並且可以使用 Kubernetes 傳入 API 或閘道 API 進行管理。如果您沒有現有的輸入資源， AWS 建議您從閘道 API 開始，因為它是定義和管理 Kubernetes 網路資源更明確且靈活的方法。Kubernetes 閘道 API 旨在標準化 Kubernetes 叢集中傳入、負載平衡和服務網格的網路資源的定義和管理方式。

當您啟用 Cilium 的傳入或閘道功能時，Cilium 運算子會協調叢集中的傳入/閘道物件，而每個節點上的 Envoy 代理會處理第 7 層 (L7) 網路流量。Cilium 不會直接佈建傳入/閘道基礎結構，例如負載平衡器。如果您計劃搭配負載平衡器使用 Cilium Ingress/Gateway，則必須使用負載平衡器的工具，通常是傳入或閘道控制器，來部署和管理負載平衡器的基礎結構。

對於傳入/閘道流量，Cilium 會處理核心網路流量和 L3/L4 政策強制執行，而整合式 Envoy 代理則負責處理 L7 網路流量。藉助 Cilium Ingress/Gateway，Envoy 負責套用 L7 路由規則、政策和請求操作、進階流量管理 (例如流量分割和鏡像) 以及 TLS 終止和起始。根據預設，Cilium 的 Envoy 代理會部署為單獨的 DaemonSet (`cilium-envoy`)，這樣一來，Envoy 和 Cilium 代理程式可分別進行更新、擴展和管理。

如需 Cilium Ingress 和 Cilium Gateway 運作方式的詳細資訊，請參閱 Cilium 文件中的 Cilium Ingress 和 Cilium Gateway 頁面。

Cilium Ingress 和 Gateway 比較

下表總結了截至 Cilium 版本 1.17.x 的 Cilium Ingress 和 Cilium Gateway 功能。

功能	Ingress	閘道
服務類型 LoadBalancer	是	是
服務類型 NodePort	是	否[1]
主機網路	是	是

功能	Ingress	閘道
共用的負載平衡器	是	是
專用負載平衡器	是	否[2]
網路政策	是	是
通訊協定	第 7 層 (HTTP(S)、gRPC)	第 7 層 (HTTP(S)、gRPC)[3]
TLS 傳遞	是	是
流量管理	路徑和主機路由	路徑和主機路由、URL 重新導向和重寫、流量分割、標頭修改

[1] Cilium Gateway 計劃在 Cilium 版本 1.18.x 中對 NodePort 服務提供支援 (#27273)

[2] Cilium Gateway 支援專用負載平衡器 (#25567)

[3] Cilium Gateway 支援 TCP/UDP (#21929)

安裝 Cilium Gateway

考量事項

- 必須設定 Cilium，即將 `nodePort.enabled` 設定為 `true`，如以下範例所示。如果您使用的是 Cilium 的 kube-proxy 取代功能，則不需要將 `nodePort.enabled` 設定為 `true`。
- 必須設定 Cilium，即將 `envoy.enabled` 設定為 `true`，如以下範例所示。
- Cilium Gateway 可以部署在負載平衡器 (預設) 或主機網路模式中。
- 在負載平衡器模式下使用 Cilium Gateway 時，必須在閘道資源上設定`service.beta.kubernetes.io/aws-load-balancer-type: "external"`註釋，以防止舊版 AWS 雲端提供者為 Cilium 為閘道資源建立的 LoadBalancer 類型的服務建立 Classic Load Balancer。 LoadBalancer
- 在主機網路模式中使用 Cilium Gateway 時，即會停用 LoadBalancer 類型的服務模式。對於沒有負載平衡器基礎結構的環境，主機網路模式非常有用，如需詳細資訊，請參閱 the section called “主機網路”。

先決條件

1. 已在命令列環境中安裝 Helm，請參閱安裝 Helm 說明。
2. 遵循 the section called “設定 CNI” 中的指示安裝 Cilium。

程序

1. 安裝 Kubernetes 閘道 API 自訂資源定義 (CRD)。

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/gateway-api/
v1.2.1/config/crd/standard/gateway.networking.k8s.io_gatewayclasses.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/gateway-api/
v1.2.1/config/crd/standard/gateway.networking.k8s.io_gateways.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/gateway-api/
v1.2.1/config/crd/standard/gateway.networking.k8s.io_httproutes.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/gateway-api/
v1.2.1/config/crd/standard/gateway.networking.k8s.io_referencegrants.yaml
kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/gateway-api/
v1.2.1/config/crd/standard/gateway.networking.k8s.io_grpcroutes.yaml
```

2. 建立稱為 `cilium-gateway-values.yaml` 的檔案，其中具有以下內容。以下範例會將 Cilium Gateway 設定為使用預設負載平衡器模式，以及為僅在混合節點上執行的 Envoy 代理使用單獨 `cilium-envoy` DaemonSet。

```
gatewayAPI:
  enabled: true
  # uncomment to use host network mode
  # hostNetwork:
  #   enabled: true
nodePort:
  enabled: true
envoy:
  enabled: true
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: eks.amazonaws.com/compute-type
            operator: In
            values:
```

```
            - hybrid
```

3. 將 Helm 值套用至叢集。

```
helm upgrade cilium oci://public.ecr.aws/eks/cilium/cilium \
  --namespace kube-system \
  --reuse-values \
  --set operator.rollOutPods=true \
  --values cilium-gateway-values.yaml
```

4. 確認 Cilium 運算子、代理程式和 Envoy Pod 正在執行。

```
kubectl -n kube-system get pods --selector=app.kubernetes.io/part-of=cilium
```

```
NAME                               READY   STATUS    RESTARTS   AGE
cilium-envoy-5pgnd                 1/1     Running   0          6m31s
cilium-envoy-6fhg4                 1/1     Running   0          6m30s
cilium-envoy-jskrk                 1/1     Running   0          6m30s
cilium-envoy-k2xtb                 1/1     Running   0          6m31s
cilium-envoy-w5s9j                 1/1     Running   0          6m31s
cilium-grwlc                       1/1     Running   0          4m12s
cilium-operator-68f7766967-5nnbl   1/1     Running   0          4m20s
cilium-operator-68f7766967-7spfz   1/1     Running   0          4m20s
cilium-pnxcv                       1/1     Running   0          6m29s
cilium-r7qkj                       1/1     Running   0          4m12s
cilium-wxhfn                       1/1     Running   0          4m1s
cilium-z7hlb                       1/1     Running   0          6m30s
```

設定 Cilium Gateway

將 `gatewayClassName` 設定為 `cilium`，即可在閘道物件上啟用 Cilium Ingress。Cilium 為閘道資源建立的服務可以使用閘道物件上的欄位進行設定。閘道控制器用來設定負載平衡器基礎結構的常見註釋可以透過閘道物件的 `infrastructure` 欄位進行設定。使用 Cilium 的 LoadBalancer IPAM 時 (請參閱 the section called “服務類型 LoadBalancer” 中的範例)，可在閘道物件的 `addresses` 欄位中設定用於 LoadBalancer 類型的服務的 IP 位址。如需閘道組態的詳細資訊，請參閱 Kubernetes 閘道 API 規格。

```
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
```

```
  name: my-gateway
spec:
  gatewayClassName: cilium
  infrastructure:
    annotations:
      service.beta.kubernetes.io/...
      service.kubernetes.io/...
  addresses:
  - type: IPAddress
    value: <LoadBalancer IP address>
  listeners:
  ...
```

Cilium 和 Kubernetes Gateway 規格支援 GatewayClass、閘道、HTTPRoute、GRPCRoute 和 ReferenceGrant 資源。

- 如需可用欄位的清單，請參閱 HTTPRoute 和 GRPCRoute 規格。
- 請參閱以下 the section called “部署 Cilium Gateway” 章節中的範例及 Cilium 文件中的範例，以了解如何使用和設定這些資源。

部署 Cilium Gateway

1. 建立範例應用程式。以下範例會使用 Istio Bookinfo 範例微服務應用程式。

```
kubectl apply -f https://raw.githubusercontent.com/istio/istio/refs/heads/master/
samples/bookinfo/platform/kube/bookinfo.yaml
```

2. 確認應用程式已成功執行。

```
kubectl get pods
```

```
NAME                              READY   STATUS    RESTARTS   AGE
details-v1-766844796b-9965p       1/1     Running   0          81s
productpage-v1-54bb874995-jmc8j   1/1     Running   0          80s
ratings-v1-5dc79b6bcd-smzxz       1/1     Running   0          80s
reviews-v1-598b896c9d-vj7gb       1/1     Running   0          80s
reviews-v2-556d6457d-xbt8v        1/1     Running   0          80s
reviews-v3-564544b4d6-cpmvq       1/1     Running   0          80s
```

3. 使用下列內容建立名為 my-gateway.yaml 的檔案。以下範例使用 service.beta.kubernetes.io/aws-load-balancer-type: "external"註釋，以防止舊版 AWS 雲端提供者為 Cilium 為閘道資源建立的 LoadBalancer 類型服務建立 Classic Load Balancer。 LoadBalancer

```
---
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
  name: my-gateway
spec:
  gatewayClassName: cilium
  infrastructure:
    annotations:
      service.beta.kubernetes.io/aws-load-balancer-type: "external"
  listeners:
  - protocol: HTTP
    port: 80
    name: web-gw
    allowedRoutes:
      namespaces:
        from: Same
---
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: http-app-1
spec:
  parentRefs:
  - name: my-gateway
    namespace: default
  rules:
  - matches:
    - path:
        type: PathPrefix
        value: /details
    backendRefs:
    - name: details
      port: 9080
```

4. 將閘道資源套用至您的叢集。

```
kubectl apply -f my-gateway.yaml
```

5. 確認閘道資源和對應的服務已建立。在此階段，閘道資源的 ADDRESS 欄位預期不會填入 IP 位址或主機名稱，而且閘道資源的 LoadBalancer 類型的服務同樣不會指派 IP 位址或主機名稱。

```
kubectl get gateway my-gateway
```

```
NAME         CLASS    ADDRESS   PROGRAMMED   AGE
my-gateway   cilium             True         10s
```

```
kubectl get svc cilium-gateway-my-gateway
```

```
NAME                          TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)
    AGE
cilium-gateway-my-gateway   LoadBalancer   172.16.227.247   <pending>     80:30912/
TCP   24s
```

6. 繼續 the section called “服務類型 LoadBalancer” 以設定閘道資源，進而使用 Cilium Load Balancer IPAM 配置的 IP 位址；以及使用 the section called “服務類型 NodePort” 或 the section called “主機網路” 以設定閘道資源，進而使用 NodePort 或主機網路位址。

安裝 Cilium Ingress

考量事項

- 必須設定 Cilium，即將 `nodePort.enabled` 設定為 `true`，如以下範例所示。如果您使用的是 Cilium 的 kube-proxy 取代功能，則不需要將 `nodePort.enabled` 設定為 `true`。
- 必須設定 Cilium，即將 `envoy.enabled` 設定為 `true`，如以下範例所示。
- 將 `ingressController.loadbalancerMode` 設定為 `dedicated` 時，Cilium 會為每個傳入資源建立專用服務。將 `ingressController.loadbalancerMode` 設定為 `shared` 時，Cilium 會為叢集中的所有傳入資源建立 LoadBalancer 類型的共用服務。使用 `shared` 負載平衡器模式時，共用服務的設定 (例如 `labels`、`annotations`、`type` 和 `loadBalancerIP`) 可在 Helm 值的 `ingressController.service` 區段中進行設定。如需詳細資訊，請參閱 Cilium Helm 值參考。
- 將 `ingressController.default` 設定 `true` 時，Cilium 會設定為叢集的預設傳入控制器，即使未在傳入資源上指定 `ingressClassName`，也會建立傳入項目。

- Cilium Ingress 可以部署在負載平衡器 (預設)、節點連接埠或主機網路模式中。在主機網路模式下安裝 Cilium 時，會停用 LoadBalancer 類型的服務和 NodePort 類型的服務模式。如需詳細資訊，請參閱the section called “主機網路”。
- 一律在 Helm 值`service.beta.kubernetes.io/aws-load-balancer-type: "external"`中`ingressController.service.annotations`將 設定為 ，以防止舊版 AWS 雲端提供者為 Cilium Helm Chart 建立的預設`cilium-ingress`服務建立 Classic Load Balancer。

先決條件

1. 已在命令列環境中安裝 Helm，請參閱安裝 Helm 說明。
2. 遵循 the section called “設定 CNI” 中的指示安裝 Cilium。

程序

1. 建立稱為 `cilium-ingress-values.yaml` 的檔案，其中具有以下內容。以下範例會將 Cilium Ingress 設定為使用預設負載平衡器 `dedicated` 模式，以及為僅在混合節點上執行的 Envoy 代理使用單獨 `cilium-envoy` DaemonSet。

```
ingressController:
  enabled: true
  loadbalancerMode: dedicated
  service:
    annotations:
      service.beta.kubernetes.io/aws-load-balancer-type: "external"
nodePort:
  enabled: true
envoy:
  enabled: true
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: eks.amazonaws.com/compute-type
            operator: In
            values:
            - hybrid
```

2. 將 Helm 值套用至叢集。

```
helm upgrade cilium oci://public.ecr.aws/eks/cilium/cilium \
  --namespace kube-system \
  --reuse-values \
  --set operator.rollOutPods=true \
  --values cilium-ingress-values.yaml
```

3. 確認 Cilium 運算子、代理程式和 Envoy Pod 正在執行。

```
kubectl -n kube-system get pods --selector=app.kubernetes.io/part-of=cilium
```

```
NAME                               READY   STATUS    RESTARTS   AGE
cilium-envoy-5pgnd                 1/1     Running   0          6m31s
cilium-envoy-6fhg4                 1/1     Running   0          6m30s
cilium-envoy-jskrk                 1/1     Running   0          6m30s
cilium-envoy-k2xtb                 1/1     Running   0          6m31s
cilium-envoy-w5s9j                 1/1     Running   0          6m31s
cilium-grwlc                       1/1     Running   0          4m12s
cilium-operator-68f7766967-5nnbl   1/1     Running   0          4m20s
cilium-operator-68f7766967-7spfz   1/1     Running   0          4m20s
cilium-pnxcv                       1/1     Running   0          6m29s
cilium-r7qkj                       1/1     Running   0          4m12s
cilium-wxhfn                       1/1     Running   0          4m1s
cilium-z7hlb                       1/1     Running   0          6m30s
```

設定 Cilium Ingress

將 `ingressClassName` 設定為 `cilium`，即可在 Ingress 物件上啟用 Cilium Ingress。當使用 `dedicated` 負載平衡器模式時，Cilium 為傳入資源建立的服務可藉助傳入物件上的注釋進行設定，而當使用 `shared` 負載平衡器模式時，則可在 Cilium/Helm 組態中進行設定。傳入控制器通常會使用這些註釋來設定負載平衡器基礎結構或服務的其他屬性，例如服務類型、負載平衡器模式、連接埠和 TLS 傳遞。重要註釋如下所述。如需支援的註釋的完整清單，請參閱 Cilium 文件中的 Cilium Ingress 註釋。

註釋	Description
`ingress.cilium.io/loadbalancer-mode`	`dedicated` ：每個傳入資源的 LoadBalancer 類型的專用服務 (預設)。

註釋	Description
	`shared`：所有傳入資源的 LoadBalancer 類型的單一服務。
`ingress.cilium.io/service-type`	`LoadBalancer` ：服務屬於 LoadBalancer 類型 (預設) `NodePort`：服務屬於 NodePort。
`service.beta.kubernetes.io/aws-load-balancer-type`	`"external"` ：防止舊版 AWS 雲端提供者為 LoadBalancer 類型的服務佈建 Classic Load Balancer。 LoadBalancer
`lbipam.cilium.io/ips`	要從 Cilium LoadBalancer IPAM 配置的 IP 位址的清單

Cilium 和 Kubernetes Ingress 規格可支援傳入路徑的精確、字首和實作特定相符規則。Cilium 支援 regex 作為其實作專屬相符規則。如需詳細資訊，請參閱 Cilium 文件中的傳入路徑類型和優先順序和路徑類型範例，以及此頁面 the section called “部署 Cilium Ingress” 不分中的範例。

範例 Cilium Ingress 物件如下所示。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress
  annotations:
    service.beta.kubernetes.io/...
    service.kubernetes.io/...
spec:
  ingressClassName: cilium
  rules:
  ...
```

部署 Cilium Ingress

1. 建立範例應用程式。以下範例會使用 Istio Bookinfo 範例微服務應用程式。

```
kubectl apply -f https://raw.githubusercontent.com/istio/istio/refs/heads/master/samples/bookinfo/platform/kube/bookinfo.yaml
```

2. 確認應用程式已成功執行。

```
kubectl get pods
```

```
NAME                              READY   STATUS    RESTARTS   AGE
details-v1-766844796b-9965p       1/1     Running   0          81s
productpage-v1-54bb874995-jmc8j   1/1     Running   0          80s
ratings-v1-5dc79b6bcd-smzxz       1/1     Running   0          80s
reviews-v1-598b896c9d-vj7gb       1/1     Running   0          80s
reviews-v2-556d6457d-xbt8v        1/1     Running   0          80s
reviews-v3-564544b4d6-cpmvq       1/1     Running   0          80s
```

3. 使用下列內容建立名為 my-ingress.yaml 的檔案。以下範例使用 service.beta.kubernetes.io/aws-load-balancer-type: "external"註釋，以防止舊版 AWS 雲端提供者為 Cilium 為輸入資源建立的 LoadBalancer 類型的服務建立 Classic Load Balancer。 LoadBalancer

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress
  namespace: default
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-type: "external"
spec:
  ingressClassName: cilium
  rules:
  - http:
      paths:
      - backend:
          service:
            name: details
            port:
              number: 9080
        path: /details
        pathType: Prefix
```

4. 將傳入資源套用至您的叢集。

```
kubectl apply -f my-ingress.yaml
```

5. 確認傳入資源和對應的服務已建立。在此階段，傳入資源的 ADDRESS 欄位預期不會填入 IP 位址或主機名稱，而且傳入資源的 LoadBalancer 類型的共用或專用服務同樣不會指派 IP 位址或主機名稱。

```
kubectl get ingress my-ingress
```

```
NAME         CLASS    HOSTS   ADDRESS   PORTS   AGE
my-ingress   cilium   *                 80      8s
```

對於負載平衡器模式 `shared`

```
kubectl -n kube-system get svc cilium-ingress
```

```
NAME             TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)
      AGE
cilium-ingress   LoadBalancer   172.16.217.48   <pending>     80:32359/TCP,443:31090/
TCP   10m
```

對於負載平衡器模式 `dedicated`

```
kubectl -n default get svc cilium-ingress-my-ingress
```

```
NAME                         TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)
                  AGE
cilium-ingress-my-ingress    LoadBalancer   172.16.193.15   <pending>     80:32088/
TCP,443:30332/TCP   25s
```

6. 繼續 the section called “服務類型 LoadBalancer” 以設定傳入資源，進而使用 Cilium Load Balancer IPAM 配置的 IP 位址；以及使用 the section called “服務類型 NodePort” 或 the section called “主機網路” 以設定傳入資源，進而使用 NodePort 或主機網路位址。

服務類型 LoadBalancer

現有的負載平衡器基礎結構

根據預設，對於 Cilium Ingress 和 Cilium Gateway，Cilium 會為傳入/閘道資源建立 LoadBalancer 類型的 Kubernetes Service。Cilium 建立的服務屬性可以透過傳入和閘道資源進行設定。當您建立傳入

或閘道資源時，傳入或閘道的外部公開 IP 位址或主機名稱會從負載平衡器基礎結構進行配置，並且通常由傳入或閘道控制器進行佈建。

許多傳入和閘道控制器會使用註釋來偵測和設定負載平衡器基礎結構。如上述範例所示，這些傳入和閘道控制器的註釋會在傳入或閘道資源上設定。如需其支援的註釋，請參閱傳入或閘道控制器的文件，而如需熱門控制器的清單，則請參閱 Kubernetes Ingress 文件和 Kubernetes Gateway 文件。

> **Important**
>
> Cilium Ingress 和 Gateway 不能與 EKS 混合節點的 AWS Load Balancer控制器和 AWS 網路負載平衡器 (NLBs) 搭配使用。嘗試將其一起使用會導致未註冊的目標，因為當 NLB 的 `target-type` 設定為 `ip` 時，NLB 會嘗試直接連接至支援 LoadBalancer 類型的服務的 Pod IP (要求搭配使用 NLB 與在 EKS 混合節點上執行的工作負載)。

沒有負載平衡器基礎結構

如果您的環境中沒有負載平衡器基礎結構和對應的傳入/閘道控制器，則可以將傳入/閘道資源和對應的 LoadBalancer 類型的服務設定為使用 Cilium Load Balancer IP 位址管理 (LB IPAM) 配置的 IP 位址。Cilium LB IPAM 可以使用來自您內部部署環境的已知 IP 位址範圍進行設定，並且可以使用 Cilium 的內建邊界閘道協定 (BGP) 支援或 L2 公告，從而向內部部署網路公告 LoadBalancer IP 位址。

以下範例說明如何使用 IP 位址設定 Cilium 的 LB IPAM 以用於您的傳入/閘道資源，以及如何設定 Cilium BGP 控制平面以向內部部署網路公告 LoadBalancer IP 位址。根據預設，Cilium 的 LB IPAM 功能已啟用，但在建立 `CiliumLoadBalancerIPPool` 資源之前不會啟動。

先決條件

- 遵循 the section called “安裝 Cilium Ingress” 或 the section called “安裝 Cilium Gateway” 中的指示安裝 Cilium Ingress 或 Gateway。
- 遵循 the section called “部署 Cilium Ingress” 或 the section called “部署 Cilium Gateway” 中的指示，部署範例應用程式的 Cilium Ingress 或 Gateway 資源。
- 遵循 the section called “設定 BPA” 中的指示啟用 Cilium BGP 控制平面。如果您不想使用 BGP，則可以略過此先決條件，但在 Cilium LB IPAM 配置的 LoadBalancer IP 位址在內部部署網路上可路由之前，您將無法存取您的傳入或閘道資源。

程序

1. 選擇性地修補傳入或閘道資源，以請求用於 LoadBalancer 類型的服務的特定 IP 位址。如果您未請求特定 IP 位址，Cilium 將從後續步驟中的 CiliumLoadBalancerIPPool 資源中設定的 IP 位址範圍配置 IP 位址。在下列命令中，使用 IP 位址取代 LB_IP_ADDRESS，以請求 LoadBalancer 類型的服務。

   閘道

   ```
   kubectl patch gateway -n default my-gateway --type=merge -p '{
     "spec": {
       "addresses": [{"type": "IPAddress", "value": "LB_IP_ADDRESS"}]
     }
   }'
   ```

   傳入

   ```
   kubectl patch ingress my-ingress --type=merge -p '{
     "metadata": {"annotations": {"lbipam.cilium.io/ips": "LB_IP_ADDRESS"}}
   }'
   ```

2. 使用 CiliumLoadBalancerIPPool 資源建立名為 cilium-lbip-pool-ingress.yaml 的檔案，以為您的傳入/閘道資源設定 Load Balancer IP 位址範圍。

   - 如果您使用的是 Cilium Ingress，Cilium 會自動將 cilium.io/ingress: "true" 標籤套用至其為傳入資源建立的服務。您可以在 CiliumLoadBalancerIPPool 資源定義的 serviceSelector 欄位中使用此標籤，以選取符合 LB IPAM 資格的服務。
   - 如果您使用的是 Cilium Gateway，則可以使用 CiliumLoadBalancerIPPool 資源定義的 serviceSelector 欄位中的 gateway.networking.k8s.io/gateway-name 標籤來選取符合 LB IPAM 資格的閘道資源。
   - 使用用於 Load Balancer IP 位址的 IP 位址範圍取代 LB_IP_CIDR。若要選取單一 IP 位址，請使用 /32 CIDR。如需詳細資訊，請參閱 Cilium 文件中的 LoadBalancer IP 位址管理。

     ```
     apiVersion: cilium.io/v2alpha1
     kind: CiliumLoadBalancerIPPool
     metadata:
       name: bookinfo-pool
     spec:
       blocks:
       - cidr: "LB_IP_CIDR"
     ```

```
  serviceSelector:
    # if using Cilium Gateway
    matchExpressions:
      - { key: gateway.networking.k8s.io/gateway-name, operator: In, values: [ my-
gateway ] }
    # if using Cilium Ingress
    matchLabels:
      cilium.io/ingress: "true"
```

3. 將 CiliumLoadBalancerIPPool 資源套用至您的叢集。

```
kubectl apply -f cilium-lbip-pool-ingress.yaml
```

4. 確認已從 Cilium LB IPAM 為傳入/閘道資源配置 IP 位址。

   閘道

```
kubectl get gateway my-gateway
```

```
NAME         CLASS    ADDRESS        PROGRAMMED   AGE
my-gateway   cilium   LB_IP_ADDRESS    True         6m41s
```

   傳入

```
kubectl get ingress my-ingress
```

```
NAME         CLASS    HOSTS   ADDRESS        PORTS   AGE
my-ingress   cilium   *       LB_IP_ADDRESS   80      10m
```

5. 使用 CiliumBGPAdvertisement 資源建立名為 cilium-bgp-advertisement-ingress.yaml 的檔案，以為您的傳入/閘道資源公告 LoadBalancer IP 位址。如果您不使用 Cilium BGP，則可以略過此步驟。用於傳入/閘道的 LoadBalancer IP 位址必須在內部部署網路上可路由，這樣您才能在下一個步驟中查詢服務。

```
apiVersion: cilium.io/v2alpha1
kind: CiliumBGPAdvertisement
metadata:
  name: bgp-advertisement-lb-ip
  labels:
    advertise: bgp
```

```
spec:
  advertisements:
    - advertisementType: "Service"
      service:
        addresses:
          - LoadBalancerIP
      selector:
        # if using Cilium Gateway
        matchExpressions:
          - { key: gateway.networking.k8s.io/gateway-name, operator: In, values:
 [ my-gateway ] }
        # if using Cilium Ingress
        matchLabels:
          cilium.io/ingress: "true"
```

6. 將 CiliumBGPAdvertisement 資源套用至您的叢集。

```
kubectl apply -f cilium-bgp-advertisement-ingress.yaml
```

7. 使用從 Cilium LB IPAM 配置的 IP 位址存取服務。

```
curl -s http://LB_IP_ADDRESS:80/details/1 | jq
```

```
{
  "id": 1,
  "author": "William Shakespeare",
  "year": 1595,
  "type": "paperback",
  "pages": 200,
  "publisher": "PublisherA",
  "language": "English",
  "ISBN-10": "1234567890",
  "ISBN-13": "123-1234567890"
}
```

服務類型 NodePort

如果您在環境中沒有負載平衡器基礎結構和對應的傳入控制器，或者您是自我管理負載平衡器基礎結構或使用 DNS 型負載平衡，則可以將 Cilium Ingress 設定為為傳入資源建立 NodePort 類型的服務。搭配 Cilium Ingress 使用 NodePort 時，NodePort 類型的服務會在連接埠範圍 30000-32767 內每個節點

的連接埠上公開。在此模式中，當流量到達 NodePort 上叢集中的任何節點時，即會轉送到可傳回服務的 Pod，而這些 Pod 可能位於相同的節點，也可能位於不同的節點。

> **Note**
>
> Cilium Gateway 計劃在 Cilium 版本 1.18.x 中對 NodePort 服務提供支援 (#27273)

先決條件

- 遵循 the section called “安裝 Cilium Ingress” 中的指示安裝 Cilium Ingress。
- 遵循 the section called “部署 Cilium Ingress” 中的指示，部署範例應用程式的 Cilium Ingress 資源。

程序

1. 修補現有的傳入資源 `my-ingress`，以將其從 LoadBalancer 類型的服務變更為 NodePort。

```
kubectl patch ingress my-ingress --type=merge -p '{
    "metadata": {"annotations": {"ingress.cilium.io/service-type": "NodePort"}}
}'
```

如果您尚未建立傳入資源，則可以將下列傳入定義套用至叢集，進而建立傳入資源。請注意，以下傳入定義會使用 the section called “部署 Cilium Ingress” 中所述的 Istio Bookinfo 範例應用程式。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress
  namespace: default
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-type: "external"
    "ingress.cilium.io/service-type": "NodePort"
spec:
  ingressClassName: cilium
  rules:
  - http:
      paths:
      - backend:
          service:
            name: details
```

```
            port:
              number: 9080
        path: /details
        pathType: Prefix
```

2. 確認傳入資源的服務已更新為使用服務類型 NodePort。請注意輸出中的 HTTP 通訊協定的連接埠。在以下範例中，此 HTTP 連接埠是 32353，且在後續步驟中，將用其來查詢服務。搭配 NodePort 類型的服務使用 Cilium Ingress 的優點是，對於 Ingress 流量，您可以套用路徑和以主機為基礎的路由以及相關網路政策，而對於沒有 Ingress 的標準 NodePort 類型的服務，您則無法如此操作。

```
kubectl -n default get svc cilium-ingress-my-ingress
```

```
NAME                          TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)
             AGE
cilium-ingress-my-ingress   NodePort   172.16.47.153   <none>        80:32353/
TCP,443:30253/TCP   27m
```

3. 獲取您叢集節點的 IP 位址。

```
kubectl get nodes -o wide
```

```
NAME                   STATUS   ROLES    AGE   VERSION               INTERNAL-IP
 EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION       CONTAINER-RUNTIME
mi-026d6a261e355fba7   Ready    <none>   23h   v1.32.3-eks-473151a   10.80.146.150
 <none>        Ubuntu 22.04.5 LTS   5.15.0-142-generic   containerd://1.7.27
mi-082f73826a163626e   Ready    <none>   23h   v1.32.3-eks-473151a   10.80.146.32
 <none>        Ubuntu 22.04.4 LTS   5.15.0-142-generic   containerd://1.7.27
mi-09183e8a3d755abf6   Ready    <none>   23h   v1.32.3-eks-473151a   10.80.146.33
 <none>        Ubuntu 22.04.4 LTS   5.15.0-142-generic   containerd://1.7.27
mi-0d78d815980ed202d   Ready    <none>   23h   v1.32.3-eks-473151a   10.80.146.97
 <none>        Ubuntu 22.04.4 LTS   5.15.0-142-generic   containerd://1.7.27
mi-0daa253999fe92daa   Ready    <none>   23h   v1.32.3-eks-473151a   10.80.146.100
 <none>        Ubuntu 22.04.4 LTS   5.15.0-142-generic   containerd://1.7.27
```

4. 使用節點的 IP 位址和上文擷取的 NodePort 存取 NodePort 類型的服務。在以下範例中，使用的節點 IP 位址是 10.80.146.32，而 NodePort 是 32353。使用您環境的值取代這些值。

```
curl -s http://10.80.146.32:32353/details/1 | jq
```

```
{
```

```
  "id": 1,
  "author": "William Shakespeare",
  "year": 1595,
  "type": "paperback",
  "pages": 200,
  "publisher": "PublisherA",
  "language": "English",
  "ISBN-10": "1234567890",
  "ISBN-13": "123-1234567890"
}
```

主機網路

類似於 NodePort 類型的服務，如果您沒有負載平衡器基礎結構和傳入或閘道控制器，或者如果您使用外部負載平衡器自我管理負載平衡，則可以將 Cilium Ingress 和 Cilium Gateway 設定為可直接在主機網路上公開傳入和閘道資源。為傳入或閘道資源啟用主機網路模式時，會自動停用 LoadBalancer 類型和 NodePort 類型的服務模式，而主機網路模式會與每個傳入或閘道資源的這些替代模式互斥。相較於 NodePort 類型的服務模式，主機網路模式能擴增可使用的連接埠範圍 (不限於 30000-32767 NodePort 範圍)，並且您可以設定 Envoy 代理在主機網路上執行的節點子集。

先決條件

- 遵循 the section called “安裝 Cilium Ingress” 或 the section called “安裝 Cilium Gateway” 中的指示安裝 Cilium Ingress 或 Gateway。

程序

閘道

1. 建立名為 `cilium-gateway-host-network.yaml` 且具有下列內容的檔案。

```
gatewayAPI:
  enabled: true
  hostNetwork:
    enabled: true
    # uncomment to restrict nodes where Envoy proxies run on the host network
    # nodes:
    #   matchLabels:
    #     role: gateway
```

2. 將主機網路 Cilium Gateway 組態套用至您的叢集。

```
helm upgrade cilium oci://public.ecr.aws/eks/cilium/cilium \
  --namespace kube-system \
  --reuse-values \
  --set operator.rollOutPods=true \
  -f cilium-gateway-host-network.yaml
```

如果您尚未建立閘道資源，則可以將下列閘道定義套用至叢集，進而建立閘道資源。以下閘道定義會使用 the section called “部署 Cilium Gateway” 中所述的 Istio Bookinfo 範例應用程式。在以下範例中，閘道資源會設定為使用 HTTP 接聽程式的 8111 連接埠，而這是主機網路上執行的 Envoy 代理程式的共用接聽程式連接埠。如果您為閘道資源使用了具有特殊權限的連接埠 (低於 1023)，請參閱 Cilium 文件以取得說明。

```
---
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
  name: my-gateway
spec:
  gatewayClassName: cilium
  listeners:
  - protocol: HTTP
    port: 8111
    name: web-gw
    allowedRoutes:
      namespaces:
        from: Same
---
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: http-app-1
spec:
  parentRefs:
  - name: my-gateway
    namespace: default
  rules:
  - matches:
    - path:
        type: PathPrefix
        value: /details
    backendRefs:
```

```
    - name: details
      port: 9080
```

您可以使用下列命令來觀測套用的 Cilium Envoy 組態。

```
kubectl get cec cilium-gateway-my-gateway -o yaml
```

您可以使用下列命令來取得 cilium-gateway-my-gateway 服務的 Envoy 接聽程式連接埠。在此範例中，共用接聽程式連接埠為 8111。

```
kubectl get cec cilium-gateway-my-gateway -o jsonpath={.spec.services[0].ports[0]}
```

3. 獲取您叢集節點的 IP 位址。

```
kubectl get nodes -o wide
```

```
NAME                    STATUS   ROLES    AGE   VERSION               INTERNAL-IP
 EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION       CONTAINER-RUNTIME
mi-026d6a261e355fba7   Ready    <none>   23h   v1.32.3-eks-473151a   10.80.146.150
 <none>        Ubuntu 22.04.5 LTS   5.15.0-142-generic   containerd://1.7.27
mi-082f73826a163626e   Ready    <none>   23h   v1.32.3-eks-473151a   10.80.146.32
 <none>        Ubuntu 22.04.4 LTS   5.15.0-142-generic   containerd://1.7.27
mi-09183e8a3d755abf6   Ready    <none>   23h   v1.32.3-eks-473151a   10.80.146.33
 <none>        Ubuntu 22.04.4 LTS   5.15.0-142-generic   containerd://1.7.27
mi-0d78d815980ed202d   Ready    <none>   23h   v1.32.3-eks-473151a   10.80.146.97
 <none>        Ubuntu 22.04.4 LTS   5.15.0-142-generic   containerd://1.7.27
mi-0daa253999fe92daa   Ready    <none>   23h   v1.32.3-eks-473151a   10.80.146.100
 <none>        Ubuntu 22.04.4 LTS   5.15.0-142-generic   containerd://1.7.27
```

4. 使用節點的 IP 位址和 cilium-gateway-my-gateway 資源的接聽程式連接埠存取服務。在以下範例中，使用的節點 IP 位址是 10.80.146.32，而接聽程式連接埠是 8111。使用您環境的值取代這些值。

```
curl -s http://10.80.146.32:8111/details/1 | jq
```

```
{
  "id": 1,
  "author": "William Shakespeare",
  "year": 1595,
  "type": "paperback",
```

```
  "pages": 200,
  "publisher": "PublisherA",
  "language": "English",
  "ISBN-10": "1234567890",
  "ISBN-13": "123-1234567890"
}
```

Ingress

由於上游 Cilium 問題 (#34028)，主機網路模式中的 Cilium Ingress 需要使用 `loadbalancerMode: shared`，而這會為叢集中的所有傳入資源建立 ClusterIP 類型的單一服務。如果您為傳入資源使用了具有特殊權限的連接埠 (低於 1023)，請參閱 Cilium 文件以取得說明。

1. 建立名為 `cilium-ingress-host-network.yaml` 且具有下列內容的檔案。

```
ingressController:
  enabled: true
  loadbalancerMode: shared
  # This is a workaround for the upstream Cilium issue
  service:
    externalTrafficPolicy: null
    type: ClusterIP
  hostNetwork:
    enabled: true
    # ensure the port does not conflict with other services on the node
    sharedListenerPort: 8111
    # uncomment to restrict nodes where Envoy proxies run on the host network
    # nodes:
    #   matchLabels:
    #     role: ingress
```

2. 將主機網路 Cilium Ingress 組態套用至您的叢集。

```
helm upgrade cilium oci://public.ecr.aws/eks/cilium/cilium \
  --namespace kube-system \
  --reuse-values \
  --set operator.rollOutPods=true \
  -f cilium-ingress-host-network.yaml
```

如果您尚未建立傳入資源，則可以將下列傳入定義套用至叢集，進而建立傳入資源。以下傳入定義會使用 the section called “部署 Cilium Ingress” 中所述的 Istio Bookinfo 範例應用程式。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress
  namespace: default
spec:
  ingressClassName: cilium
  rules:
  - http:
      paths:
      - backend:
          service:
            name: details
            port:
              number: 9080
        path: /details
        pathType: Prefix
```

您可以使用下列命令來觀測套用的 Cilium Envoy 組態。

```
kubectl get cec -n kube-system cilium-ingress -o yaml
```

您可以使用下列命令來取得 `cilium-ingress` 服務的 Envoy 接聽程式連接埠。在此範例中，共用接聽程式連接埠為 8111。

```
kubectl get cec -n kube-system cilium-ingress -o
 jsonpath={.spec.services[0].ports[0]}
```

3. 獲取您叢集節點的 IP 位址。

```
kubectl get nodes -o wide
```

```
NAME                   STATUS   ROLES    AGE   VERSION               INTERNAL-IP
 EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION       CONTAINER-RUNTIME
mi-026d6a261e355fba7   Ready    <none>   23h   v1.32.3-eks-473151a   10.80.146.150
 <none>        Ubuntu 22.04.5 LTS   5.15.0-142-generic   containerd://1.7.27
mi-082f73826a163626e   Ready    <none>   23h   v1.32.3-eks-473151a   10.80.146.32
 <none>        Ubuntu 22.04.4 LTS   5.15.0-142-generic   containerd://1.7.27
mi-09183e8a3d755abf6   Ready    <none>   23h   v1.32.3-eks-473151a   10.80.146.33
 <none>        Ubuntu 22.04.4 LTS   5.15.0-142-generic   containerd://1.7.27
```

```
mi-0d78d815980ed202d   Ready    <none>   23h   v1.32.3-eks-473151a   10.80.146.97
 <none>        Ubuntu 22.04.4 LTS   5.15.0-142-generic   containerd://1.7.27
mi-0daa253999fe92daa   Ready    <none>   23h   v1.32.3-eks-473151a   10.80.146.100
 <none>        Ubuntu 22.04.4 LTS   5.15.0-142-generic   containerd://1.7.27
```

4. 使用節點的 IP 位址和 `cilium-ingress` 資源的 `sharedListenerPort` 存取服務。在以下範例中，使用的節點 IP 位址是 `10.80.146.32`，而接聽程式連接埠是 8111。使用您環境的值取代這些值。

```
curl -s http://10.80.146.32:8111/details/1 | jq
```

```
{
  "id": 1,
  "author": "William Shakespeare",
  "year": 1595,
  "type": "paperback",
  "pages": 200,
  "publisher": "PublisherA",
  "language": "English",
  "ISBN-10": "1234567890",
  "ISBN-13": "123-1234567890"
}
```

為混合節點設定 LoadBalancer 類型的服務

本主題會說明如何為在 Amazon EKS 混合節點上執行的應用程式設定第 4 層 (L4) 負載平衡。LoadBalancer 類型的 Kubernetes Service 可用於向叢集外部公開 Kubernetes 應用程式。LoadBalancer 類型的服務通常可與雲端或內部部署環境中的實體負載平衡器基礎結構搭配使用，進而提供工作負載的流量。此負載平衡器基礎結構通常使用環境特定的控制器進行佈建。

AWS 支援針對 EKS 混合節點上執行的 LoadBalancer 類型的服務使用 AWS Network Load Balancer (NLB) 和 Cilium。使用 NLB 還是 Cilium 的決定取決於應用程式流量的來源。如果應用程式流量來自 AWS 區域，則 AWS 會建議使用 AWS NLB 和 AWS Load Balancer 控制器。如果應用程式流量來自本機內部部署或邊緣環境，則 AWS 會建議使用 Cilium 的內建負載平衡功能，而該功能可在您的環境中搭配或不搭配負載平衡器基礎結構使用。

如需第 7 層 (L7) 應用程式流量負載平衡，請參閱 the section called “設定 Ingress”。如需使用 EKS 進行負載平衡的一般資訊，請參閱負載平衡的最佳實務。

AWS Network Load Balancer

對於在混合節點上執行的工作負載，您可以搭配目標類型 `ip` 使用 AWS Load Balancer 控制器和 NLB。使用目標類型 `ip` 時，NLB 可繞過服務層網路路徑，將流量直接轉送至 Pod。若要讓 NLB 達到混合節點上的 Pod IP 目標，您的內部部署 Pod CIDR 必須在內部部署網路上可路由。此外，AWS Load Balancer 控制器使用 Webhook 並且需要從 EKS 控制平面直接通訊。如需詳細資訊，請參閱 the section called “設定 Webhook”。

- 如需有關 AWS Network Load Balancer 和 AWS Load Balancer 控制器的其他資訊，請參閱 the section called “網路負載平衡” 以取得子網路組態需求，並參閱 the section called “使用 Helm 安裝” 和負載平衡的最佳實務。
- 請參閱 AWS Load Balancer 控制器 NLB 組態，了解可套用至具有 AWS Network Load Balancer 的 `LoadBalancer` 類型的服務的組態。

先決條件

- 遵循 the section called “設定 CNI” 中的指示安裝 Cilium。
- 遵循 the section called “設定 BPA” 中的指示啟用 Cilium BGP 控制平面。如果您不想使用 BGP，則必須使用替代方法，以讓內部部署 Pod CIDR 在內部部署網路上可路由，如需詳細資訊，請參閱 the section called “可路由的遠端 Pod CIDR”。
- 已在命令列環境中安裝 Helm，請參閱安裝 Helm 說明。
- 已在命令列環境中安裝 eksctl，請參閱安裝 eksctl 說明。

程序

1. 下載適用於 AWS Load Balancer 控制器的 IAM 政策，允許它代表您呼叫 AWS API。

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-
controller/refs/heads/main/docs/install/iam_policy.json
```

2. 使用上一個步驟中下載的政策，建立 IAM 政策。

```
aws iam create-policy \
    --policy-name AWSLoadBalancerControllerIAMPolicy \
    --policy-document file://iam_policy.json
```

3. 使用您的設定取代下列值：叢集名稱 (`CLUSTER_NAME`)、AWS 區域 (`AWS_REGION`) 和 AWS 帳戶 ID (`AWS_ACCOUNT_ID`)，然後執行下列命令。

```
eksctl create iamserviceaccount \
    --cluster=CLUSTER_NAME \
    --namespace=kube-system \
    --name=aws-load-balancer-controller \
    --attach-policy-arn=arn:aws:iam::AWS_ACCOUNT_ID:policy/
AWSLoadBalancerControllerIAMPolicy \
    --override-existing-serviceaccounts \
    --region AWS_REGION \
    --approve
```

4. 新增 eks-charts Helm Chart 儲存庫。AWS 會在 GitHub 上維護此儲存庫。

```
helm repo add eks https://aws.github.io/eks-charts
```

5. 更新您的本機 Helm 儲存庫，以確保您擁有最近的圖表。

```
helm repo update eks
```

6. 安裝 AWS Load Balancer 控制器。使用您的設定取代下列值：叢集名稱 (CLUSTER_NAME)、AWS 區域 (AWS_REGION)、VPC ID (VPC_ID) 和 AWS Load Balancer 控制器 Helm Chart 版本 (AWS_LBC_HELM_VERSION)。您可以透過執行 `helm search repo eks/aws-load-balancer-controller --versions` 來尋找最新版本的 Helm Chart。如果您執行具有混合節點和 AWS 雲端中的節點的混合模式叢集，則您可以遵循 the section called “AWS Load Balancer控制器” 中的指示，在雲端節點上執行 AWS Load Balancer 控制器。

```
helm install aws-load-balancer-controller eks/aws-load-balancer-controller \
  -n kube-system \
  --version AWS_LBC_HELM_VERSION \
  --set clusterName=CLUSTER_NAME \
  --set region=AWS_REGION \
  --set vpcId=VPC_ID \
  --set serviceAccount.create=false \
  --set serviceAccount.name=aws-load-balancer-controller
```

7. 驗證 AWS Load Balancer 控制器是否已成功安裝。

```
kubectl get -n kube-system deployment aws-load-balancer-controller
```

```
NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
```

```
aws-load-balancer-controller   2/2     2            2           84s
```

8. 在名為 tcp-sample-app.yaml 的檔案中定義範例應用程式。以下範例使用具有 TCP 連接埠的簡單的 NGINX 部署。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: tcp-sample-app
  namespace: default
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: public.ecr.aws/nginx/nginx:1.23
          ports:
            - name: tcp
              containerPort: 80
```

9. 將部署套用至叢集。

```
kubectl apply -f tcp-sample-app.yaml
```

10.在名為 tcp-sample-service.yaml 的檔案中定義部署的 LoadBalancer 類型服務。

```
apiVersion: v1
kind: Service
metadata:
  name: tcp-sample-service
  namespace: default
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-type: external
    service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: ip
    service.beta.kubernetes.io/aws-load-balancer-scheme: internet-facing
spec:
```

```
  ports:
    - port: 80
      targetPort: 80
      protocol: TCP
  type: LoadBalancer
  selector:
    app: nginx
```

11將服務組態套用至叢集。

```
kubectl apply -f tcp-sample-service.yaml
```

12為服務佈建 NLB 可能需要幾分鐘的時間。佈建 NLB 後，服務將會為其指派一個與 NLB 部署的 DNS 名稱對應的地址。

```
kubectl get svc tcp-sample-service
```

```
NAME                TYPE           CLUSTER-IP       EXTERNAL-IP
                                                 PORT(S)        AGE
tcp-sample-service   LoadBalancer   172.16.115.212   k8s-default-tcpsampl-xxxxxxxxxx-
xxxxxxxxxxxxxxxx.elb.<region>.amazonaws.com   80:30396/TCP   8s
```

13.使用 NLB 的地址存取服務。

```
curl k8s-default-tcpsampl-xxxxxxxxxx-xxxxxxxxxxxxxxxx.elb.<region>.amazonaws.com
```

範例輸出如下。

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
[...]
```

14清除您建立的 資源。

```
kubectl delete -f tcp-sample-service.yaml
kubectl delete -f tcp-sample-app.yaml
```

Cilium 叢集內負載平衡

Cilium 可用作在 EKS 混合節點上執行的工作負載的叢集內負載平衡器，而這對於沒有負載平衡器基礎結構的環境非常有用。Cilium 的負載平衡功能以 Cilium 功能的組合為建置基礎，包括 kube-proxy 取代、Load Balancer IP 位址管理 (IPAM) 和 BGP 控制平面。這些功能的責任詳述如下：

- Cilium kube-proxy 取代：處理將服務流量路由到後端 Pod。
- Cilium Load Balancer IPAM：管理可指派給 `LoadBalancer` 類型的服務的 IP 位址。
- Cilium BGP 控制平面：將 Load Balancer IPAM 配置的 IP 位址公告至內部部署網路。

如果您沒有使用 Cilium 的 kube-proxy 取代，您仍然可以使用 Cilium Load Balancer IPAM 和 BGP 控制平面，以為 LoadBalancer 類型的服務配置和指派 IP 位址。如果您沒有使用 Cilium 的 kube-proxy 取代，則根據預設，EKS 中服務至後端 Pod 的負載平衡將由 kube-proxy 和 iptables 處理。

先決條件

- 在啟用或未啟用 kube-proxy 取代的情況下，遵循 the section called “設定 CNI” 中的指示安裝 Cilium。Cilium 的 kube-proxy 取代需要執行至少與 v4.19.57、v5.1.16 或 v5.2.0 相近的 Linux 核心作業系統。除了 Red Hat Enterprise Linux (RHEL) 8.x 外，所有支援與混合節點搭配使用的最新版作業系統皆符合此條件。
- 遵循 the section called “設定 BPA” 中的指示啟用 Cilium BGP 控制平面。如果您不想使用 BGP，則必須使用替代方法，以讓內部部署 Pod CIDR 在內部部署網路上可路由，如需詳細資訊，請參閱 the section called “可路由的遠端 Pod CIDR”。
- 已在命令列環境中安裝 Helm，請參閱安裝 Helm 說明。

程序

1. 使用 `CiliumLoadBalancerIPPool` 資源建立名為 `cilium-lbip-pool-loadbalancer.yaml` 的檔案，以為 Load Balancer 類型的服務設定 Load Balancer IP 位址範圍。
   - 使用用於 Load Balancer IP 位址的 IP 位址範圍取代 `LB_IP_CIDR`。若要選取單一 IP 位址，請使用 `/32` CIDR。如需詳細資訊，請參閱 Cilium 文件中的 LoadBalancer IP 位址管理。
   - `serviceSelector` 欄位設定為與您將在後續步驟中建立的服務名稱相符。使用此組態，此集區的 IP 將僅會配置給名稱為 `tcp-sample-service` 的服務。

```
apiVersion: cilium.io/v2alpha1
kind: CiliumLoadBalancerIPPool
metadata:
```

```
  name: tcp-service-pool
spec:
  blocks:
  - cidr: "LB_IP_CIDR"
  serviceSelector:
    matchLabels:
      io.kubernetes.service.name: tcp-sample-service
```

2. 將 CiliumLoadBalancerIPPool 資源套用至您的叢集。

```
kubectl apply -f cilium-lbip-pool-loadbalancer.yaml
```

3. 確認集區中至少有一個 IP 位址可用。

```
kubectl get ciliumloadbalancerippools.cilium.io
```

```
NAME               DISABLED   CONFLICTING   IPS AVAILABLE   AGE
tcp-service-pool   false      False         1               24m
```

4. 使用 CiliumBGPAdvertisement 資源建立名為 cilium-bgp-advertisement-loadbalancer.yaml 的檔案，以公告您將在下一步驟中建立的服務的負載平衡器 IP 位址。如果您不使用 Cilium BGP，則可以略過此步驟。用於服務的負載平衡器 IP 位址必須在內部部署網路上可路由，這樣您才能在最後一個步驟中查詢服務。
   - advertisementType 欄位設定為 Service，而 service.addresses 則設定為 LoadBalancerIP，以便僅為 LoadBalancer 類型的服務公告 LoadBalancerIP。
   - selector 欄位設定為與您將在後續步驟中建立的服務名稱相符。使用此組態時，將僅會公告名稱為 tcp-sample-service 的服務的 LoadBalancerIP。

```
apiVersion: cilium.io/v2alpha1
kind: CiliumBGPAdvertisement
metadata:
  name: bgp-advertisement-tcp-service
  labels:
    advertise: bgp
spec:
  advertisements:
    - advertisementType: "Service"
      service:
        addresses:
          - LoadBalancerIP
      selector:
```

```
        matchLabels:
          io.kubernetes.service.name: tcp-sample-service
```

5. 將 CiliumBGPAdvertisement 資源套用至您的叢集。如果您不使用 Cilium BGP，則可以略過此步驟。

```
kubectl apply -f cilium-bgp-advertisement-loadbalancer.yaml
```

6. 在名為 tcp-sample-app.yaml 的檔案中定義範例應用程式。以下範例使用具有 TCP 連接埠的簡單的 NGINX 部署。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: tcp-sample-app
  namespace: default
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: public.ecr.aws/nginx/nginx:1.23
          ports:
            - name: tcp
              containerPort: 80
```

7. 將部署套用至叢集。

```
kubectl apply -f tcp-sample-app.yaml
```

8. 在名為 tcp-sample-service.yaml 的檔案中定義部署的 LoadBalancer 類型服務。
   - 您可以從負載平衡器 IP 集區請求特定的 IP 位址，並在服務物件上加上 lbipam.cilium.io/ips 註釋。如果您不想為服務請求特定的 IP 位址，則可移除此註釋。
   - 需要 loadBalancerClass 規格欄位，才能防止舊版 AWS 雲端提供者為服務建立 Classic Load Balancer。在以下範例中，這會設定為 io.cilium/bgp-control-plane，以使用 Cilium 的

BGP 控制平面作為負載平衡器類別。或者，此欄位可設定為 `io.cilium/l2-announcer`，以使用 Cilium 的 L2 功能公告 (目前為 beta 版且未受 AWS 正式支援)。

```
apiVersion: v1
kind: Service
metadata:
  name: tcp-sample-service
  namespace: default
  annotations:
    lbipam.cilium.io/ips: "LB_IP_ADDRESS"
spec:
  loadBalancerClass: io.cilium/bgp-control-plane
  ports:
    - port: 80
      targetPort: 80
      protocol: TCP
  type: LoadBalancer
  selector:
    app: nginx
```

9. 將服務套用至叢集。服務將使用外部 IP 位址建立，而您可用其來存取應用程式。

```
kubectl apply -f tcp-sample-service.yaml
```

10確認服務已成功建立，並從上一個步驟中建立的 `CiliumLoadBalancerIPPool` 為其指派 IP。

```
kubectl get svc tcp-sample-service
```

```
NAME                TYPE           CLUSTER-IP      EXTERNAL-IP     PORT(S)
 AGE
tcp-sample-service   LoadBalancer   172.16.117.76   LB_IP_ADDRESS   80:31129/TCP
 14m
```

11如果您以 kube-proxy 取代模式使用 Cilium，您可以執行下列命令，以確認 Cilium 正在處理服務的負載平衡。在以下輸出中，`10.86.2.x` 位址是服務的後端 Pod 的 Pod IP 位址。

```
kubectl -n kube-system exec ds/cilium -- cilium-dbg service list
```

```
ID   Frontend              Service Type   Backend
...
41   LB_IP_ADDRESS:80/TCP   LoadBalancer   1 => 10.86.2.76:80/TCP (active)
```

```
                                                  2 => 10.86.2.130:80/TCP (active)
                                                  3 => 10.86.2.141:80/TCP (active)
```

12.確認 Cilium 正透過 BGP 將 IP 位址公告至內部部署網路。在以下範例中，有五個混合節點，並且每個節點都會向內部部署網路公告 tcp-sample-service 服務的 LB_IP_ADDRESS。

```
Node                    VRouter      Prefix             NextHop   Age     Attrs
mi-026d6a261e355fba7    NODES_ASN
                   LB_IP_ADDRESS/32   0.0.0.0   12m3s   [{Origin: i} {Nexthop:
 0.0.0.0}]
mi-082f73826a163626e    NODES_ASN
                   LB_IP_ADDRESS/32   0.0.0.0   12m3s   [{Origin: i} {Nexthop:
 0.0.0.0}]
mi-09183e8a3d755abf6    NODES_ASN
                   LB_IP_ADDRESS/32   0.0.0.0   12m3s   [{Origin: i} {Nexthop:
 0.0.0.0}]
mi-0d78d815980ed202d    NODES_ASN
                   LB_IP_ADDRESS/32   0.0.0.0   12m3s   [{Origin: i} {Nexthop:
 0.0.0.0}]
mi-0daa253999fe92daa    NODES_ASN
                   LB_IP_ADDRESS/32   0.0.0.0   12m3s   [{Origin: i} {Nexthop:
 0.0.0.0}]
```

13.使用指派的 load balancerIP 位址存取服務。

```
curl LB_IP_ADDRESS
```

範例輸出如下。

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
[...]
```

14.清除您建立的 資源。

```
kubectl delete -f tcp-sample-service.yaml
kubectl delete -f tcp-sample-app.yaml
kubectl delete -f cilium-lb-ip-pool.yaml
kubectl delete -f cilium-bgp-advertisement.yaml
```

設定混合節點的 Kubernetes 網路政策

AWS 使用 Cilium 做為 CNI 搭配 EKS 混合節點時，支援 Pod 輸入和輸出流量的 Kubernetes 網路政策（第 3 層/第 4 層）。如果您使用 AWS 雲端中的節點執行 EKS 叢集，則 AWS 支援 Amazon VPC CNI for Kubernetes 網路政策。

本主題涵蓋如何使用 EKS 混合節點設定 Cilium 和 Kubernetes 網路政策。如需 Kubernetes 網路政策的詳細資訊，請參閱 Kubernetes 文件中的 Kubernetes 網路政策。

設定網路政策

考量事項

- AWS 支援 Pod 輸入和輸出的上游 Kubernetes 網路政策和規格。 AWS 目前不支援 `CiliumNetworkPolicy`或 `CiliumClusterwideNetworkPolicy`。
- `policyEnforcementMode` Helm 值可用於控制預設的 Cilium 政策強制執行行為。預設行為允許所有輸出和輸入流量。當網路政策選取端點時，其會轉換為預設拒絕狀態，而該狀態僅允許明確允許的流量。如需預設政策模式和政策強制執行模式的詳細資訊，請參閱 Cilium 文件。
- 如果您要變更現有 Cilium 安裝的 `policyEnforcementMode`，則您必須重新啟動 Cilium 代理程式 DaemonSet，以套用新的政策強制執行模式。
- 使用 `namespaceSelector` 和 `podSelector` 允許或拒絕往返具有相符標籤的命名空間和 Pod 之間的流量。`namespaceSelector` 和 `podSelector` 可與 `matchLabels` 或 `matchExpressions` 搭配使用，以根據其標籤選取命名空間和 Pod。
- 使用 `ingress.ports` 和 `egress.ports` 允許或拒絕往返連接埠和通訊協定之間的流量。
- `ipBlock` 欄位無法用於選擇性地允許或拒絕往返 Pod IP 位址 (#9209) 的流量。針對節點 IPs 使用`ipBlock`選擇器是 Cilium 中的 Beta 功能，不受 支援 AWS。
- 如需 Kubernetes 網路政策可用欄位的資訊，請參閱 Kubernetes 文件中的 NetworkPolicy 資源。

先決條件

- 遵循 the section called “設定 CNI” 中的指示安裝 Cilium。
- 已在命令列環境中安裝 Helm，請參閱安裝 Helm 說明。

程序

下列程序會為範例微服務應用程式設定網路政策，以便元件只能與應用程式運作所需的其他元件通訊。程序會使用 Istio Bookinfo 範例微服務應用程式。

Bookinfo 應用程式由四個單獨的微服務組成，且其關係如下：

- productpage。productpage 微服務會呼叫詳細資訊並檢閱微服務來填入頁面。
- 詳細資訊。詳細資訊微服務包含書籍資訊。
- 檢閱。檢閱微服務包含書籍檢閱。其還會呼叫評分微服務。
- 評分。評分微服務包含隨附書籍檢閱的書籍排名資訊。

1. 建立範例應用程式。

```
kubectl apply -f https://raw.githubusercontent.com/istio/istio/refs/heads/master/
samples/bookinfo/platform/kube/bookinfo.yaml
```

2. 確認應用程式已成功執行，並記下 productpage 微服務的 Pod IP 位址。在後續步驟中，您將使用此 Pod IP 位址來查詢每個微服務。

```
kubectl get pods -o wide
```

```
NAME                              READY   STATUS    RESTARTS   AGE   IP
 NODE
details-v1-766844796b-9wff2       1/1     Running   0          7s    10.86.3.7
 mi-0daa253999fe92daa
productpage-v1-54bb874995-lwfgg   1/1     Running   0          7s    10.86.2.193
 mi-082f73826a163626e
ratings-v1-5dc79b6bcd-59njm       1/1     Running   0          7s    10.86.2.232
 mi-082f73826a163626e
reviews-v1-598b896c9d-p2289       1/1     Running   0          7s    10.86.2.47
 mi-026d6a261e355fba7
reviews-v2-556d6457d-djktc        1/1     Running   0          7s    10.86.3.58
 mi-0daa253999fe92daa
reviews-v3-564544b4d6-g8hh4       1/1     Running   0          7s    10.86.2.69
 mi-09183e8a3d755abf6
```

3. 建立將用於測試網路政策的 Pod。請注意，Pod 是在具有標籤 `access: true` 的 `default` 命名空間中建立的。

```
kubectl run curl-pod --image=curlimages/curl -i --tty --labels=access=true --
namespace=default --overrides='{"spec": { "nodeSelector": {"eks.amazonaws.com/
compute-type": "hybrid"}}}' -- /bin/sh
```

4. 測試對 productpage 微服務的存取。在以下範例中，我們使用 productpage Pod (`10.86.2.193`) 的 Pod IP 位址來查詢微服務。將此取代為您環境中 productpage Pod 的 Pod IP 位址。

```
curl -s http://10.86.2.193:9080/productpage | grep -o "<title>.*</title>"
```

```
<title>Simple Bookstore App</title>
```

5. 您可以輸入 `exit` 來結束測試 curl Pod，並執行下列命令來重新連接至 Pod。

```
kubectl attach curl-pod -c curl-pod -i -t
```

6. 為了在下列步驟中示範網路政策的效果，我們會先建立一個網路政策，其中該政策會拒絕 BookInfo 微服務的所有流量。建立稱為 `network-policy-deny-bookinfo.yaml` 的檔案，其中會定義拒絕網路政策。

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: deny-bookinfo
  namespace: default
spec:
  podSelector:
    matchExpressions:
    - key: app
      operator: In
      values: ["productpage", "details", "reviews", "ratings"]
  policyTypes:
  - Ingress
  - Egress
```

7. 將拒絕網路政策套用至您的叢集。

```
kubectl apply -f network-policy-default-deny-bookinfo.yaml
```

8. 測試對 BookInfo 應用程式的存取。在以下範例中，我們使用 productpage Pod (`10.86.2.193`) 的 Pod IP 位址來查詢微服務。將此取代為您環境中 productpage Pod 的 Pod IP 位址。

```
curl http://10.86.2.193:9080/productpage --max-time 10
```

```
curl: (28) Connection timed out after 10001 milliseconds
```

9. 建立稱為 network-policy-productpage.yaml 的檔案，其中會定義 productpage 網路政策。政策具有下列規則：
   - 允許來自具有標籤 access: true 的 Pod 的輸入流量 (在上一個步驟中建立的 curl Pod)
   - 允許連接埠 9080 上的輸出 TCP 流量，以取得詳細資訊、檢閱和評分微服務
   - 允許在 kube-system 命名空間中執行的 CoreDNS 連接埠 53 上的輸出 TCP/UDP 流量

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: productpage-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: productpage
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          access: "true"
  egress:
  - to:
    - podSelector:
        matchExpressions:
        - key: app
          operator: In
          values: ["details", "reviews", "ratings"]
    ports:
    - port: 9080
      protocol: TCP
  - to:
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: kube-system
      podSelector:
        matchLabels:
          k8s-app: kube-dns
    ports:
    - port: 53
```

```
      protocol: UDP
    - port: 53
      protocol: TCP
```

10將 productpage 網路政策套用至您的叢集。

```
kubectl apply -f network-policy-productpage.yaml
```

11連線至 curl Pod 並測試對 Bookinfo 應用程式的存取。現在允許存取 productpage 微服務，但其他微服務仍然被拒絕，因為它們仍須遵循拒絕網路政策。在以下範例中，我們使用 productpage Pod (10.86.2.193) 的 Pod IP 位址來查詢微服務。將此取代為您環境中 productpage Pod 的 Pod IP 位址。

```
kubectl attach curl-pod -c curl-pod -i -t
```

```
curl -s http://10.86.2.193:9080/productpage | grep -o "<title>.*</title>"
<title>Simple Bookstore App</title>
```

```
curl -s http://10.86.2.193:9080/api/v1/products/1
{"error": "Sorry, product details are currently unavailable for this book."}
```

```
curl -s http://10.86.2.193:9080/api/v1/products/1/reviews
{"error": "Sorry, product reviews are currently unavailable for this book."}
```

```
curl -s http://10.86.2.193:9080/api/v1/products/1/ratings
{"error": "Sorry, product ratings are currently unavailable for this book."}
```

12建立稱為 network-policy-details.yaml 的檔案，其中會定義詳細資訊網路政策。此政策僅允許來自 productpage 微服務的輸入流量。

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: details-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: details
```

```
  policyTypes:
  - Ingress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: productpage
```

13.建立稱為 network-policy-reviews.yaml 的檔案，其中會定義檢閱網路政策。此政策僅允許來自 productpage 微服務輸入流量，且僅允許至評分微服務和 CoreDNS 的輸出流量。

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: reviews-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: reviews
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: productpage
  egress:
  - to:
    - podSelector:
        matchLabels:
          app: ratings
  - to:
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: kube-system
      podSelector:
        matchLabels:
          k8s-app: kube-dns
    ports:
    - port: 53
      protocol: UDP
```

```
    - port: 53
      protocol: TCP
```

14建立稱為 network-policy-ratings.yaml 的檔案，其中會定義評分網路政策。此政策僅允許來自檢閱微服務的輸入流量。

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: ratings-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: ratings
  policyTypes:
  - Ingress
  ingress:
  - from:
    - podSelector:
        matchExpressions:
        - key: app
          operator: In
          values: ["productpage", "reviews"]
```

15將詳細資訊、檢閱和評分網路政策套用至您的叢集。

```
kubectl apply -f network-policy-details.yaml
kubectl apply -f network-policy-reviews.yaml
kubectl apply -f network-policy-ratings.yaml
```

16連線至 curl Pod 並測試對 Bookinfo 應用程式的存取。在以下範例中，我們使用 productpage Pod (10.86.2.193) 的 Pod IP 位址來查詢微服務。將此取代為您環境中 productpage Pod 的 Pod IP 位址。

```
kubectl attach curl-pod -c curl-pod -i -t
```

測試詳細資訊微服務。

```
curl -s http://10.86.2.193:9080/api/v1/products/1
```

```
{"id": 1, "author": "William Shakespeare", "year": 1595, "type": "paperback",
 "pages": 200, "publisher": "PublisherA", "language": "English", "ISBN-10":
 "1234567890", "ISBN-13": "123-1234567890"}
```

測試檢閱微服務。

```
curl -s http://10.86.2.193:9080/api/v1/products/1/reviews
```

```
{"id": "1", "podname": "reviews-v1-598b896c9d-p2289", "clustername": "null",
 "reviews": [{"reviewer": "Reviewer1", "text": "An extremely entertaining play
 by Shakespeare. The slapstick humour is refreshing!"}, {"reviewer": "Reviewer2",
 "text": "Absolutely fun and entertaining. The play lacks thematic depth when
 compared to other plays by Shakespeare."}]}
```

測試評分微服務。

```
curl -s http://10.86.2.193:9080/api/v1/products/1/ratings
```

```
{"id": 1, "ratings": {"Reviewer1": 5, "Reviewer2": 4}}
```

17清除您在本程序中建立的資源。

```
kubectl delete -f network-policy-deny-bookinfo.yaml
kubectl delete -f network-policy-productpage.yaml
kubectl delete -f network-policy-details.yaml
kubectl delete -f network-policy-reviews.yaml
kubectl delete -f network-policy-ratings.yaml
kubectl delete -f https://raw.githubusercontent.com/istio/istio/refs/heads/master/
samples/bookinfo/platform/kube/bookinfo.yaml
kubectl delete pod curl-pod
```

混合節點的概念

使用 Amazon EKS 混合節點，您可以將內部部署或邊緣環境中執行的實體或虛擬機器加入 AWS 雲端中執行的 Amazon EKS 叢集。這種方法可帶來許多好處，但同時也會為熟悉在單一網路環境中執行 Kubernetes 叢集的使用者引進新的聯網概念和架構。

以下章節將深入探討 EKS 混合節點的 Kubernetes 和聯網概念，並詳細說明流量如何流經混合式架構。這些章節會要求您熟悉基本的 Kubernetes 網路知識，例如 Pod、節點、服務、Kubernetes 控制平面、kubelet 和 kube-proxy 的概念。

我們建議您按照順序閱讀這些頁面，即從 the section called “聯網概念” 開始，接著是 the section called “Kubernetes 概念”，最後是 the section called “流量流程”。

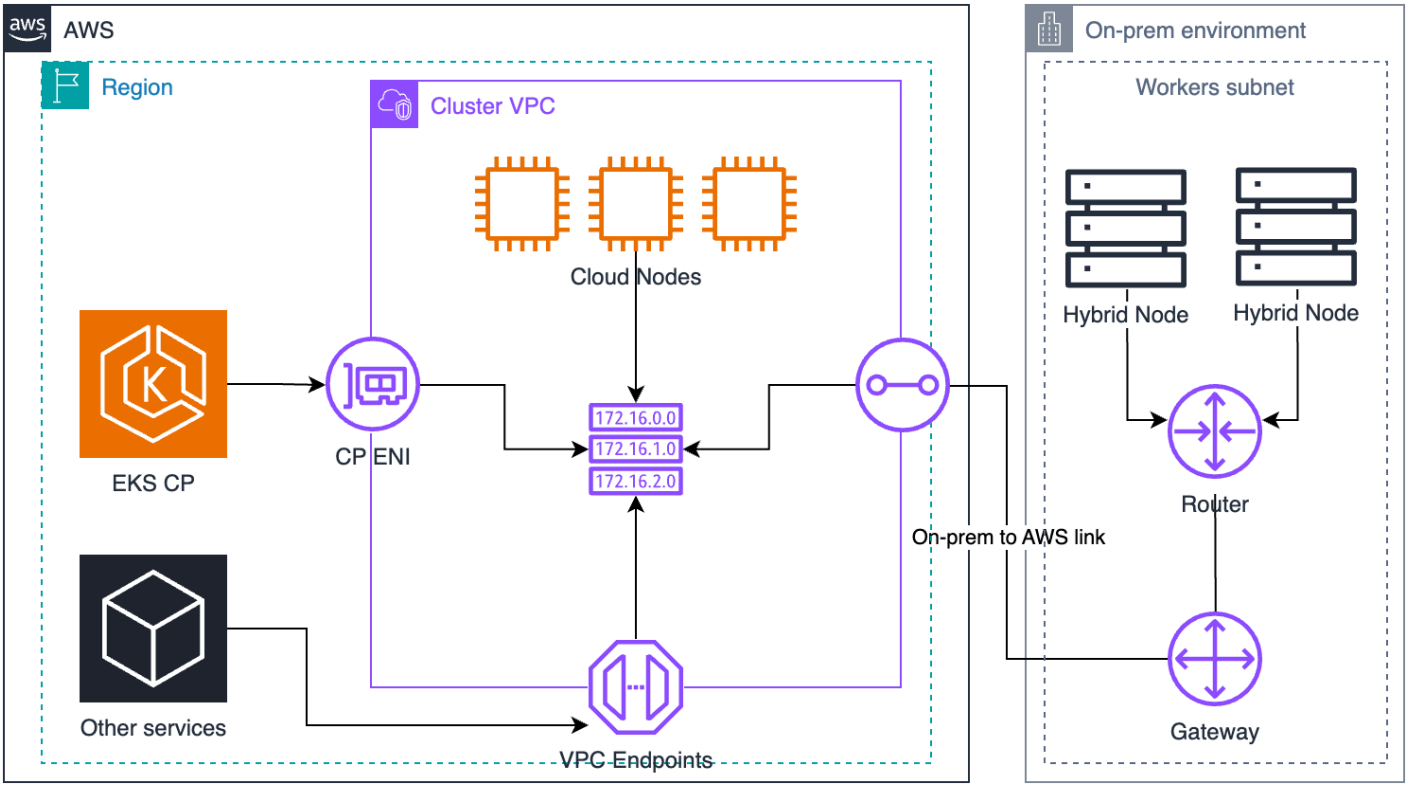
主題

- 混合節點的聯網概念
- 混合節點的 Kubernetes 概念
- 混合節點的網路流量流程

混合節點的聯網概念

本節詳細說明設計 EKS 混合節點的網路拓撲時，必須考量的核心聯網概念和限制條件。

EKS 混合節點的聯網概念



VPC 作為網路集線器

所有跨雲端邊界路由的流量都會通過您的 VPC。這包括在 中執行的 EKS 控制平面或 Pod AWS 與在其上執行的混合節點或 Pod 之間的流量。您可以將叢集的 VPC 視為混合節點與叢集其餘部分之間的網路集線器。此架構可讓您完全控制流量及其路由，但也可要求您須負責正確設定 VPC 的路由、安全群組和防火牆。

EKS 控制平面到 VPC

EKS 控制平面會將彈性網路介面 (ENI) 連接至您的 VPC。這些 ENI 會處理進出 EKS API 伺服器的流量。您可以在設定叢集時控制 EKS 控制平面 ENI 的置放，因為 EKS 會將 ENI 連接到您在叢集建立期間傳遞的子網路。

EKS 會將安全群組與 EKS 連接到子網路的 ENI 建立關聯。這些安全群組允許流量透過 ENI 進出 EKS 控制平面。這對 EKS 混合節點很重要，因為您必須允許流量從混合節點和在其上執行的 Pod 進入 EKS 控制平面 ENI。

遠端節點網路

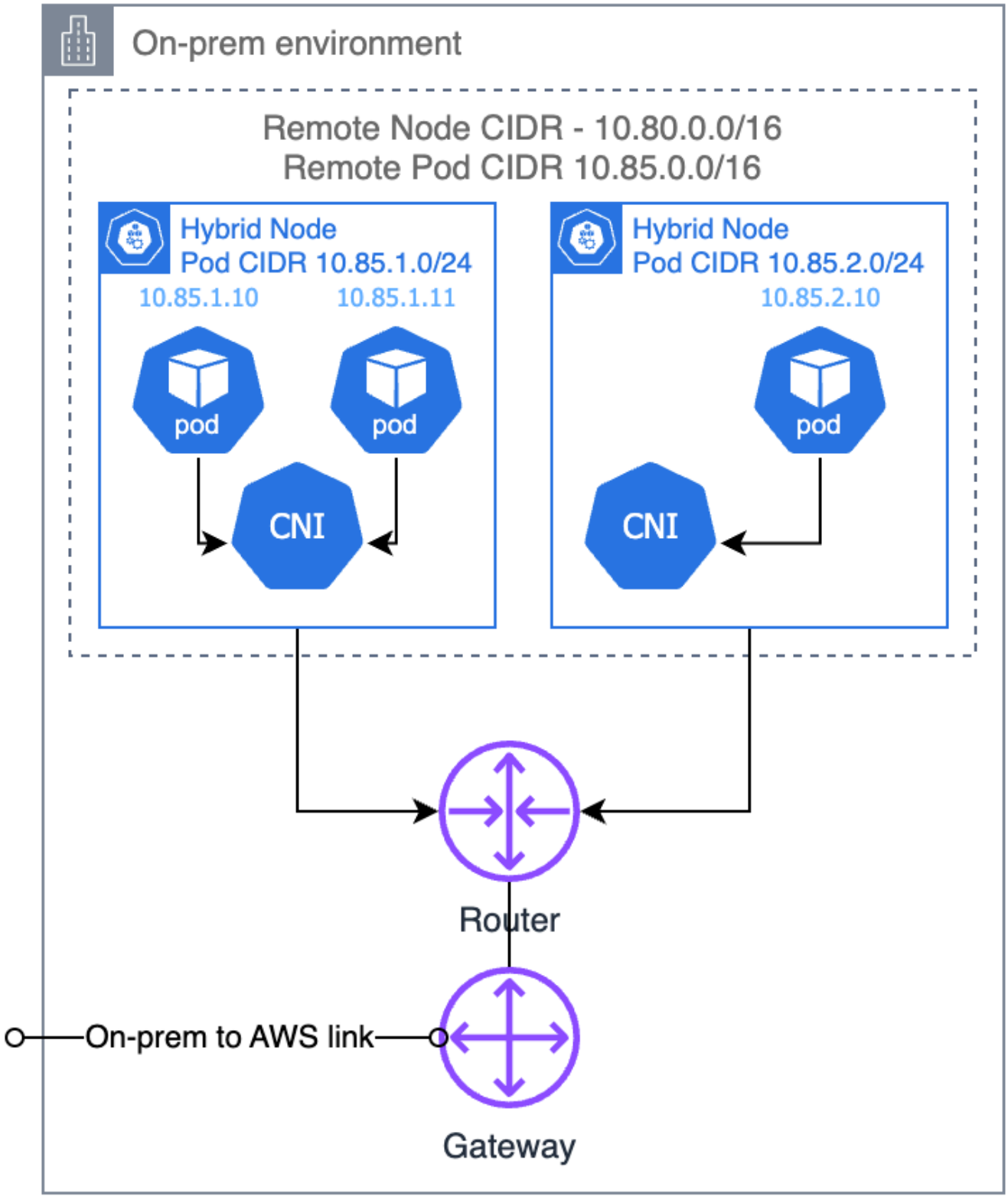
遠端節點網路，特別是遠端節點 CIDR，是指派給您用作混合節點的機器的 IP 範圍。當您佈建混合節點時，它們位於您的內部部署資料中心或邊緣節點，這是與 EKS 控制平面和 VPC 不同的網路網域。每個混合節點都有來自遠端節點 CIDR 的 IP 位址或地址，而該 CIDR 與您 VPC 中的子網路不同。

您可以使用這些遠端節點 CIDR 設定 EKS 叢集，以便 EKS 知道透過叢集 VPC 路由所有以混合節點 IP 為目標的流量，例如請求到 kubelet API。API 的連線`kubelet`用於 `kubectl attach`、`kubectl cp`、`kubectl logs`、 `kubectl exec`和 `kubectl port-forward`命令。

遠端 Pod 網路

遠端 Pod 網路是指派給在混合節點上執行的 Pod 的 IP 範圍。一般而言，您可以使用這些範圍設定 CNI，而 CNI 的 IP 位址管理 (IPAM) 功能會負責將這些範圍的配量指派給每個混合節點。當您建立 Pod 時，CNI 會從配置給已排程 Pod 的節點的配量，將 IP 指派給 Pod。

您可以使用這些遠端 Pod CIDR 設定 EKS 叢集，因此 EKS 控制平面知道透過叢集的 VPC 路由所有以混合節點上執行的 Pod 為目標的流量，例如與 Webhook 的通訊。


On-prem environment
Remote Node CIDR - 10.80.0.0/16
Remote Pod CIDR 10.85.0.0/16
Hybrid Node
Pod CIDR 10.85.1.0/24
10.85.1.10
10.85.1.11
pod
pod
CNI
Hybrid Node
Pod CIDR 10.85.2.0/24
10.85.2.10
pod
CNI
Router
On-prem to AWS link
Gateway

內部部署到 VPC

您用於混合節點的內部部署網路必須路由到您用於 EKS 叢集的 VPC。若要將內部部署網路連接到 VPC，有數個網路到 Amazon VPC 連線選項可用。您也可以使用自己的 VPN 解決方案。

請務必在 VPC 和內部部署網路的 AWS 雲端端正確設定路由，以便兩個網路透過兩個網路的連線路由正確的流量。

在 VPC 中，所有流向遠端節點和遠端 Pod 網路的流量都必須透過連線路由至內部部署網路 (稱為「閘道」)。如果您的部分子網路具有不同的路由表，則您必須使用混合節點的路由和在其上執行的 Pod 來設定每個路由表。對於連接 EKS 控制平面 ENI 的子網路，以及包含必須與混合節點通訊的 EC2 節點或 Pod 的子網路而言，都是如此。

在內部部署網路中，您必須設定網路，以允許進出 EKS 叢集 VPC 的流量，以及混合節點 AWS 所需的其他服務。EKS 叢集的流量可雙向周遊閘道。

聯網限制條件

完全路由的網路

主要限制條件是 EKS 控制平面和所有節點 (雲端或混合節點) 需要形成一個完全路由的網路。這表示所有節點都必須能夠透過 IP 位址在第三層彼此連接。

由於 EKS 控制平面和雲端節點位於平面網路 (VPC) 中，因此它們可以彼此連接。不過，混合節點會位於不同的網路網域中。因此，您需要在 VPC 和內部部署網路中設定額外的路由，以在混合節點和叢集的其餘部分之間路由流量。如果混合節點可從彼此和 VPC 連接，則您的混合節點可以位於單一平面網路或多個分段網路中。

可路由的遠端 Pod CIDR

若要讓 EKS 控制平面與在混合節點 (例如 Webhook 或指標伺服器) 上執行的 Pod 通訊，或讓在雲端節點上執行的 Pod 與在混合節點上執行的 Pod 通訊 (工作負載東西通訊)，您的遠端 Pod CIDR 必須可從 VPC 路由。這表示 VPC 必須能夠透過閘道將流量路由至內部部署網路的 Pod CIDR，並且您的內部部署網路必須能夠將 Pod 的流量路由至正確的節點。

請務必注意 VPC 和內部部署中 Pod 路由要求之間的差異。VPC 只需要知道任何流向遠端 Pod 的流量均應通過閘道。如果您只有一個遠端 Pod CIDR，則僅需一個路由。

此要求適用於您內部部署網路中的所有躍點，直到與您的混合節點位於相同子網路的本機路由器為止。這是唯一需要了解指派給每個節點的 Pod CIDR 配量的路由器，從而確保特定 Pod 的流量會交付至已排程 Pod 的節點。

您可以選擇將這些內部部署 Pod CIDR 的路由從本機內部部署路由器傳播到 VPC 路由表，但這並非必要。如果您的內部部署 Pod CIDR 經常變更，且您的 VPC 路由表需要更新以反映不斷變化的 Pod CIDR，我們建議您將內部部署 Pod CIDR 傳播至 VPC 路由表，不過這並不常見。

請注意，將內部部署 Pod CIDR 設為可路由的限制條件為選用項。如果您不需要在混合節點上執行 Webhook，或讓雲端節點上的 Pod 與混合節點上的 Pod 通訊，則您無需為內部部署網路上的 Pod CIDR 設定路由。

為什麼內部部署 Pod CIDR 需要使用混合節點進行路由？

將 EKS 與雲端節點的 VPC CNI 搭配使用時，VPC CNI 會將 IP 直接從 VPC 指派給 Pod。這表示不需要任何特殊路由，因為雲端 Pod 和 EKS 控制平面都可以直接連接 Pod IP。

當在內部部署執行 (以及與雲端中的其他 CNI 一起執行) 時，Pod 通常會在隔離的覆蓋網路中執行，而 CNI 則負責在 Pod 之間交付流量。這通常透過封裝來完成：CNI 會將 pod 至 pod 流量轉換為節點至節點流量，並負責兩端的封裝和解封裝。如此一來，就節點和路由器上就不需要額外的組態。

與混合節點的聯網是唯一的，因為它會提供兩種拓撲的組合：EKS 控制平面和雲端節點 (使用 VPC CNI) 預期包括節點和 Pod 的平坦網路，而混合節點上執行的 Pod 則位於覆蓋網路 (預設位於 Cilium) 中，可透過使用 VXLAN 進行封裝。在混合節點上執行的 Pod 可以連接在雲端節點上執行的 EKS 控制平面和 Pod，前提是內部部署網路可路由到 VPC。不過，如果沒有內部部署網路上的 Pod CIDR 的路由，而且網路不知道如何連接至覆蓋網路以及路由到正確的節點，則最終會捨棄任何返回內部部署 Pod IP 的流量。

混合節點的 Kubernetes 概念

此頁面詳細說明支援 EKS 混合節點系統架構的重要 Kubernetes 概念。

VPC 中的 EKS 控制平面

EKS 控制平面 ENI 的 IP 會存放在 `default` 命名空間的 `kubernetes` `Endpoints` 物件中。當 EKS 建立新的 ENI 或移除較舊的 ENI 時，EKS 會更新此物件，以便 IP 清單永遠是保持最新。

您可以透過 `kubernetes` 服務使用這些端點，也可以在 `default` 命名空間中使用這些端點。此服務為 `ClusterIP` 類型，且一律會獲指派叢集服務 CIDR 的第一個 IP。例如，對於服務 CIDR `172.16.0.0/16`，服務 IP 將為 `172.16.0.1`。

一般而言，這就是 Pod (無論是在雲端還是在混合節點中執行) 存取 EKS Kubernetes API 伺服器的方式。Pod 使用服務 IP 最為目的地 IP，且其會轉譯為其中一個 EKS 控制平面 ENI 的實際 IP。主要例外狀況是 `kube-proxy`，因為它會設定轉譯。

EKS API 伺服器端點

kubernetes 服務 IP 並非存取 EKS API 伺服器的唯一方式。當您建立叢集時，EKS 也會建立一個 Route53 DNS 名稱。這是呼叫 EKS DescribeCluster API 動作時 EKS 叢集的 endpoint 欄位。

```
{
    "cluster": {
        "endpoint": "https://xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.gr7.us-
west-2.eks.amazonaws.com",
        "name": "my-cluster",
        "status": "ACTIVE"
    }
}
```

在公有端點存取或公有和私有端點存取叢集中，根據預設，您的混合節點會將此 DNS 名稱解析為公有 IP，且可透過網際網路路由。在私有端點存取叢集中，DNS 名稱會解析為 EKS 控制平面 ENI 的私有 IP。

這是 kubelet 和 kube-proxy 存取 Kubernetes API 伺服器的方式。如果您希望所有 Kubernetes 叢集流量流經 VPC，您需要以私有存取模式設定叢集，或修改內部部署 DNS 伺服器，以將 EKS 叢集端點解析為 EKS 控制平面 ENI 的私有 IP。

kubelet 端點

kubelet 會公開數個 REST 端點，從而允許系統的其他部分與每個節點互動並收集資訊。在大多數叢集中，前往 kubelet 伺服器的大多數流量都來自控制平面，但某些監控代理程式也可能會與其互動。

透過此介面，kubelet 會處理各種請求：擷取日誌 (kubectl logs)、在容器內執行命令 (kubectl exec) 以及連接埠轉送流量 (kubectl port-forward)。其中每個請求皆會透過 kubelet 與基礎容器執行時期互動，而對叢集管理員和開發人員來說，這似乎是無縫的。

此 API 最常見的取用者是 Kubernetes API 伺服器。當您使用上述任何 kubectl 命令時，kubectl 會向 API 伺服器發出 API 請求，然後呼叫執行 Pod 的節點的 kubelet API。這就是需要從 EKS 控制平面連接至節點 IP，以及即使您的 Pod 正在執行，如果節點路由設定錯誤，您也無法存取其日誌或 exec 的主要原因。

節點 IP

當 EKS 控制平面與節點通訊時，其會使用以 Node 物件狀態 (status.addresses) 回報的其中一個地址。

使用 EKS 雲端節點時，kubelet 通常會在節點註冊期間將 EC2 執行個體的私有 IP 報告為 `InternalIP`。然後，雲端控制器管理員 (CCM) 會驗證此 IP，進而確保其屬於 EC2 執行個體。此外，CCM 通常會將執行個體的公有 IP (例如 `ExternalIP`) 和 DNS 名稱 (`InternalDNS` 和 `ExternalDNS`) 新增至節點狀態。

不過，混合節點沒有 CCM。當您向 EKS 混合節點 CLI (nodeadm) 註冊混合節點時，它會將 kubelet 設定為在節點狀態中直接報告機器的 IP，且不需要 CCM。

```
apiVersion: v1
kind: Node
metadata:
  name: my-node-1
spec:
  providerID: eks-hybrid:///us-west-2/my-cluster/my-node-1
status:
  addresses:
  - address: 10.1.1.236
    type: InternalIP
  - address: my-node-1
    type: Hostname
```

如果您的機器具有多個 IP，則 kubelet 會依照其自己的邏輯選取其中一個 IP。您可以使用 `--node-ip` 旗標來控制選取的 IP，而您可以在 `spec.kubelet.flags` 中的 nodeadm 組態內傳遞該旗標。只有 Node 物件中報告的 IP 需要來自 VPC 的路由。您的機器可能擁有無法從雲端連線的其他 IP。

kube-proxy

kube-proxy 負責在每個節點的網路層實作服務抽象。它可作為通往 Kubernetes Services 的流量的網路代理和負載平衡器。透過持續監看 Kubernetes API 伺服器中是否存在與服務和端點相關的變更，kube-proxy 會動態更新基礎主機的聯網規則，進而確保流量導向正確無誤。

在 `iptables` 模式中，kube-proxy 會設定數個 `netfilter` 鏈結來處理服務流量。這些規則可組成下列階層：

1. KUBE-SERVICES 鏈結：所有服務流量的進入點。它具有與每個服務的 `ClusterIP` 和連接埠相符的規則。
2. KUBE-SVC-XXX 鏈結：服務特定鏈結針對每個服務都設有負載平衡規則。
3. KUBE-SEP-XXX 鏈結：端點特定鏈結設有實際 DNAT 規則。

讓我們檢查 default 命名空間中服務 test-server 的情況：* 服務 ClusterIP：172.16.31.14 * 服務連接埠：80 * 支援 Pod：10.2.0.110、10.2.1.39 和 10.2.2.254

當我們檢查 iptables 規則時 (使用 iptables-save 0 grep -A10 KUBE-SERVICES)：

1. 在 KUBE-SERVICES 鏈結中，我們可找到與服務相符的規則：

```
-A KUBE-SERVICES -d 172.16.31.14/32 -p tcp -m comment --comment "default/test-server
 cluster IP" -m tcp --dport 80 -j KUBE-SVC-XYZABC123456
```

   - 此規則與目的地為 172.16.31.14:80 的封包相符
   - 該註解指出此規則的用途：default/test-server cluster IP
   - 相符的封包會跳至 KUBE-SVC-XYZABC123456 鏈結
2. KUBE-SVC-XYZABC123456 鏈結設有機率型負載平衡規則：

```
-A KUBE-SVC-XYZABC123456 -m statistic --mode random --probability 0.33333333349 -j
 KUBE-SEP-POD1XYZABC
-A KUBE-SVC-XYZABC123456 -m statistic --mode random --probability 0.50000000000 -j
 KUBE-SEP-POD2XYZABC
-A KUBE-SVC-XYZABC123456 -j KUBE-SEP-POD3XYZABC
```

   - 第一項規則：33.3% 的機會跳至 KUBE-SEP-POD1XYZABC
   - 第二項規則：50% 的剩餘流量 (總計的 33.3%) 會跳至 KUBE-SEP-POD2XYZABC
   - 最後一項規則：所有剩餘的流量 (總計的 33.3%) 會跳至 KUBE-SEP-POD3XYZABC
3. 個別 KUBE-SEP-XXX 鏈結會執行 DNAT (目的地 NAT)：

```
-A KUBE-SEP-POD1XYZABC -p tcp -m tcp -j DNAT --to-destination 10.2.0.110:80
-A KUBE-SEP-POD2XYZABC -p tcp -m tcp -j DNAT --to-destination 10.2.1.39:80
-A KUBE-SEP-POD3XYZABC -p tcp -m tcp -j DNAT --to-destination 10.2.2.254:80
```

   - 這些 DNAT 規則會重寫目的地 IP 和連接埠，以將流量導向特定的 Pod。
   - 每項規則可處理大約 33.3% 的流量，以便在 10.2.0.110、10.2.1.39 和 10.2.2.254 之間提供均勻的負載平衡。

此多層級鏈結結構讓 kube-proxy 能夠透過核心層級封包操作高效實作服務負載平衡和重新導向，而不需要資料路徑中的代理。

對 Kubernetes 操作的影響

節點上中斷的 kube-proxy 可防止該節點正常路由服務流量，進而導致依賴叢集服務的 Pod 逾時或連線失敗。首次註冊節點時，這尤其會造成干擾。在設定任何 Pod 聯網之前，CNI 需要與 Kubernetes API 伺服器通訊以取得資訊，例如節點的 Pod CIDR。為此，它會使用 kubernetes 服務 IP。不過，如果 kube-proxy 無法啟動或無法設定正確的 iptables 規則，則傳送至 kubernetes 服務 IP 的請求不會轉譯為 EKS 控制平面 ENI 的實際 IP。因此，CNI 將進入損毀循環，而且任何 Pod 都無法正常執行。

我們知道 Pod 會使用 kubernetes 服務 IP 與 Kubernetes API 伺服器通訊，但 kube-proxy 需要先設定 iptables 規則才能讓其運作。

kube-proxy 如何與 API 伺服器通訊？

必須將 kube-proxy 設定為使用 Kubernetes API 伺服器的實際 IP 或解析為它們的 DNS 名稱。若為 EKS，則 EKS 會將預設的 kube-proxy 設定為指向 EKS 在您建立叢集時建立的 Route53 DNS 名稱。您可以在 kube-system 命名空間的 kube-proxy ConfigMap 中看到此值。此 ConfigMap 的內容是注入 kube-proxy Pod 中的 kubeconfig，因此請尋找 clusters0.cluster.server 欄位。此值將與您 EKS 叢集的 endpoint 欄位相符 (呼叫 EKS DescribeCluster API 時)。

```
apiVersion: v1
data:
  kubeconfig: |-
    kind: Config
    apiVersion: v1
    clusters:
    - cluster:
        certificate-authority: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
        server: https://xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.gr7.us-
west-2.eks.amazonaws.com
      name: default
    contexts:
    - context:
        cluster: default
        namespace: default
        user: default
      name: default
    current-context: default
    users:
    - name: default
      user:
        tokenFile: /var/run/secrets/kubernetes.io/serviceaccount/token
```

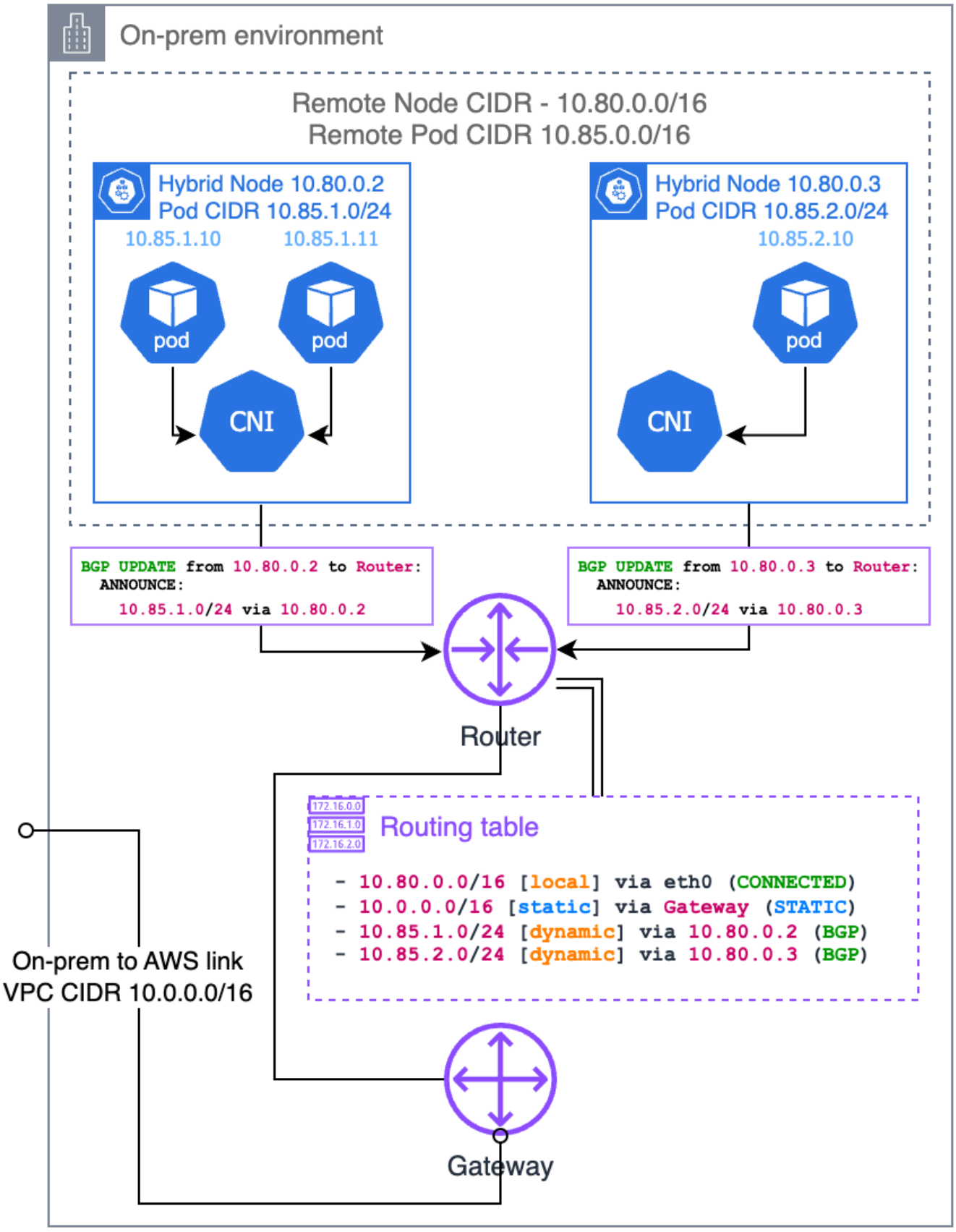
```
kind: ConfigMap
metadata:
  name: kube-proxy
  namespace: kube-system
```

可路由的遠端 Pod CIDR

此 the section called “聯網概念” 頁面會詳細說明在混合節點上執行 Webhook 或讓在雲端節點上執行的 Pod 與在混合節點上執行的 Pod 進行通訊的需求。關鍵要求是，內部部署路由器需要知道哪個節點負責特定的 Pod IP。有幾種方法可以實現這一點，包括邊界閘道協定 (BGP)、靜態路由和位址解析通訊協定 (ARP) 代理。詳細方法請參閱下列各節。

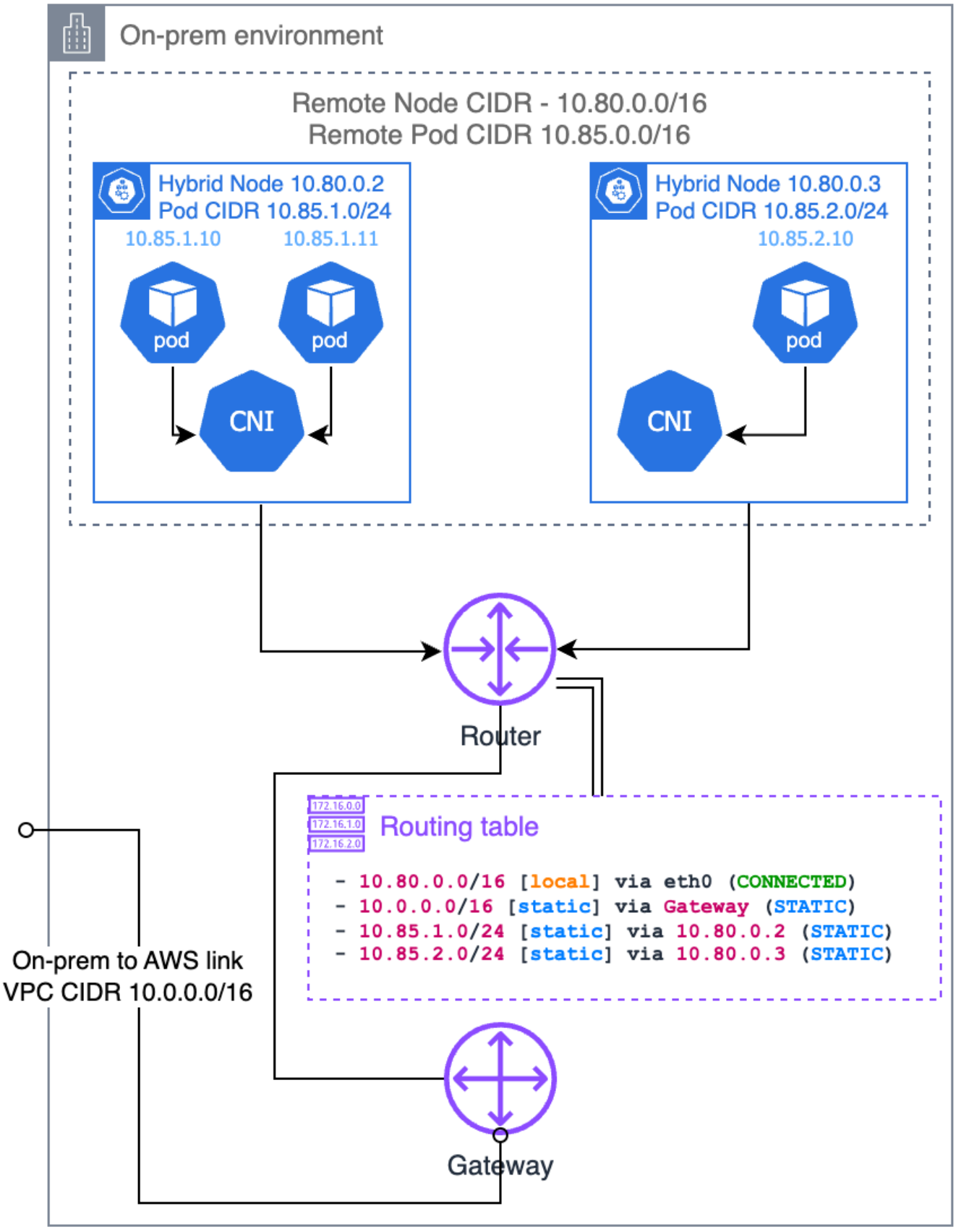
邊界閘道協定 (BGP)

如果您的 CNI 對其提供支援 (例如 Cilium 和 Calico)，您可以使用 CNI 的 BGP 模式，以將路由傳播到每個節點的 Pod CIDR (從節點到本機路由器)。使用 CNI 的 BGP 模式時，您的 CNI 可作為虛擬路由器，因此您的本機路由器會認為 Pod CIDR 屬於不同的子網路，而您的節點是該子網路的閘道。


On-prem environment
Remote Node CIDR - 10.80.0.0/16
Remote Pod CIDR 10.85.0.0/16
Hybrid Node 10.80.0.2
Pod CIDR 10.85.1.0/24
10.85.1.10
10.85.1.11
pod
pod
CNI
Hybrid Node 10.80.0.3
Pod CIDR 10.85.2.0/24
10.85.2.10
pod
CNI
BGP UPDATE from 10.80.0.2 to Router:
ANNOUNCE:
10.85.1.0/24 via 10.80.0.2
BGP UPDATE from 10.80.0.3 to Router:
ANNOUNCE:
10.85.2.0/24 via 10.80.0.3
Router
Routing table
- 10.80.0.0/16 [local] via eth0 (CONNECTED)
- 10.0.0.0/16 [static] via Gateway (STATIC)
- 10.85.1.0/24 [dynamic] via 10.80.0.2 (BGP)
- 10.85.2.0/24 [dynamic] via 10.80.0.3 (BGP)
On-prem to AWS link
VPC CIDR 10.0.0.0/16
Gateway

靜態路由

或者，您可以在本機路由器中設定靜態路由。這是將內部部署 Pod CIDR 路由至 VPC 的最簡單方法，但這也是最容易出錯且難以維護的方法。您需要確保路由始終與現有節點及其指派的 Pod CIDR 保持同步。如果您的節點數量很小且基礎結構為靜態，則此方案可行，並且不需要路由器中的 BGP 支援。如果您選擇此方案，那麼建議您使用要指派給每個節點的 Pod CIDR 配量來設定 CNI，而不是讓其 IPAM 決定。


On-prem environment
Remote Node CIDR - 10.80.0.0/16
Remote Pod CIDR 10.85.0.0/16
Hybrid Node 10.80.0.2
Pod CIDR 10.85.1.0/24
10.85.1.10
10.85.1.11
pod
pod
CNI
Hybrid Node 10.80.0.3
Pod CIDR 10.85.2.0/24
10.85.2.10
pod
CNI
Router
172.16.0.0
172.16.1.0
172.16.2.0
Routing table
- 10.80.0.0/16 [local] via eth0 (CONNECTED)
- 10.0.0.0/16 [static] via Gateway (STATIC)
- 10.85.1.0/24 [static] via 10.80.0.2 (STATIC)
- 10.85.2.0/24 [static] via 10.80.0.3 (STATIC)
On-prem to AWS link
VPC CIDR 10.0.0.0/16
Gateway

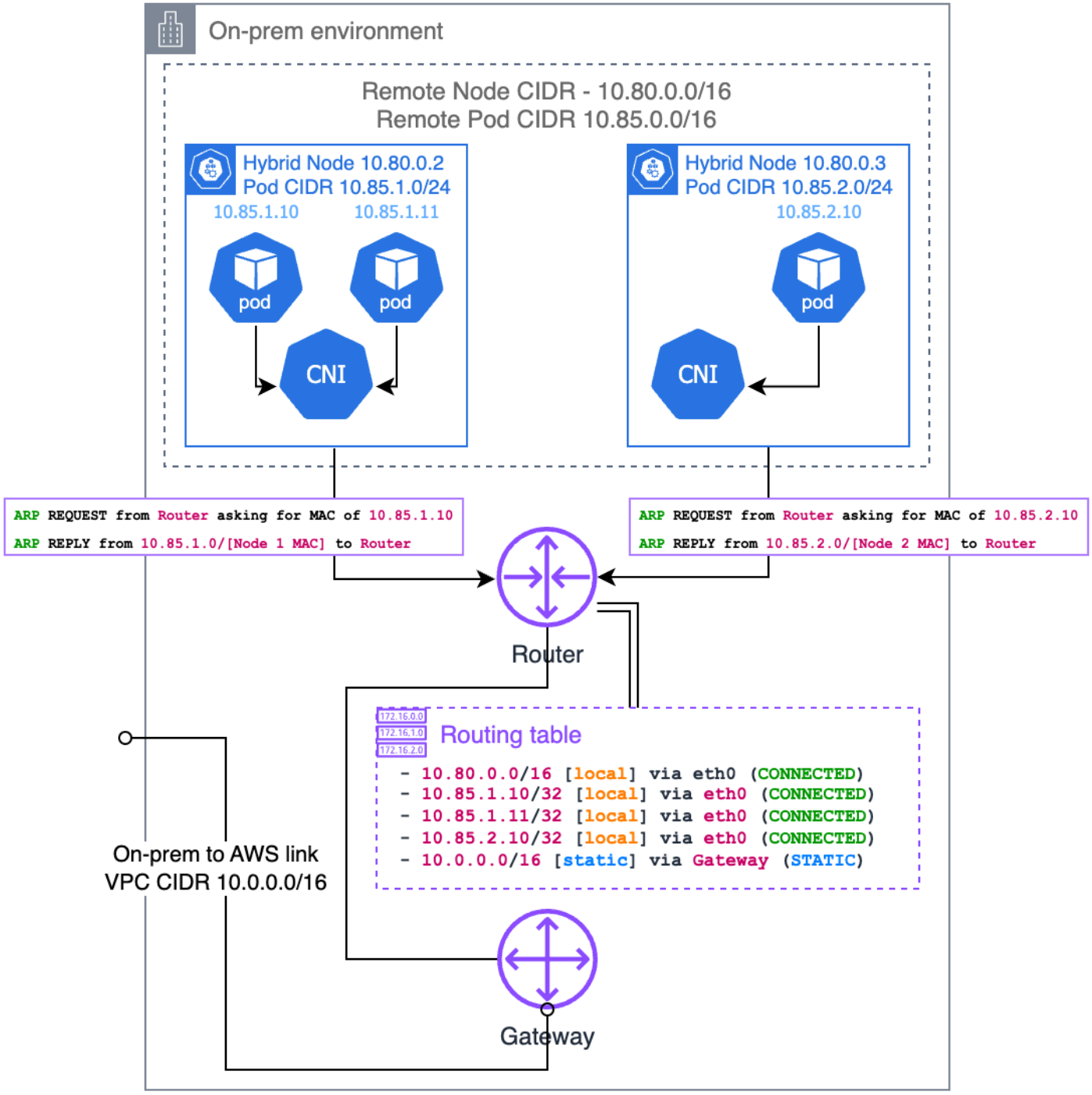
位址解析通訊協定 (ARP) 代理

ARP 代理是讓內部部署 Pod IP 可路由的另一種方法，並且當您的混合節點與本機路由器位於相同的第 2 層網路上時，特別有用。啟用 ARP 代理後，節點會回應其託管的 Pod IP 的 ARP 請求，即使這些 IP 屬於不同的子網路。

當您本機網路上的裝置嘗試連接 Pod IP 時，會先傳送 ARP 請求，並詢問「誰有此 IP？」。託管該 Pod 的混合節點將以其自己的 MAC 位址回應，表示「我可以處理該 IP 的流量」。這會在本機網路上的裝置與 Pod 之間建立一條直接路徑，而無需路由器組態。

若要使其運作，您的 CNI 必須支援代理 ARP 功能。Cilium 內建對代理 ARP 的支援，並且您可以透過組態將其啟用。關鍵考量是，Pod CIDR 不得與您環境中的任何其他網路重疊，因為這可能會導致路由衝突。

此方法有多個優點：* 無需使用 BGP 設定路由器或維護靜態路由 * 適用於您無法控制路由器組態的環境



Pod 至 Pod 封裝

在內部部署環境中，CNI 通常會使用封裝通訊協定來建立可在實體網路上運行的覆蓋網路，而無需重新設定。本節說明此封裝如何運作。請注意，某些詳細資訊可能會因您使用的 CNI 而有所不同。

封裝會將原始 Pod 網路封包包裝在另一個可透過基礎實體網路路由的網路封包內。這可讓 Pod 跨執行相同 CNI 的節點進行通訊，而不需要實體網路知道如何路由這些 Pod CIDR。

與 Kubernetes 搭配使用的最常見封裝通訊協定是虛擬局域網擴展 (VXLAN)，但視您的 CNI 而定，也可使用其他通訊協定 (例如 Geneve)。

VXLAN 封裝

VXLAN 將第 2 層乙太網路訊框封裝在 UDP 封包內。當 Pod 將流量傳送至不同節點上的另一個 Pod 時，CNI 會執行下列動作：

1. CNI 攔截來自 Pod A 的封包
2. CNI 將原始封包包裝在 VXLAN 標頭中
3. 然後，此已包裝的封包會透過節點的一般網路堆疊傳送至目的地節點
4. 目的地節點上的 CNI 會取消包裝封包並將其交付至 Pod B

以下是 VXLAN 封裝期間封包結構的相關情況：

原始 Pod 至 Pod 封包：

```
+-----------------+---------------+-------------+-----------------+
| Ethernet Header | IP Header     | TCP/UDP     | Payload         |
| Src: Pod A MAC  | Src: Pod A IP | Src Port    |                 |
| Dst: Pod B MAC  | Dst: Pod B IP | Dst Port    |                 |
+-----------------+---------------+-------------+-----------------+
```

VXLAN 封裝後：

```
+-----------------+-------------+--------------+------------
+---------------------------+
| Outer Ethernet  | Outer IP    | Outer UDP    | VXLAN      | Original Pod-to-Pod
 |
| Src: Node A MAC | Src: Node A | Src: Random  | VNI: xx    | Packet (unchanged
 |
| Dst: Node B MAC | Dst: Node B | Dst: 4789    |            | from above)
 |
+-----------------+-------------+--------------+------------
+---------------------------+
```

VXLAN 網路識別符 (VNI) 可區分不同的覆蓋網路。

Pod 通訊案例

相同混合節點上的 Pod

當相同混合節點上的 Pod 彼此通訊時，通常不需要封裝。CNI 會設定本機路由，而該路由可透過節點的內部虛擬介面來引導 Pod 之間的流量：

```
Pod A -> veth0 -> node's bridge/routing table -> veth1 -> Pod B
```

封包永遠不會離開節點，並且也不需要封裝。

不同混合節點上的 Pod

不同混合節點上的 Pod 之間的通訊需要封裝：

```
Pod A -> CNI -> [VXLAN encapsulation] -> Node A network -> router or gateway -> Node B
 network -> [VXLAN decapsulation] -> CNI -> Pod B
```

這可讓 Pod 流量周遊實體網路基礎結構，而無需實體網路來了解 Pod IP 路由。

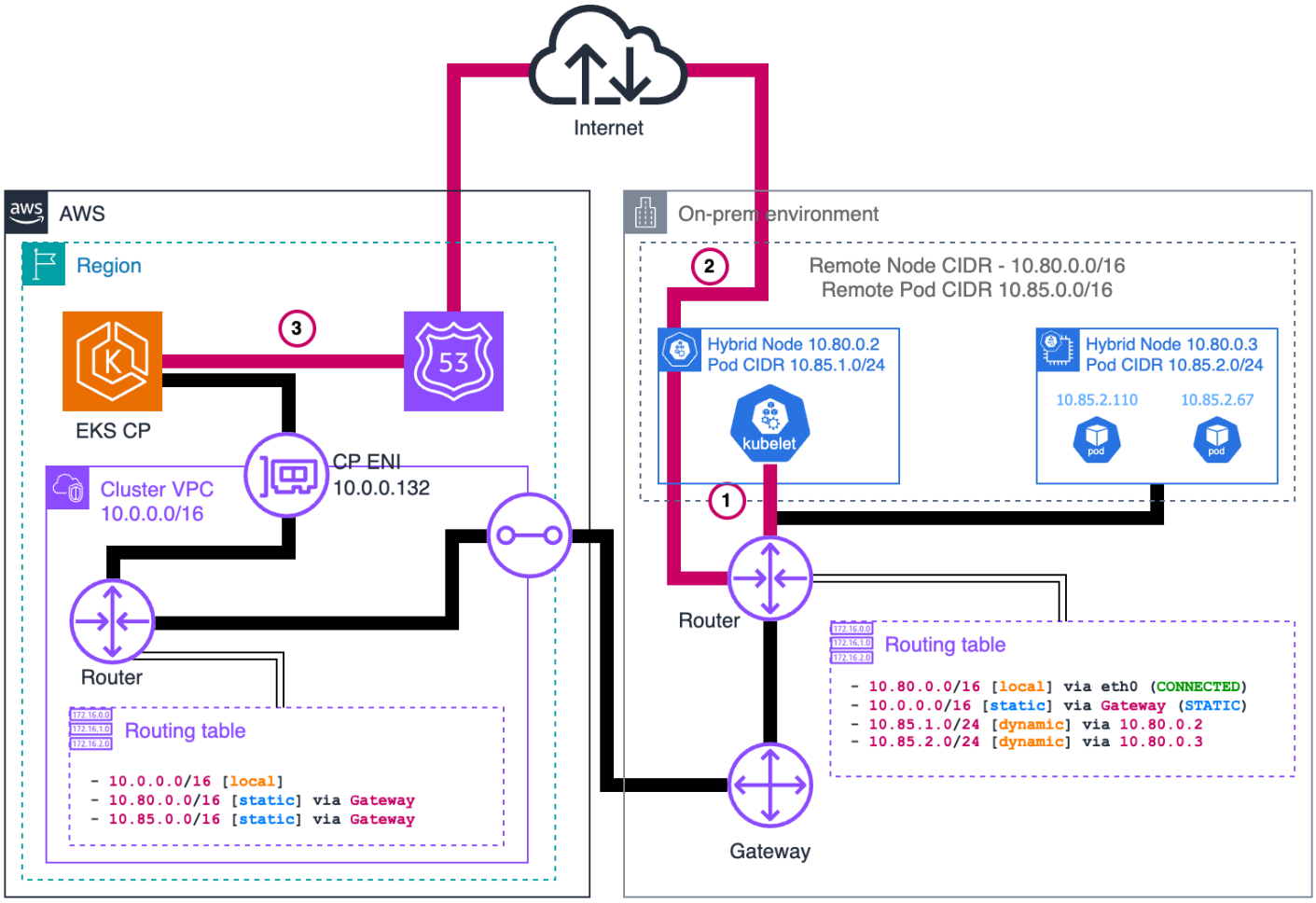
混合節點的網路流量流程

此頁面詳細說明 EKS 混合節點的網路流量流程，其中各圖表會顯示不同流量類型的端到端網路路徑。

涵蓋下列流量流程：

- the section called “混合節點 kubelet 到 EKS 控制平面”
- the section called “EKS 控制平面到混合節點 (kubelet 伺服器)”
- the section called “在混合節點上執行的 Pod 到 EKS 控制平面”
- the section called “EKS 控制平面到混合節點上執行的 Pod (Webhook)”
- the section called “在混合節點上執行的 Pod 到 Pod”
- the section called “雲端節點上的 Pod 到混合節點上的 Pod (東西流量)”

混合節點 **kubelet** 到 EKS 控制平面



請求

1**kubelet**. 啟動請求

當混合節點上的 kubelet 需要與 EKS 控制平面進行通訊時 (例如，報告節點狀態或取得 Pod 規格)，它會使用節點註冊期間提供的 kubeconfig 檔案。該 kubeconfig 具有 API 伺服器端點 URL (Route53 DNS 名稱)，而不是直接 IP 位址。

kubelet 會執行端點的 DNS 查詢 (例如，https://xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.gr7.us-west-2.eks.amazonaws.com)。在公有存取叢集中，這會解析為屬於在 AWS中執行的 EKS 服務的公有 IP 位址 (例如 54.239.118.52)。然後，kubelet 會為此端點建立安全的 HTTPS 請求。初始封包如下所示：

```
+--------------------+---------------------+-----------------+
| IP Header          | TCP Header          | Payload         |
```

```
| Src: 10.80.0.2     | Src: 52390 (random) |                 |
| Dst: 54.239.118.52 | Dst: 443            |                 |
+--------------------+---------------------+-----------------+
```

2. 本機路由器路由

由於目的地 IP 是公有 IP 位址並且不屬於本機網路，因此 kubelet 會將此封包傳送至其預設閘道 (本機內部部署路由器)。路由器會檢查目的地 IP，並判定其是否為公有 IP 位址。

對於公有流量，路由器通常會將封包轉送至網際網路閘道或邊界路由器，而它們可處理向網際網路的傳出流量。此內容未在圖表中顯示，且具體取決於內部部署網路的設定方式。封包會在您的內部部署網路基礎結構上傳輸，最終連接網際網路服務提供商的網路。

3. 交付至 EKS 控制平面

封包會通過公有網際網路和傳輸網路，直到到達 AWS網路。 AWS的網路會將封包路由到適當區域中的 EKS 服務端點。當該封包連接 EKS 服務時，它會轉送到叢集的實際 EKS 控制平面。

透過公有網際網路的路由不同於我們在其他流量流程中看到的私有 VPC 路由路徑。主要差異在於，使用公有存取模式時，從內部部署 kubelet (儘管並非從 Pod) 到 EKS 控制平面的流量不會經過您的 VPC，而是會使用全球網際網路基礎結構。

回應

在 EKS 控制平面處理 kubelet 請求後，它會傳送回應：

3. EKS 控制平面傳送回應

EKS 控制平面會建立回應封包。此封包將公有 IP 作為來源，並將混合節點的 IP 作為目的地：

```
+--------------------+---------------------+-----------------+
| IP Header          | TCP Header          | Payload         |
| Src: 54.239.118.52 | Src: 443            |                 |
| Dst: 10.80.0.2     | Dst: 52390          |                 |
+--------------------+---------------------+-----------------+
```

2. 網際網路路由

回應封包會透過網際網路傳回，並遵循網際網路服務提供商確定的路由路徑，直到連接您的內部部署網路邊緣路由器為止。

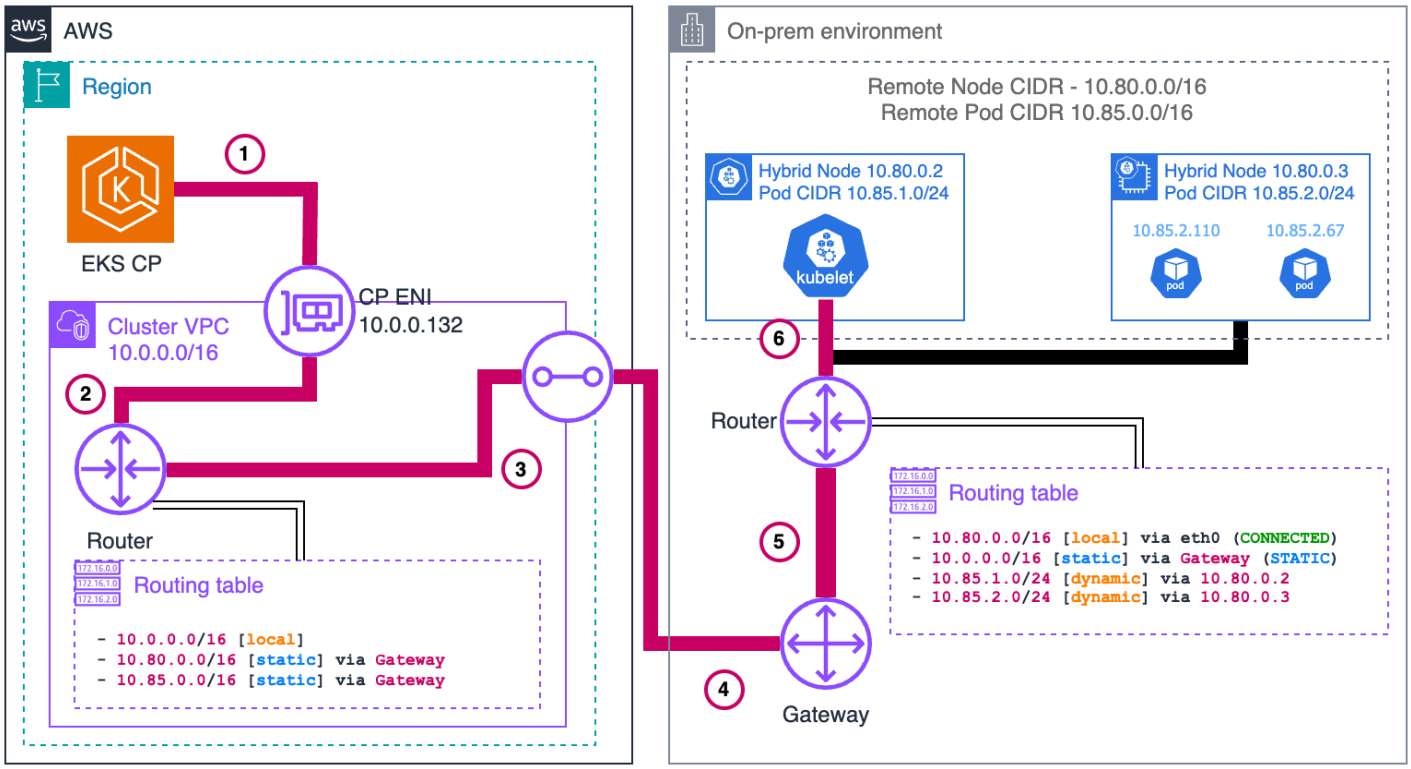
1。本機交付

您的內部部署路由器會接收封包，並將目的地 IP (`10.80.0.2`) 識別為屬於您的本機網路。它會透過您的本機網路基礎結構轉送封包，直到它連接目標混合節點為止，其中 `kubelet` 會接收和處理回應。

混合節點 **`kube-proxy`** 到 EKS 控制平面

如果您啟用叢集的公有端點存取，則傳回流量會使用公有網際網路。此流量源自混合節點`kube-proxy`上的 到 EKS 控制平面，並遵循與從 `kubelet`到 EKS 控制平面的流量相同的路徑。

EKS 控制平面到混合節點 (**`kubelet`** 伺服器)



請求

1。EKS Kubernetes API 伺服器啟動請求

EKS Kubernetes API 伺服器會從節點物件的狀態擷取節點的 IP 位址 (`10.80.0.2`)。然後，它會透過 VPC 中的 ENI 路由此請求，因為目的地 IP 屬於已設定的遠端節點 CIDR (`10.80.0.0/16`)。初始封包如下所示：

```
+-----------------+---------------------+-----------------+
| IP Header       | TCP Header          | Payload         |
```

```
| Src: 10.0.0.132 | Src: 67493 (random) |                 |
| Dst: 10.80.0.2  | Dst: 10250          |                 |
+-----------------+---------------------+-----------------+
```

2. VPC 網路處理

該封包會離開 ENI 並進入 VPC 網路層，而其會導向子網路的閘道，以便進行進一步路由。

3. VPC 路由表查詢

包含 EKS 控制平面 ENI 的子網路的 VPC 路由表具有適用於遠端節點 CIDR 的特定路由 (圖表中的第二個路由)。根據此路由規則，該封包會導向至 VPC-to-onprem 閘道。

4. 跨邊界傳輸

閘道會透過您已建立的連線 (例如 Direct Connect 或 VPN)，將封包跨雲端邊界傳輸到您的內部部署網路。

5. 內部部署網路接收

該封包會抵達本機內部部署路由器，以處理混合節點所在的子網路的流量。

6. 最終交付

本機路由器會識別目的地 IP (`10.80.0.2`) 位址屬於其直接連接的網路，並將封包直接轉送至目標混合節點，而 `kubelet` 會在其中接收和處理請求。

回應

在混合節點的 `kubelet` 處理請求後，它會遵循相同路徑反向傳回回應：

```
+-----------------+---------------------+-----------------+
| IP Header       | TCP Header          | Payload         |
| Src: 10.80.0.2  | Src: 10250          |                 |
| Dst: 10.0.0.132 | Dst: 67493          |                 |
+-----------------+---------------------+-----------------+
```

6. **`kubelet`** 傳送回應

混合節點 (`10.80.0.2`) 上的 `kubelet` 會使用原始來源 IP 作為目的地來建立回應封包。目的地不屬於本機網路，因此會傳送到主機的預設閘道，也就是本機路由器。

5. 本機路由器路由

該路由器會判定目的地 IP (`10.0.0.132`) 屬於 `10.0.0.0/16`，並且其具有會指向連接到 AWS的閘道的路由。

4. 跨邊界傳回

該封包會傳回相同的內部部署至 VPC 連線 (例如 Direct Connect 或 VPN)，並以相反方向跨越雲端邊界。

3. VPC 路由

當該封包抵達 VPC 時，路由表會識別目的地 IP 屬於 VPC CIDR。該封包會在 VPC 內路由。

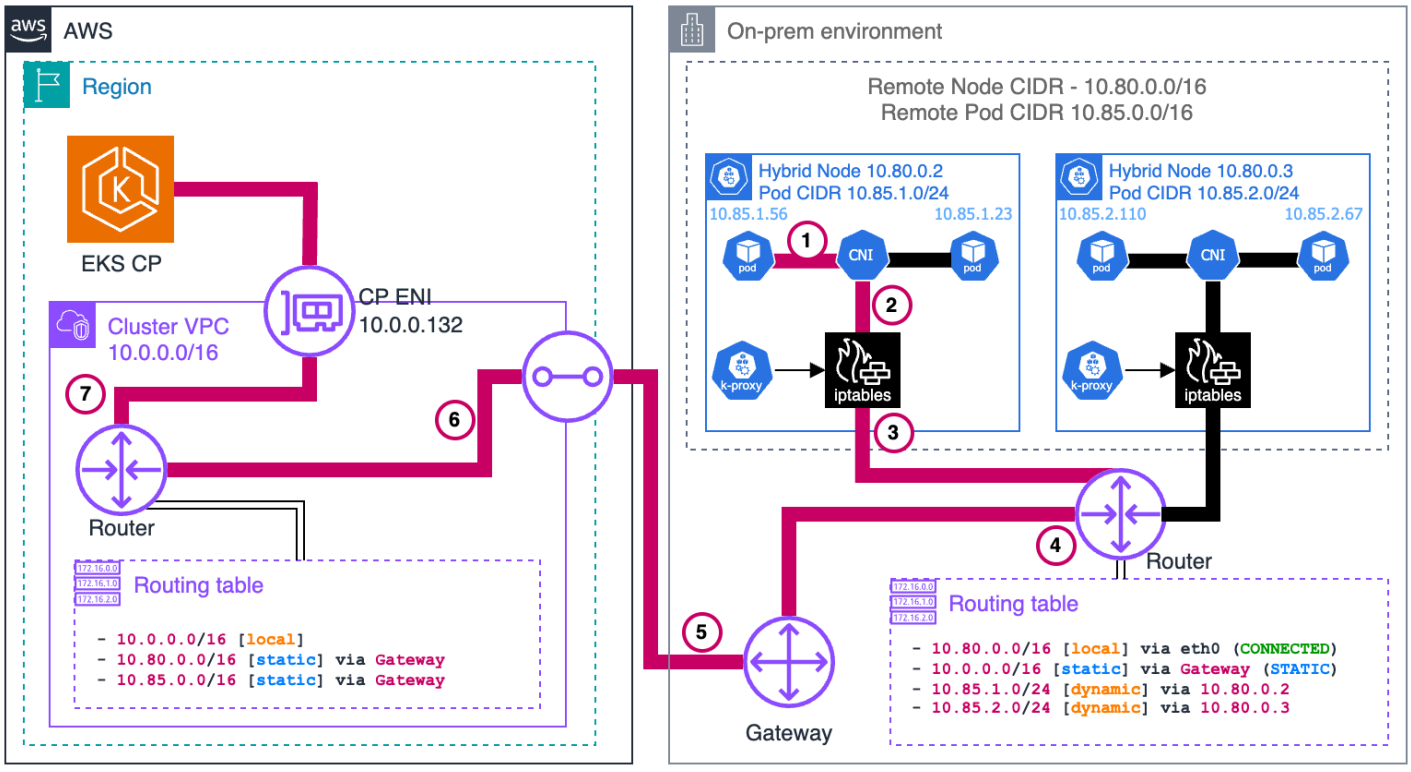
2. VPC 網路交付

VPC 網路層會使用 EKS 控制平面 ENI (`10.0.0.132`) 將封包轉送至子網路。

1。ENI 接收

該封包會到達連接到 Kubernetes API 伺服器的 EKS 控制平面 ENI，以便完成往返。

在混合節點上執行的 Pod 到 EKS 控制平面



不使用 CNI NAT

請求

Pod 通常會透過 kubernetes 服務與 Kubernetes API 伺服器進行通訊。服務 IP 是叢集的服務 CIDR 的第一個 IP。此慣例允許需要在 CoreDNS 可用之前執行的 Pod，以便連接 API 伺服器，例如 CNI。請求以服務 IP 作為目的地離開 Pod。例如，如果服務 CIDR 為 172.16.0.0/16，則服務 IP 將為 172.16.0.1。

1。Pod 啟動請求

Pod 會從隨機來源連接埠傳送請求至 API 伺服器連接埠 (443) 上的 kubernetes 服務 IP (172.16.0.1)。封包如下所示：

```
+-----------------+---------------------+-----------------+
| IP Header       | TCP Header          | Payload         |
| Src: 10.85.1.56 | Src: 67493 (random) |                 |
| Dst: 172.16.0.1 | Dst: 443            |                 |
+-----------------+---------------------+-----------------+
```

2. CNI 處理

CNI 偵測到目的地 IP 不屬於其管理的任何 Pod CIDR。由於傳出 NAT 已停用，CNI 會將該封包傳遞至主機網路堆疊，而且不會進行修改。

3. 節點網路處理

該封包會進入節點的網路堆疊，其中 netfilter 勾點會觸發由 kube-proxy 設定的 iptables 規則。按下列順序依次適用數條規則：

1. 該封包會先命中 KUBE-SERVICES 鏈結，其中包含與每個服務的 ClusterIP 和連接埠相符的規則。
2. 該相符規則會跳至 kubernetes 服務的 KUBE-SVC-XXX 鏈結 (以 172.16.0.1:443 為目的地的封包)，其中包含負載平衡規則。
3. 該負載平衡規則會隨機選取控制平面 ENI IP (10.0.0.132 或 10.0.1.23) 的其中一個 KUBE-SEP-XXX 鏈結。
4. 選取的 KUBE-SEP-XXX 鏈結具有實際規則，可將目的地 IP 從服務 IP 變更為選取的 IP。這就是所謂的目的地網路位址轉譯 (DNAT)。

套用這些規則後，假設選取的 EKS 控制平面 ENI 的 IP 為 10.0.0.132，且封包會如下所示：

```
+-----------------+---------------------+-----------------+
| IP Header       | TCP Header          | Payload         |
| Src: 10.85.1.56 | Src: 67493 (random) |                 |
| Dst: 10.0.0.132 | Dst: 443            |                 |
+-----------------+---------------------+-----------------+
```

節點會將封包轉送至其預設閘道，因為目的地 IP 不在本機網路中。

4. 本機路由器路由

本機路由器會判定目的地 IP (10.0.0.132) 屬於 VPC CIDR (10.0.0.0/16)，並將其轉送至與 AWS 連接的閘道。

5. 跨邊界傳輸

封包會透過您已建立的連線 (例如 Direct Connect 或 VPN) 跨雲端邊界傳輸到 VPC。

6. VPC 網路交付

VPC 網路層會將封包路由至 EKS 控制平面 ENI (10.0.0.132) 所在的正確子網路。

7. ENI 接收

該封包會到達連接到 Kubernetes API 伺服器的 EKS 控制平面 ENI。

回應

在 EKS 控制平面處理請求後，它會傳送回應至 Pod：

7. API 伺服器傳送回應

EKS Kubernetes API 伺服器會使用原始來源 IP 作為目的地，進而建立回應封包。封包如下所示：

```
+-----------------+---------------------+-----------------+
| IP Header       | TCP Header          | Payload         |
| Src: 10.0.0.132 | Src: 443            |                 |
| Dst: 10.85.1.56 | Dst: 67493          |                 |
+-----------------+---------------------+-----------------+
```

由於目的地 IP 屬於已設定的遠端 Pod CIDR (10.85.0.0/16)，因此它會透過其在 VPC 中的 ENI 傳送，並將子網路的路由器作為下一個躍點。

6. VPC 路由

VPC 路由表包含遠端 Pod CIDR (10.85.0.0/16) 的項目，並且會將此流量導向 VPC-to-onprem 閘道。

5. 跨邊界傳輸

閘道會透過您已建立的連線 (例如 Direct Connect 或 VPN)，將封包跨雲端邊界傳輸到您的內部部署網路。

4. 內部部署網路接收

封包送達本機內部部署路由器。

3. 交付至節點

路由器的資料表有一個 10.85.1.0/24 的項目，其中 10.80.0.2 將作為下一個躍點，負責將封包交付至我們的節點。

2. 節點網路處理

當封包由節點的網路堆疊處理時，conntrack (netfilter 的一部分) 會將封包與 Pod 最初建立的連線進行比對。由於最初套用了 DNAT，conntrack 透過將來源 IP 從 EKS 控制平面 ENI 的 IP 重寫到 kubernetes 服務 IP，進而反轉 DNAT：

```
+-----------------+---------------------+-----------------+
| IP Header       | TCP Header          | Payload         |
| Src: 172.16.0.1 | Src: 443            |                 |
| Dst: 10.85.1.56 | Dst: 67493          |                 |
+-----------------+---------------------+-----------------+
```

1。CNI 處理

CNI 會識別目的地 IP 屬於其網路中的 Pod，並將封包交付至正確的 Pod 網路命名空間。

此流程展示為什麼遠端 Pod CIDR 必須能夠從 VPC 一直正確路由到託管每個 Pod 的特定節點，整個傳回路徑取決於跨雲端和內部部署網路的適當 Pod IP 路由。

使用 CNI NAT

此流程與不使用 CNI NAT 的流程非常相似，但有一個重要差異：CNI 會先將來源 NAT (SNAT) 套用至封包，然後再將其傳送至節點的網路堆疊。這樣一來，即會將封包的來源 IP 變更為節點的 IP，從而允許封包路由回節點，而不需要額外的路由組態。

請求

1。Pod 啟動請求

Pod 會從隨機來源連接埠傳送請求至 EKS Kubernetes API 伺服器連接埠 (443) 上的 kubernetes 服務 IP (172.16.0.1)。封包如下所示：

```
+-----------------+---------------------+-----------------+
| IP Header       | TCP Header          | Payload         |
| Src: 10.85.1.56 | Src: 67493 (random) |                 |
| Dst: 172.16.0.1 | Dst: 443            |                 |
+-----------------+---------------------+-----------------+
```

2. CNI 處理

CNI 偵測到目的地 IP 不屬於其管理的任何 Pod CIDR。由於傳出 NAT 已啟用，CNI 會將 SNAT 套用至封包，將來源 IP 變更為節點的 IP，然後再將其傳遞至節點的網路堆疊：

```
+-----------------+---------------------+-----------------+
| IP Header       | TCP Header          | Payload         |
| Src: 10.80.0.2  | Src: 67493 (random) |                 |
| Dst: 172.16.0.1 | Dst: 443            |                 |
+-----------------+---------------------+-----------------+
```

注意：為了清楚起見，在範例中，CNI 和 iptables 會顯示為獨立的區塊，但實際上，有些 CNI 可能會使用 iptables 來套用 NAT。

3. 節點網路處理

在這裡，由 kube-proxy 設定的 iptables 規則的行為與上一個範例中的行為相同，並且會將封包負載平衡到其中一個 EKS 控制平面 ENI。初始封包現如下所示：

```
+-----------------+---------------------+-----------------+
| IP Header       | TCP Header          | Payload         |
| Src: 10.80.0.2  | Src: 67493 (random) |                 |
| Dst: 10.0.0.132 | Dst: 443            |                 |
+-----------------+---------------------+-----------------+
```

節點會將封包轉送至其預設閘道，因為目的地 IP 不在本機網路中。

4. 本機路由器路由

本機路由器會判定目的地 IP (10.0.0.132) 屬於 VPC CIDR (10.0.0.0/16)，並將其轉送至與 AWS 連接的閘道。

5. 跨邊界傳輸

封包會透過您已建立的連線 (例如 Direct Connect 或 VPN) 跨雲端邊界傳輸到 VPC。

6. VPC 網路交付

VPC 網路層會將封包路由至 EKS 控制平面 ENI (10.0.0.132) 所在的正確子網路。

7. ENI 接收

該封包會到達連接到 Kubernetes API 伺服器的 EKS 控制平面 ENI。

回應

在 EKS 控制平面處理請求後，它會傳送回應至 Pod：

7. API 伺服器傳送回應

EKS Kubernetes API 伺服器會使用原始來源 IP 作為目的地，進而建立回應封包。封包如下所示：

```
+-----------------+---------------------+-----------------+
| IP Header       | TCP Header          | Payload         |
| Src: 10.0.0.132 | Src: 443            |                 |
| Dst: 10.80.0.2  | Dst: 67493          |                 |
+-----------------+---------------------+-----------------+
```

由於目的地 IP 屬於已設定的遠端節點 CIDR (10.80.0.0/16)，因此它會透過其在 VPC 中的 ENI 傳送，並將子網路的路由器作為下一個躍點。

6. VPC 路由

VPC 路由表包含遠端節點 CIDR (10.80.0.0/16) 的項目，並且會將此流量導向 VPC-to-onprem 閘道。

5. 跨邊界傳輸

閘道會透過您已建立的連線 (例如 Direct Connect 或 VPN)，將封包跨雲端邊界傳輸到您的內部部署網路。

4. 內部部署網路接收

封包送達本機內部部署路由器。

3. 交付至節點

本機路由器會識別目的地 IP (`10.80.0.2`) 位址屬於其直接連接的網路，並將封包直接轉送至目標混合節點。

2. 節點網路處理

當封包由節點的網路堆疊處理時，`conntrack` (`netfilter` 的一部分) 會將封包與 Pod 最初建立的連線進行比對，而且由於最初套用了 DNAT，其透過將來源 IP 從 EKS 控制平面 ENI 的 IP 重寫到 `kubernetes` 服務 IP，進而實現反轉：

```
+-----------------+---------------------+-----------------+
| IP Header       | TCP Header          | Payload         |
| Src: 172.16.0.1 | Src: 443            |                 |
| Dst: 10.80.0.2  | Dst: 67493          |                 |
+-----------------+---------------------+-----------------+
```

1。CNI 處理

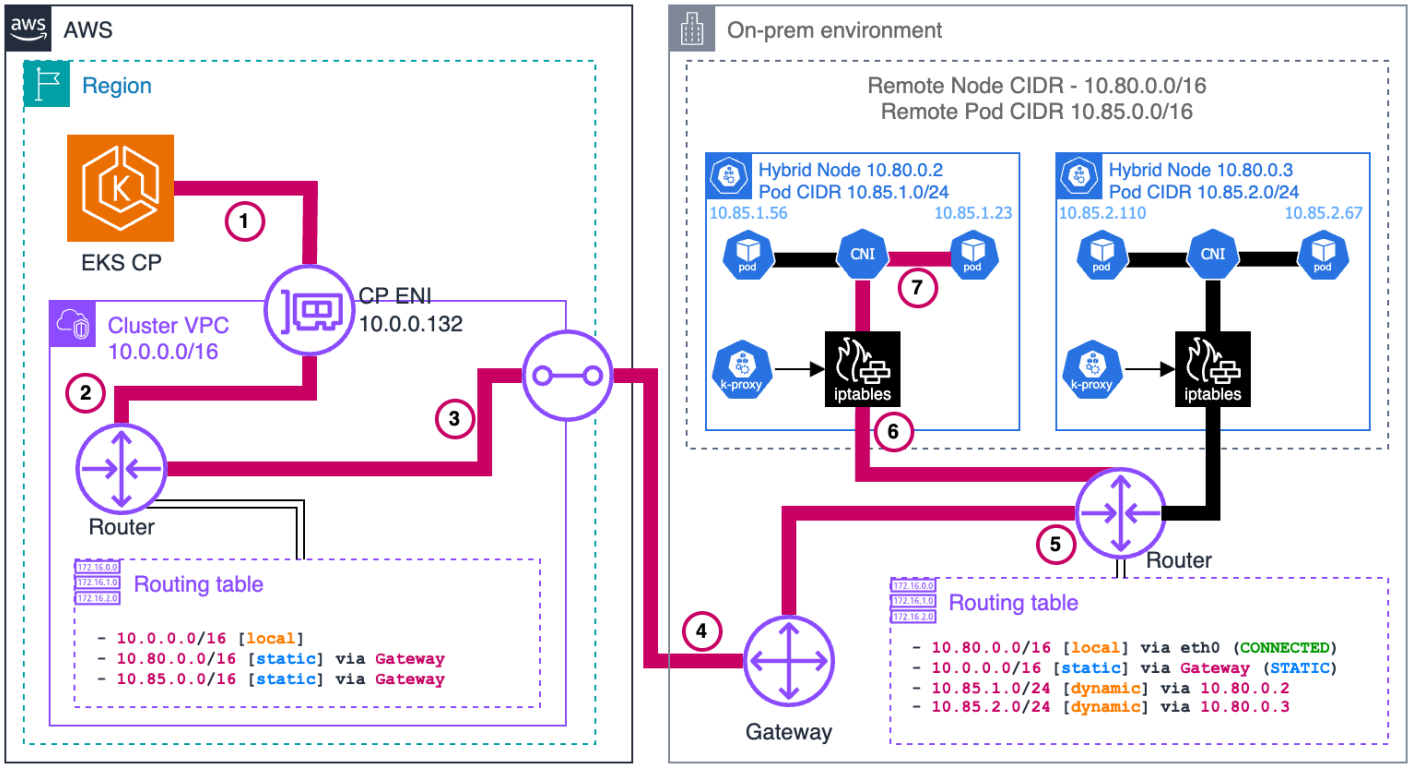
CNI 會識別此封包屬於先前已套用 SNAT 的連線。其會反轉 SNAT，並將目的地 IP 變更回 Pod 的 IP：

```
+-----------------+---------------------+-----------------+
| IP Header       | TCP Header          | Payload         |
| Src: 172.16.0.1 | Src: 443            |                 |
| Dst: 10.85.1.56 | Dst: 67493          |                 |
+-----------------+---------------------+-----------------+
```

CNI 偵測到目的地 IP 屬於其網路中的 Pod，並將封包交付至正確的 Pod 網路命名空間。

此流程會示範 CNI NAT-ing 可如何透過允許封包路由回節點來簡化組態，並且無需額外的 Pod CIDR 路由。

EKS 控制平面到混合節點上執行的 Pod (Webhook)



此流量模式最常見於 Webhook 中，而 EKS 控制平面需要直接啟動與混合節點上 Pod 中執行的 Webhook 伺服器的連線。範例包括驗證和變換許可 Webhook，其中這些 Webhook 會在資源驗證或變換程序期間由 API 伺服器呼叫。

請求

1。EKS Kubernetes API 伺服器啟動請求

在叢集中設定 Webhook 並且相關 API 操作將其觸發後，EKS Kubernetes API 伺服器需要直接連線至 Webhook 伺服器 Pod。API 伺服器會先從與 Webhook 相關聯的服務或端點資源中查詢 Pod 的 IP 位址。

假設 Webhook Pod 在具有 IP `10.85.1.23` 的混合節點上執行，EKS Kubernetes API 伺服器會建立對 Webhook 端點的 HTTPS 請求。初始封包會透過 VPC 中的 EKS 控制平面 ENI 傳送，因為目的地 IP `10.85.1.23` 屬於已設定的遠端 Pod CIDR (`10.85.0.0/16`)。封包如下所示：

```
+-----------------+---------------------+-----------------+
| IP Header       | TCP Header          | Payload         |
| Src: 10.0.0.132 | Src: 41892 (random) |                 |
| Dst: 10.85.1.23 | Dst: 8443           |                 |
```

```
+-----------------+--------------------+-----------------+
```

2. VPC 網路處理

該封包會離開 EKS 控制平面 ENI 並進入 VPC 網路層，同時會將子網路的路由器作為下一個躍點。

3. VPC 路由表查詢

包含 EKS 控制平面 ENI 的子網路的 VPC 路由表包含適用於遠端 Pod CIDR 的特定路由 (`10.85.0.0/16`)。此路由規則會將封包導向 VPC-to-onprem 閘道 (例如，適用於 Direct Connect 的虛擬私有閘道或 VPN 連線)：

```
Destination     Target
10.0.0.0/16     local
10.85.0.0/16    vgw-id (VPC-to-onprem gateway)
```

4. 跨邊界傳輸

閘道會透過您已建立的連線 (例如 Direct Connect 或 VPN)，將封包跨雲端邊界傳輸到您的內部部署網路。該封包會在其於連線傳輸時，維護其原始來源和目的地 IP 位址。

5. 內部部署網路接收

封包送達本機內部部署路由器。路由器會參考其路由表，以判定連接 10.85.1.23 地址的方式。為此，您的內部部署網路必須具有將流量導向適當混合節點的 Pod CIDR 的路由。

在此情況下，路由器的路由表包含一個項目，其中會指出可透過具有 IP `10.80.0.2` 的混合節點來連接 `10.85.1.0/24` 子網路：

```
Destination     Next Hop
10.85.1.0/24    10.80.0.2
```

6. 交付至節點

根據路由表項目，路由器會將封包轉送到混合節點 (`10.80.0.2`)。當封包抵達節點時，它看起來與 EKS Kubernetes API 伺服器傳送它時相同，而目的地 IP 仍然是 Pod 的 IP。

7. CNI 處理

節點的網路堆疊會接收封包，並發現目的地 IP 不是節點的自有 IP，然後將其傳遞給 CNI 以進行處理。CNI 會識別目的地 IP 屬於在此節點上以本機執行的 Pod，並透過適當的虛擬介面將封包轉送至正確的 Pod：

```
Original packet -> node routing -> CNI -> Pod's network namespace
```

Pod 中的 Webhook 伺服器會接收請求並進行處理。

回應

在 Webhook Pod 處理請求後，它會遵循相同路徑反向傳回回應：

7. Pod 傳送回應

Webhook Pod 會使用其自己的 IP 作為來源建立回應封包，並將原始請求者 (EKS 控制平面 ENI) 作為目的地：

```
+-----------------+---------------------+-----------------+
| IP Header       | TCP Header          | Payload         |
| Src: 10.85.1.23 | Src: 8443           |                 |
| Dst: 10.0.0.132 | Dst: 41892          |                 |
+-----------------+---------------------+-----------------+
```

CNI 會識別此封包會移至外部網路 (而非本機 Pod)，並將封包傳遞至保留原始來源 IP 的節點的網路堆疊。

6. 節點網路處理

節點會判定目的地 IP (`10.0.0.132`) 不在本機網路中，並將封包轉送至其預設閘道 (本機路由器)。

5. 本機路由器路由

本機路由器會參考其路由表，並判定目的地 IP (`10.0.0.132`) 屬於 VPC CIDR (`10.0.0.0/16`)。它會將封包轉送到連接至 AWS的閘道。

4. 跨邊界傳輸

該封包會傳回相同的內部部署至 VPC 連線，並以相反方向跨越雲端邊界。

3. VPC 路由

當該封包抵達 VPC 時，路由表會識別目的地 IP 屬於 VPC 內的子網路。封包會在 VPC 中進行相應路由。

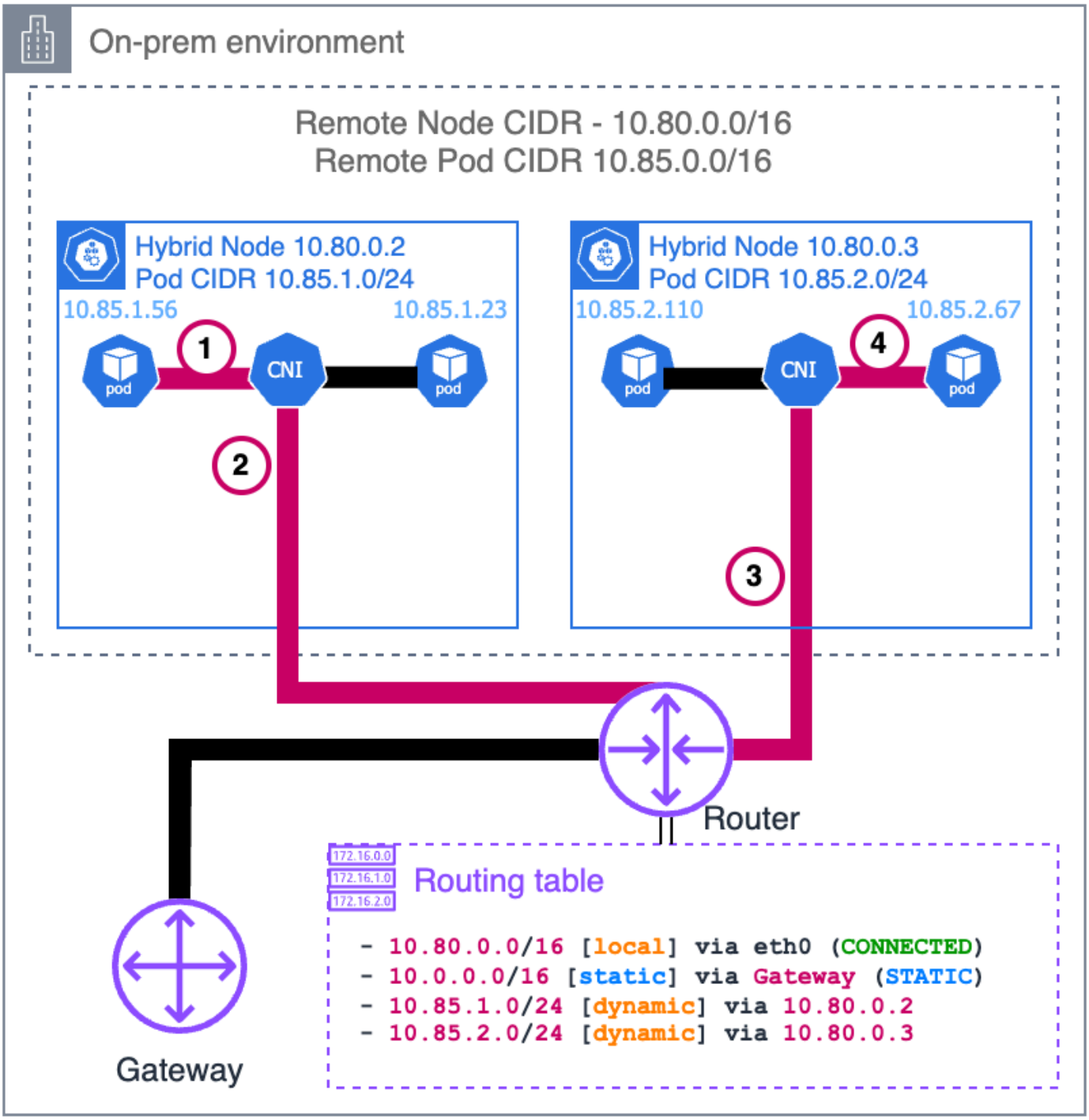
2. 和 1. EKS 控制平面 ENI 接收

該封包會到達連接到 EKS Kubernetes API 伺服器的 ENI，以便完成往返。API 伺服器會接收 Webhook 回應，並繼續根據此回應處理原始 API 請求。

此流量流程示範為什麼遠端 Pod CIDR 必須正確進行設定和路由：

- VPC 必須具有指向內部部署閘道的遠端 Pod CIDR 的路由
- 您的內部部署網路必須具有 Pod CIDR 的路由，並將流量導向託管這些 Pod 的特定節點
- 如果沒有此路由組態，將無法從 EKS 控制平面連接混合節點上 Pod 中執行的 Webhook 和其他類似服務。

在混合節點上執行的 Pod 到 Pod



本節說明在不同混合節點上執行的 Pod 會如何彼此進行通訊。此範例假設您的 CNI 使用 VXLAN 進行封裝，而這在 Cilium 或 Calico 等 CNI 中很常見。整體程序與其他封裝通訊協定，例如 Geneve 或 IP-in-IP。

請求

1。Pod A 啟動通訊

節點 1 上的 Pod A (10.85.1.56) 想要將流量傳送至節點 2 上的 Pod B (10.85.2.67)。初始封包如下所示：

```
+------------------+-----------------+-------------+-----------------+
| Ethernet Header  | IP Header       | TCP/UDP     | Payload         |
| Src: Pod A MAC   | Src: 10.85.1.56 | Src: 43721  |                 |
| Dst: Gateway MAC | Dst: 10.85.2.67 | Dst: 8080   |                 |
+------------------+-----------------+-------------+-----------------+
```

2. CNI 攔截和處理封包

當 Pod A 的封包離開其網路命名空間時，CNI 會將其攔截。CNI 會參考其路由表並判定： - 目的地 IP (10.85.2.67) 屬於 Pod CIDR - 此 IP 不在本機節點上，但屬於節點 2 (10.80.0.3) - 該封包需要使用 VXLAN 進行封裝。

封裝的決策至關重要，因為底層實體網路不知道如何直接路由 Pod CIDR，它只知道如何在節點 IP 之間路由流量。

CNI 會將整個原始封包封裝在 VXLAN 架構內。這實際上建立了有效地具有新標頭的「封包內封包」：

```
+-----------------+---------------+-------------+-----------
+--------------------------+
| Outer Ethernet  | Outer IP      | Outer UDP   | VXLAN     | Original Pod-to-Pod
    |
| Src: Node1 MAC  | Src: 10.80.0.2 | Src: Random  | VNI: 42    | Packet (unchanged
    |
| Dst: Router MAC | Dst: 10.80.0.3 | Dst: 8472    |            | from above)
    |
+-----------------+---------------+-------------+-----------
+--------------------------+
```

此封裝的要點：- 將外部封包從節點 1 (10.80.0.2) 傳送至節點 2 (10.80.0.3) - 根據預設，UDP 連接埠 8472 是 Cilium 使用的 VXLAN 連接埠 - VXLAN 網路識別符 (VNI) 可識別此封包所屬的覆蓋網路 - 整個原始封包 (將 Pod A 的 IP 作為來源，將 Pod B 的 IP 作為目的地) 可在內部完整保留

已封裝的封包現在會進入節點 1 的常規網路堆疊，並以與任何其他封包相同的方式進行處理：

1. 節點網路處理：節點 1 的網路堆疊會根據其目的地 (10.80.0.3) 路由封包

2. 本機網路交付：
   - 如果兩個節點位於相同的第 2 層網路上，則該封包會直接傳送至節點 2
   - 如果封包位於不同的子網路上，則會先將該封包轉送至本機路由器
3. 路由器處理：路由器會根據其路由表轉送封包，並將其交付至節點 2

3. 接收節點處理

當封裝的封包抵達節點 2 (`10.80.0.3`) 時：

1. 節點的網路堆疊會接收此封包並將其識別為 VXLAN 封包 (UDP 連接埠 4789)
2. 該封包會傳遞至 CNI 的 VXLAN 介面，以進行處理

4. VXLAN 解封裝

節點 2 上的 CNI 會處理 VXLAN 封包：

1. 它會移除外部標頭 (乙太網路、IP、UDP 和 VXLAN)
2. 它會擷取原始內部封包
3. 該封包現恢復為其原始格式：

```
+------------------+-----------------+-------------+-----------------+
| Ethernet Header  | IP Header       | TCP/UDP     | Payload         |
| Src: Pod A MAC   | Src: 10.85.1.56 | Src: 43721  |                 |
| Dst: Gateway MAC | Dst: 10.85.2.67 | Dst: 8080   |                 |
+------------------+-----------------+-------------+-----------------+
```

節點 2 上的 CNI 會檢查目的地 IP (`10.85.2.67`) 和：

1. 識別此 IP 屬於本機 Pod
2. 透過適當的虛擬介面路由封包
3. 將封包交付至 Pod B 的網路命名空間

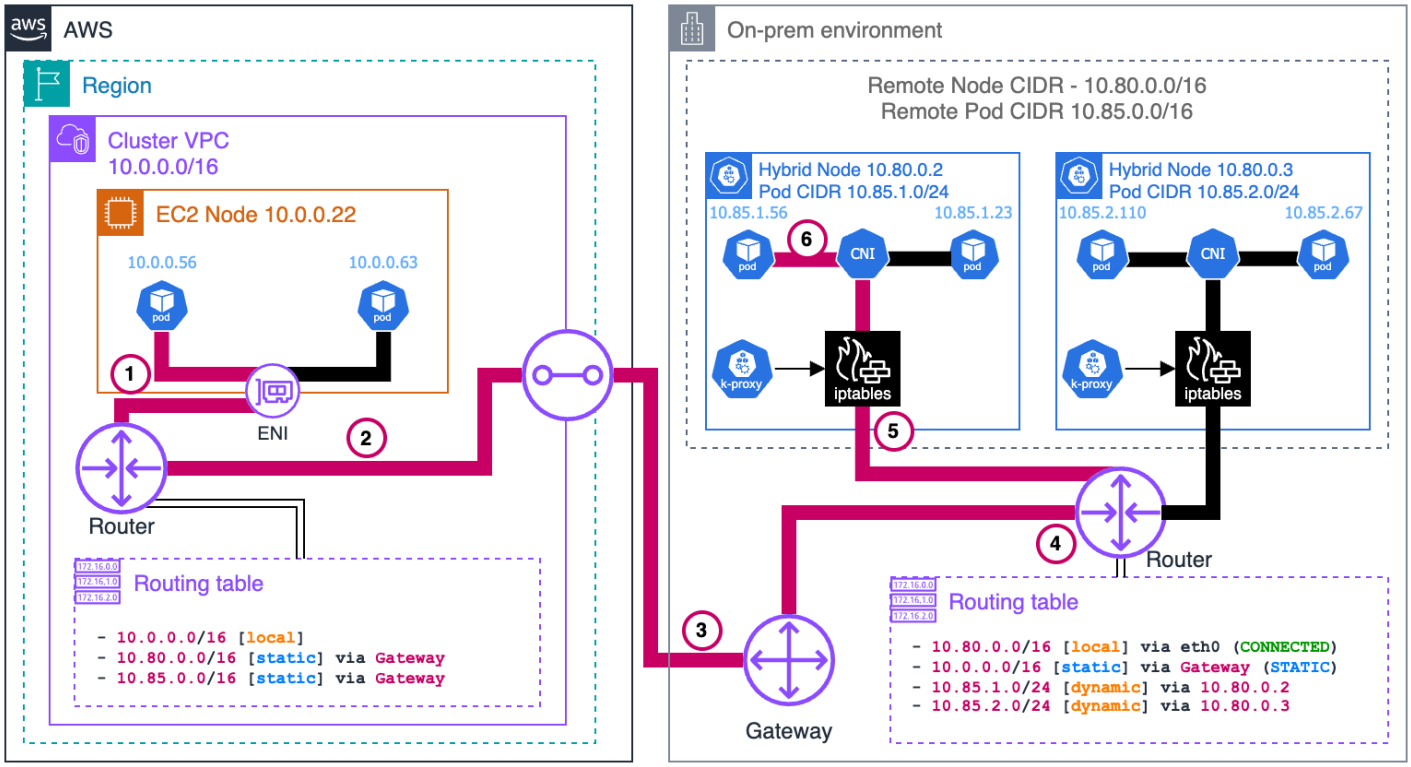
回應

當 Pod B 回應 Pod A 時，整個程序會發生反轉，即以相反順序執行：

4. Pod B 會將封包傳送至 Pod A (`10.85.1.56`)

5. 節點 2 的 CNI 會使用 VXLAN 進行封裝，並將目的地設定為節點 1 (10.80.0.2)
6. 封裝的封包會交付至節點 1
7. 節點 1 的 CNI 會將其解封裝，並將原始回應交付給 Pod A

雲端節點上的 Pod 到混合節點上的 Pod (東西流量)



請求

1。Pod A 啟動通訊

EC2 節點上的 Pod A (10.0.0.56) 想要將流量傳送至混合節點上的 Pod B (10.85.1.56)。初始封包如下所示：

```
+-----------------+---------------------+-----------------+
| IP Header       | TCP Header          | Payload         |
| Src: 10.0.0.56  | Src: 52390 (random) |                 |
| Dst: 10.85.1.56 | Dst: 8080           |                 |
+-----------------+---------------------+-----------------+
```

使用 VPC CNI 時，Pod A 具有來自 VPC CIDR 的 IP，並會直接連接到 EC2 執行個體上的 ENI。Pod 的網路命名空間已連接至 VPC 網路，因此該封包會直接進入 VPC 路由基礎結構。

2. VPC 路由

VPC 路由表包含遠端 Pod CIDR (10.85.0.0/16) 的特定路由，並且會將此流量導向 VPC-to-onprem 閘道：

```
Destination     Target
10.0.0.0/16     local
10.85.0.0/16    vgw-id (VPC-to-onprem gateway)
```

根據此路由規則，該封包會導向至連接到內部部署網路的閘道。

3. 跨邊界傳輸

閘道會透過您已建立的連線 (例如 Direct Connect 或 VPN)，將封包跨雲端邊界傳輸到您的內部部署網路。該封包會在其傳輸過程中，維護其原始來源和目的地 IP 位址。

4. 內部部署網路接收

封包送達本機內部部署路由器。路由器會參考其路由表，以判定連接 10.85.1.56 地址的下一個躍點。您的內部部署路由器必須具有將流量導向適當混合節點的 Pod CIDR 的路由。

該路由器的資料表包含一個項目，其中會指出可透過具有 IP 10.80.0.2 的混合節點來連接 10.85.1.0/24 子網路：

```
Destination     Next Hop
10.85.1.0/24    10.80.0.2
```

5. 節點網路處理

路由器會將封包轉送至混合節點 (10.80.0.2)。當封包抵達節點時，它仍然將 Pod A 的 IP 作為來源，並將 Pod B 的 IP 作為目的地。

6. CNI 處理

節點的網路堆疊會接收封包，並發現目的地 IP 不是其自有 IP，然後將其傳遞給 CNI 以進行處理。CNI 會識別目的地 IP 屬於在此節點上以本機執行的 Pod，並透過適當的虛擬介面將封包轉送至正確的 Pod：

```
Original packet -> node routing -> CNI -> Pod B's network namespace
```

Pod B 會接收封包並視需要進行處理。

回應

6. Pod B 傳送回應

Pod B 會以其自有 IP 作為來源建立回應封包，而會將 Pod A 的 IP 作為目的地：

```
+-----------------+---------------------+-----------------+
| IP Header       | TCP Header          | Payload         |
| Src: 10.85.1.56 | Src: 8080           |                 |
| Dst: 10.0.0.56  | Dst: 52390          |                 |
+-----------------+---------------------+-----------------+
```

CNI 會識別此封包以外部網路為目的地，並將其傳遞至節點的網路堆疊。

5. 節點網路處理

節點會判定目的地 IP (`10.0.0.56`) 不屬於本機網路，並將封包轉送至其預設閘道 (本機路由器)。

4. 本機路由器路由

本機路由器會參考其路由表，並判定目的地 IP (`10.0.0.56`) 屬於 VPC CIDR (`10.0.0.0/16`)。它會將封包轉送到連接至 AWS的閘道。

3. 跨邊界傳輸

該封包會傳回相同的內部部署至 VPC 連線，並以相反方向跨越雲端邊界。

2. VPC 路由

當該封包抵達 VPC 時，路由系統會識別目的地 IP 屬於 VPC 內的子網路。該封包會透過 VPC 網路路由至託管 Pod A 的 EC2 執行個體。

1。Pod A 接收回應

該封包抵達 EC2 執行個體，並透過其連接的 ENI 直接交付至 Pod A。由於 VPC CNI 不會為 VPC 中的 Pod 使用覆蓋網路，因此不需要額外的解封裝 - 封包送達時，其原始標題會完整保留。

此東西流量流程示範為什麼遠端 Pod CIDR 必須正確設定並可從雙向路由：

- VPC 必須具有指向內部部署閘道的遠端 Pod CIDR 的路由
- 您的內部部署網路必須具有 Pod CIDR 的路由，並將流量導向託管這些 Pod 的特定節點。

混合節點 nodeadm 參考

Amazon EKS 混合節點 CLI (nodeadm) 可簡化混合節點元件的安裝、組態、註冊和解除安裝。您可以在作業系統映像中包含 nodeadm，以自動化混合節點引導，如需詳細資訊，請參閱 the section called "準備作業系統"。

混合節點的 nodeadm 版本與用於將 Amazon EC2 執行個體引導為 Amazon EKS 叢集中節點的 nodeadm 版本不同。遵循相應 nodeadm 版本的文件和參考。本文件頁面適用於混合節點 nodeadm 版本。

混合節點的原始程式碼nodeadm會發佈在 https://github.com/aws/eks-hybrid GitHub 儲存庫中。

Important

您必須與擁有 root/sudo 權限的使用者一同執行 nodeadm。

SSM 登入資料提供者所需的 nodeadm 版本

如果您使用 AWS Systems Manager (SSM) 作為混合節點的憑證提供者，則必須使用 nodeadm 版本 `1.0.19`或更新版本進行新的安裝和升級。舊版 nodeadm包含過期的 SSM 簽署金鑰，並在 `nodeadm install`期間失敗，並`nodeadm upgrade`出現下列簽章驗證錯誤：

```
"msg":"Command failed","error":"failed to install ssm installer: validating ssm-setup-cli signature: Signature Verification Error: No matching signature"
```

若要解決此錯誤，請在執行 `nodeadm install`或 nodeadm之前下載最新版本的 `nodeadm upgrade`。

下載 nodeadm

nodeadm 的混合節點版本託管於前端為 Amazon CloudFront 的 Amazon S3 中。若要在每個內部部署主機上安裝 nodeadm，您可以從內部部署主機執行下列命令。

對於 x86_64 主機

```
curl -OL 'https://hybrid-assets.eks.amazonaws.com/releases/latest/bin/linux/amd64/nodeadm'
```

對於 ARM 主機

```
curl -OL 'https://hybrid-assets.eks.amazonaws.com/releases/latest/bin/linux/arm64/nodeadm'
```

在每個主機上，為下載的二進位檔新增可執行檔許可。

```
chmod +x nodeadm
```

nodeadm install

nodeadm install 命令可用於安裝執行混合節點並將其加入 Amazon EKS 叢集所需的成品和相依性。nodeadm install 命令可在每個混合節點上分別執行，也可以在映像建置管道期間執行，進而在作業系統映像中預先安裝混合節點相依性。

用途

```
nodeadm install [KUBERNETES_VERSION] [flags]
```

位置引數

(必要) KUBERNETES_VERSION 要安裝的 EKS Kubernetes major.minor 版本，例如 1.32

Flags

名稱	必要	說明
-p, --credential-provider	TRUE	要安裝的憑證提供者。支援的值為 iam-ra 和 ssm。如需詳細資訊，請參閱the section called “準備憑證”。
-s, --containerd-source	FALSE	containerd 的來源。nodeadm 支援從 OS distro、Docker 套件安裝 containerd ，並略過 containerd 安裝。 Values (數值)

名稱	必要	說明
		distro - 這是預設值。nodeadm將安裝由與 EKS Kubernetes 版本相容的節點作業系統分發的最新containerd 套件。 distro 不是 Red Hat Enterprise Linux (RHEL) 作業系統支援的值。 docker - nodeadm將安裝由 Docker 建置和分發且與 EKS Kubernetes 版本相容的最新containerd 套件。 docker 不是 Amazon Linux 2023 支援的值。 none - nodeadm 不會安裝 containerd 套件。您必須先手動安裝 containerd ，然後才能執行 nodeadm init。
-r, --region	FALSE	指定下載成品 AWS 的區域，例如 SSM Agent。預設為 us-west-2 。
-t, --timeout	FALSE	安裝命令持續時間上限。輸入遵循持續時間格式。例如 1h23m。安裝命令的預設下載逾時設定為 20 分鐘。
-h, --help	FALSE	顯示說明訊息，其中包含可用的旗標、子命令和定位值參數。

範例

1.32 使用 AWS Systems Manager (SSM) 做為登入資料提供者來安裝 Kubernetes 版本

```
nodeadm install 1.32 --credential-provider ssm
```

1.32 使用 AWS Systems Manager (SSM) 作為登入資料提供者安裝 Kubernetes 版本，Docker 作為容器化來源，下載逾時為 20 分鐘。

```
nodeadm install 1.32 --credential-provider ssm --containerd-source docker --timeout 20m
```

1.32 使用 AWS IAM Roles Anywhere 做為登入資料提供者來安裝 Kubernetes 版本

```
nodeadm install 1.32 --credential-provider iam-ra
```

nodeadm config check

nodeadm config check 命令會檢查提供的節點組態是否存在錯誤。此命令可用於確認和驗證混合節點組態檔案的正確性。

用途

```
nodeadm config check [flags]
```

Flags

名稱	必要	說明
-c, --config-source	TRUE	nodeadm 組態的來源。對於混合節點，輸入應該遵循具有檔案結構描述的 URI。
-h, --help	FALSE	顯示說明訊息，其中包含可用的旗標、子命令和定位值參數。

範例

```
nodeadm config check -c file://nodeConfig.yaml
```

nodeadm init

nodeadm init 命令會啟動混合節點並將其與設定的 Amazon EKS 叢集進行連接。如需如何設定 nodeConfig.yaml 檔案的詳細資訊，請參閱 the section called “SSM 混合啟用的節點組態” 或 the section called “IAM Roles Anywhere 的節點組態”。

用途

```
nodeadm init [flags]
```

Flags

名稱	必要	說明
-c, --config-source	TRUE	nodeadm 組態的來源。對於混合節點，輸入應該遵循具有檔案結構描述的 URI。
-s, --skip	FALSE	要略過的 init 階段。除非有助於修正問題，否則不建議略過任何階段。 Values (數值) install-validation 會略過檢查上述安裝命令是否已成功執行。 如果節點上已啟用防火牆，則 cni-validation 會略過檢查 Cilium 或 Calico CNI 的 VXLAN 連接埠是否已開啟 node-ip-validation 會略過檢查節點 IP 是否落在遠端節點網路中的 CIDR 內

名稱	必要	說明
-h, --help	FALSE	顯示說明訊息，其中包含可用的旗標、子命令和定位值參數。

範例

```
nodeadm init -c file://nodeConfig.yaml
```

nodeadm upgrade

nodeadm upgrade 命令會將所有已安裝的成品升級到最新版本，並引導節點來設定升級的成品，並在 AWS上加入 EKS 叢集。升級是對節點上執行的工作負載的干擾命令。請先將工作負載移至另一個節點，然後再執行升級。

用途

```
nodeadm upgrade [KUBERNETES_VERSION] [flags]
```

位置引數

(必要) KUBERNETES_VERSION 要安裝的 EKS Kubernetes major.minor 版本，例如 1.32

Flags

名稱	必要	說明
-c, --config-source	TRUE	nodeadm 組態的來源。對於混合節點，輸入應該遵循具有檔案結構描述的 URI。
-t, --timeout	FALSE	下載成品逾時。輸入遵循持續時間格式。例如 1h23m。升級命令的預設下載逾時設定為 10 分鐘。

名稱	必要	說明
-s, --skip	FALSE	要略過的升級階段。除非有助於修正問題，否則不建議略過任意階段。 Values (數值) pod-validation 會略過檢查節點上是否所有 Pod 都在執行，但常駐程式集和靜態 Pod 除外。 node-validation 會略過檢查是否已包圍隔離節點。 init-validation 會略過檢查節點是否在執行升級之前已成功初始化。 containerd-major-version-upgrade 可防止在節點升級期間進行容器主要版本升級。
-h, --help	FALSE	顯示說明訊息，其中包含可用的旗標、子命令和定位值參數。

範例

```
nodeadm upgrade 1.32 -c file://nodeConfig.yaml
```

```
nodeadm upgrade 1.32 -c file://nodeConfig.yaml --timeout 20m
```

nodeadm uninstall

nodeadm uninstall 命令會停止並移除 nodeadm 在 nodeadm install 期間安裝的成品，包括 kubelet 和 containerd。請注意，解除安裝命令不會耗盡或刪除叢集中的混合節點。您必須分別執行耗盡和刪除操作，如需詳細資訊，請參閱 the section called “刪除混合節點”。根據預設，如果節點上還有剩餘的 Pod，則 nodeadm uninstall 不會繼續。同樣地，nodeadm uninstall 不會移除 CNI 相依性或您在叢集上執行的其他 Kubernetes 附加元件的相依性。若要從主機完全移除 CNI 安裝，請參閱 the section called “設定 CNI” 中的指示。如果您使用 AWS SSM 混合啟用做為內部部署憑證提供者，nodeadm uninstall命令會將您的主機取消註冊為 AWS SSM 受管執行個體。

用途

```
nodeadm uninstall [flags]
```

Flags

名稱	必要	說明
-s, --skip	FALSE	要略過的解除安裝階段。除非有助於修正問題，否則不建議略過任何階段。 Values (數值) pod-validation 會略過檢查節點上是否所有 Pod 都在執行，但常駐程式集和靜態 Pod 除外。 node-validation 會略過檢查是否已包圍隔離節點。 init-validation 會略過檢查節點是否在執行解除安裝之前已成功初始化。
-h, --help	FALSE	顯示說明訊息，其中包含可用的旗標、子命令和定位值參數。

名稱	必要	說明
`-f`, `--force`	FALSE	強制刪除可能包含 Kubernetes 和 CNI 元件剩餘檔案的其他目錄。 WARNING 這將刪除預設 Kubernetes 和 CNI 目錄中的所有內容 (`/var/lib/cni` 、`/etc/cni/net.d` 等等)。如果您在這些位置存放您自己的資料，則請勿使用此旗標。 從 nodeadm `v1.0.9` 開始，`./nodeadm uninstall --skip node-validation,pod-validation --force` 命令不會再刪除 `/var/lib/kubelet` 目錄。這是因為它可能包含 Pod 磁碟區和磁碟區子路徑目錄，而這些目錄有時會包含掛載的節點檔案系統。 安全處理秘訣 - 刪除掛載的路徑可能會導致意外刪除實際掛載的節點檔案系統。在手動刪除 `/var/lib/kubelet` 目錄之前，請仔細檢查所有作用中的掛載並安全地卸載磁碟區，以避免資料遺失。

範例

```
nodeadm uninstall
```

```
nodeadm uninstall --skip node-validation,pod-validation
```

nodeadm debug

nodeadm debug 命令可用於對運作狀態不佳或設定錯誤的混合節點進行故障診斷。它會驗證下列需求是否已就位。

- 節點具有必要 AWS APIs的網路存取權，以取得登入資料，
- 節點可以取得所設定混合節點 IAM 角色的 AWS 登入資料，
- 節點具有對 EKS Kubernetes API 端點的網路存取權以及 EKS Kubernetes API 端點憑證的有效性，
- 節點能夠使用 EKS 叢集進行身分驗證，其在叢集中的身分有效，而且該節點可以透過為 EKS 叢集設定的 VPC 存取 EKS 叢集。

如果發現錯誤，命令的輸出會建議故障診斷步驟。某些驗證步驟會顯示子程序。如果失敗，輸出會顯示在驗證錯誤下的 stderr 區段中。

用途

```
nodeadm debug [flags]
```

Flags

名稱	必要	說明
-c, --config-source	TRUE	nodeadm 組態的來源。對於混合節點，輸入應該遵循具有檔案結構描述的 URI。
--no-color	FALSE	停用顏色輸出。適用於自動化。
-h, --help	FALSE	顯示說明訊息，其中包含可用的旗標、子命令和定位值參數。

範例

```
nodeadm debug -c file://nodeConfig.yaml
```

Nodeadm 檔案位置

nodeadm 安裝

執行 `nodeadm install` 時，會設定下列檔案和檔案位置。

Artifact	路徑
IAM Roles Anywhere CLI	/usr/local/bin/aws_signing_helper
Kubelet 二進位檔	/usr/bin/kubelet
Kubectl 二進位檔	usr/local/bin/kubectl
ECR 憑證提供者	/etc/eks/image-credential-provider/ecr-credential-provider
AWS IAM 驗證器	/usr/local/bin/aws-iam-authenticator
SSM Setup CLI	/opt/ssm/ssm-setup-cli
SSM Agent	在 Ubuntu 上 - /snap/amazon-ssm-agent/current/amazon-ssm-agent 在 RHEL 和 AL2023 上 - /usr/bin/amazon-ssm-agent
containerd	在 Ubuntu 和 AL2023 上 - /usr/bin/containerd 在 RHEL 上 - /bin/containerd
Iptables	在 Ubuntu 和 AL2023 上 - /usr/sbin/iptables 在 RHEL 上 - /sbin/iptables
CNI 外掛程式	/opt/cni/bin

Artifact	路徑
已安裝的成品追蹤器	/opt/nodeadm/tracker

nodeadm init

執行 `nodeadm init` 時，會設定下列檔案和檔案位置。

名稱	路徑
Kubelet kubeconfig	/var/lib/kubelet/kubeconfig
Kubelet 組態	/etc/kubernetes/kubelet/config.json
Kubelet systemd 單位	/etc/systemd/system/kubelet.service
映像憑證提供者組態	/etc/eks/image-credential-provider/config.json
Kubelet env 檔案	/etc/eks/kubelet/environment
Kubelet 憑證	/etc/kubernetes/pki/ca.crt
Containerd 組態	/etc/containerd/config.toml
Containerd 核心模組組態	/etc/modules-load.d/containerd.conf
AWS 組態檔案	/etc/aws/hybrid/config
AWS 登入資料檔案（如果啟用登入資料檔案）	/eks-hybrid/.aws/credentials
AWS 簽署協助程式系統單位	/etc/systemd/system/aws_signing_helper_update.service
Sysctl conf 檔案	/etc/sysctl.d/99-nodeadm.conf
Ca-certificates	/etc/ssl/certs/ca-certificates.crt
Gpg 金鑰檔案	/etc/apt/keyrings/docker.asc
Docker 儲存庫來源檔案	/etc/apt/sources.list.d/docker.list

SSM 混合啟用的節點組態

以下是針對混合節點登入資料使用 AWS SSM 混合啟用nodeConfig.yaml的範例。

```
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name:              # Name of the EKS cluster
    region:            # AWS Region where the EKS cluster resides
  hybrid:
    ssm:
      activationCode: # SSM hybrid activation code
      activationId:   # SSM hybrid activation id
```

IAM Roles Anywhere 的節點組態

以下是混合節點登入nodeConfig.yaml資料 AWS IAM Roles Anywhere 的範例。

使用 AWS IAM Roles Anywhere 做為內部部署憑證提供者時，nodeName您在nodeadm組態中使用的 必須符合您為混合節點 IAM 角色設定範圍的許可。例如，如果您的混合節點 IAM 角色許可僅允許 AWS IAM Roles Anywhere 在角色工作階段名稱等於主機憑證的 CN 時擔任角色，則nodeadm組態nodeName中的 必須與您憑證的 CN 相同。您使用的 nodeName 不可超過 64 個字元。如需詳細資訊，請參閱the section called “準備憑證”。

```
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name:               # Name of the EKS cluster
    region:             # AWS Region where the EKS cluster resides
  hybrid:
    iamRolesAnywhere:
      nodeName:         # Name of the node
      trustAnchorArn:   # ARN of the IAM Roles Anywhere trust anchor
      profileArn:       # ARN of the IAM Roles Anywhere profile
      roleArn:          # ARN of the Hybrid Nodes IAM role
      certificatePath: # Path to the certificate file to authenticate with the IAM
 Roles Anywhere trust anchor
      privateKeyPath:   # Path to the private key file for the certificate
```

用於自訂 kubelet 的節點組態 (選用)

您可以在 nodeadm 組態中傳遞 kubelet 組態和旗標。請參閱下列範例，了解如何新增其他節點標籤，abc.example.com/test-label並將 kubelet 組態shutdownGracePeriod設定為 30 秒。如需 kubelet 組態選項的詳細資訊，請參閱 Kubernetes 文件中的 Kubelet 組態 (v1beta1) 參考。如需 kubelet 命令列旗標的詳細資訊，請參閱 Kubernetes 文件中的 kubelet CLI 參考。

```
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name:             # Name of the EKS cluster
    region:           # AWS Region where the EKS cluster resides
  kubelet:
    config:           # Map of kubelet config and values
       shutdownGracePeriod: 30s
    flags:            # List of kubelet flags
       - --node-labels=abc.example.com/test-label=true
  hybrid:
    ssm:
      activationCode: # SSM hybrid activation code
      activationId:   # SSM hybrid activation id
```

用於自訂 containerd 的節點組態 (選用)

您可以在 nodeadm 組態中傳遞自訂 containerd 組態。nodeadm 的 containerd 組態接受內嵌 TOML。請參閱以下範例，了解如何設定 containerd 以停用刪除 containerd 內容存放區中已解壓縮的映像層。

```
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name:             # Name of the EKS cluster
    region:           # AWS Region where the EKS cluster resides
  containerd:
    config: |         # Inline TOML containerd additional configuration
       [plugins."io.containerd.grpc.v1.cri".containerd]
       discard_unpacked_layers = false
  hybrid:
    ssm:
      activationCode: # SSM hybrid activation code
      activationId:   # SSM hybrid activation id
```

> **Note**
>
> 容器版本 1.x 和 2.x 使用不同的組態格式。Containerd 1.x 使用組態版本 2，而 Containerd 2.x 使用組態版本 3。雖然 containerd 2.x 仍與組態第 2 版回溯相容，但建議使用組態第 3 版，以獲得最佳效能。使用 檢查您的容器版本`containerd --version`或檢閱`nodeadm`安裝日誌。如需組態版本控制的詳細資訊，請參閱 https：//https://containerd.io/releases/

您也可以使用 containerd 組態來啟用 SELinux 支援。在 containerd 上啟用 SELinux 後，請確保在節點上排程的 Pod 已啟用適當的 securityContext 和 seLinuxOptions。如需設定安全性內容的詳細資訊，請參閱 Kubernetes 文件。

> **Note**
>
> 根據預設，Red Hat Enterprise Linux (RHEL) 8 和 RHEL 9 已啟用 SELinux，並在主機上設定為嚴格。根據預設，Amazon Linux 2023 已啟用 SELinux，並設定為寬容模式。在主機上將 SELinux 設定為寬容模式時，在 containerd 上啟用它不會封鎖請求，但會根據主機上的 SELinux 組態進行記錄。

```
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name:             # Name of the EKS cluster
    region:           # AWS Region where the EKS cluster resides
  containerd:
    config: |         # Inline TOML containerd additional configuration
       [plugins."io.containerd.grpc.v1.cri"]
       enable_selinux = true
  hybrid:
    ssm:
      activationCode: # SSM hybrid activation code
      activationId:   # SSM hybrid activation id
```

故障診斷混合節點

本主題涵蓋一些使用 Amazon EKS 混合節點時可能遇到的常見錯誤與修正這些錯誤的方法。如需其他疑難排解資訊，請參閱 AWS re：Post 上的 疑難排解和 Amazon EKS 知識中心標籤。如果您無法解決問題，請聯絡 AWS Support。

⚠ SSM 登入資料提供者所需的 nodeadm 版本

如果您使用 AWS Systems Manager (SSM) 作為混合節點的憑證提供者，則必須使用 nodeadm 版本 `1.0.19`或更新版本進行新的安裝和升級。舊版 nodeadm包含過期的 SSM 簽署金鑰，並在 `nodeadm install`期間失敗，並`nodeadm upgrade`出現下列簽章驗證錯誤：

```
"msg":"Command failed","error":"failed to install ssm installer: validating ssm-setup-cli signature: Signature Verification Error: No matching signature"
```

若要解決此錯誤，請在執行 或 之前下載最新版本的 `nodeadm upgrade`。 `nodeadm nodeadm install`

使用 **`nodeadm debug`** 進行節點故障診斷 您可以從混合節點執行 `nodeadm debug` 命令，以驗證是否符合聯網和憑證要求。如需有關 `nodeadm debug` 命令的詳細資訊，請參閱 the section called “混合節點 nodeadm”。

使用叢集洞察偵測混合節點的問題 Amazon EKS 叢集洞察包括洞察檢查，可偵測叢集中 EKS 混合節點組態的常見問題。您可以從 AWS 管理主控台、 AWS CLI 和 AWS SDKs 檢視所有洞見檢查的結果。如需有關叢集洞察的詳細資訊，請參閱 the section called “叢集洞察”。

安裝混合節點故障診斷

下列故障診斷主題與使用 `nodeadm install` 命令在主機上安裝混合節點相依性相關。

`nodeadm` 命令失敗 **`must run as root`**

`nodeadm install` 命令必須由主機上具有 root 或 sudo 權限的使用者執行。如果您利用沒有 root 或 sudo 權限的使用者執行 `nodeadm install`，則會在 nodeadm 輸出中看到下列錯誤。

```
"msg":"Command failed","error":"must run as root"
```

無法連接至相依性

nodeadm install 命令可安裝混合節點所需的相依性。混合節點相依性包括 containerd、kubectl、 kubelet和 AWS SSM 或 AWS IAM Roles Anywhere 元件。您必須擁有從執行 nodeadm install 的位置的存取權，才能下載這些相依性。如需您必須能夠存取的位置清單的詳細資訊，請參閱 the section called “準備聯網”。如果您沒有存取權，則會在 nodeadm install 輸出中看到類似以下錯誤。

```
"msg":"Command failed","error":"failed reading file from url: ...: max retries achieved
 for http request"
```

無法更新套件管理工具

在安裝混合節點相依性之前，nodeadm install 命令會執行 apt update 或 yum update 或 dnf update。如果此步驟不成功，您可能會看到類似以下錯誤。若要修復，您可以在執行 nodeadm install 之前執行 apt update、yum update 或 dnf update，也可以嘗試重新執行 nodeadm install。

```
failed to run update using package manager
```

逾時或超過內容截止日期

執行 nodeadm install 時，如果您在安裝程序的各個階段看到錯誤，其中指出逾時或超過內容截止日期，則可能是連接速度緩慢，進而導致在逾時之前無法安裝混合節點相依性。若要解決這些問題，您可以嘗試使用 nodeadm 中的 --timeout 旗標來延長下載相依性的逾時持續時間。

```
nodeadm install K8S_VERSION --credential-provider CREDS_PROVIDER --timeout 20m0s
```

連接混合節點故障診斷

本節中的故障診斷主題與使用 nodeadm init 命令將混合節點連接至 EKS 叢集的程序有關。

操作錯誤或不支援的結構描述

執行 nodeadm init 時，如果您看到與 operation error 或 unsupported scheme 相關的錯誤，請檢查您的 nodeConfig.yaml，以確保其格式正確並傳遞給 nodeadm。如需有關 nodeConfig.yaml 的格式和選項的詳細資訊，請參閱 the section called “混合節點 nodeadm”。

```
"msg":"Command failed","error":"operation error ec2imds: GetRegion, request canceled,
 context deadline exceeded"
```

混合節點 IAM 角色缺少對 **eks:DescribeCluster** 動作的許可

執行 時nodeadm init， 會呼叫 EKS DescribeCluster動作，nodeadm嘗試收集 EKS 叢集的相關資訊。如果您的混合節點 IAM 角色沒有 eks:DescribeCluster動作的許可，則您必須在執行 nodeadm時傳遞節點組態中的 Kubernetes API 端點、叢集 CA 套件和服務 IPv4 CIDRnodeadm init。如需有關混合節點 IAM 角色所需許可的詳細資訊，請參閱 the section called “準備憑證”。

```
"msg":"Command failed","error":"operation error EKS: DescribeCluster, https response
 error StatusCode: 403 ... AccessDeniedException"
```

混合節點 IAM 角色缺少對 **eks:ListAccessEntries** 動作的許可

執行 時nodeadm init， 會呼叫 EKS ListAccessEntries動作，nodeadm嘗試驗證 EKS 叢集是否具有與混合節點 IAM 角色HYBRID_LINUX相關聯的 類型存取項目。如果您的混合節點 IAM 角色沒有 eks:ListAccessEntries動作的許可，則您必須在執行 nodeadm init命令時傳遞 --skip cluster-access-validation旗標。如需有關混合節點 IAM 角色所需許可的詳細資訊，請參閱 the section called “準備憑證”。

```
"msg":"Command failed","error":"operation error EKS: ListAccessEntries, https response
 error StatusCode: 403 ... AccessDeniedException"
```

節點 IP 不在遠端節點網路 CIDR 中

執行 nodeadm init 時，如果節點的 IP 位址不在指定的遠端節點網路 CIDR 內，則您可能會遇到錯誤。錯誤看起來與下列範例類似：

```
node IP 10.18.0.1 is not in any of the remote network CIDR blocks [10.0.0.0/16
 192.168.0.0/16]
```

此範例顯示 IP 10.18.0.1 的節點嘗試加入具有遠端網路 CIDR 10.0.0.0/16 和 192.168.0.0/16 的叢集。發生錯誤，因為 10.18.0.1 不在任一範圍內。

確認您已正確設定 RemoteNodeNetworks 以包含所有節點 IP 位址。如需有關聯網組態的詳細資訊，請參閱 the section called “準備聯網”。

- 在叢集所在的區域中執行下列命令，以檢查您的 RemoteNodeNetwork 組態。確認輸出中列出的 CIDR 區塊包含節點的 IP 範圍，並且與錯誤訊息中列出的 CIDR 區塊相同。如果它們不相符，請確認您 nodeConfig.yaml 中的叢集名稱和區域與您預期的叢集相符。

```
aws eks describe-cluster --name CLUSTER_NAME --region REGION_NAME --query
 cluster.remoteNetworkConfig.remoteNodeNetworks
```

- 驗證您正在處理的是預期節點：
  - 檢查其主機名稱和 IP 位址是否與您要向叢集註冊的 IP 位址相符，以確認您使用正確的節點。
  - 確認此節點位於正確的內部部署網路中 (在叢集設定期間，其 CIDR 範圍已註冊為 RemoteNodeNetwork 的網路)。

如果您的節點 IP 仍然不符合預期，請檢查下列項目：

- 如果您使用的是 IAM Roles Anywhere，kubelet 會在 IAM Roles Anywhere nodeName 上執行 DNS 查詢，並在可用時使用與節點名稱相關聯的 IP。如果您維護節點的 DNS 項目，請確認這些項目會指向遠端節點網路 CIDR 內的 IP。
- 如果您的節點有多個網路介面，kubelet 可能會預設選取遠端節點網路 CIDR 外部的 IP 位址的介面。若要使用不同的介面，請使用 nodeConfig.yaml 中的 --node-ip kubelet 旗標指定其 IP 位址。如需詳細資訊，請參閱the section called “混合節點 nodeadm”。您可以透過在節點上執行下列命令，來檢視節點的網路介面及其 IP 位址：

```
ip addr show
```

混合節點未出現在 EKS 叢集中

如果您執行 nodeadm init 且已完成，但您的混合節點未出現在叢集中，混合節點與 EKS 控制平面之間的網路連線可能會發生問題，您可能並未設定必要的安全群組許可，或者您可能沒有混合節點 IAM 角色與 Kubernetes 角色型存取控制 (RBAC) 的必要映射。您可以使用下列命令檢查 kubelet 和 kubelet 日誌的狀態，進而開始偵錯程序。從無法加入叢集的混合節點執行下列命令。

```
systemctl status kubelet
```

```
journalctl -u kubelet -f
```

無法與叢集通訊

如果您的混合節點無法與叢集控制平面通訊，您可能會看到類似如下的日誌。

```
"Failed to ensure lease exists, will retry" err="Get ..."
```

```
"Unable to register node with API server" err="Post ..."
```

```
Failed to contact API server when waiting for CSINode publishing ... dial tcp <ip
 address>: i/o timeout
```

如果您看到這些訊息，請檢查下列項目，以確保其符合 the section called “準備聯網” 中詳述的混合節點要求。

- 確認傳遞給 EKS 叢集的 VPC 已設定為可路由到適用於內部部署節點或選用的 Pod CIDR 的傳輸閘道 (TGW) 或虛擬私有閘道 (VGW)。
- 確認您的 EKS 叢集具有其他安全群組，且該安全群組包含適用於內部部署節點 CIDR 的傳入規則，也選擇性地包含適用於 Pod CIDR 的傳入規則。
- 確認您的內部部署路由器已設定為允許流量往返 EKS 控制平面。

未授權

如果您的混合節點能夠與 EKS 控制平面通訊但無法註冊，您可能會看到類似如下的日誌。請注意，以下日誌訊息中的主要差異是 Unauthorized 錯誤。這表明該節點無法執行其任務，因為它沒有必要許可。

```
"Failed to ensure lease exists, will retry" err="Unauthorized"
```

```
"Unable to register node with API server" err="Unauthorized"
```

```
Failed to contact API server when waiting for CSINode publishing: Unauthorized
```

如果您看到這些訊息，請檢查下列項目，以確保其符合 the section called “準備憑證” 和 the section called “準備叢集存取” 中詳述的混合節點要求。

- 確認混合節點的身分與您預期的混合節點 IAM 角色相符。為此，可以從混合節點執行 `sudo aws sts get-caller-identity`。
- 確認您的混合節點 IAM 角色具有必要許可。
- 確認您的叢集中具有適用於混合節點 IAM 角色的 EKS 存取項目，或確認您的 `aws-auth` ConfigMap 具有適用於混合節點 IAM 角色的項目。如果您使用的是 EKS 存取項目，則請確認適用於混合節點 IAM 角色的存取項目具有 `HYBRID_LINUX` 存取類型。如果您使用的是 `aws-auth`

ConfigMap，則請確認適用於混合節點 IAM 角色的項目符合 the section called “準備叢集存取” 中詳述的要求和格式。

向 EKS 叢集註冊但顯示狀態為 **Not Ready** 的混合節點

如果您的混合節點已成功向 EKS 叢集註冊，但混合節點顯示狀態為 Not Ready，則首先要檢查的是容器聯網介面 (CNI) 狀態。如果您尚未安裝 CNI，則預期您的混合節點的狀態為 Not Ready。成功安裝並執行 CNI 後，節點狀態會更新為 Ready。如果您嘗試安裝 CNI，但未成功執行，請參閱此頁面上的 the section called “混合節點 CNI 故障診斷”。

憑證簽署請求 (CSR) 卡在待定狀態

將混合節點連接到 EKS 叢集後，如果您看到混合節點有待定的 CSR，則您的混合節點不符合自動核准的要求。如果混合節點的 CSR 是由具有 eks.amazonaws.com/compute-type: hybrid 標籤的節點建立，且 CSR 具有下列主體替代名稱 (SAN)，則會自動核准及核發混合節點的 CSR：至少一個 DNS SAN 等於節點名稱且 IP SAN 屬於遠端節點網路 CIDR。

混合設定檔已存在

如果您變更 nodeadm 組態並嘗試使用新組態重新註冊節點，您可能會發現錯誤，其中會指出混合設定檔已存在，但其內容已變更。不要在組態變更之間執行 nodeadm init，而是要先執行 nodeadm uninstall，然後再執行 nodeadm install 和 nodeadm init。如此可確保正確清除組態中的變更。

```
"msg":"Command failed","error":"hybrid profile already exists at /etc/aws/hybrid/config
 but its contents do not align with the expected configuration"
```

混合節點無法解析私有 API

執行 nodeadm init 後，如果您在 kubelet 日誌中發現錯誤，其中顯示由於存在 no such host 的情況而無法聯絡 EKS Kubernetes API 伺服器，您可能需要變更內部部署網路或主機層級的 EKS Kubernetes API 端點的 DNS 項目。請參閱 AWS Route53 文件中的轉送傳入 DNS 查詢到您的 VPC。

```
Failed to contact API server when waiting for CSINode publishing: Get ... no such host
```

無法在 EKS 主控台中檢視混合節點

如果您已註冊混合節點，但無法在 EKS 主控台中進行檢視，請檢查您用來檢視主控台的 IAM 主體的許可。您使用的 IAM 主體必須具有特定的最低 IAM 和 Kubernetes 許可，才能在主控台中檢視資源。如需詳細資訊，請參閱the section called “存取叢集資源”。

執行混合節點故障診斷

如果您的混合節點已向 EKS 叢集註冊，狀態為 Ready，然後轉換為狀態 Not Ready，則可能存在各種問題會導致運作狀態不佳，例如節點缺少充足的 CPU、記憶體或可用磁碟空間資源，或節點與 EKS 控制平面中斷連線。您可以使用下列步驟對節點進行故障診斷，如果無法解決問題，請聯絡 AWS Support。

從混合節點執行 nodeadm debug，以驗證是否符合聯網和憑證要求。如需有關 nodeadm debug 命令的詳細資訊，請參閱 the section called “混合節點 nodeadm”。

取得節點狀態

```
kubectl get nodes -o wide
```

檢查節點條件和事件

```
kubectl describe node NODE_NAME
```

取得 Pod 狀態

```
kubectl get pods -A -o wide
```

檢查 Pod 條件和事件

```
kubectl describe pod POD_NAME
```

檢查 Pod 日誌

```
kubectl logs POD_NAME
```

檢查 **kubelet** 日誌

```
systemctl status kubelet
```

```
journalctl -u kubelet -f
```

Pod 即時性探查失敗或 Webhook 無法運作

如果混合節點上執行的應用程式、附加元件或 Webhook 未正確啟動，您可能會遇到封鎖與 Pod 的通訊的聯網問題。若要讓 EKS 控制平面聯絡在混合節點上執行的 Webhook，您必須使用遠端 Pod 網路設定 EKS 叢集，並將 VPC 路由表中的現場部署 Pod CIDR 路由，目標為 Transit Gateway (TGW)、虛擬私有閘道 (VGW) 或您用來連接 VPC 與現場部署網路的其他閘道。如需有關混合節點的聯網需求的詳細資訊，請參閱 the section called “準備聯網”。此外，您還必須在內部部署防火牆中允許此流量，並確保您的路由器可正確路由到您的 Pod。如需有關在混合節點上執行 Webhook 的需求的詳細資訊，請參閱 the section called “設定 Webhook”。

此案例的常見 Pod 日誌訊息如下所示，其中 ip-address 是 Kubernetes Service 的叢集 IP。

```
dial tcp <ip-address>:443: connect: no route to host
```

`kubectl logs` 或 **`kubectl exec`** 命令無法運作 (**`kubelet`** API 命令)

如果 `kubectl attach`、`kubectl cp`、`kubectl logs`、 和 `kubectl exec`kubectl port-forward命令在其他kubectl命令成功時逾時，問題可能與遠端網路組態有關。這些命令會透過叢集連接至節點上的 `kubelet` 端點。如需更多資訊，請參閱the section called “`kubelet` 端點”。

確認您的節點 IP 和 Pod IP 位於為叢集設定的遠端節點網路和遠端 Pod 網路 CIDR 內。使用以下命令來檢查 IP 指派。

```
kubectl get nodes -o wide
```

```
kubectl get pods -A -o wide
```

將這些 IP 與您設定的遠端網路 CIDR 進行比較，以確保正確路由。有關網路組態需求，請參閱 the section called “準備聯網”。

混合節點 CNI 故障診斷

如果您一開始使用混合節點啟動 Cilium 或 Calico 時就遇到問題，最常見的原因是混合節點或混合節點上執行的 CNI Pod 與 EKS 控制平面之間存在聯網問題。請確保您的環境符合準備混合節點的聯網要求。將問題分成幾個部分非常實用。

EKS 叢集組態

RemoteNodeNetwork 和 RemotePodNetwork 組態是否正確無誤？

VPC 組態

VPC 路由表中是否有以傳輸閘道或虛擬私有閘道為目標的 RemoteNodeNetwork and RemotePodNetwork 的路由？

安全群組組態

RemoteNodeNetwork 和 RemotePodNetwork 是否設有傳入和傳出規則？

現場部署網路

是否有往返 EKS 控制平面以及往返混合節點和混合節點上執行的 Pod 的路由和存取權？

CNI 組態

如果使用覆蓋網路，CNI 的 IP 集區組態是否會與使用 Webhook 時為 EKS 叢集設定的 RemotePodNetwork 相符？

混合節點的狀態為 **Ready** 且 CNI 未安裝

如果您的混合節點顯示狀態為 Ready，但您尚未在叢集上安裝 CNI，則混合節點上可能存在舊的 CNI 成品。根據預設，當您使用 Helm 等工具解除安裝 Cilium 和 Calico 時，不會從實體或虛擬機器中移除磁碟上的資源。此外，這些 CNI 的自訂資源定義 (CRD) 仍可能存在於舊安裝的叢集上。如需詳細資訊，請參閱 the section called “設定 CNI” 的「刪除 Cilium」和「刪除 Calico」章節。

Cilium 故障診斷

如果您在混合節點上執行 Cilium 時遇到問題，請參閱 Cilium 文件中的故障診斷步驟。以下各節涵蓋在混合節點上部署 Cilium 時可能特有的問題。

Cilium 未啟動

如果每個混合節點上執行的 Cilium 代理程式未啟動，請檢查 Cilium 代理程式 Pod 的日誌是否存在錯誤。Cilium 代理程式需要連線到 EKS Kubernetes API 端點才能啟動。如果未正確設定此連線，則 Cilium 代理程式啟動會失敗。在此情況下，您會在 Cilium 代理程式 Pod 日誌中看到類似如下的日誌訊息。

```
msg="Unable to contact k8s api-server"
```

```
level=fatal msg="failed to start: Get \"https://<k8s-cluster-ip>:443/api/v1/namespaces/
kube-system\": dial tcp <k8s-cluster-ip>:443: i/o timeout"
```

Cilium 代理程式會在主機網路上執行。您的 EKS 叢集必須使用 RemoteNodeNetwork 進行設定，如此才能實現 Cilium 連線。確認您的 EKS 叢集具有其他安全群組，且該安全群組包含適用於 RemoteNodeNetwork 的傳入規則，您的 VPC 中為您的 RemoteNodeNetwork 設定了路由，並且您的內部部署網路已正確設定為允許連線至 EKS 控制平面。

如果 Cilium 運算子正在執行，且部分 Cilium 代理程式也正在執行 (並非全部)，則請確認您有可用的 Pod IP 可配置給叢集中的所有節點。您可以在 Cilium 組態中搭配使用叢集集區 IPAM 和 clusterPoolIPv4PodCIDRList 時，設定可配置的 Pod CIDR 大小。每個節點 CIDR 大小是使用 Cilium 組態中的 clusterPoolIPv4MaskSize 設定進行設定的。如需詳細資訊，請參閱 Cilium 文件中的展開叢集集區。

Cilium BGP 無法運作

如果您使用 Cilium BGP 控制平面向內部部署網路公告您的 Pod 或服務地址，則您可以使用下列 Cilium CLI 命令來檢查 BGP 是否會向資源公告路由。有關安裝 Cilium CLI 的步驟，請參閱 Cilium 文件中的安裝 Cilium CLI。

如果 BGP 正常運作，您應該會在輸出established中看到工作階段狀態為 的混合節點。您可能需要與聯網團隊合作，以便為環境的本機 AS、對等 AS 和對等地址確定正確的值。

```
cilium bgp peers
```

```
cilium bgp routes
```

如果您使用 Cilium BGP 來公告類型為 LoadBalancer 的 服務 IP，則您的 CiliumLoadBalancerIPPool 和 服務必須具有相同的標籤，而該標籤應該在您的 CiliumBGPAdvertisement 選取器中使用。以下所示為範例。請注意，如果您使用 Cilium BGP 來公告 LoadBalancer 類型的服務 IP，則 BGP 路由可能會在 Cilium 代理程式重新啟動期間中斷。如需詳細資訊，請參閱 Cilium 文件中的失敗案例。

服務

```
kind: Service
apiVersion: v1
metadata:
```

```
  name: guestbook
  labels:
    app: guestbook
spec:
  ports:
  - port: 3000
    targetPort: http-server
  selector:
    app: guestbook
  type: LoadBalancer
```

CiliumLoadBalancerIPPool

```
apiVersion: cilium.io/v2alpha1
kind: CiliumLoadBalancerIPPool
metadata:
  name: guestbook-pool
  labels:
    app: guestbook
spec:
  blocks:
  - cidr: <CIDR to advertise>
  serviceSelector:
    matchExpressions:
      - { key: app, operator: In, values: [ guestbook ] }
```

CiliumBGPAdvertisement

```
apiVersion: cilium.io/v2alpha1
kind: CiliumBGPAdvertisement
metadata:
  name: bgp-advertisements-guestbook
  labels:
    advertise: bgp
spec:
  advertisements:
    - advertisementType: "Service"
      service:
        addresses:
          - ExternalIP
          - LoadBalancerIP
      selector:
        matchExpressions:
```

```
          - { key: app, operator: In, values: [ guestbook ] }
```

Calico 故障診斷

如果您在混合節點上執行 Calico 時遇到問題，請參閱 Calico 文件中的故障診斷步驟。以下各節涵蓋在混合節點上部署 Calico 時可能特有的問題。

下表摘要說明了 Calico 元件以及它們是否會依預設在節點或 Pod 網路上執行。如果您將 Calico 設定為使用 NAT 傳出 Pod 流量，則必須將內部部署網路設定為將流量路由到內部部署節點 CIDR，而您的 VPC 路由表必須設定有為內部部署節點 CIDR 的路由，且這些路由會以傳輸閘道 (TGW) 或虛擬私有閘道 (VGW) 為目標。如果您未將 Calico 設定為使用 NAT 傳出 Pod 流量，則必須將內部部署網路設定為將流量路由到內部部署 Pod CIDR，而您的 VPC 路由表必須設定有為內部部署 Pod CIDR 的路由，且這些路由會以傳輸閘道 (TGW) 或虛擬私有閘道 (VGW) 為目標。

元件	網路
Calico API 伺服器	節點
Kubernetes 專用 Calico 控制器	Pod
Calico 節點代理程式	節點
Calico `typha`	節點
Calico CSI 節點驅動程式	Pod
Calico 運算子	節點

Calico 資源已在封鎖節點上排程或執行

根據預設，Calico 資源未以 DaemonSet 執行並且具有靈活的容錯能力，可在尚未準備好排程或執行 Pod 的封鎖節點上排程這些資源。您可以變更您的運算子安裝以包含下列內容，進而限制非 DaemonSet Calico 資源的容錯。

```
installation:
  ...
  controlPlaneTolerations:
  - effect: NoExecute
    key: node.kubernetes.io/unreachable
    operator: Exists
```

```
    tolerationSeconds: 300
  - effect: NoExecute
    key: node.kubernetes.io/not-ready
    operator: Exists
    tolerationSeconds: 300
  calicoKubeControllersDeployment:
    spec:
      template:
        spec:
          tolerations:
          - effect: NoExecute
            key: node.kubernetes.io/unreachable
            operator: Exists
            tolerationSeconds: 300
          - effect: NoExecute
            key: node.kubernetes.io/not-ready
            operator: Exists
            tolerationSeconds: 300
  typhaDeployment:
    spec:
      template:
        spec:
          tolerations:
          - effect: NoExecute
            key: node.kubernetes.io/unreachable
            operator: Exists
            tolerationSeconds: 300
          - effect: NoExecute
            key: node.kubernetes.io/not-ready
            operator: Exists
            tolerationSeconds: 300
```

憑證故障診斷

對於 AWS SSM 混合啟用和 AWS IAM Roles Anywhere，您可以從混合節點執行下列命令，驗證混合節點 IAM 角色的登入資料是否已在您的混合節點上正確設定。確認節點名稱和混合節點 IAM 角色名稱與您的預期相符。

```
sudo aws sts get-caller-identity
```

```
{
    "UserId": "ABCDEFGHIJKLM12345678910:<node-name>",
```

```
    "Account": "<aws-account-id>",
    "Arn": "arn:aws: sts::<aws-account-id>:assumed-role/<hybrid-nodes-iam-role/<node-
name>"
}
```

AWS Systems Manager (SSM) 故障診斷

如果您針對混合節點登入資料使用 AWS SSM 混合啟用，請注意 安裝在混合節點上的下列 SSM 目錄和成品nodeadm。如需 SSM 代理程式的詳細資訊，請參閱 AWS Systems Manager 使用者指南中的使用 SSM 代理程式。

說明	Location
SSM 代理程式	Ubuntu - `/snap/amazon-ssm-agent/current/amazon-ssm-agent` RHEL 和 AL2023 - `/usr/bin/amazon-ssm-agent`
SSM 代理程式日誌	`/var/log/amazon/ssm`
AWS 登入資料	`/root/.aws/credentials`
SSM Setup CLI	`/opt/ssm/ssm-setup-cli`

重新啟動 SSM 代理程式

重新啟動 SSM 代理程式即可解決某些問題。您可以使用以下命令來重新啟動。

AL2023 和其他作業系統

```
systemctl restart amazon-ssm-agent
```

Ubuntu

```
systemctl restart snap.amazon-ssm-agent.amazon-ssm-agent
```

檢查 SSM 端點的連線

確認您可以從混合節點連接至 SSM 端點。如需 SSM 端點的清單，請參閱 AWS Systems Manager 端點和配額。將以下命令us-west-2中的 取代為 AWS SSM 混合啟用 AWS 的區域。

```
ping ssm.us-west-2.amazonaws.com
```

檢視已註冊的 SSM 執行個體的連線狀態

您可以使用下列 CLI 命令，檢查已向 SSM AWS 混合啟用註冊之執行個體的連線狀態。使用您執行個體的機器 ID 取代機器 ID。

```
aws ssm get-connection-status --target mi-012345678abcdefgh
```

SSM Setup CLI 檢查總和不相符

執行 `nodeadm install` 時，如果您看到 `ssm-setup-cli` 檢查總和不相符的問題，則您應該確認主機上的現有 SSM 安裝並非較舊版本。如果您的主機上有較舊的 SSM 安裝，則請將其移除並重新執行 `nodeadm install`，以解決問題。

```
Failed to perform agent-installation/on-prem registration: error while verifying
 installed ssm-setup-cli checksum: checksum mismatch with latest ssm-setup-cli.
```

SSM **`InvalidActivation`**

如果您看到向 AWS SSM 註冊執行個體時發生錯誤，請確認 `activationId`中的 `activationCode`、 `region`和 `nodeConfig.yaml` 正確無誤。EKS 叢集 AWS 的區域必須符合 SSM 混合啟用的區域。如果這些值設定錯誤，您可能會看到類似如下的錯誤。

```
ERROR Registration failed due to error registering the instance with AWS SSM.
 InvalidActivation
```

SSM **`ExpiredTokenException`**：包含在請求中的安全性權杖已過期

如果 SSM 代理程式無法重新整理憑證，您可能會看到 `ExpiredTokenException`。在這種情況下，如果您能夠從混合節點連接至 SSM 端點，您可能需要重新啟動 SSM 代理程式，以強制執行憑證重新整理。

```
"msg":"Command failed","error":"operation error SSM: DescribeInstanceInformation, https
 response error StatusCode: 400, RequestID: eee03a9e-f7cc-470a-9647-73d47e4cf0be, api
 error ExpiredTokenException: The security token included in the request is expired"
```

執行 register machine 命令時發生 SSM 錯誤

如果您發現向 SSM 註冊機器時出現錯誤，您可能需要重新執行 nodeadm install，以確保已正確安裝所有 SSM 相依性。

```
"error":"running register machine command: , error: fork/exec /opt/aws/ssm-setup-cli:
 no such file or directory"
```

SSM **ActivationExpired**

執行 nodeadm init 時，如果您發現因為過期的啟用而向 SSM 註冊執行個體發生錯誤，您需要建立新的 SSM 混合啟用、使用新的 SSM 混合啟用的 activationCode 和 activationId 更新 nodeConfig.yaml，並重新執行 nodeadm init。

```
"msg":"Command failed","error":"SSM activation expired. Please use a valid activation"
```

```
ERROR Registration failed due to error registering the instance with AWS SSM.
 ActivationExpired
```

SSM 無法重新整理快取的憑證

如果您發現重新整理快取的憑證失敗，/root/.aws/credentials 檔案可能已從您的主機上刪除。首先檢查您的 SSM 混合啟用，並確保其處於作用中狀態，且您的混合節點已正確設定為使用啟用。檢查位於 /var/log/amazon/ssm 的 SSM 代理程式日誌，並在 SSM 端解決問題後重新執行 nodeadm init 命令。

```
"Command failed","error":"operation error SSM: DescribeInstanceInformation, get
 identity: get credentials: failed to refresh cached credentials"
```

清除 SSM

若要從主機移除 SSM 代理程式，您可以執行下列命令。

```
dnf remove -y amazon-ssm-agent
sudo apt remove --purge amazon-ssm-agent
snap remove amazon-ssm-agent
rm -rf /var/lib/amazon/ssm/Vault/Store/RegistrationKey
```

AWS IAM Roles Anywhere 故障診斷

如果您將 AWS IAM Roles Anywhere 用於混合節點憑證，請注意 安裝在混合節點上的下列目錄和成品nodeadm。如需故障診斷 IAM Roles Anywhere 的詳細資訊，請參閱《IAM Roles Anywhere AWS 使用者指南》中的故障診斷 IAM Roles Anywhere 身分和存取權。 AWS

說明	Location
IAM Roles Anywhere CLI	`/usr/local/bin/aws_signing_helper`
預設憑證位置和名稱	`/etc/iam/pki/server.pem`
預設金鑰位置和名稱	`/etc/iam/pki/server.key`

IAM Roles Anywhere 無法重新整理快取的憑證

如果您發現重新整理快取的憑證失敗，請檢閱 `/etc/aws/hybrid/config` 的內容，並確認已正確設定 nodeadm 組態中的 IAM Roles Anywhere。確認存在 `/etc/iam/pki`。每個節點都必須有唯一的憑證和金鑰。根據預設，使用 IAM Roles Anywhere 作為憑證提供者時，nodeadm 會將 `/etc/iam/pki/server.pem` 用於憑證位置和名稱，以及將 `/etc/iam/pki/server.key` 用於私有金鑰。您可能需要先建立目錄，然後再將憑證和金鑰放入具有 `sudo mkdir -p /etc/iam/pki` 的目錄中。您可以使用以下命令來驗證憑證的內容。

```
openssl x509 -text -noout -in server.pem
```

```
open /etc/iam/pki/server.pem: no such file or directory
could not parse PEM data
Command failed {"error": "... get identity: get credentials: failed to refresh cached
 credentials, process provider error: error in credential_process: exit status 1"}
```

IAM Roles Anywhere 未獲授權執行 **`sts:AssumeRole`**

在 `kubelet` 日誌中，如果您在使用 IAM Roles Anywhere 時發現 `sts:AssumeRole` 操作的存取遭拒問題，請檢查混合節點 IAM 角色的信任政策，以確認允許 IAM Roles Anywhere 服務主體擔任混合節點 IAM 角色。此外，請確認您的混合節點 IAM 角色信任政策中已正確設定信任錨 ARN，並且您的混合節點 IAM 角色已新增至您的 IAM Roles Anywhere 設定檔。

```
could not get token: AccessDenied: User: ... is not authorized to perform:
 sts:AssumeRole on resource: ...
```

IAM Roles Anywhere 未獲授權設定 **roleSessionName**

在 kubelet 日誌中，如果您發現設定 roleSessionName 時出現存取遭拒問題，請確認您已為 IAM Roles Anywhere 設定檔將 acceptRoleSessionName 設為 true。

```
AccessDeniedException: Not authorized to set roleSessionName
```

作業系統故障診斷

RHEL

權利或訂閱管理員註冊失敗

如果您正在執行 nodeadm install，且由於權利註冊問題而無法安裝混合節點相依性，則請確保您已在主機上正確設定您的 Red Hat 使用者名稱和密碼。

```
This system is not registered with an entitlement server
```

Ubuntu

找不到 GLIBC

如果您將 Ubuntu 用於作業系統，並將 IAM Roles Anywhere 用作具有混合節點的憑證提供者，同時還發現找不到 GLIBC 的問題，則您可以手動安裝該相依性，進而解決問題。

```
GLIBC_2.32 not found (required by /usr/local/bin/aws_signing_helper)
```

執行下列命令以安裝相依性：

```
ldd --version
sudo apt update && apt install libc6
sudo apt install glibc-source
```

Bottlerocket

如果您已啟用 Bottlerocket 管理員容器，則可使用 SSH 來進行存取，以利用更高的權限進行進階偵錯和故障診斷。下列各節包含需要在 Bottlerocket 主機環境中執行的命令。進入管理員容器後，您可以執行 sheltie，以取得 Bottlerocket 主機中的完整根 shell。

```
sheltie
```

您也可以在每個命令前加上字首 `sudo chroot /.bottlerocket/rootfs`，進而從管理員容器 Shell 的下列區段中執行命令。

```
sudo chroot /.bottlerocket/rootfs <command>
```

使用 logdog 進行日誌收集

Bottlerocket 提供 `logdog` 公用程式來高效收集日誌和系統資訊，進而供故障診斷之用。

```
logdog
```

`logdog` 公用程式會從 Bottlerocket 主機上的不同位置收集日誌，並將其合併為 tarball。根據預設，tarball 會在 `/var/log/support/bottlerocket-logs.tar.gz` 中建立，並且可從 `/.bottlerocket/support/bottlerocket-logs.tar.gz` 的託管容器存取。

使用 journalctl 存取系統日誌

您可以使用下列命令檢查各種系統服務的狀態 (例如 `kubelet`、`containerd` 等等)，並檢視其日誌。`-f` 旗標會即時追蹤日誌。

若要檢查 `kubelet` 服務狀態並擷取 `kubelet` 日誌，您可以執行：

```
systemctl status kubelet
journalctl -u kubelet -f
```

若要檢查 `containerd` 服務狀態並擷取協調的 `containerd` 執行個體的日誌，您可以執行：

```
systemctl status containerd
journalctl -u containerd -f
```

若要檢查 `host-containerd` 服務狀態並擷取主體 `containerd` 執行個體的日誌，您可以執行：

```
systemctl status host-containerd
journalctl -u host-containerd -f
```

如需擷取引導容器和託管容器的日誌，您可以執行：

```
journalctl _COMM=host-ctr -f
```

Amazon EKS 混合節點閘道

Amazon EKS 混合節點閘道可自動化 Amazon EKS 叢集 VPC 與在 EKS 混合節點上執行的 Kubernetes Pod 之間的聯網。閘道不需要讓內部部署 Pod 網路可從 VPC 路由或協調網路基礎設施變更。它會在 VPC 中的 EC2-based閘道節點與現場部署環境中的 Cilium 受管混合節點之間建立 VXLAN 通道，並自動維護 VPC 路由表項目，讓流量到達正確的閘道執行個體。

使用案例

混合節點閘道可啟用 VPC 與內部部署環境之間的下列流量流程：

- 控制平面到 Webhook 通訊 — Kubernetes API 伺服器可以連接在混合節點上執行的 Webhook 端點。如果沒有閘道，除非 Pod CIDRs 在內部部署環境中變成可路由，否則混合節點上的 Webhook 無法從控制平面連線。
- 跨雲端和內部部署的 Pod-to-pod 流量 — 在 VPC 中 EC2 節點上執行的 Pod 可以直接與在混合節點上執行的 Pod 通訊，反之亦然。
- AWS 與混合式 Pod 的服務連線 – Application Load Balancer、Network Load Balancer 和 Amazon Managed Service for Prometheus 等 AWS 服務可以連接在混合節點上執行的 Pod。

Architecture

兩個閘道 Pod 會在標記的 EC2 節點上執行為部署。Kubernetes 租用型領導者選擇會決定哪些 Pod 處於作用中狀態。兩個 Pod 都會在啟動時建立 VXLAN 界面，並執行可監看`CiliumNode`物件的節點對帳程式，因此待命隨時準備好在容錯移轉時在 3-5 秒內轉送流量。只有領導層特定的動作 (VPC 路由表更新`CiliumVTEPConfig`和管理） 會在領導層變更時轉移。

運作方式

混合節點閘道使用四種機制來啟用連線：

VXLAN 通道 — 閘道會在 UDP 連接埠 8472 (Cilium 預設`hybrid_vxlan0`) 上建立具有 VNI 2 的 VXLAN 介面 ()。它透過程式設計 VXLAN 界面上的 FDB 項目、ARP 項目和路由，為每個混合節點建立通道。節點控制器會監看`CiliumNode`物件，並在混合節點聯結或離開叢集時自動新增或移除通道。

VPC 路由表管理 — 當閘道成為領導者時，它會在指定的 VPC 路由表中建立或取代路由。每個路由會將混合 Pod CIDR 指向領導者的主要 ENI，因此以混合 Pod 為目標的 VPC 流量會轉送至作用中閘道執行個體。

Cilium VTEP 整合 — 閘道會建立`CiliumVTEPConfig`自訂資源，告知混合節點上的 Cilium 代理程式傳送 VPC 繫結流量的位置。組態包含領導者的節點 IP 做為通道端點和 VXLAN 介面 MAC 地址。當混合式 Pod 將流量傳送至 VPC 地址時，Cilium 會將流量封裝在 VXLAN 封包中，並將其傳送至閘道。

領導者選擇 — 閘道使用以 Kubernetes Lease 為基礎的領導者選擇搭配作用中待命模型。兩個閘道 Pod 會在由 Pod 反親和性強制執行的個別節點上執行。兩個 Pod 都會在啟動時建立 VXLAN 界面，並執行節點調校程式，以維護所有混合節點的 VTEP 項目。領導者會執行 VPC 路由表更新和 Cilium VTEP 組態。如果領導者失敗，待命會偵測到租用過期、取得租用，並執行領導者設定序列。預期的容錯移轉時間約為 3–5 秒。

部署模型

混合節點閘道會在 VPC 中的 EC2 執行個體上執行，並使用 Helm Chart 部署。閘道支援下列部署目標：

- EKS 自動模式 — 您可以建立 `NodePool`和 `NodeClass`，以使用正確的標籤、污點和來源/目的地檢查組態自動佈建閘道節點。這是建議的組態。
- EKS 受管節點群組 — 您可以在停用閘道標籤、污點和來源/目的地檢查的情況下建立專用受管節點群組，然後在 Helm 值`autoMode.enabled=false`中設定 。

對於所有部署目標，建議使用至少兩個節點以獲得高可用性。如需詳細資訊，請參閱the section called “開始使用”。

定價

Amazon EKS 混合節點閘道不收取額外費用，但您需要支付執行閘道的基礎設施成本，包括適用的 EC2 執行個體和 EKS Auto Mode 管理費。如需詳細資訊，請參閱 Amazon EKS 定價。

區域可用性

Amazon EKS 混合節點閘道適用於所有可使用 EKS 混合節點 AWS 的區域，但中國區域除外。如需支援區域的目前清單，請參閱 the section called “混合節點”。

開放原始碼

Amazon EKS 混合節點閘道程式碼庫是開放原始碼。您可以在 GitHub 儲存庫檢視原始程式碼、報告問題和貢獻。

限制及考量

部署混合節點閘道之前，請考慮下列事項：

- 無流量加密 — 閘道建立的 VXLAN 通道不會加密流量。如果您需要 VPC 與內部部署環境之間的傳輸中加密，請使用加密傳輸，例如 AWS Direct Connect with MACsec 或 VPN 連接。如需詳細資訊，請參閱 Amazon EKS 中的資料保護。
- 單一叢集 — 每個閘道部署都提供單一 EKS 叢集。如果您有多個具有混合節點的叢集，請為每個叢集部署個別閘道。
- 需要 Cilium VTEP — 閘道需要 Cilium CNI 的 EKS 版本，並在混合節點上啟用 VTEP 支援。不支援其他 CNI 外掛程式。

後續步驟

- the section called “開始使用”
- the section called “Configuration”
- the section called “作業”
- the section called “疑難排解”

EKS 混合節點閘道入門

此頁面會逐步解說 Amazon EKS 混合節點閘道的先決條件、環境準備、安裝、驗證和移除。如需閘道及其架構的簡介，請參閱 the section called “混合節點閘道”。

先決條件

安裝混合節點閘道之前，請確認您的環境符合下列要求：

- EKS 叢集支援 Cilium CNI 和 VTEP — 您的 EKS 叢集必須使用 EKS 版本的 Cilium 作為混合節點上的 CNI，並且必須啟用 Cilium VTEP。如需詳細資訊，請參閱the section called “設定閘道的 CNI”。
- AWS 雲端節點上的 VPC CNI — 叢集中的閘道節點和其他雲端節點必須使用 AWS VPC CNI。閘道依賴 VPC 原生路由來轉送 VPC 和 VXLAN 通道之間的流量。
- 混合連線 — 您的 VPC 與內部部署環境之間需要私有連線。您可以使用 AWS Direct Connect、AWS Site-to-Site VPN 或您自己的 VPN 解決方案。如需詳細資訊，請參閱the section called “準備聯網”。
- 允許 VXLAN 流量 — 連接到閘道 EC2 執行個體的安全群組必須允許連接埠 8472 上的傳入和傳出 UDP 流量。在混合節點端，內部部署防火牆規則也必須允許 UDP 連接埠 8472 往返閘道節點 IP 地址的流量。
- 路由表管理的 IAM 許可 — 閘道需要下列 EC2 動作來管理 VPC 路由表：
  - `ec2:DescribeRouteTables`

- ec2:CreateRoute
- ec2:ReplaceRoute
- ec2:DescribeInstances

您可以使用下列其中一種方法授予這些許可：

EKS Pod Identity (recommended)

使用 EKS Pod Identity 僅將許可授予閘道 Pod 的服務帳戶。

a. 如果尚未安裝 EKS Pod Identity Agent 附加元件，請在叢集上安裝：

```
aws eks create-addon \
  --cluster-name CLUSTER_NAME \
  --addon-name eks-pod-identity-agent
```

b. 使用 EKS Pod Identity 所需的 EC2 許可和信任政策建立 IAM 角色：

```
cat > gateway-trust-policy.json << 'EOF'
{
  "Version": "2012-10-17"        ,
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": { "Service": "pods.eks.amazonaws.com" },
      "Action": ["sts:AssumeRole", "sts:TagSession"]
    }
  ]
}
EOF

aws iam create-role \
  --role-name EKSHybridNodesGatewayRole \
  --assume-role-policy-document file://gateway-trust-policy.json

aws iam put-role-policy \
  --role-name EKSHybridNodesGatewayRole \
  --policy-name HybridNodesGatewayRouteTable \
  --policy-document '{
    "Version": "2012-10-17"        ,
    "Statement": [{
      "Effect": "Allow",
      "Action": [
```

```
        "ec2:DescribeRouteTables",
        "ec2:CreateRoute",
        "ec2:ReplaceRoute",
        "ec2:DescribeInstances"
      ],
      "Resource": "*"
    }]
  }'
```

c. 建立將 IAM 角色連結至閘道服務帳戶的 Pod 身分關聯：

```
aws eks create-pod-identity-association \
  --cluster-name CLUSTER_NAME \
  --namespace eks-hybrid-nodes-gateway \
  --service-account eks-hybrid-nodes-gateway \
  --role-arn arn:aws:iam::ACCOUNT_ID:role/EKSHybridNodesGatewayRole
```

Node IAM role

將具有必要許可的內嵌或受管政策連接至與閘道節點執行個體描述檔相關聯的 IAM 角色：

```
aws iam put-role-policy \
  --role-name NODE_ROLE_NAME \
  --policy-name HybridNodesGatewayRouteTable \
  --policy-document '{
    "Version": "2012-10-17"        ,
    "Statement": [{
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeRouteTables",
        "ec2:CreateRoute",
        "ec2:ReplaceRoute",
        "ec2:DescribeInstances"
      ],
      "Resource": "*"
    }]
  }'
```

閘道節點上的所有 Pod 都將具有這些許可。

- EKS 自動模式（如果閘道節點使用自動模式）— 如果您計劃使用 EKS 自動模式來佈建閘道節點，則必須在 EKS 叢集上啟用自動模式。如需詳細資訊，請參閱啟用 EKS 自動模式。

準備閘道節點

閘道至少需要兩個 EC2 節點才能實現高可用性。閘道支援兩個節點組態選項：

- the section called “EKS Auto 模式 （ 建議 ）” — 節點會使用 NodePool和 自動佈建NodeClass。來源/目的地檢查、標籤和污點都是以宣告方式設定。
- the section called “受管節點群組” — 您可以使用受管節點群組或自我管理節點佈建節點。可以透過受管節點群組 API 設定標籤，也可以使用自訂啟動範本搭配使用者資料來停用來源/目的地檢查。

EKS Auto 模式 （ 建議 ）

使用 EKS Auto Mode 時，您必須先建立 NodePool和 ，NodeClass再安裝 Helm Chart。NodePool會使用正確的標籤、污點和來源/目的地檢查組態來佈建 EC2 執行個體。您不需要手動佈建或設定節點。

將下列資源套用至您的叢集，取代預留位置值：

- *YOUR_NODE_ROLE* — EKS Auto Mode 用來佈建節點的節點 IAM 角色名稱。這是您在叢集上啟用自動模式時所設定 （ 或 EKS 建立的角色 ）。如需詳細資訊，請參閱建立 EKS Auto Mode 的節點 IAM 角色。
- *YOUR_CLUSTER_NAME* — EKS 叢集名稱。
- *SUBNET_ID_1*、*SUBNET_ID_2* — 不同可用區域中IDs，其中將佈建閘道節點。

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: hybrid-gateway
spec:
  advancedNetworking:
    sourceDestCheck: DisabledPrimaryENI
  role: YOUR_NODE_ROLE
  securityGroupSelectorTerms:
    - tags:
        aws:eks:cluster-name: YOUR_CLUSTER_NAME
  subnetSelectorTerms:
    - id: SUBNET_ID_1
    - id: SUBNET_ID_2
---
apiVersion: karpenter.sh/v1
kind: NodePool
```

```
metadata:
  name: hybrid-gateway
spec:
  template:
    metadata:
      labels:
        hybrid-gateway-node: "true"
    spec:
      expireAfter: 336h
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: hybrid-gateway
      requirements:
        - key: karpenter.sh/capacity-type
          operator: In
          values:
            - on-demand
        - key: eks.amazonaws.com/instance-category
          operator: In
          values:
            - c
            - m
            - r
        - key: eks.amazonaws.com/instance-generation
          operator: Gt
          values:
            - "4"
        - key: kubernetes.io/arch
          operator: In
          values:
            - amd64
        - key: kubernetes.io/os
          operator: In
          values:
            - linux
      taints:
        - key: hybrid-gateway-node
          effect: NoSchedule
      terminationGracePeriod: 24h0m0s
  disruption:
    budgets:
      - nodes: 10%
    consolidateAfter: 30s
```

```
    consolidationPolicy: WhenEmptyOrUnderutilized
```

此組態中的金鑰欄位：

- advancedNetworking.sourceDestCheck: DisabledPrimaryENI — 停用節點主要 ENI 的 EC2 來源/目的地檢查，讓閘道可以將未定址的流量轉送給自己。
- taints — hybrid-gateway-node: NoSchedule 污點可確保這些節點上只有具有相符容錯排程的閘道 Pod。
- labels — Helm Chart 的節點選擇器會使用 hybrid-gateway-node: "true"標籤，將閘道 Pod 目標設為這些節點。
- nodeClassRef — 使用來源/目的地檢查組態將 NodePool 連結至 NodeClass。

受管節點群組

使用受管節點群組時，請為閘道建立專用節點群組，其中包含必要的標籤、污點和自訂啟動範本，以在啟動時停用來源/目的地檢查。

步驟 1：建立啟動範本

使用使用者資料建立啟動範本，以在執行個體開機時停用主要 ENI 的來源/目的地檢查：

```
# Create the launch template with user data to disable source/dest check
USERDATA=$(cat <<'SCRIPT' | base64 -w 0
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="==BOUNDARY=="

--==BOUNDARY==
Content-Type: text/x-shellscript; charset="us-ascii"

#!/bin/bash
TOKEN=$(curl -s -X PUT "http://169.254.169.254/latest/api/token" \
  -H "X-aws-ec2-metadata-token-ttl-seconds: 60")
MAC=$(curl -s -H "X-aws-ec2-metadata-token: $TOKEN" \
  http://169.254.169.254/latest/meta-data/mac)
ENI_ID=$(curl -s -H "X-aws-ec2-metadata-token: $TOKEN" \
  "http://169.254.169.254/latest/meta-data/network/interfaces/macs/${MAC}/interface-
id")
REGION=$(curl -s -H "X-aws-ec2-metadata-token: $TOKEN" \
  http://169.254.169.254/latest/meta-data/placement/region)

aws ec2 modify-network-interface-attribute \
```

```
  --network-interface-id "$ENI_ID" \
  --no-source-dest-check \
  --region "$REGION"

--==BOUNDARY==--
SCRIPT
)

aws ec2 create-launch-template \
  --launch-template-name YOUR_CLUSTER_NAME-gateway-lt \
  --launch-template-data "{\"UserData\":\"${USERDATA}\",\"MetadataOptions\":
{\"HttpTokens\":\"required\",\"HttpPutResponseHopLimit\":2}}"
```

> **Note**
>
> 節點的 IAM 角色必須具有ec2:ModifyNetworkInterfaceAttribute許可，使用者資料指令碼才能成功。MIME 分段格式可確保使用者資料與 EKS 引導指令碼一起執行。

步驟 2：建立受管節點群組

從步驟 1 建立具有閘道標籤、污點和啟動範本的專用受管節點群組：

```
aws eks create-nodegroup \
  --cluster-name YOUR_CLUSTER_NAME \
  --nodegroup-name YOUR_CLUSTER_NAME-gateway-nodes \
  --subnets SUBNET_ID_1 SUBNET_ID_2 \
  --node-role YOUR_NODE_ROLE_ARN \
  --instance-types INSTANCE_TYPE \
  --ami-type AL2023_x86_64_STANDARD \
  --scaling-config desiredSize=2,maxSize=2,minSize=2 \
  --labels hybrid-gateway-node=true \
  --taints "key=hybrid-gateway-node,effect=NO_SCHEDULE" \
  --launch-template "name=YOUR_CLUSTER_NAME-gateway-lt,version=1"
```

這會建立 2 節點受管節點群組，其中：

- `--labels` 會設定 ，hybrid-gateway-node=true讓 Helm Chart 的節點選擇器以這些節點為目標。
- `--taints` NoSchedule 新增了污點，因此只有閘道 Pod 在這些節點上具有相符的容錯排程。
- `--launch-template` 連接啟動範本，該範本會在開機時停用來源/目的地檢查。

在不同可用區域中使用子網路以獲得高可用性。

使用 Helm 安裝

EKS 自動模式

執行下列命令以使用 EKS Auto 模式安裝閘道：

```
helm install eks-hybrid-nodes-gateway \
  oci://public.ecr.aws/eks/eks-hybrid-nodes-gateway \
  --version 1.0.0 \
  --namespace eks-hybrid-nodes-gateway \
  --create-namespace \
  --set vpcCIDR=VPC_CIDR \
  --set podCIDRs=POD_CIDRS \
  --set routeTableIDs=ROUTE_TABLE_IDS
```

受管節點群組或自我管理節點

對於受管節點群組或自我管理節點，設定 `autoMode.enabled=false`：

```
helm install eks-hybrid-nodes-gateway \
  oci://public.ecr.aws/eks/eks-hybrid-nodes-gateway \
  --version 1.0.0 \
  --namespace eks-hybrid-nodes-gateway \
  --create-namespace \
  --set autoMode.enabled=false \
  --set vpcCIDR=VPC_CIDR \
  --set podCIDRs=POD_CIDRS \
  --set routeTableIDs=ROUTE_TABLE_IDS
```

必要的 Helm 值

所有安裝都需要下列值：

Value	說明
`vpcCIDR`	EKS 叢集 VPC 的 CIDR 區塊（例如 `10.0.0.0/16`）。用於 Cilium VTEP 組態，以便混合節點透過閘道路由 VPC 繫結流量。

Value	說明
podCIDRs	Cilium 在混合節點上使用的 Pod CIDRs 逗號分隔清單（例如 10.100.0.0/16,10.101.0.0/16 ）。閘道會為這些 CIDRs 建立 VPC 路由表項目和 VXLAN 路由。
routeTableIDs	要程式設計的 VPC 路由表 IDs 逗號分隔清單（例如 rtb-0abc1234def567890,rtb-0fed9876cba543210 ）。閘道會在這些資料表中建立將混合 Pod CIDRs 指向作用中閘道執行個體的路由。

如需可設定值的完整清單，請參閱 the section called “Configuration”。

驗證安裝

安裝閘道之後，請確認閘道執行正常。

檢查 Pod 狀態

確認兩個閘道 Pod 正在執行：

```
kubectl get pods -n eks-hybrid-nodes-gateway
```

您應該會看到類似以下的輸出：

```
NAME                                          READY   STATUS    RESTARTS   AGE
eks-hybrid-nodes-gateway-5d4f6a7b8c-abc12   1/1     Running   0          2m
eks-hybrid-nodes-gateway-5d4f6a7b8c-def34   1/1     Running   0          2m
```

檢查領導者選擇

確認一個 Pod 已取得領導者選擇租用：

```
kubectl get lease -n eks-hybrid-nodes-gateway
```

輸出會在 HOLDER欄中顯示目前的領導者：

```
NAME                     HOLDER                                          AGE
hybrid-gateway-leader    eks-hybrid-nodes-gateway-5d4f6a7b8c-abc12   2m
```

檢查運作狀態端點

驗證運作狀態端點是否使用連接埠轉送來回應領導 Pod：

```
kubectl port-forward -n eks-hybrid-nodes-gateway LEADER_POD_NAME 8088:8088 &
curl -s http://localhost:8088/healthz
```

運作狀態良好的閘道會傳回 HTTP 200 回應。

驗證 VPC 路由表項目

在 Amazon VPC 主控台或使用 AWS CLI 中，確認您的 VPC 路由表包含指向領導閘道執行個體 ENI 的混合式 Pod CIDRs 項目：

```
aws ec2 describe-route-tables \
  --route-table-ids ROUTE_TABLE_ID \
  --query "RouteTables[].Routes[?DestinationCidrBlock=='POD_CIDR']"
```

每個混合 Pod CIDR 都應有一個路由，並將 NetworkInterfaceId設定為領導執行個體的主要 ENI。

解除安裝

若要移除混合節點閘道，請執行：

```
helm uninstall eks-hybrid-nodes-gateway --namespace eks-hybrid-nodes-gateway
```

> **Note**
>
> 解除安裝 Helm Chart 不會自動移除閘道建立的 VPC 路由表項目。解除安裝後，從 VPC 路由表手動刪除混合 Pod CIDRs的路由，以避免將流量路由到不再執行閘道的執行個體。您可以使用 CLI AWS 移除路由：
>
> ```
> aws ec2 delete-route \
>   --route-table-id ROUTE_TABLE_ID \
>   --destination-cidr-block POD_CIDR
> ```

後續步驟

- the section called “Configuration” — 自訂 Helm 值、CLI 旗標和領導者選擇參數。
- the section called “作業” — 監控閘道、了解容錯移轉行為，以及規劃擴展。
- the section called “疑難排解” — 診斷並解決常見問題。

設定混合節點閘道的 CNI

Amazon EKS 混合節點閘道需要特定的 Cilium 組態，才能在 VPC 和混合節點之間啟用 VXLAN 型連線。您必須啟用 Cilium VTEP (VXLAN 通道端點） 支援並停用 L7 代理。

先決條件

- Cilium 已安裝在您的混合節點上。如需安裝指示，請參閱the section called “設定 CNI”。

> **Note**
>
> 僅下列最低 EKS Cilium 版本支援 Cilium VTEP：
>
> - Cilium 1.17.13-1 或更新版本
> - Cilium 1.18.8-1 或更新版本
> - Cilium 1.19.2-1 或更新版本
>
> 如果您執行的是較舊的版本，請在啟用 VTEP 之前升級 Cilium。如需升級說明，請參閱 the section called “設定 CNI”。

啟用 VTEP 和停用 L7 代理

設定 Cilium 版本並執行 `helm upgrade`命令以啟用 VTEP，並在您現有的 Cilium 安裝上停用 L7 代理。您可以在 Amazon ECR Public 找到最新的可用修補程式版本：eks/cilium/cilium。

```
helm upgrade cilium oci://public.ecr.aws/eks/cilium/cilium \
  --version CILIUM_VERSION \
  --namespace kube-system \
  --reuse-values \
  --set vtep.enabled=true \
  --set l7Proxy=false
```

升級完成後，重新啟動 Cilium DaemonSet 以套用新組態：

```
kubectl rollout restart daemonset/cilium -n kube-system
kubectl rollout status daemonset/cilium -n kube-system
```

必要設定

設定	Value	Description
vtep.enabled	true	啟用 Cilium 的 VXLAN 通道端點支援。這可讓混合節點上的 Cilium 代理程式封裝 VPC 繫結流量，並將其傳送至閘道。
l7Proxy	false	停用 Cilium 的 L7 代理。這是 VTEP 正常運作的必要項目。如果啟用 L7 代理，可能會攔截並捨棄 VTEP 流量。

驗證組態

升級 Cilium 之後，請確認 VTEP 已啟用且 L7 代理已停用：

```
kubectl get configmap cilium-config -n kube-system -o yaml | grep -E "enable-vtep|
enable-l7-proxy"
```

您應該會看到類似以下的輸出：

```
enable-l7-proxy: "false"
enable-vtep: "true"
```

設定混合 Pod 流量的 VPC CNI

根據預設， AWS VPC CNI 會將來源 NAT (SNAT) 套用至離開節點的所有 Pod 流量。為了確保具有混合 Pod 端點的 Kubernetes ClusterIP 服務能從雲端節點正常運作，請從 SNAT 中排除混合 Pod CIDRs。這可讓 kube-proxy DNAT 的流量正確使用 VPC 路由。

將 aws-node DaemonSet 上的AWS_VPC_K8S_CNI_EXCLUDE_SNAT_CIDRS環境變數設定為您的混合 Pod CIDR (s)：

```
kubectl set env daemonset aws-node -n kube-system \
  AWS_VPC_K8S_CNI_EXCLUDE_SNAT_CIDRS=POD_CIDRS
```

將值取代為您實際的混合 CIDRs （如果是多個，則以逗號分隔）。

此設定會將混合 Pod CIDRs 新增至 VPC CNI 的 ip 規則，讓目的地為混合 Pod 的流量透過主路由表路由，其中包含通往閘道 ENI 的 VPC 路由。

> **Note**
>
> 從雲端 Pod 到混合 Pod 端點的 ClusterIP 服務流量需要此步驟。依 IP 地址的直接 pod-to-pod 連線可在沒有此設定的情況下運作。

後續步驟

設定 Cilium 和 VPC CNI 之後，請繼續安裝混合節點閘道。請參閱the section called “開始使用”。

Amazon EKS 混合節點閘道組態參考

此頁面記錄 Amazon EKS 混合節點閘道的所有可設定參數。如需安裝指示，請參閱the section called “開始使用”。如需閘道的概觀，請參閱 the section called “混合節點閘道”。

Helm Chart 值

下表列出 `eks-hybrid-nodes-gateway` Helm Chart 中的所有值。您可以在 或 期間使用`--set`旗標`helm install`或自訂`values.yaml`檔案來設定這些值`helm upgrade`。

Value	Type	預設	必要	描述
`image.repository`	string	`public.ecr.aws/eks/eks-hybrid-nodes-gateway`	否	閘道映像的容器映像儲存庫。預設為公有 Amazon ECR 登錄檔。
`image.tag`	string	圖表`appVersion`	否	影像標籤。預設為 中`appVersion` 定義的 `Chart.yaml` 。

Value	Type	預設	必要	描述
image.pullPolicy	string	IfNotPresent	否	映像提取政策。有效值：Always、IfNotPresent 、Never。
replicas	integer	2	否	閘道 Pod 複本的數量。兩個複本是生產環境中高可用性的建議組態。
nodeLabel	string	hybrid-gateway-node	否	用於選取閘道節點的節點標籤金鑰。節點必須將此標籤設為 "true"。
vpcCIDR	string	""	是	用於 Cilium VTEP 組態的 VPC CIDR 區塊。範例：10.0.0.0/16 。
podCIDRs	string	""	是	Cilium 在混合節點上使用的 Pod CIDRs 逗號分隔清單。範例：10.85.0.0/16,10.86.0.0/16 。
routeTableIDs	string	""	是	以逗號分隔的 VPC 路由表 IDs 清單，以混合式 Pod 路由進行程式設計。範例：rtb-0123456789abcdef0,rtb-0123456789abcdef1 。
autoMode.enabled	布林值	true	否	啟用 EKS Auto Mode 整合。當時true，圖表會新增eks.amazonaws.com/compute-type: auto節點選擇器，並使用零停機時間滾動更新策略。針對受false管節點群組或自我管理節點，將 設定為 。

安裝命令範例

EKS 自動模式（預設）：

```
helm install eks-hybrid-nodes-gateway oci://public.ecr.aws/eks/eks-hybrid-nodes-gateway
 \
  --version 1.0.0 \
  --namespace eks-hybrid-nodes-gateway \
  --create-namespace \
  --set vpcCIDR=VPC_CIDR \
  --set podCIDRs=POD_CIDRS \
  --set routeTableIDs=ROUTE_TABLE_IDS
```

受管節點群組或自我管理節點：

```
helm install eks-hybrid-nodes-gateway oci://public.ecr.aws/eks/eks-hybrid-nodes-gateway
 \
  --version 1.0.0 \
  --namespace eks-hybrid-nodes-gateway \
  --create-namespace \
  --set autoMode.enabled=false \
  --set vpcCIDR=VPC_CIDR \
  --set podCIDRs=POD_CIDRS \
  --set routeTableIDs=ROUTE_TABLE_IDS
```

自動模式與受管節點群組組態

`autoMode.enabled` Helm 值控制部署範本中的兩種行為：

Behavior (行為)	**autoMode.enabled=true** (default)	**autoMode.enabled=false**
節點選擇器	除了閘道節點標籤之外，`eks.amazonaws.com/compute-type: auto`新增 。	僅使用閘道節點標籤 (`hybrid-gateway-node: "true"`)。
滾動更新策略	`maxSurge: 1`，`maxUnavailable: 0` — 在終止舊的 Pod 之前，先啟動新的 Pod，以確保升級期間不會停機。	`maxSurge: 0`，`maxUnavailable: 1` — 會在啟動新的 Pod 之前終止舊的 Pod，因為節點容量已預先佈建。

使用 EKS Auto Mode 時，您必須先建立 NodePool和 ，NodeClass再安裝 Helm Chart。NodePool 會使用正確的標籤、污點和來源/目的地檢查組態來佈建 EC2 執行個體。如需必要的 YAML，請參閱 the section called “開始使用”。使用受管節點群組或自我管理節點時，您必須在安裝圖表之前自行佈建和標記節點。

對於所有部署目標，部署會使用 hostNetwork: true和 需要 NET_ADMIN功能。Pod 反親和性可確保兩個閘道 Pod 在不同的節點上執行。

領導者選擇調校

閘道使用以 Kubernetes Lease 為基礎的領導者選擇來維護作用中待命模型。預設參數會針對快速容錯移轉進行調校：

參數	預設	Description
--leader-election-lease-duration	3s	非領導層在上次觀察到的租約續約後等待多久，再嘗試取得租約。較低的值可更快地偵測領導者故障，但在暫時性網路問題期間增加錯誤容錯移轉的風險。
--leader-election-renew-deadline	2s	主動領導者在放棄領導之前嘗試續約租約的時間上限。必須小於租用持續時間。
--leader-election-retry-period	1s	候選者重試取得或續約租約的頻率。

使用預設值時，作用中閘道失敗時的預期容錯移轉時間約為 3–5 秒。這包括待命偵測租用過期、取得租用，以及執行領導者設定序列 (VPC 路由表更新和 Cilium VTEP 組態） 的時間。

何時調整領導者選擇參數

- 如果您發現閘道 Pod 和 Kubernetes API 伺服器之間發生暫時性網路延遲，導致領導者頻繁轉換，請延長租用持續時間。

- 如果您需要更快的容錯移轉偵測，且閘道節點和 API 伺服器之間的網路可靠且低延遲，請縮短租用持續時間。
- 增加重試期間，以減少大型叢集中領導者選擇的 API 伺服器負載。

> **Note**
>
> `--leader-election-renew-deadline` 一律必須小於 `--leader-election-lease-duration`。如果續約期限超過租賃期間，領導者可能會在續約之前遺失租賃。

相關主題

- the section called “混合節點閘道”
- the section called “開始使用”
- the section called “作業”
- the section called “疑難排解”

Amazon EKS 混合節點閘道操作

此頁面涵蓋 Amazon EKS 混合節點閘道的第 2 天操作，包括高可用性、容錯移轉行為、監控、擴展和 VXLAN 通道生命週期。如需安裝指示，請參閱the section called “開始使用”。

高可用性和容錯移轉

混合節點閘道使用主動待命模型搭配以 Kubernetes Lease 為基礎的領導者選擇。兩個閘道 Pod 在單獨的 EC2 節點上執行，由 Pod 反親和性強制執行。兩個 Pod 都會在啟動時建立 VXLAN 界面，並執行節點調校程式，以維護所有混合節點的 VTEP 項目。只有領導者 Pod 會管理 VPC 路由表和 `CiliumVTEPConfig` CRD。待命 Pod 在容錯移轉時隨時準備好在 3-5 秒內轉送流量，因為它已經有一組完整的通道項目。

容錯移轉序列

當作用中閘道執行個體失敗時，會發生下列序列：

1. 待命 Pod 偵測到領導者租用已過期。
2. 待命 Pod 會取得租用，並成為新的領導者。
3. 新的領導者會執行領導者設定序列：

- 更新 VPC 路由表項目，將混合 Pod CIDRs 指向新領導者的主要 ENI。
- 使用新領導者的節點 IP 和 VXLAN MAC 地址 Upsert `CiliumVTEPConfig`自訂資源。

4. 流量會繼續流經新的領導者。

由於兩個 Pod 都會隨時維護 VXLAN 介面和 VTEP 項目，因此新領導者不需要在容錯移轉期間重新建立 VXLAN 介面或重新設定通道項目。只需要 VPC 路由表和`CiliumVTEPConfig`更新。

預期的容錯移轉時間約為 3–5 秒。在容錯移轉期間，VPC 和混合 Pod 之間的流量會中斷。

可用區域建議

將閘道節點分散到兩個可用區域，讓 AZ 失敗不會同時佔用領導者和待命。使用 EKS Auto 模式時，`NodeClass`使用跨多個 AZs子網路選擇器來設定您的 。對於受管節點群組或自我管理節點，請在標記節點時選擇不同AZs節點。

> **Note**
>
> 閘道與 VPC 中其他資源之間的跨可用區流量會產生標準 AWS 跨可用區資料傳輸費用。

領導者選擇參數

預設領導者選擇參數會針對快速容錯移轉進行調校：

參數	預設	Description
`--leader-election-lease-duration`	3s	在領導者停止續約後，非領導者嘗試取得租用前的等待時間。
`--leader-election-renew-deadline`	2s	領導者嘗試續約租約多久才放棄。
`--leader-election-`	1s	候選者重試取得租用的頻率。

參數	預設	Description
retry-period		

降低這些值可減少容錯移轉時間，但會增加網路分割區下錯誤容錯移轉的風險。對於大多數部署，預設值是適當的。如需詳細資訊，請參閱the section called “Configuration”。

VPC 路由表管理

閘道會管理 VPC 路由表項目，以便目的地為混合 Pod CIDRs流量到達作用中的閘道執行個體。

如何管理路由

當閘道 Pod 成為領導者時，它會在每個設定的 VPC 路由表中建立或取代路由。每個路由都會將目的地 CIDR 設定為混合 Pod CIDR，並將目標設定為領導者的主要 ENI。如果路由已存在並指向正確的 ENI，閘道會略過更新。

在容錯移轉期間，新領導者會取代現有的路由，使其指向自己的 ENI。這是將 VPC 流量重新導向至新作用中閘道的機制。

路由表項目範例

閘道設定路由之後，您的 VPC 路由表會包含類似下列的項目：

目標	Target	狀態
10.0.0.0/16	本機	active
HYBRID_POD_CIDR	*eni-LEADER_ENI_ID*	active

IAM 許可

閘道需要下列 IAM 動作來管理路由表：

- ec2:DescribeRouteTables
- ec2:CreateRoute
- ec2:ReplaceRoute
- ec2:DescribeInstances

將這些許可連接到與閘道節點的執行個體描述檔、Pod 身分或 IRSA 組態相關聯的 IAM 角色。

監控

運作狀態和準備程度端點

閘道會在連接埠 上公開運作狀態和準備程度端點8088：

Endpoint	路徑	Description
運作狀態檢查	`/healthz`	當閘道程序運作狀態良好時，傳回 HTTP 200。供 Kubernetes 即時性探查使用。
準備度檢查	`/readyz`	當閘道準備好提供流量時，傳回 HTTP 200。供 Kubernetes 整備探查使用。

您可以執行暫時偵錯容器或連接埠轉送，手動查詢這些端點以進行診斷：

```
kubectl port-forward -n eks-hybrid-nodes-gateway POD_NAME 8088:8088 &
curl -s http://localhost:8088/healthz
curl -s http://localhost:8088/readyz
```

指標端點

閘道會在`/metrics`路徑`10080`的連接埠上公開 Prometheus 相容指標。除了標準控制器執行期指標之外，還提供下列自訂指標。

閘道資訊：

指標	Type	說明
`hybrid_gateway_info`	量測計	閘道執行個體的靜態資訊。一律為 1。標籤：`node_ip`、`node_name` 、`vxlan_interface` 、`vpc_cidr`、`pod_cidr`。

混合節點：

指標	Type	說明
hybrid_gateway_hybrid_nodes_configured	量測計	已設定 VTEP 項目的混合節點目前數量。

VTEP 操作：

指標	Type	說明
hybrid_gateway_vtep_add_total	計數器	成功的 VTEP 新增操作總數。
hybrid_gateway_vtep_add_errors_total	計數器	失敗的 VTEP 新增操作總數。
hybrid_gateway_vtep_remove_total	計數器	成功的 VTEP 移除操作總數。
hybrid_gateway_vtep_remove_errors_total	計數器	失敗的 VTEP 移除操作總數。

領導者選擇和路由表：

指標	Type	說明
hybrid_gateway_leader_is_active	量測計	如果此 Pod 是作用中的領導者，則為 1；如果為待命，則為 0。
hybrid_gateway_leader_setup_duration_seconds	直方圖	領導者設定操作（路由表 + CiliumVTEPConfig) 的持續時間，以秒為單位。

指標	Type	說明
hybrid_gateway_aws_route_table_update_total	計數器	總成功 AWS 路由表更新操作。
hybrid_gateway_aws_route_table_update_errors_total	計數器	失敗的 AWS 路由表更新操作總數。
hybrid_gateway_aws_route_table_update_duration_seconds	直方圖	AWS 路由表更新操作的持續時間，以秒為單位。

網路統計資料（每個湊集的隨需收集）：

指標	Type	說明
hybrid_gateway_vxlan_rx_bytes_total	量測計	VXLAN 界面上接收的總位元組數。
hybrid_gateway_vxlan_tx_bytes_total	量測計	在 VXLAN 界面上傳輸的總位元組數。
hybrid_gateway_vxlan_rx_packets_total	量測計	VXLAN 界面上收到的封包總數。
hybrid_gateway_vxlan_tx_packets_total	量測計	在 VXLAN 界面上傳輸的總封包數。
hybrid_gateway_vxlan_rx_dropped_total	量測計	在 VXLAN 界面接收時捨棄的封包總數。
hybrid_gateway_vxlan_tx_dropped_total	量測計	VXLAN 介面傳輸時捨棄的封包總數。
hybrid_gateway_vxlan_rx_errors_total	量測計	VXLAN 界面上的總接收錯誤。

指標	Type	說明
hybrid_gateway_vxlan_tx_errors_total	量測計	VXLAN 界面上的傳輸錯誤總數。
hybrid_gateway_vxlan_interface_up	量測計	如果 VXLAN 介面為 UP，則為 1，否則為 0。
hybrid_gateway_vxlan_fdb_entries	量測計	VXLAN 界面上 FDB 項目的目前數量。
hybrid_gateway_vxlan_route_count	量測計	透過 VXLAN 界面的目前路由數。
hybrid_gateway_primary_nic_rx_bytes_total	量測計	在主要網路界面上接收的總位元組數。
hybrid_gateway_primary_nic_tx_bytes_total	量測計	在主要網路界面上傳輸的總位元組數。
hybrid_gateway_primary_nic_rx_packets_total	量測計	在主要網路介面上收到的封包總數。
hybrid_gateway_primary_nic_tx_packets_total	量測計	在主要網路介面上傳輸的總封包數。
hybrid_gateway_primary_nic_rx_dropped_total	量測計	在主要 NIC 接收時捨棄的封包總數。
hybrid_gateway_primary_nic_tx_dropped_total	量測計	主要 NIC 在傳輸時捨棄的封包總數。

指標	Type	說明
hybrid_gateway_primary_nic_rx_errors_total	量測計	主要 NIC 的總接收錯誤。
hybrid_gateway_primary_nic_tx_errors_total	量測計	主要 NIC 上的傳輸錯誤總數。
hybrid_gateway_primary_nic_info	量測計	主要 NIC 名稱。一律為 1。標籤：interface_name 。

CloudWatch 可觀測性附加元件

您可以使用 Amazon CloudWatch Observability 附加元件來收集閘道指標和日誌。設定附加元件以抓取連接埠 上的閘道命名空間 (eks-hybrid-nodes-gateway)10080。如需正確的組態格式，請參閱上面連結的附加元件文件。

擴展考量

混合節點閘道使用主動待命模型搭配領導者選擇，因此只有一個 Pod 會在任何指定時間處理流量。水平擴展閘道 （透過增加複本數量） 可以提供可在容錯移轉期間接管的額外待命 Pod，但它不會改善效能或輸送量，因為流量不會分散到複本。若要擴展效能，請為您的流量選擇具有足夠網路頻寬的 EC2 執行個體類型，以垂直擴展。

執行個體類型指引

閘道輸送量受限於 EC2 執行個體網路效能。選取執行個體類型時，請考慮下列事項：

- 網路頻寬 — 閘道會轉送 VPC 和混合 Pod 之間的所有流量。選擇網路頻寬符合您尖峰流量需求的執行個體類型。
- 每秒封包數 (PPS) — VXLAN 封裝會增加每個封包的額外負荷。具有許多小型封包的工作負載 （例如，具有高請求率的微服務） 受益於具有較高 PPS 限制的執行個體類型。
- 混合節點數量 — 每個混合節點都會新增 VXLAN 通道端點，閘道會透過此端點轉送流量。隨著混合節點數量的擴展，透過閘道的彙總流量也會增加。選取具有足夠網路頻寬的執行個體類型，以處理叢集的尖峰跨網路流量。

建議的執行個體類型

生產 (10–100 個混合節點、中等流量）

適用於具有穩定跨網路流量的標準生產工作負載。

執行個體類型	vCPUs	記憶體	網路	備註
c6i.xlarge	4	8 GiB	最高 12.5 Gbps	成本和效能的良好平衡
c6in.xlarge	4	8 GiB	高達 30 Gbps	網路最佳化；建議用於生產
c7i.xlarge	4	8 GiB	最高 12.5 Gbps	最新一代的運算最佳化
m6i.xlarge	4	16 GiB	最高 12.5 Gbps	適合在閘道節點上同時放置其他工作負載時

高輸送量生產（超過 100 個混合節點、大量流量）

對於具有重大跨網路頻寬需求的環境，例如資料密集型工作負載或許多並行連線。

執行個體類型	vCPUs	記憶體	網路	備註
c6in.2xlarge	8	16 GiB	高達 40 Gbps	建議用於高輸送量生產
c5n.2xlarge	8	21 GiB	最高 25 Gbps	上一代網路最佳化、經濟實惠
c6in.4xlarge	16	32 GiB	最高 50 Gbps	非常繁重的工作負載的最大輸送量
c5n.4xlarge	16	42 GiB	最高 25 Gbps	極高封包速率的高 vCPU 計數

使用閘道指標監控網路使用率（請參閱the section called “指標端點”），並視需要調整執行個體類型。

VXLAN 通道生命週期

閘道會在混合節點加入或離開叢集時，自動維護混合節點的 VXLAN 通道。

如何管理通道

節點控制器會監看叢集中的`CiliumNode`物件。控制器會在每個閘道 Pod（不只是領導者）上執行，因此領導者和待命都具有up-to-date通道狀態。`CiliumNode` 事件發生時，控制器會尋找`eks.amazonaws.com/compute-type: hybrid`標籤來檢查節點是否為混合節點。

當混合節點加入叢集時：

1. 控制器會偵測新的`CiliumNode`物件。
2. 它會從`CiliumNode`規格擷取節點的內部 IP 地址和 Pod CIDR。
3. 它會在 VXLAN 界面上設定下列項目：
   - 透過 VXLAN 界面透過節點 IP 的節點 Pod CIDR 路由。
   - 將節點 IP 映射至決定性 MAC 地址的靜態 ARP 項目。
   - 告知 VXLAN 模組將封裝封包傳送至節點 IP 的 FDB 項目。

當混合節點離開叢集時：

1. 控制器會偵測`CiliumNode`刪除。
2. 它會從 VXLAN 介面移除該節點的路由、ARP 項目和 FDB 項目。

此生命週期是全自動的。新增或移除混合節點時，您不需要手動設定通道。

後續步驟

- the section called “Configuration” — 自訂 Helm 值、CLI 旗標和領導者選擇參數。
- the section called “疑難排解” — 診斷並解決常見問題。
- the section called “混合節點閘道” — 返回概觀頁面。

Amazon EKS 混合節點閘道疑難排解

此頁面提供診斷和解決 Amazon EKS 混合節點閘道常見問題的指引。每個區段都會描述症狀、可能的原因、診斷步驟和解決方案。如需操作詳細資訊，請參閱 the section called “作業”。

無法從 VPC 連線混合節點上的 Pod

混合節點上執行的 Pod 無法從 VPC 中的資源連線，例如 EC2 執行個體、負載平衡器或 Kubernetes 控制平面。

可能原因：

- VPC 路由表項目遺失或指向錯誤的 ENI。
- 閘道領導者 Pod 未執行或未完成設定。
- 混合節點上未啟用或設定 Cilium VTEP。
- 閘道 EC2 執行個體上已啟用來源/目的地檢查。

診斷步驟：

1. 檢查 VPC 路由表項目。確認混合 Pod CIDRs路由是否存在，並指向作用中閘道執行個體的主要 ENI：

   ```
   aws ec2 describe-route-tables \
     --route-table-ids ROUTE_TABLE_ID \
     --query "RouteTables[].Routes[?DestinationCidrBlock=='POD_CIDR']"
   ```

   如果路由遺失，請檢查閘道日誌是否有路由表錯誤。如果路由指向錯誤的 ENI，則容錯移轉可能尚未成功完成。
2. 檢查閘道 Pod 狀態和領導者選擇。確認兩個閘道 Pod 正在執行中，一個保留領導者租用：

   ```
   kubectl get pods -n eks-hybrid-nodes-gateway
   kubectl get lease -n eks-hybrid-nodes-gateway
   ```

   如果沒有 Pod 保留租用，請參閱 the section called “領導者選擇問題”。
3. 檢查混合節點上的 Cilium VTEP 組態。驗證`CiliumVTEPConfig`資源是否存在，並包含領導者的節點 IP：

   ```
   kubectl get ciliumvtepconfig hybrid-gateway -o yaml
   ```

   `spec.endpoints[0].tunnelEndpoint` 應該符合領導閘道節點的 IP 地址。如果資源遺失或 IP 過時，閘道可能尚未完成領導者設定。
4. 檢查來源/目的地檢查。確認閘道 EC2 執行個體上的來源/目的地檢查已停用：

```
aws ec2 describe-instance-attribute \
  --instance-id GATEWAY_INSTANCE_ID \
  --attribute sourceDestCheck
```

如果 `sourceDestCheck`是 `true`，請將其停用。請參閱 the section called “開始使用”。

Webhook 對混合節點的呼叫失敗

Kubernetes API 伺服器無法連線到在混合節點上執行的 Webhook 端點。Webhook 許可請求逾時或傳回連線錯誤。

可能原因：

- 閘道不會將流量從控制平面路由到混合式 Pod。
- `CiliumVTEPConfig` 資源遺失或端點 IP 過時。

診斷步驟：

1. 確認控制平面可以到達閘道節點 IP。控制平面會將流量傳送至 VPC 路由表，再將其轉送至閘道的 ENI。使用 中的步驟，確認 VPC 路由表項目正確無誤the section called “無法從 VPC 連線混合節點上的 Pod”。
2. 檢查 CiliumVTEPConfig 資源。驗證資源是否存在，且 `tunnelEndpoint`符合目前領導者的節點 IP：

   ```
   kubectl get ciliumvtepconfig hybrid-gateway -o yaml
   ```

   如果通道端點過時（指向先前的領導者），閘道可能尚未完成領導者設定序列。檢查閘道日誌在 `CiliumVTEPConfig` upsert 期間是否有錯誤。

VPC 路由表更新失敗

閘道日誌會顯示與 VPC 路由表操作相關的錯誤，而且混合 Pod CIDRs路由不會建立或更新。

可能原因：

- 閘道的 IAM 角色沒有必要的 EC2 許可。
- 組態中的路由表 IDs 不正確或路由表不存在。

- 閘道無法連線到 EC2 API 端點。

診斷步驟：

1. 驗證 IAM 許可。閘道需要下列 IAM 動作：
   - `ec2:DescribeRouteTables`
   - `ec2:CreateRoute`
   - `ec2:ReplaceRoute`
   - `ec2:DescribeInstances`

   檢查連接至閘道節點執行個體描述檔或 Pod 身分組態的 IAM 角色。
2. 檢查組態中的路由表 IDs。確認`ROUTE_TABLE_IDS`環境變數在閘道部署中包含有效的路由表 IDs：

   ```
   kubectl get deployment eks-hybrid-nodes-gateway -n eks-hybrid-nodes-gateway -o
    jsonpath='{.spec.template.spec.containers[0].env}' | jq .
   ```

   確認您的 VPC 中存在路由表 IDs：

   ```
   aws ec2 describe-route-tables --route-table-ids ROUTE_TABLE_ID
   ```

3. 檢查閘道日誌是否有路由表錯誤。尋找與路由表操作相關的錯誤訊息：

   ```
   kubectl logs -n eks-hybrid-nodes-gateway LEADER_POD | grep -i "route table"
   ```

   常見的錯誤訊息包括：
   - `Failed to verify route table access` — 閘道無法描述路由表。檢查 IAM 許可和路由表 IDs。
   - `Failed to update route tables` — 閘道無法建立或取代路由。檢查 IAM 許可。
   - `failed to access route table` — 路由表 ID 可能不正確或 IAM 角色缺少 `ec2:DescribeRouteTables`。

閘道 Pod 無法啟動或運作狀態不佳

閘道 Pod 處於 `CrashLoopBackOff`、 `Error`或 `Pending` 狀態，或運作狀態端點傳回錯誤。

可能原因：

- 未設定必要的環境變數 (VPC_CIDR、POD_CIDRS、ROUTE_TABLE_IDS)。
- 閘道節點上未啟用 IP 轉送。
- 節點標籤或反親和性限制會阻止排程。

診斷步驟：

1. 檢查 Pod 日誌。檢視失敗 Pod 的日誌，以識別錯誤：

```
kubectl logs -n eks-hybrid-nodes-gateway LEADER_POD
```

2. 檢查必要的環境變數。閘道需要 NODE_IP、 VPC_CIDR和 POD_CIDRS。如果有任何遺失，閘道會立即結束。驗證 Pod 規格：

```
kubectl get pod -n eks-hybrid-nodes-gateway LEADER_POD -o
 jsonpath='{.spec.containers[0].env}' | jq .
```

   - NODE_IP 在 Pod 規格status.hostIP中自動從 設定 。如果為空，可能尚未在節點上排程 Pod。
   - VPC_CIDR 和 POD_CIDRS 來自 Helm 值。確認設定正確。

3. 檢查 IP 轉送。閘道會在啟動時檢查 IP 轉送是否已啟用，如果未啟用則會結束。在 Pod 日誌IP forwarding is not enabled中尋找錯誤訊息。在節點上啟用 IP 轉送：

```
# Check current setting
cat /proc/sys/net/ipv4/ip_forward

# Enable if not set
sudo sysctl -w net.ipv4.ip_forward=1
```

   對於持久性設定，請透過 kubelet 設定 IP 轉送或net.ipv4.ip_forward=1新增至 /etc/sysctl.d/。

4. 檢查節點標籤和排程限制。閘道 Pod 需要具有 hybrid-gateway-node=true標籤的節點。Pod 反親和性可確保每個 Pod 在個別節點上執行。如果 Pod 是 Pending，請檢查排程問題：

```
kubectl describe pod -n eks-hybrid-nodes-gateway LEADER_POD
```

尋找指出節點不足、缺少標籤或反親和性衝突的事件。

領導者選擇問題

閘道 Pod 正在執行，但沒有任何 Pod 會取得領導者租用，或經常發生領導轉換。

可能原因：

- 缺少租用物件的 RBAC 許可。
- 閘道 Pod 和 Kubernetes API 伺服器之間的網路連線不可靠。
- 領導者選擇參數設定錯誤。

診斷步驟：

1. 檢查租用物件。驗證租用是否存在，並檢查其目前持有者：

   ```
   kubectl get lease -n eks-hybrid-nodes-gateway hybrid-gateway-leader -o yaml
   ```

   `spec.holderIdentity` 欄位會顯示目前的領導者。`spec.renewTime` 會顯示上次續約的時間。如果 `renewTime` 過時，領導者可能會失去與 API 伺服器的連線。
2. 檢查 RBAC 許可。閘道服務帳戶需要許可，才能取得、建立和更新閘道命名空間中的租用物件。驗證角色和 RoleBinding：

   ```
   kubectl get role -n eks-hybrid-nodes-gateway
   kubectl get rolebinding -n eks-hybrid-nodes-gateway
   ```

   角色應包含 `coordination.k8s.io` API 群組中`leases`資源的 `get`、`create`、 `update` 和動詞。
3. 檢查 Pod 日誌是否有租用錯誤。在 Pod 日誌中尋找領導者選擇錯誤：

   ```
   kubectl logs -n eks-hybrid-nodes-gateway LEADER_POD | grep -i "leader\|lease"
   ```

   常見問題包括：

   - `Failed to acquire lease` — Pod 無法建立或更新租用物件。檢查 RBAC 許可。
   - 經常`Leadership ended`跟隨`Leader setup complete`訊息 — 領導者遺失並重新取得租用。這可能表示 Pod 和 API 伺服器之間的網路不穩定。考慮增加 `--leader-election-lease-duration`。

4. 檢查領導者選擇參數。驗證設定的值：

```
kubectl get deployment eks-hybrid-nodes-gateway -n eks-hybrid-nodes-gateway -o
 jsonpath='{.spec.template.spec.containers[0].args}'
```

確保 --leader-election-renew-deadline小於 --leader-election-lease-duration。如果續約期限超過租賃期間，領導者會失去租賃，然後才能續約。如需詳細資訊，請參閱the section called “領導者選擇調校”。

常見錯誤訊息

下表列出您可能會在閘道 Pod 日誌中看到的錯誤訊息及其解決方法。

錯誤訊息	原因	Resolution
IP forwarding is not enabled	核心參數net.ipv4.ip_forward 未在閘道節點1上設定為 。	透過 kubelet 組態或執行 來啟用 IP 轉送sysctl -w net.ipv4.ip_forward=1 。
Failed to setup VXLAN	閘道無法建立 VXLAN 網路介面。這通常發生在 Pod 缺少 NET_ADMIN 功能時。	驗證 部署規格是否包含在 NET_ADMIN 中securityContext.capabilities.add 。檢查 Helm Chart 是否正確部署。
Failed to verify route table access	閘道無法在啟動時描述一或多個 VPC 路由表。	確認 IAM 角色具有ec2:DescribeRouteTables 許可，且組態中的路由表 IDs 正確。
Failed to update route tables	閘道無法在 VPC 路由表中建立或取代路由。	驗證 IAM 角色具有 ec2:CreateRoute 和 ec2:ReplaceRoute 許可。
Failed to create route table manager	閘道無法初始化 AWS EC2 用戶端或擷取執行個體的主要 ENI。	確認 IAM 角色具有 ec2:DescribeInstances 許可，且可存取執行個體中繼資料服務 (IMDS)。
NODE_IP is required	未設定NODE_IP環境變數或--node-ip 旗標。	status.hostIP 使用 驗證NODE_IP來自 的 Pod 規格

錯誤訊息	原因	Resolution
		集fieldRef。檢查 Helm Chart 是否正確部署。
Invalid NODE_IP	為 提供的值NODE_IP不是有效的 IP 地址。	檢查 Pod 規格中的NODE_IP環境變數值。
pod-cidrs and vpc-cidr are required	POD_CIDRS 或 VPC_CIDR環境變數為空。	在安裝期間設定 podCIDRs和 vpcCIDR Helm 值。
No valid route table IDs provided	ROUTE_TABLE_IDS 值已設定，但在剖析後不包含有效的路由表 IDs。	檢查 routeTableIDs Helm 值是否有格式錯誤。路由表 IDs應以逗號分隔 （ 例如 rtb-abc123,rtb-def456 ）。
Failed to auto-detect AWS region	閘道無法從 EC2 執行個體中繼資料擷取 AWS 區域。	驗證執行個體中繼資料服務 (IMDS) 是否可存取。或者，明確設定--aws-region 旗標或AWS_REGION 環境變數。
Failed to auto-detect AWS instance ID	閘道無法從 EC2 執行個體中繼資料擷取執行個體 ID。	驗證執行個體中繼資料服務 (IMDS) 是否可存取。或者，明確設定--aws-instance-id 旗標或AWS_INSTANCE_ID 環境變數。
CiliumNode has no internal IP	混合節點的CiliumNode 物件在其規格中沒有內部 IP 地址。	確認混合節點已正確註冊，且 Cilium 代理程式正在執行。檢查節點CiliumNode 的資源。
CiliumNode <name> has no pod CIDRs allocated	混合節點的CiliumNode 物件沒有由 Cilium IPAM 配置的 Pod CIDRs。	確認混合節點上的 Cilium IPAM 設定正確。檢查CiliumNode 資源是否有節點的 IPAM 狀態。
Failed to upsert CiliumVTEPConfig	閘道無法建立或更新 CiliumVTEPConfig 自訂資源。	確認 CRD 已安裝在叢集中，且閘道服務帳戶具有管理 CiliumVTEPConfig 資源的許可。

錯誤訊息	原因	Resolution
`Unable to create manager`	控制器執行期管理員無法初始化。	檢查 Pod 日誌以取得其他內容。常見原因包括 kubeconfig 無效或無法連線到 Kubernetes API 伺服器。
`Failed to add gateway setup`	領導者選擇的可執行檔無法向控制器管理員註冊。	這通常是內部錯誤。檢查完整的 Pod 日誌以取得其他內容，並在 GitHub 儲存庫上報告問題。
`Unable to create Node controller`	CiliumNode 對帳程式無法向控制器管理員註冊。	檢查 Pod 日誌以取得其他內容。確認叢集中已安裝 CiliumNode CRD。
`Problem running manager`	控制器管理員意外結束。	檢查 Pod 日誌是否有基礎錯誤。常見原因包括 Kubernetes API 伺服器的連線中斷，或指標或運作狀態探查繫結地址發生連接埠衝突。
`failed to access route table <id>`	在啟動驗證檢查期間，閘道無法描述特定的 VPC 路由表。	確認 IAM 角色具有`ec2:DescribeRouteTables` 許可，且路由表 ID 正確。路由表必須存在於與閘道執行個體相同的區域中。

相關主題

- the section called “混合節點閘道” — 閘道架構和使用案例的概觀。
- the section called “開始使用” — 先決條件和安裝指示。
- the section called “Configuration” — Helm 值、CLI 旗標和環境變數的完整參考。
- the section called “作業” — 監控、容錯移轉行為和擴展指引。

針對您的叢集使用應用程式資料儲存

您可以搭配 Amazon EKS 使用一系列 AWS 儲存服務，以滿足應用程式的儲存需求。透過容器儲存介面 (CSI) 驅動程式的 AWS支援廣度，您可以輕鬆使用 Amazon EBS、Amazon S3、Amazon S3 檔案、Amazon EFS、Amazon FSx 和 Amazon File Cache，以滿足在 Amazon EKS 上執行之應用程式的儲存需求。若要管理 Amazon EKS 叢集的備份，請參閱 AWS Amazon EKS 的備份支援。

本章涵蓋 Amazon EKS 叢集的儲存選項。

主題

- 透過 Amazon EBS 使用 Kubernetes 磁碟區儲存
- 適用於 Kubernetes 磁碟區儲存的 Amazon EC2 執行個體存放區
- 透過 Amazon EFS 使用彈性檔案系統儲存
- 搭配 Amazon EFS CSI 驅動程式使用 Amazon S3 檔案系統儲存 EFS
- 搭配 Amazon FSx for Lustre 使用高效能應用程式儲存
- 搭配 FSx for NetApp ONTAP 使用高效能應用程式儲存
- 搭配 Amazon FSx for OpenZFS 使用資料儲存
- 使用 Amazon File Cache 將延遲降到最低
- 使用適用於 Amazon S3 的掛載點 CSI 驅動程式存取 Amazon S3 物件
- 啟用 CSI 磁碟區的快照功能

透過 Amazon EBS 使用 Kubernetes 磁碟區儲存

> **Note**
>
> 新功能：Amazon EKS 自動模式現可自動執行區塊儲存的常規任務。了解如何 the section called “部署具狀態工作負載”。

Amazon Elastic Block Store (Amazon EBS) 容器儲存介面 (CSI) 驅動程式可管理 Amazon EBS 磁碟區的生命週期，作為您所建立 Kubernetes 磁碟區的儲存空間。Amazon EBS CSI 驅動程式為以下類型的 Kubernetes 磁碟區製作 Amazon EBS 磁碟區：一般暫時性磁碟區與持續性磁碟區。

考量事項

- 您不需要在 EKS 自動模式叢集上安裝 Amazon EBS CSI 控制器。
- 您無法將 Amazon EBS 磁碟區掛載到 Fargate Pod。
- 您可以在 Fargate 節點上執行 Amazon EBS CSI 控制器，但是 Amazon EBS CSI 節點 `DaemonSet` 僅能在 Amazon EC2 執行個體上執行。
- Amazon EBS 磁碟區及 Amazon EBS CSI 驅動程式與 Amazon EKS 混合節點不相容。
- 將為最新的附加元件版本和一個先前版本提供支援。最新版本中發現的錯誤或漏洞修正會以新的次要版本形式回溯至先前的版本。
- EKS 自動模式需要使用儲存類別 `ebs.csi.eks.amazonaws.com` 作為佈建程式。標準 Amazon EBS CSI 驅動程式 (`ebs.csi.aws.com`) 會單獨管理其自己的磁碟區。要將現有磁碟區與 EKS 自動模式一起使用，請使用磁碟區快照將其移轉到使用自動模式佈建程式的儲存類別。

> **Important**
>
> 要使用 Amazon EBS CSI 驅動程式的快照功能，您必須先安裝 CSI 快照控制器。如需詳細資訊，請參閱the section called “CSI 快照器控制器”。

先決條件

- 現有的叢集。若要查看所需的平台版本，請執行下列命令。

```
aws eks describe-addon-versions --addon-name aws-ebs-csi-driver
```

- EBS CSI 驅動程式需要 AWS IAM 許可。
  - AWS 建議使用 EKS Pod 身分。如需詳細資訊，請參閱the section called “設定 EKS Pod 身分識別的概觀”。
  - 如需有關服務帳戶 IAM 角色的詳細資訊，請參閱 the section called “IAM OIDC 提供商”。

步驟 1：建立 IAM 角色

Amazon EBS CSI 外掛程式需要 IAM 許可，才能代表您呼叫 AWS APIs。如果您沒有完成這些步驟，則嘗試安裝附加元件並執行 `kubectl describe pvc` 將會顯示 `failed to provision volume`

`with StorageClass` 以及 `could not create volume in EC2: UnauthorizedOperation` 錯誤。如需詳細資訊，請參閱 GitHub 上的設定驅動程式許可。

> **Note**
>
> Pod 可以存取指派給 IAM 角色的許可，除非您封鎖對 IMDS 的存取。如需詳細資訊，請參閱the section called “最佳實務”。

下列程序說明如何建立 IAM 角色，並將 AWS 受管政策連接至該角色。若要實作此程序，您可使用下列其中一個工具：

- the section called “`eksctl`”
- the section called “AWS 管理主控台”
- the section called “AWS CLI”

> **Note**
>
> 您可以建立具有進一步縮小範圍許可的自我管理政策。檢閱`AmazonEBSCSIDriverPolicyV2`並建立具有較低許可的自訂 IAM 政策。如果從 遷移`AmazonEBSCSIDriverPolicy`，請參閱 EBS CSI 驅動程式政策遷移。

> **Note**
>
> 本程序中的特定步驟是針對將驅動程式用作 Amazon EKS 附加元件而編寫。需要執行不同的步驟，才能將驅動程式用作自我管理附加元件。如需詳細資訊，請參閱 GitHub 上的設定驅動程式許可。

`eksctl`

1. 建立 IAM 角色並連接政策。 AWS 維護 AWS 受管政策，或者您可以建立自己的自訂政策。您可以使用下列命令建立 IAM 角色並連接 AWS 受管政策。使用您叢集的名稱取代 *`my-cluster`*。命令會部署建立 IAM 角色並將 IAM 政策連接至其中的 AWS CloudFormation 堆疊。

```
eksctl create iamserviceaccount \
```

```
        --name ebs-csi-controller-sa \
        --namespace kube-system \
        --cluster my-cluster \
        --role-name AmazonEKS_EBS_CSI_DriverRole \
        --role-only \
        --attach-policy-arn arn:aws: iam::aws:policy/service-role/
AmazonEBSCSIDriverPolicyV2 \
        --approve
```

2. 如果您不使用自訂 KMS 金鑰，可以跳過此步驟。如果在 Amazon EBS 磁碟區上使用一個進行加密，請視需要自訂 IAM 角色。例如，請執行以下操作：

   a. 複製以下程式碼並貼到新 kms-key-for-encryption-on-ebs.json 檔案中。將 *custom-key-arn* 取代為自訂 KMS 金鑰 ARN。

```
{
      "Version":"2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Action": [
            "kms:CreateGrant",
            "kms:ListGrants",
            "kms:RevokeGrant"
          ],
          "Resource": ["arn:aws:kms:us-
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"],
          "Condition": {
            "Bool": {
              "kms:GrantIsForAWSResource": "true"
            }
          }
        },
        {
          "Effect": "Allow",
          "Action": [
            "kms:Encrypt",
            "kms:Decrypt",
            "kms:ReEncrypt*",
            "kms:GenerateDataKey*",
            "kms:DescribeKey"
          ],
          "Resource": ["arn:aws:kms:us-
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"]
```

```
        }
      ]
    }
```

b. 建立政策。您可將 *KMS_Key_For_Encryption_On_EBS_Policy* 變更為不同的名稱。但是，如果您這樣做，請務必在稍後的步驟中也進行變更。

```
aws iam create-policy \
      --policy-name KMS_Key_For_Encryption_On_EBS_Policy \
      --policy-document file://kms-key-for-encryption-on-ebs.json
```

c. 使用以下命令將 IAM 政策連接至角色。使用您的帳戶 ID 取代 *111122223333*。

```
aws iam attach-role-policy \
      --policy-arn arn:aws: iam::111122223333:policy/
KMS_Key_For_Encryption_On_EBS_Policy \
      --role-name AmazonEKS_EBS_CSI_DriverRole
```

AWS 管理主控台

1. 前往 https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。
2. 在左側導覽窗格中，選擇 Roles (角色)。
3. 在 Roles (角色) 頁面上，選擇 Create role (建立角色)。
4. 在 Select trusted entity (選取信任的實體) 頁面上，執行以下作業：
   a. 在 Trusted entity type (信任的實體類型) 區段中，選擇 Web identity (Web 身分)。
   b. 對於 Identity provider (身分提供者)，為您的叢集選擇 OpenID Connect provider URL (OpenID Connect 供應商 URL)(如 Amazon EKS Overview (概觀) 下所示)。
   c. 針對 Audience (對象)，選擇 `sts.amazonaws.com`。
   d. 選擇下一步。
5. 在 Add permissions (新增許可) 頁面上，執行以下作業：
   a. 在 Filter policies (篩選政策) 方塊中，輸入 `AmazonEBSCSIDriverPolicyV2`。
   b. 勾選在搜尋中傳回的 `AmazonEBSCSIDriverPolicyV2` 左側的核取方塊。
   c. 選擇下一步。
6. 在 Name, review, and create (命名、檢閱和建立) 頁面上，執行以下作業：
   a. 針對角色名稱，為您的角色輸入唯一名稱 (例如 *AmazonEKS_EBS_CSI_DriverRole*)。

b. 藉由連接標籤做為鍵值對，在新增標籤 (選用) 下將中繼資料新增至角色。如需有關在 IAM 中使用標籤的詳細資訊，請參閱《IAM 使用者指南》中的標記 IAM 資源。

c. 選擇建立角色。

7. 建立角色之後，在主控台中選擇角色，以開啟角色進行編輯。

8. 選擇 Trust Relationships (信任關係) 標籤，然後選擇 Edit Trust Relationship (編輯信任政策)。

9. 查找與下列行相似的行：

```
"oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE:aud":
 "sts.amazonaws.com"
```

在上一行的末尾新增一個逗號，然後在上一行之後新增下一行。將 *region-code* 取代為您的叢集所在的 AWS 區域。使用叢集的 OIDC 提供商 ID 取代 *EXAMPLED539D4633E53DE1B71EXAMPLE*。

```
"oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE:sub":
 "system:serviceaccount:kube-system:ebs-csi-controller-sa"
```

10選擇 Update policy (更新政策) 以完成操作。

11如果在 Amazon EBS 磁碟區上使用自訂 KMS 金鑰進行加密，請視需要自訂 IAM 角色。例如，請執行以下操作：

a. 在左側導覽窗格中選擇 Policies (政策)。

b. 在 Policies (政策) 頁面上，選擇 Create a policy (建立政策)。

c. 在建立政策頁面上，選擇 JSON 標籤。

d. 複製以下程式碼並貼到編輯器中，將 *custom-key-arn* 取代為自訂 KMS 金鑰 ARN。

```
{
      "Version":"2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Action": [
            "kms:CreateGrant",
            "kms:ListGrants",
            "kms:RevokeGrant"
          ],
          "Resource": ["arn:aws:kms:us-
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"],
```

```
          "Condition": {
            "Bool": {
              "kms:GrantIsForAWSResource": "true"
            }
          }
        },
        {
          "Effect": "Allow",
          "Action": [
            "kms:Encrypt",
            "kms:Decrypt",
            "kms:ReEncrypt*",
            "kms:GenerateDataKey*",
            "kms:DescribeKey"
          ],
          "Resource": ["arn:aws:kms:us-
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"]
        }
      ]
    }
```

e. 選擇下一步：標籤。

f. 在 Add tags (Optional) (新增標籤 (選用)) 頁面上，選擇 Next: Review (下一步：檢閱)。

g. 對於名稱，輸入政策的唯一名稱 (例如 *KMS_Key_For_Encryption_On_EBS_Policy*)。

h. 選擇建立政策。

i. 在左側導覽窗格中，選擇 Roles (角色)。

j. 在主控台中選擇 ***AmazonEKS_EBS_CSI_DriverRole***，將其開啟以進行編輯。

k. 在新增許可下拉式清單中，選擇連接政策。

l. 在篩選政策方塊中，輸入 *KMS_Key_For_Encryption_On_EBS_Policy*。

m. 選取在搜尋中傳回的 *KMS_Key_For_Encryption_On_EBS_Policy* 左側的核取方塊。

n. 選擇連接政策。

AWS CLI

1. 檢視叢集的 OIDC 提供商 URL。使用您的叢集名稱取代 *my-cluster*。如果來自命令的輸出是 None，則請檢閱先決條件。

```
aws eks describe-cluster --name my-cluster --query "cluster.identity.oidc.issuer" --
output text
```

範例輸出如下。

```
https://oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE
```

2. 建立 IAM 角色，授予其 AssumeRoleWithWebIdentity 動作。

   a. 將以下內容複製到名為 aws-ebs-csi-driver-trust-policy.json 的檔案。使用您的帳戶 ID 取代 *111122223333*。使用前一個步驟傳回的值取代 *EXAMPLED539D4633E53DE1B71EXAMPLE* 和 *region-code*。

```
{
      "Version":"2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Federated": "arn:aws:iam::111122223333:oidc-provider/oidc.eks.us-
east-1.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
          },
          "Action": "sts:AssumeRoleWithWebIdentity",
          "Condition": {
            "StringEquals": {
              "oidc.eks.us-east-1.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:aud": "sts.amazonaws.com",
              "oidc.eks.us-east-1.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:sub": "system:serviceaccount:kube-system:ebs-csi-
controller-sa"
            }
          }
        }
      ]
    }
```

   b. 建立角色。您可將 *AmazonEKS_EBS_CSI_DriverRole* 變更為不同的名稱。若要變更名稱，請務必在稍後的步驟中進行變更。

```
aws iam create-role \
      --role-name AmazonEKS_EBS_CSI_DriverRole \
```

```
      --assume-role-policy-document file://"aws-ebs-csi-driver-trust-policy.json"
```

3. 連接政策。 會 AWS 維護 AWS 受管政策，或者您可以建立自己的自訂政策。使用下列命令將 AWS 受管政策連接至角色。

```
aws iam attach-role-policy \
      --policy-arn arn:aws: iam::aws:policy/service-role/AmazonEBSCSIDriverPolicyV2 \
      --role-name AmazonEKS_EBS_CSI_DriverRole
```

4. 如果在 Amazon EBS 磁碟區上使用自訂 KMS 金鑰進行加密，請視需要自訂 IAM 角色。例如，請執行以下操作：

   a. 複製以下程式碼並貼到新 `kms-key-for-encryption-on-ebs.json` 檔案中。將 *`custom-key-arn`* 取代為自訂 KMS 金鑰 ARN。

```
{
      "Version":"2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Action": [
            "kms:CreateGrant",
            "kms:ListGrants",
            "kms:RevokeGrant"
          ],
          "Resource": ["arn:aws:kms:us-
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"],
          "Condition": {
            "Bool": {
              "kms:GrantIsForAWSResource": "true"
            }
          }
        },
        {
          "Effect": "Allow",
          "Action": [
            "kms:Encrypt",
            "kms:Decrypt",
            "kms:ReEncrypt*",
            "kms:GenerateDataKey*",
            "kms:DescribeKey"
          ],
```

```
          "Resource": ["arn:aws:kms:us-
east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"]
        }
      ]
    }
```

b. 建立政策。您可將 *KMS_Key_For_Encryption_On_EBS_Policy* 變更為不同的名稱。但是，如果您這樣做，請務必在稍後的步驟中也進行變更。

```
aws iam create-policy \
      --policy-name KMS_Key_For_Encryption_On_EBS_Policy \
      --policy-document file://kms-key-for-encryption-on-ebs.json
```

c. 使用以下命令將 IAM 政策連接至角色。使用您的帳戶 ID 取代 *111122223333*。

```
aws iam attach-role-policy \
      --policy-arn arn:aws: iam::111122223333:policy/
KMS_Key_For_Encryption_On_EBS_Policy \
      --role-name AmazonEKS_EBS_CSI_DriverRole
```

現在，您已建立 Amazon EBS CSI 驅動程式 IAM 角色，即可繼續進行下一節。當您使用此 IAM 角色部署附加元件時，即會建立並設定為使用名為 `ebs-csi-controller-sa` 的服務帳戶。該服務帳戶會繫結至獲指派所需 Kubernetes 許可的 Kubernetes `clusterrole`。

步驟 2：取得 Amazon EBS CSI 驅動程式

建議您透過 Amazon EKS 附加元件安裝 Amazon EBS CSI 驅動程式，以提升安全性並減少工作量。若要將 Amazon EKS 附加元件新增至叢集，請參閱 the section called “建立附加元件”。如需附加元件的詳細資訊，請參閱 the section called “Amazon EKS 附加元件”。

> **Important**
>
> 將 Amazon EBS 驅動程式新增為 Amazon EKS 附加元件之前，請確認您的叢集上並未安裝自我管理的驅動程式版本。如果是這樣，請參閱 GitHub 上的解除安裝自我管理的 Amazon EBS CSI 驅動程式。

> **Note**
>
> 根據預設，EBS CSI 使用的 RBAC 角色具有變更節點以支援其污點移除功能的許可。由於 Kubernetes RBAC 的限制，這也允許它變更叢集中的任何其他節點。Helm Chart 具有參數 (`node.serviceAccount.disableMutation`)，可停用 ebs-csi-node 服務帳戶的變動節點 RBAC 許可。啟用時，清除污點等驅動程式功能將無法運作。

或者，如果您想要 Amazon EBS CSI 驅動程式的自我管理安裝，請參閱 GitHub 上的安裝。

步驟 3：部署範例應用程式

您可以部署各種範例應用程式，再視需要進行修改。如需詳細資訊，請參閱 GitHub 上的 Kubernetes 範例。

適用於 Kubernetes 磁碟區儲存的 Amazon EC2 執行個體存放區

Amazon EC2 執行個體存放區 CSI 驅動程式是一種容器儲存介面 (CSI) 外掛程式，可讓 Kubernetes 使用 EC2 執行個體存放磁碟區。執行個體存放區磁碟區提供實體連接至主機電腦的暫時性區塊層級儲存。驅動程式會管理這些 NVMe 儲存磁碟區的生命週期，並將其做為 Kubernetes 持久性磁碟區提供。

Amazon EC2 Instance Store CSI 驅動程式會透過本機 Linux 操作與後端 NVMe 裝置互動。不涉及 AWS API 呼叫，因此不需要 IAM 角色。

考量事項

> **Important**
>
> Amazon EC2 Instance Store CSI 驅動程式會在安裝期間清除暫時性磁碟上的所有資料。如果另一個 CSI 驅動程式或磁碟區管理系統（例如 LVM、原始檔案系統、LUKS 等）正在管理暫時性磁碟，請在安裝 Amazon EC2 執行個體存放區 CSI 驅動程式之前備份您的資料，以避免資料遺失。

- Amazon EC2 執行個體存放區 CSI 驅動程式不適用於 EKS Auto 模式。
- 您無法將 Amazon EC2 執行個體存放區磁碟區掛載至 Fargate Pod。

- 下列執行個體類型無法使用 Amazon EC2 執行個體存放區 CSI 驅動程式：C1、C3、C4、C5d、C5ad、C6gd、D2、D3、D3en、DL1、E3、Edge1gd、F1、G2、G3、G4ad、G4dn R5adR5dn、R6gd、T1T2、X1、X1d、X2gdZ1dX1e
- 支援最新的附加元件版本和一個舊版。針對最新版本中發現的錯誤或漏洞的修正，會以新的次要版本的形式回溯至先前的版本。
- 執行個體儲存體是暫時性的。當節點終止時，資料會遺失。實作關鍵資料的應用程式層級複寫或備份。
- 不支援磁碟區擴展。
- 根據預設，所有磁碟區都會啟用 I/O 限流，以確保跨工作負載的公平頻寬共用。如需詳細資訊，請參閱the section called “I/O 限流”。
- 相較於相同類型的較舊執行個體，新啟動的執行個體已更新 NVMe 識別命令回應。如果您的工具取決於特定的 NVMe 識別值，請檢閱邏輯的相容性。如需詳細資訊，請參閱the section called “新啟動的執行個體上的 NVMe 命名空間變更”。

先決條件

開始前，請確定您具有下列項目：

- 現有 Amazon EKS 叢集。若要查看所有可用的附加元件版本，請執行下列 CLI AWS 命令：

```
aws eks describe-addon-versions --addon-name aws-ec2-local-instance-store-csi-driver
```

安裝 Amazon EC2 執行個體存放區 CSI 驅動程式

Amazon EC2 執行個體存放區 CSI 驅動程式可作為 Amazon EKS 附加元件使用。安裝驅動程式可讓您的叢集使用執行個體儲存體磁碟區進行 Pod 儲存。

若要安裝驅動程式，請將 Amazon EC2 Instance Store CSI 驅動程式附加元件新增至您的叢集。如需說明，請參閱建立 Amazon EKS 附加元件。如需 EKS 附加元件的詳細資訊，請參閱 Amazon EKS 附加元件。

部署範例應用程式

安裝驅動程式後，您可以部署範例應用程式，以驗證執行個體存放區磁碟區是否與叢集正常運作。您可以部署各種範例應用程式，再視需要進行修改。

I/O 限流

Amazon EC2 Instance Store CSI 驅動程式包含內建 I/O 限流，以確保在同一 NVMe 控制器上跨工作負載公平共用頻寬。預設會針對所有磁碟區啟用調節，防止任何單一工作負載使用不成比例的 I/O 頻寬，而犧牲相同節點上的其他工作負載。

設定限流

您可以在兩個層級設定限流：

- StorageClass 參數 – 在 StorageClass 定義`"false"`中將 `throttling` 參數設定為 `"true"`（預設）或 。
- PVC 註釋 – 將`lis.csi.aws.com/throttling`註釋設定為個別 PersistentVolumeClaims `"false"`上的 `"true"`或 。這會在存在時覆寫 StorageClass 設定。

下表顯示 StorageClass 參數和 PVC 註釋如何互動：

PVC 註釋	StorageClass 參數	結果
未設定	未設定	啟用調節 （預設）
未設定	`"false"`	調節已停用
`"false"`	`"true"` 或未設定	調節已停用
`"true"`	`"false"`	啟用調節

建立未編排的 StorageClass

若要停用 StorageClass 中所有磁碟區的限流：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ec2-instance-store-unthrottled
provisioner: lis.csi.aws.com
parameters:
  throttling: "false"
volumeBindingMode: WaitForFirstConsumer
```

停用單一 PVC 的限流

若要在使用限流 StorageClass 時停用個別 PVC 上的限流：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: high-perf-pvc
  annotations:
    lis.csi.aws.com/throttling: "false"
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: ec2-instance-store-sc
  resources:
    requests:
      storage: 100Gi
```

調節考量

- 調節預設為啟用，以為所有共用 NVMe 控制器的工作負載提供可預測的效能。
- 在磁碟區上停用限流可讓它消耗更多 I/O 頻寬，這可能會影響相同控制器上其他限流磁碟區的效能。
- 考慮在專用節點上隔離未分配的工作負載。
- 對於跨越多個控制器的 RAID-0 磁碟區，調節會套用至所有成員分割區。
- 調節狀態會在節點重新啟動期間持續存在。

新啟動的執行個體上的 NVMe 命名空間變更

具有本機執行個體儲存體的新啟動 EC2 執行個體已更新 NVMe 識別控制器和識別命名空間命令回應。這些變更支援 EC2 本機執行個體儲存體上的 NVMe 命名空間管理。現有的執行中執行個體不會受到影響，且值會在任何執行個體的整個生命週期內保持穩定。

更新欄位值

與相同類型的較舊執行個體相比，新啟動的執行個體上的下列欄位已變更：

識別控制器 (`nvme id-ctrl`)

- OACS 的位元 3：從 0 變更為 1

- OAES 的位元 8：從 0 變更為 1
- NN：從 1 變更為值 <= 256
- TNVMCAP：從 0 變更為本機儲存的實際大小

識別命名空間 (`nvme id-ns`)

- NVMCAP：從 0 變更為本機儲存的實際大小
- NGUID：從所有零變更為有效的 NGUID

所有更新的值都符合 NVMe 規格。

變更範圍

這些變更僅適用於 EC2 本機執行個體儲存體。Amazon EBS 磁碟區不受影響。若要區分 EBS 磁碟區和本機執行個體儲存體，請執行下列命令：

```
sudo nvme list
```

在輸出中：

- `Amazon Elastic Block Store` = EBS 磁碟區
- `Amazon EC2 NVMe Instance Storage` = 本機執行個體儲存體

擷取 NVMe 識別值

若要使用 查詢這些值`nvme-cli`：

```
# Identify Controller
sudo nvme id-ctrl /dev/nvme1

# Identify Namespace
sudo nvme id-ns /dev/nvme1n1
```

需執行的動作

無需採取任何動作。這些變更可回溯相容，並遵循 NVMe 規格。不過，如果您的應用程式或工具驗證或依賴 NVMe 識別命令的特定值，請檢閱您的邏輯，以確保與新啟動執行個體上更新的值相容。

透過 Amazon EFS 使用彈性檔案系統儲存

Amazon Elastic File System (Amazon EFS) 提供無伺服器、完全彈性的檔案儲存功能，讓您無需佈建或管理儲存容量和效能，即可分享檔案資料。Amazon EFS 容器儲存介面 (CSI) 驅動程式允許在 上執行的 Kubernetes 叢集 AWS 將 Amazon EFS 檔案系統掛載為持久性磁碟區。本主題旨在說明如何將 Amazon EFS CSI 驅動程式部署到您的 Amazon EKS 叢集。

考量事項

- Amazon EFS CSI 驅動程式與以 Windows 為基礎的容器映像不相容。
- 您不能將動態持續性磁碟區佈建與 Fargate 節點搭配使用，但可以使用靜態佈建。
- 動態佈建需要 1.2 或更新版本的驅動程序。您可以在任何支援的 Amazon EKS 叢集版本上，使用版本 `1.1` 驅動程式靜態佈建持續性磁碟區 (請參閱 Amazon EKS 支援的版本)。
- 此驅動程式的 1.3.2 或更新版本支援 Arm64 架構，包括基於 Amazon EC2 Graviton 的執行個體。
- 此驅動程式的 1.4.2 版或更新版本皆支援使用 FIPS 掛載檔案系統。
- 記下 Amazon EFS 的資源配額。如需詳細資訊，請參閱 Amazon EFS 配額。
- 從 2.0.0 版開始，此驅動程式會從使用 `stunnel` 切換至 `efs-proxy` 以進行 TLS 連線。當使用 `efs-proxy` 時，它將開啟的執行緒數量等於其執行節點的核心數加一。
- Amazon EFS CSI 驅動程式與 Amazon EKS 混合節點不相容。

先決條件

- Amazon EFS CSI 驅動程式需要 AWS Identity and Access Management (IAM) 許可。
  - AWS 建議使用 EKS Pod 身分。如需詳細資訊，請參閱the section called “設定 EKS Pod 身分識別的概觀”。
  - 如需有關服務帳戶的 IAM 角色，以及為叢集設定 IAM OpenID Connect (OIDC) 供應商的資訊，請參閱 the section called “IAM OIDC 提供商”。
- 在您的裝置或 AWS CloudShell 上安裝和設定的 AWS 命令列界面 (AWS CLI) 版本 `1.27.160` `2.12.3`或更新版本。若要檢查您目前的版本，請使用 `aws --version | cut -d / -f2 | cut -d ' ' -f1`。適用於 macOS 的 `yum`、 `apt-get`或 Homebrew 等套件管理員通常是最新版本 CLI AWS 後面的幾個版本。若要安裝最新版本，請參閱《 AWS 命令列界面使用者指南》中的使用 aws 設定安裝 和快速組態。 https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-quickstart.html#cli-configure-quickstart-config在 AWS CloudShell 中安裝的 AWS CLI 版本也可能是最新版本後面的幾個版本。若要更新它，請參閱《CloudShell AWS 使用者指南》中的將 CLI 安裝到您的主目錄。 AWS CloudShell

- kubectl 命令列工具安裝在您的裝置或 AWS CloudShell 上。版本可以與您的叢集 Kubernetes 版本相同，或是為最多比該版本更舊一版或更新一版的次要版本。例如，如果您的叢集版本為 1.29，則可以搭配使用 kubectl 1.28、1.29 或 1.30 版。若要安裝或升級 kubectl，請參閱 the section called “設定 kubectl 和 eksctl”。

> **Note**
>
> 在 Fargate 上執行的 Pod 會自動掛載 Amazon EFS 檔案系統，而無需執行手動驅動程式安裝步驟。

步驟 1：建立 IAM 角色

Amazon EFS CSI 驅動程式需要 IAM 許可才能與檔案系統互動。建立 IAM 角色，並將 arn:aws:iam::aws:policy/service-role/AmazonEFSCSIDriverPolicy受管政策連接至該角色。

> **Note**
>
> 如果您想要同時使用 Amazon EFS 和 Amazon S3 檔案系統儲存，您必須同時將 AmazonEFSCSIDriverPolicy和 AmazonS3FilesCSIDriverPolicy受管政策連接到您的 IAM 角色。如需 Amazon S3 檔案系統儲存體的詳細資訊，請參閱 the section called “Amazon S3 Files”。

若要實作此程序，您可使用下列其中一個工具：

- the section called “eksctl”
- the section called “AWS 管理主控台”
- the section called “AWS CLI”

> **Note**
>
> 本程序中的特定步驟是針對將驅動程式用作 Amazon EKS 附加元件而編寫。如需有關自我管理安裝的詳細資訊，請參閱 GitHub 上的設定驅動程式許可一節。

eksctl

如果使用 Pod 身分識別

執行以下命令以使用 eksctl 建立 IAM 角色和 Pod 身分識別關聯。使用您的叢集名稱取代 my-cluster。也可使用不同名稱取代 AmazonEKS_EFS_CSI_DriverRole。

```
export cluster_name=my-cluster
export role_name=AmazonEKS_EFS_CSI_DriverRole
eksctl create podidentityassociation \
    --service-account-name efs-csi-controller-sa \
    --namespace kube-system \
    --cluster $cluster_name \
    --role-name $role_name \
    --permission-policy-arns arn:aws: iam::aws:policy/service-role/
AmazonEFSCSIDriverPolicy
```

如果為服務帳戶使用 IAM 角色

執行以下命令以使用 eksctl 建立 IAM 角色。使用您的叢集名稱取代 my-cluster。也可使用不同名稱取代 AmazonEKS_EFS_CSI_DriverRole。

```
export cluster_name=my-cluster
export role_name=AmazonEKS_EFS_CSI_DriverRole
eksctl create iamserviceaccount \
    --name efs-csi-controller-sa \
    --namespace kube-system \
    --cluster $cluster_name \
    --role-name $role_name \
    --role-only \
    --attach-policy-arn arn:aws: iam::aws:policy/service-role/AmazonEFSCSIDriverPolicy
 \
    --approve
TRUST_POLICY=$(aws iam get-role --output json --role-name $role_name --query
 'Role.AssumeRolePolicyDocument' | \
    sed -e 's/efs-csi-controller-sa/efs-csi-*/' -e 's/StringEquals/StringLike/')
aws iam update-assume-role-policy --role-name $role_name --policy-document
 "$TRUST_POLICY"
```

AWS 管理主控台

執行下列操作以使用 建立 IAM 角色 AWS 管理主控台。

1. 前往 https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。
2. 在左側導覽窗格中，選擇 Roles (角色)。
3. 在 Roles (角色) 頁面上，選擇 Create role (建立角色)。
4. 在 Select trusted entity (選取信任的實體) 頁面上，執行以下作業：
   a. 如果使用 EKS Pod 身分識別：
      i. 在信任的實體類型區段中，選擇 AWS 服務。
      ii. 在服務或使用案例下拉式清單中，選擇 EKS。
      iii. 在使用案例區段中，選擇 EKS - Pod 身分識別。
      iv. 選擇下一步。
   b. 如果為服務帳戶使用 IAM 角色：
      i. 在 Trusted entity type (信任的實體類型) 區段中，選擇 Web identity (Web 身分)。
      ii. 對於 Identity provider (身分提供者)，為您的叢集選擇 OpenID Connect provider URL (OpenID Connect 供應商 URL)(如 Amazon EKS Overview (概觀) 下所示)。
      iii. 針對 Audience (對象)，選擇 `sts.amazonaws.com`。
      iv. 選擇下一步。
5. 在 Add permissions (新增許可) 頁面上，執行以下作業：
   a. 在 Filter policies (篩選政策) 方塊中，輸入 `AmazonEFSCSIDriverPolicy`。
   b. 勾選在搜尋中傳回的 `AmazonEFSCSIDriverPolicy` 左側的核取方塊。
   c. 選擇下一步。
6. 在 Name, review, and create (命名、檢閱和建立) 頁面上，執行以下作業：
   a. 針對 Role name (角色名稱)，為您的角色輸入唯一名稱 (例如 `AmazonEKS_EFS_CSI_DriverRole`)。
   b. 藉由連接標籤做為鍵值對，在新增標籤 (選用) 下將中繼資料新增至角色。如需有關在 IAM 中使用標籤的詳細資訊，請參閱《IAM 使用者指南》中的標記 IAM 資源。
   c. 選擇建立角色。
7. 建立角色之後：
   a. 如果使用 EKS Pod 身分識別：
      i. 開啟 Amazon EKS 主控台。
      ii. 在左側導覽窗格中，選取叢集，然後選取您要為其設定 EKS Pod 身分識別關聯之叢集的名稱。
      iii. 選擇存取索引標籤。

iv. 在 Pod 身分識別關聯中，選擇建立。

v. 選擇 IAM 角色下拉式清單，然後選取新建立的角色。

vi. 選擇 Kubernetes 命名空間欄位，並輸入 kube-system。

vii.選擇 Kubernetes 服務帳戶欄位，並輸入 efs-csi-controller-sa。

viii選擇建立。

ix. 如需建立 Pod 身分識別關聯的詳細資訊，請參閱 the section called “建立 Pod Identity 關聯 AWS （主控台）”。

b. 如果為服務帳戶使用 IAM 角色：

i. 選擇角色以開啟進行編輯。

ii. 選擇 Trust Relationships (信任關係) 標籤，然後選擇 Edit Trust Relationship (編輯信任政策)。

iii. 查找與下列行相似的行：

```
"oidc.eks.region-code.amazonaws.com/id/<EXAMPLED539D4633E53DE1B71EXAMPLE>:aud":
 "sts.amazonaws.com"
```

在前一行上方加入下列行。<region-code> 將 取代為叢集所在的 AWS 區域。將 <EXAMPLED539D4633E53DE1B71EXAMPLE> 取代為叢集的 OIDC 提供商 ID。

```
"oidc.eks.<region-code>.amazonaws.com/id/
<EXAMPLED539D4633E53DE1B71EXAMPLE>:sub": "system:serviceaccount:kube-system:efs-
csi-*",
```

iv. 將 Condition 運算子從 "StringEquals" 修改為 "StringLike"。

v. 選擇 Update policy (更新政策) 以完成操作。

AWS CLI

執行下列命令以使用 CLI 建立 IAM AWS 角色。

如果使用 Pod 身分識別

1. 建立將 AssumeRole 和 TagSession 動作授予 pods.eks.amazonaws.com 服務的 IAM 角色。

a. 將下列內容複製到名為 aws-efs-csi-driver-trust-policy-pod-identity.json 的檔案。

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowEksAuthToAssumeRoleForPodIdentity",
            "Effect": "Allow",
            "Principal": {
                "Service": "pods.eks.amazonaws.com"
            },
            "Action": [
                "sts:AssumeRole",
                "sts:TagSession"
            ]
        }
    ]
}
```

b. 建立角色。使用您的叢集名稱取代 my-cluster。也可使用不同名稱取代 AmazonEKS_EFS_CSI_DriverRole。

```
export cluster_name=my-cluster
export role_name=AmazonEKS_EFS_CSI_DriverRole
aws iam create-role \
  --role-name $role_name \
  --assume-role-policy-document file://"aws-efs-csi-driver-trust-policy-pod-
identity.json"
```

2. 使用下列命令將所需的 AWS 受管政策連接至角色。

```
aws iam attach-role-policy \
  --policy-arn arn:aws: iam::aws:policy/service-role/AmazonEFSCSIDriverPolicy \
  --role-name $role_name
```

3. 執行下列命令以建立 Pod 身分識別關聯。使用您的帳戶 ID 取代 <111122223333>。

```
aws eks create-pod-identity-association --cluster-name $cluster_name --role-arn {arn-
aws}iam::<111122223333>:role/$role_name --namespace kube-system --service-account
 efs-csi-controller-sa
```

4. 如需建立 Pod 身分識別關聯的詳細資訊，請參閱 the section called “建立 Pod Identity 關聯AWS（主控台）”。

如果為服務帳戶使用 IAM 角色

1. 檢視叢集的 OIDC 提供商 URL。使用您的叢集名稱取代 my-cluster。也可使用不同名稱取代 AmazonEKS_EFS_CSI_DriverRole。

```
export cluster_name=my-cluster
export role_name=AmazonEKS_EFS_CSI_DriverRole
aws eks describe-cluster --name $cluster_name --query "cluster.identity.oidc.issuer"
 --output text
```

   範例輸出如下。

```
https://oidc.eks.<region-code>.amazonaws.com/id/<EXAMPLED539D4633E53DE1B71EXAMPLE>
```

   如果來自命令的輸出是 None，則請檢閱先決條件。

2. 建立授予 AssumeRoleWithWebIdentity 動作許可的 IAM 角色。

   a. 將下列內容複製到名為 aws-efs-csi-driver-trust-policy.json 的檔案。使用您的帳戶 ID 取代 <111122223333>。使用前一個步驟傳回的值取代 <EXAMPLED539D4633E53DE1B71EXAMPLE> 和 <region-code>。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::123456789012:oidc-provider/oidc.eks.us-
east-1.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringLike": {
          "oidc.eks.us-east-1.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:sub": "system:serviceaccount:kube-system:efs-csi-
*",
          "oidc.eks.us-east-1.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:aud": "sts.amazonaws.com"
        }
      }
    }
```

```
  ]
}
```

b. 建立角色。

```
aws iam create-role \
  --role-name $role_name \
  --assume-role-policy-document file://"aws-efs-csi-driver-trust-policy.json"
```

3. 使用下列命令將所需的 AWS 受管政策連接至角色。

```
aws iam attach-role-policy \
  --policy-arn arn:aws: iam::aws:policy/service-role/AmazonEFSCSIDriverPolicy \
  --role-name $role_name
```

步驟 2：取得 Amazon EFS CSI 驅動程式

我們建議您透過 Amazon EKS 附加元件安裝 Amazon EFS CSI 驅動程式。若要將 Amazon EKS 附加元件新增至叢集，請參閱 the section called “建立附加元件”。如需附加元件的詳細資訊，請參閱 the section called “Amazon EKS 附加元件”。如果您無法使用 Amazon EKS 附加元件，我們建議您提交有關為何無法存取容器藍圖 GitHub 儲存庫的問題。

> **Important**
>
> 將 Amazon EFS 驅動程式新增為 Amazon EKS 附加元件之前，請確認您的叢集上並未安裝自我管理的驅動程式版本。若是如此，請參閱 GitHub 上解除安裝 Amazon EFS CSI 驅動程式。

或者，如果您想要 Amazon EFS CSI 驅動程式的自我管理安裝，請參閱 GitHub 上的安裝。

步驟 3：建立 Amazon EFS 檔案系統

如需建立 Amazon EFS 檔案系統，請參閱 GitHub 上的為 Amazon EKS 建立 Amazon EFS 檔案系統。

步驟 4：部署範例應用程式

您可以部署各種範例應用程式，再視需要進行修改。如需詳細資訊，請參閱 GitHub 上的範例。

搭配 Amazon EFS CSI 驅動程式使用 Amazon S3 檔案系統儲存 EFS

S3 檔案是一種共用檔案系統，可直接將任何 AWS 運算與 Amazon S3 中的資料連線。它可讓您以具有完整檔案系統語意和低延遲效能的檔案形式，快速直接存取所有 S3 資料，而不會讓您的資料離開 S3。這表示以檔案為基礎的應用程式、客服人員和團隊可以使用他們已經依賴的工具，以檔案系統的形式存取和使用 S3 資料。Amazon EFS 容器儲存界面 (CSI) 驅動程式允許在 上執行的 Kubernetes 叢集 AWS 將 Amazon S3 檔案系統掛載為從 3.0.0 版開始的持久性磁碟區。本主題說明如何使用 Amazon EFS CSI 驅動程式來管理 Amazon EKS 叢集上的 Amazon S3 檔案系統。

考量事項

- Amazon EFS CSI 驅動程式與以 Windows 為基礎的容器映像不相容。
- EKS Fargate 不支援 S3 檔案。
- Amazon EFS CSI 驅動程式與 Amazon EKS 混合節點不相容。
- Amazon EFS CSI 驅動程式中的 Amazon S3 檔案支援從 3.0.0 版開始。 EFS

先決條件

- Amazon EFS CSI 驅動程式需要 AWS Identity and Access Management (IAM) 許可。
  - AWS 建議使用 EKS Pod 身分。如需詳細資訊，請參閱the section called “設定 EKS Pod 身分識別的概觀”。
  - 如需有關服務帳戶的 IAM 角色，以及為叢集設定 IAM OpenID Connect (OIDC) 供應商的資訊，請參閱 the section called “IAM OIDC 提供商”。
- 在您的裝置或 AWS CloudShell 上安裝和設定的 AWS 命令列界面 (AWS CLI) 版本 `1.27.160` `2.12.3`或更新版本。若要檢查您目前的版本，請使用 `aws --version | cut -d / -f2 | cut -d ' ' -f1`。適用於 macOS 的 `yum`、 `apt-get`或 Homebrew 等套件管理員通常是最新版本 CLI AWS 後面的幾個版本。若要安裝最新版本，請參閱《 AWS 命令列界面使用者指南》中的使用 aws 設定安裝 和 快速組態。 https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-quickstart.html#cli-configure-quickstart-config在 AWS CloudShell 中安裝的 AWS CLI 版本也可能是最新版本後面的幾個版本。若要更新它，請參閱《CloudShell AWS 使用者指南》中的將 CLI 安裝到您的主目錄。 AWS CloudShell
- `kubectl` 命令列工具安裝在您的裝置或 AWS CloudShell 上。版本可以與您的叢集 Kubernetes 版本相同，或是為最多比該版本更舊一版或更新一版的次要版本。例如，如果您的叢集版本為 `1.29`，

則可以搭配使用 kubectl 1.28、1.29 或 1.30 版。若要安裝或升級 kubectl，請參閱 the section called “設定 kubectl 和 eksctl”。

步驟 1：建立 IAM 角色

Amazon EFS CSI 驅動程式需要 IAM 許可才能與檔案系統互動。EFS CSI 驅動程式使用兩個具有不同 IAM 角色的服務帳戶：

- efs-csi-controller-sa — 供控制器使用，需要 AmazonS3FilesCSIDriverPolicy和 AmazonS3FilesClientFullAccess。
- efs-csi-node-sa — 由節點協助程式集使用， 需要：
  - AmazonS3FilesClientFullAccess — 授予掛載和寫入 S3 檔案系統的許可 (s3files:ClientMount、s3files:ClientWrite、s3files:ClientRootAccess)。
  - AmazonS3ReadOnlyAccess — 啟用直接從 S3 儲存貯體進行串流讀取，以獲得更高的輸送量。
  - AmazonElasticFileSystemsUtils — 可將 efs-utils 日誌發佈至 Amazon CloudWatch，以便查看掛載操作並更輕鬆地進行故障診斷。

> **Note**
>
> 如果您想要同時使用 Amazon S3 檔案系統和 Amazon EFS 儲存，您必須同時將 AmazonS3FilesCSIDriverPolicy和 AmazonEFSCSIDriverPolicy受管政策連接到控制器角色。如需 Amazon EFS 儲存體的詳細資訊，請參閱 the section called “Amazon EFS”。

若要實作此程序，您可使用下列其中一個工具：

- the section called “eksctl”
- the section called “AWS 管理主控台”
- the section called “AWS CLI”

> **Note**
>
> 本程序中的特定步驟是針對將驅動程式用作 Amazon EKS 附加元件而編寫。如需有關自我管理安裝的詳細資訊，請參閱 GitHub 上的設定驅動程式許可一節。

eksctl

如果使用 Pod 身分識別

執行下列命令，以建立與 的 IAM 角色和 Pod 身分關聯eksctl。將 *my-cluster* 取代為您的值。

```
export cluster_name=my-cluster

# Create the controller role
eksctl create podidentityassociation \
    --service-account-name efs-csi-controller-sa \
    --namespace kube-system \
    --cluster $cluster_name \
    --role-name AmazonEKS_EFS_CSI_ControllerRole \
    --permission-policy-arns arn:aws: iam::aws:policy/service-role/
AmazonS3FilesCSIDriverPolicy,arn:aws: iam::aws:policy/AmazonS3FilesClientFullAccess

# Create the node role
eksctl create podidentityassociation \
    --service-account-name efs-csi-node-sa \
    --namespace kube-system \
    --cluster $cluster_name \
    --role-name AmazonEKS_EFS_CSI_NodeRole \
    --permission-policy-arns arn:aws: iam::aws:policy/
AmazonS3FilesClientFullAccess,arn:aws: iam::aws:policy/AmazonS3ReadOnlyAccess,arn:aws:
 iam::aws:policy/AmazonElasticFileSystemsUtils
```

如果為服務帳戶使用 IAM 角色

執行下列命令以使用 建立 IAM 角色eksctl。將 *my-cluster* 取代為您的叢集名稱，並將 *region-code* 取代為您的 AWS 區域代碼。

```
export cluster_name=my-cluster
export region_code=region-code

# Create the controller role
export controller_role_name=AmazonEKS_EFS_CSI_ControllerRole
eksctl create iamserviceaccount \
    --name efs-csi-controller-sa \
    --namespace kube-system \
    --cluster $cluster_name \
    --role-name $controller_role_name \
```

```
    --attach-policy-arn arn:aws: iam::aws:policy/service-role/
AmazonS3FilesCSIDriverPolicy \
    --attach-policy-arn arn:aws: iam::aws:policy/AmazonS3FilesClientFullAccess \
    --approve \
    --region $region_code

# Create the node role
export node_role_name=AmazonEKS_EFS_CSI_NodeRole
eksctl create iamserviceaccount \
    --name efs-csi-node-sa \
    --namespace kube-system \
    --cluster $cluster_name \
    --role-name $node_role_name \
    --attach-policy-arn arn:aws: iam::aws:policy/AmazonS3FilesClientFullAccess \
    --attach-policy-arn arn:aws: iam::aws:policy/AmazonS3ReadOnlyAccess \
    --attach-policy-arn arn:aws: iam::aws:policy/AmazonElasticFileSystemsUtils \
    --approve \
    --region $region_code
```

AWS 管理主控台

執行下列操作以使用 建立 IAM 角色 AWS 管理主控台。

1. 前往 https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。
2. 在左側導覽窗格中，選擇 Roles (角色)。
3. 在 Roles (角色) 頁面上，選擇 Create role (建立角色)。
4. 在 Select trusted entity (選取信任的實體) 頁面上，執行以下作業：
   a. 如果使用 EKS Pod 身分識別：
      i. 在信任的實體類型區段中，選擇 AWS 服務。
      ii. 在服務或使用案例下拉式清單中，選擇 EKS。
      iii. 在使用案例區段中，選擇 EKS - Pod 身分識別。
      iv. 選擇下一步。
   b. 如果為服務帳戶使用 IAM 角色：
      i. 在 Trusted entity type (信任的實體類型) 區段中，選擇 Web identity (Web 身分)。
      ii. 對於 Identity provider (身分提供者)，為您的叢集選擇 OpenID Connect provider URL (OpenID Connect 供應商 URL)(如 Amazon EKS Overview (概觀) 下所示)。
      iii. 針對 Audience (對象)，選擇 `sts.amazonaws.com`。
      iv. 選擇下一步。

5. 在 Add permissions (新增許可) 頁面上，執行以下作業：
   a. 在 Filter policies (篩選政策) 方塊中，輸入 `AmazonS3FilesCSIDriverPolicy`。選取搜尋中傳回的政策左側的核取方塊。然後搜尋並選取 `AmazonS3FilesClientFullAccess`。
   b. 選擇下一步。
6. 在 Name, review, and create (命名、檢閱和建立) 頁面上，執行以下作業：
   a. 針對 Role name (角色名稱)，為您的角色輸入唯一名稱 (例如 `AmazonEKS_EFS_CSI_ControllerRole`)。
   b. 藉由連接標籤做為鍵值對，在新增標籤 (選用) 下將中繼資料新增至角色。如需有關在 IAM 中使用標籤的詳細資訊，請參閱《IAM 使用者指南》中的標記 IAM 資源。
   c. 選擇建立角色。
7. 建立角色之後：
   a. 如果使用 EKS Pod 身分識別：
      i. 開啟 Amazon EKS 主控台。
      ii. 在左側導覽窗格中，選取叢集，然後選取您要為其設定 EKS Pod 身分識別關聯之叢集的名稱。
      iii. 選擇存取索引標籤。
      iv. 在 Pod 身分識別關聯中，選擇建立。
      v. 選擇 IAM 角色下拉式清單，然後選取新建立的角色。
      vi. 選擇 Kubernetes 命名空間欄位，並輸入 `kube-system`。
      vii. 選擇 Kubernetes 服務帳戶欄位，並輸入 `efs-csi-controller-sa`。
      viii. 選擇建立。
      ix. 如需建立 Pod 身分識別關聯的詳細資訊，請參閱 the section called “建立 Pod Identity 關聯 AWS （主控台）”。
      x. 重複上述步驟，為節點服務帳戶建立第二個角色。在新增許可頁面上，`AmazonElasticFileSystemsUtils`改為連接 `AmazonS3FilesClientFullAccess`和 `AmazonS3ReadOnlyAccess`。然後，為 Kubernetes 服務帳戶欄位建立與 `efs-csi-node-sa`的 Pod Identity 關聯。
   b. 如果為服務帳戶使用 IAM 角色：
      i. 選擇角色以開啟進行編輯。
      ii. 選擇 Trust Relationships (信任關係) 標籤，然後選擇 Edit Trust Relationship (編輯信任政策)。
      iii. 查找與下列行相似的行：

```
"oidc.eks.region-code.amazonaws.com/id/<EXAMPLED539D4633E53DE1B71EXAMPLE>:aud":
 "sts.amazonaws.com"
```

在前一行上方加入下列行。<region-code> 將 取代為叢集所在的 AWS 區域。將 <EXAMPLED539D4633E53DE1B71EXAMPLE> 取代為叢集的 OIDC 提供商 ID。

```
"oidc.eks.<region-code>.amazonaws.com/id/
<EXAMPLED539D4633E53DE1B71EXAMPLE>:sub": "system:serviceaccount:kube-system:efs-
csi-controller-sa",
```

iv. 選擇 Update policy (更新政策) 以完成操作。

v. 重複上述步驟，為節點服務帳戶建立第二個角色。在新增許可頁面上，AmazonElasticFileSystemsUtils改為連接 AmazonS3FilesClientFullAccessAmazonS3ReadOnlyAccess和 。在信任政策中，使用 efs-csi-node-sa 做為:sub條件值。

AWS CLI

執行下列命令，以使用 CLI 建立 IAM AWS 角色。

如果使用 Pod 身分識別

1. 建立將 AssumeRole 和 TagSession 動作授予 pods.eks.amazonaws.com 服務的 IAM 角色。

   a. 將下列內容複製到名為 aws-efs-csi-driver-trust-policy-pod-identity.json 的檔案。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowEksAuthToAssumeRoleForPodIdentity",
            "Effect": "Allow",
            "Principal": {
                "Service": "pods.eks.amazonaws.com"
            },
            "Action": [
                "sts:AssumeRole",
                "sts:TagSession"
            ]
```

```
        }
    ]
}
```

b. 建立角色。使用您的叢集名稱取代 *my-cluster*。

```
export cluster_name=my-cluster
export controller_role_name=AmazonEKS_EFS_CSI_ControllerRole
aws iam create-role \
  --role-name $controller_role_name \
  --assume-role-policy-document file://"aws-efs-csi-driver-trust-policy-pod-
identity.json"
```

2. 將所需的 AWS 受管政策連接至控制器角色。

```
aws iam attach-role-policy \
  --policy-arn arn:aws: iam::aws:policy/service-role/AmazonS3FilesCSIDriverPolicy \
  --role-name $controller_role_name

aws iam attach-role-policy \
  --policy-arn arn:aws: iam::aws:policy/AmazonS3FilesClientFullAccess \
  --role-name $controller_role_name
```

3. 使用相同的信任政策建立節點 IAM 角色。

```
export node_role_name=AmazonEKS_EFS_CSI_NodeRole
aws iam create-role \
  --role-name $node_role_name \
  --assume-role-policy-document file://"aws-efs-csi-driver-trust-policy-pod-
identity.json"
```

4. 將所需的 AWS 受管政策連接至節點角色。

```
aws iam attach-role-policy \
  --policy-arn arn:aws: iam::aws:policy/AmazonS3FilesClientFullAccess \
  --role-name $node_role_name

aws iam attach-role-policy \
  --policy-arn arn:aws: iam::aws:policy/AmazonS3ReadOnlyAccess \
  --role-name $node_role_name

aws iam attach-role-policy \
  --policy-arn arn:aws: iam::aws:policy/AmazonElasticFileSystemsUtils \
```

```
  --role-name $node_role_name
```

5. 執行下列命令來建立 Pod Identity 關聯。使用您的帳戶 ID 取代 <111122223333>。

```
aws eks create-pod-identity-association --cluster-name $cluster_name --role-arn {arn-
aws}iam::<111122223333>:role/$controller_role_name --namespace kube-system --service-
account efs-csi-controller-sa
aws eks create-pod-identity-association --cluster-name $cluster_name --role-arn
 {arn-aws}iam::<111122223333>:role/$node_role_name --namespace kube-system --service-
account efs-csi-node-sa
```

6. 如需建立 Pod 身分識別關聯的詳細資訊，請參閱 the section called “建立 Pod Identity 關聯AWS（主控台）”。

如果為服務帳戶使用 IAM 角色

1. 檢視叢集的 OIDC 提供商 URL。使用您的叢集名稱取代 *my-cluster*。

```
export cluster_name=my-cluster
aws eks describe-cluster --name $cluster_name --query "cluster.identity.oidc.issuer"
 --output text
```

   範例輸出如下。

```
https://oidc.eks.<region-code>.amazonaws.com/id/<EXAMPLED539D4633E53DE1B71EXAMPLE>
```

   如果來自命令的輸出是 None，則請檢閱先決條件。

2. 建立控制器服務帳戶的 IAM 角色。

   a. 將下列內容複製到名為 controller-trust-policy.json 的檔案。使用您的帳戶 ID 取代 <111122223333>。使用前一個步驟傳回的值取代 <EXAMPLED539D4633E53DE1B71EXAMPLE> 和 <region-code>。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws: iam::<111122223333>:oidc-provider/
oidc.eks.<region-code>.amazonaws.com/id/<EXAMPLED539D4633E53DE1B71EXAMPLE>"
```

```
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "oidc.eks.<region-code>.amazonaws.com/id/
<EXAMPLED539D4633E53DE1B71EXAMPLE>:aud": "sts.amazonaws.com",
                    "oidc.eks.<region-code>.amazonaws.com/id/
<EXAMPLED539D4633E53DE1B71EXAMPLE>:sub": "system:serviceaccount:kube-system:efs-
csi-controller-sa"
                }
            }
        }
    ]
}
```

b. 建立角色。

```
export controller_role_name=AmazonEKS_EFS_CSI_ControllerRole
aws iam create-role \
  --role-name $controller_role_name \
  --assume-role-policy-document file://"controller-trust-policy.json"
```

3. 將所需的 AWS 受管政策連接至控制器角色。

```
aws iam attach-role-policy \
  --policy-arn arn:aws: iam::aws:policy/service-role/AmazonS3FilesCSIDriverPolicy \
  --role-name $controller_role_name

aws iam attach-role-policy \
  --policy-arn arn:aws: iam::aws:policy/AmazonS3FilesClientFullAccess \
  --role-name $controller_role_name
```

4. 建立節點服務帳戶的 IAM 角色。

a. 將下列內容複製到名為 node-trust-policy.json 的檔案。使用您的帳戶 ID 取代 <111122223333>。將 <EXAMPLED539D4633E53DE1B71EXAMPLE>和 取代<region-code>為步驟 1 中傳回的值。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Principal": {
                "Federated": "arn:aws: iam::<111122223333>:oidc-provider/
oidc.eks.<region-code>.amazonaws.com/id/<EXAMPLED539D4633E53DE1B71EXAMPLE>"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "oidc.eks.<region-code>.amazonaws.com/id/
<EXAMPLED539D4633E53DE1B71EXAMPLE>:sub": "system:serviceaccount:kube-system:efs-
csi-node-sa",
                    "oidc.eks.<region-code>.amazonaws.com/id/
<EXAMPLED539D4633E53DE1B71EXAMPLE>:aud": "sts.amazonaws.com"
                }
            }
        }
    ]
}
```

b. 建立角色。

```
export node_role_name=AmazonEKS_EFS_CSI_NodeRole
aws iam create-role \
  --role-name $node_role_name \
  --assume-role-policy-document file://"node-trust-policy.json"
```

5. 將所需的 AWS 受管政策連接至節點角色。

```
aws iam attach-role-policy \
  --policy-arn arn:aws: iam::aws:policy/AmazonS3FilesClientFullAccess \
  --role-name $node_role_name

aws iam attach-role-policy \
  --policy-arn arn:aws: iam::aws:policy/AmazonS3ReadOnlyAccess \
  --role-name $node_role_name

aws iam attach-role-policy \
  --policy-arn arn:aws: iam::aws:policy/AmazonElasticFileSystemsUtils \
  --role-name $node_role_name
```

> **Note**
>
> `AmazonS3ReadOnlyAccess` 政策會授予所有 S3 儲存貯體的讀取存取權。若要限制對特定儲存貯體的存取，您可以將其分離，並以標籤型內嵌政策取代它。如需詳細資訊，請參閱 GitHub 上的 Amazon EFS CSI 驅動程式 IAM 政策文件。

步驟 2：取得 Amazon EFS CSI 驅動程式

我們建議您透過 Amazon EKS 附加元件安裝 Amazon EFS CSI 驅動程式。若要將 Amazon EKS 附加元件新增至叢集，請參閱 the section called “建立附加元件”。如需附加元件的詳細資訊，請參閱 the section called “Amazon EKS 附加元件”。如果您無法使用 Amazon EKS 附加元件，我們建議您提交有關為何無法存取容器藍圖 GitHub 儲存庫的問題。

> **Important**
>
> 將 Amazon EFS 驅動程式新增為 Amazon EKS 附加元件之前，請確認您的叢集上並未安裝自我管理的驅動程式版本。若是如此，請參閱 GitHub 上解除安裝 Amazon EFS CSI 驅動程式。

或者，如果您想要 Amazon EFS CSI 驅動程式的自我管理安裝，請參閱 GitHub 上的安裝。

步驟 3：建立 Amazon S3 檔案系統

若要建立 Amazon S3 檔案系統，請參閱在 GitHub 上為 Amazon EKS 建立 Amazon S3 檔案系統。

步驟 4：部署範例應用程式

您可以部署各種範例應用程式，再視需要進行修改。如需詳細資訊，請參閱 GitHub 上的範例。

搭配 Amazon FSx for Lustre 使用高效能應用程式儲存

Amazon FSx for Lustre 容器儲存介面 (CSI) 驅動程式提供 CSI 介面，允許 Amazon EKS 叢集管理 Amazon FSx for Lustre 檔案系統的生命週期。如需詳細資訊，請參閱《Amazon FSx for Lustre 使用者指南》。

如需有關如何將 Amazon FSx for Lustre CSI 驅動程式部署到您的 Amazon EKS 叢集並確認其是否正常運作的詳細資訊，請參閱 the section called “部署驅動程式”。

部署 FSx for Lustre 驅動程式

本主題說明如何將 FSx for Lustre CSI 驅動程式部署到您的 Amazon EKS 叢集，並確認是否正常運作。我們建議您使用最新版的驅動程式。如需可用版本，請參閱 GitHub 上的 CSI 規格相容性對照表。

> **Note**
>
> Fargate 不支援驅動程式。

如需可用參數的詳細說明，以及示範驅動程式功能的完整範例，請參閱 GitHub 上的 FSx for Lustre 容器儲存介面 (CSI) 驅動程式專案。

先決條件

- 現有的叢集。
- Amazon FSx CSI 驅動程式 EKS 附加元件支援透過 EKS Pod Identity 或 IAM Roles for Service Accounts (IRSA) 進行身分驗證。若要使用 EKS Pod Identity，請在部署 FSx CSI 驅動程式附加元件之前或之後安裝 Pod Identity 代理程式。如需詳細資訊，請參閱the section called “設定代理程式”。若要改為使用 IRSA，請參閱 the section called “IAM OIDC 提供商”。
- 在您的裝置或 AWS CloudShell 上安裝和設定的 AWS 命令列界面 (AWS CLI) 版本 `1.27.160` `2.12.3`或更新版本。若要檢查您目前的版本，請使用 `aws --version | cut -d / -f2 | cut -d ' ' -f1`。適用於 macOS 的 `yum`、 `apt-get`或 Homebrew 等套件管理員通常是最新版本 CLI AWS 後面的幾個版本。若要安裝最新版本，請參閱《 AWS 命令列界面使用者指南》中的使用 aws 設定安裝 和快速組態。 https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-quickstart.html#cli-configure-quickstart-config在 AWS CloudShell 中安裝的 AWS CLI 版本也可能是最新版本後面的幾個版本。若要更新它，請參閱《CloudShell AWS 使用者指南》中的將 CLI 安裝到您的主目錄。 AWS CloudShell
- 裝置或 AWS CloudShell 上安裝的`eksctl`命令列工具版本 `0.215.0`或更新版本。如需有關安裝或更新 `eksctl` 的指示，請參閱 `eksctl` 文件中的安裝一節。
- `kubectl` 命令列工具安裝在您的裝置或 AWS CloudShell 上。版本可以與您的叢集 Kubernetes 版本相同，或是為最多比該版本更舊一版或更新一版的次要版本。例如，如果您的叢集版本為 `1.29`，則可以搭配使用 `kubectl` `1.28`、`1.29` 或 `1.30` 版。若要安裝或升級 `kubectl`，請參閱 the section called “設定 kubectl 和 eksctl”。

步驟 1：建立 IAM 角色

Amazon FSx CSI 外掛程式需要 IAM 許可，才能代表您呼叫 AWS APIs。

> **Note**
>
> Pod 可以存取指派給 IAM 角色的許可，除非您封鎖對 IMDS 的存取。如需詳細資訊，請參閱the section called “最佳實務”。

下列程序說明如何建立 IAM 角色，並將 AWS 受管政策連接至該角色。

1. 建立 IAM 角色，並使用下列命令連接 AWS 受管政策。使用您要使用的叢集名稱取代 `my-cluster`。命令會部署建立 IAM 角色並將 IAM 政策連接至其中的 AWS CloudFormation 堆疊。

   ```
   eksctl create iamserviceaccount \
       --name fsx-csi-controller-sa \
       --namespace kube-system \
       --cluster my-cluster \
       --role-name AmazonEKS_FSx_CSI_DriverRole \
       --role-only \
       --attach-policy-arn arn:aws: iam::aws:policy/AmazonFSxFullAccess \
       --approve
   ```

   建立服務帳戶時，您會看到數行輸出。輸出的最後一行類似於下列內容。

   ```
   [#]  1 task: {
       2 sequential sub-tasks: {
           create IAM role for serviceaccount "kube-system/fsx-csi-controller-sa",
           create serviceaccount "kube-system/fsx-csi-controller-sa",
       } }
   [#]  building iamserviceaccount stack "eksctl-my-cluster-addon-iamserviceaccount-
   kube-system-fsx-csi-controller-sa"
   [#]  deploying stack "eksctl-my-cluster-addon-iamserviceaccount-kube-system-fsx-csi-
   controller-sa"
   [#]  waiting for CloudFormation stack "eksctl-my-cluster-addon-iamserviceaccount-
   kube-system-fsx-csi-controller-sa"
   [#]  created serviceaccount "kube-system/fsx-csi-controller-sa"
   ```

   請注意已部署的 AWS CloudFormation 堆疊名稱。在先前的範例輸出中，堆疊被命名為 `eksctl-my-cluster-addon-iamserviceaccount-kube-system-fsx-csi-controller-sa`。

現在，您已建立 Amazon FSx CSI 驅動程式 IAM 角色，即可繼續進行下一節。當您使用此 IAM 角色部署附加元件時，即會建立並設定為使用名為 `fsx-csi-controller-sa` 的服務帳戶。該服務帳戶會繫結至獲指派所需 Kubernetes 許可的 Kubernetes `clusterrole`。

步驟 2：安裝 Amazon FSx CSI 驅動程式

建議您透過 Amazon EKS 附加元件安裝 Amazon FSx CSI 驅動程式，以提升安全性並減少工作量。若要將 Amazon EKS 附加元件新增至叢集，請參閱 the section called “建立附加元件”。如需附加元件的詳細資訊，請參閱 the section called “Amazon EKS 附加元件”。

Important

叢集中預先存在的 Amazon FSx CSI 驅動程式安裝可能會導致附加元件安裝失敗。當您嘗試安裝 Amazon EKS 附加元件版本而又存在非 EKS FSx CSI 驅動程式時，安裝會因資源衝突而失敗。在安裝期間使用 `OVERWRITE` 旗標以解決此問題。

```
aws eks create-addon --addon-name aws-fsx-csi-driver --cluster-name my-cluster
 --resolve-conflicts OVERWRITE
```

或者，如果您想要 Amazon FSx CSI 驅動程式的自我管理安裝，請參閱 GitHub 上的安裝。

步驟 3：部署儲存類別、持久性磁碟區宣告和範例應用程式

此程序使用 FSx for Lustre 容器儲存介面 (CSI) 驅動程式 GitHub 儲存庫，來耗用動態佈建的 FSx for Lustre 磁碟區。

1. 記下叢集的安全群組。您可以在聯網區段 AWS 管理主控台 下的 中或使用下列 CLI AWS 命令來查看。使用您要使用的叢集名稱取代 `my-cluster`。

   ```
   aws eks describe-cluster --name my-cluster --query
    cluster.resourcesVpcConfig.clusterSecurityGroupId
   ```

2. 根據《Amazon FSx for Lustre 使用者指南》中的 Amazon VPC 安全群組中顯示的標準，為您的 Amazon FSx 檔案系統建立安全群組。對於 VPC，選擇叢集的 VPC，如聯網區段所示。對於「與 Lustre 用戶端關聯的安全群組」，請使用您的叢集安全群組。您可以單獨保留傳出規則，以允許 All traffic (所有流量)。
3. 執行以下命令，下載儲存類別清單檔案。

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-fsx-csi-driver/master/
examples/kubernetes/dynamic_provisioning/specs/storageclass.yaml
```

4. 編輯 `storageclass.yaml` 檔案的參數區段。使用您自己的值取代每一個範例值。

```
parameters:
  subnetId: subnet-0eabfaa81fb22bcaf
  securityGroupIds: sg-068000ccf82dfba88
  deploymentType: PERSISTENT_1
  automaticBackupRetentionDays: "1"
  dailyAutomaticBackupStartTime: "00:00"
  copyTagsToBackups: "true"
  perUnitStorageThroughput: "200"
  dataCompressionType: "NONE"
  weeklyMaintenanceStartTime: "7:09:00"
  fileSystemTypeVersion: "2.12"
```

- **`subnetId`** ：應該在其中建立 Amazon FSx for Lustre 檔案系統的子網路 ID。並非所有可用區域都支援 Amazon FSx for Lustre。開啟位於 https://console.aws.amazon.com/fsx/ 的 Amazon FSx for Lustre 主控台，確認您要使用的子網路位於支援的可用區域中。子網路可以包含您的節點，也可以是不同的子網路或 VPC：
  - 您可以在運算區段下選取節點群組， AWS 管理主控台 以檢查 中的節點子網路。
  - 若您指定的子網路與您擁有節點的子網路不同，則您的 VPC 必須已連線，且您必須確認安全群組中必要的連接埠已開啟。
- **`securityGroupIds`** ：您為檔案系統所建立的安全群組 ID。
- **`deploymentType`** (選用)：檔案系統部署類型。有效值為 `SCRATCH_1`、`SCRATCH_2`、`PERSISTENT_1`、`PERSISTENT_2`。如需部署類型的詳細資訊，請參閱建立您的 Amazon FSx for Lustre 檔案系統。
- 其他參數 (選用)：如需有關其他參數的詳細資訊，請參閱 GitHub 上的編輯 StorageClass。

5. 建立儲存類別清單檔案。

```
kubectl apply -f storageclass.yaml
```

範例輸出如下。

```
storageclass.storage.k8s.io/fsx-sc created
```

6. 下載持續性磁碟區宣告清單檔案。

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-fsx-csi-driver/master/examples/kubernetes/dynamic_provisioning/specs/claim.yaml
```

7. (選用) 編輯 claim.yaml 檔案。根據您的儲存要求和您在上一個步驟中選取的 deploymentType，將 1200Gi 變更為下列某一個增量值。

```
storage: 1200Gi
```

   - SCRATCH_2 和 PERSISTENT：1.2 TiB 或 2.4 TiB，超過 2.4 TiB 則以 2.4 TiB 為單位遞增。
   - SCRATCH_1：1.2 TiB、2.4 TiB 或 3.6 TiB，超過 3.6 TiB 則以 3.6 TiB 為單位遞增。

8. 建立持續性磁碟區宣告。

```
kubectl apply -f claim.yaml
```

   範例輸出如下。

```
persistentvolumeclaim/fsx-claim created
```

9. 確認已佈建檔案系統。

```
kubectl describe pvc
```

   範例輸出如下。

```
Name:          fsx-claim
Namespace:     default
StorageClass:  fsx-sc
Status:        Bound
[...]
```

> **Note**
>
> Status 可能會顯示為 Pending 約 5-10 分鐘，然後變更為 Bound。在 Status 成為 Bound 之前，請勿繼續下一步。如果 Status 顯示 Pending 超過 10 分鐘，請使用 Events 中的警告訊息作為解決任何問題的參考。

10部署範例應用程式。

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/aws-fsx-csi-
driver/master/examples/kubernetes/dynamic_provisioning/specs/pod.yaml
```

11確認範例應用程式正在執行。

```
kubectl get pods
```

範例輸出如下。

```
NAME      READY   STATUS              RESTARTS   AGE
fsx-app   1/1     Running             0          8s
```

12驗證應用程式是否正確掛載了檔案系統。

```
kubectl exec -ti fsx-app -- df -h
```

範例輸出如下。

```
Filesystem                  Size  Used Avail Use% Mounted on
overlay                      80G  4.0G   77G   5% /
tmpfs                        64M     0   64M   0% /dev
tmpfs                       3.8G     0  3.8G   0% /sys/fs/cgroup
192.0.2.0@tcp:/abcdef01     1.1T  7.8M  1.1T   1% /data
/dev/nvme0n1p1               80G  4.0G   77G   5% /etc/hosts
shm                          64M     0   64M   0% /dev/shm
tmpfs                       6.9G   12K  6.9G   1% /run/secrets/kubernetes.io/
serviceaccount
tmpfs                       3.8G     0  3.8G   0% /proc/acpi
tmpfs                       3.8G     0  3.8G   0% /sys/firmware
```

13確認範例應用程式是否已將資料寫入 FSx for Lustre 檔案系統。

```
kubectl exec -it fsx-app -- ls /data
```

範例輸出如下。

```
out.txt
```

此範例輸出顯示範例應用程式成功地將 `out.txt` 檔案寫入檔案系統。

> **Note**
>
> 刪除叢集之前，請確認刪除 FSx for Lustre 檔案系統。如需詳細資訊，請參閱 《FSx for Lustre 使用者指南》中的清除資源。

FSx for Lustre 的效能調校

搭配 Amazon EKS 使用 FSx for Lustre 時，您可以在節點初始化期間套用 Lustre 調校，進而最佳化效能。建議的方法是使用啟動範本使用者資料，以確保所有節點的組態一致。

這些調校包括：

- 網路和 RPC 最佳化
- Lustre 模組管理
- LRU (鎖定資源單位) 調校
- 用戶端快取控制設定
- OST 和 MDC 的 RPC 控制

如需實作這些效能調校的詳細說明：

- 如需最佳化標準節點 (非 EFA) 的效能，請參閱 the section called “最佳化 (非 EFA)”，以取得可新增至啟動範本使用者資料的完整指令碼。
- 如需最佳化已啟用 EFA 節點的效能，請參閱 the section called “最佳化 (EFA)”。

最佳化節點 (EFA) 上的 Amazon FSx for Lustre 效能

本主題會說明如何使用 Amazon EKS 和 Amazon FSx for Lustre 設定 Elastic Fabric Adapter (EFA) 調校。

> **Note**
>
> - 如需建立和部署 FSx for Lustre CSI 驅動程式的資訊，請參閱 the section called “部署驅動程式”。

- 如需不使用 EFA 最佳化標準節點的資訊，請參閱 the section called “最佳化 (非 EFA)”。

步驟 1. 建立 EKS 叢集

使用提供的組態檔案建立叢集：

```
# Create cluster using efa-cluster.yaml
eksctl create cluster -f efa-cluster.yaml
```

範例 `efa-cluster.yaml`：

```
#efa-cluster.yaml

apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: csi-fsx
  region: us-east-1
  version: "1.30"

iam:
  withOIDC: true

availabilityZones: ["us-east-1a", "us-east-1d"]

managedNodeGroups:
  - name: my-efa-ng
    instanceType: c6gn.16xlarge
    minSize: 1
    desiredCapacity: 1
    maxSize: 1
    availabilityZones: ["us-east-1b"]
    volumeSize: 300
    privateNetworking: true
    amiFamily: Ubuntu2204
    efaEnabled: true
    preBootstrapCommands:
      - |
        #!/bin/bash
        eth_intf="$(/sbin/ip -br -4 a sh | grep $(hostname -i)/ | awk '{print $1}')"
```

```
        efa_version=$(modinfo efa | awk '/^version:/ {print $2}' | sed 's/[^0-9.]//g')
        min_efa_version="2.12.1"

        if [[ "$(printf '%s\n' "$min_efa_version" "$efa_version" | sort -V | head -
n1)" != "$min_efa_version" ]]; then
            sudo curl -O https://efa-installer.amazonaws.com/aws-efa-
installer-1.37.0.tar.gz
            tar -xf aws-efa-installer-1.37.0.tar.gz && cd aws-efa-installer
            echo "Installing EFA driver"
            sudo apt-get update && apt-get upgrade -y
            sudo apt install -y pciutils environment-modules libnl-3-dev libnl-
route-3-200 libnl-route-3-dev dkms
            sudo ./efa_installer.sh -y
            modinfo efa
        else
            echo "Using EFA driver version $efa_version"
        fi

        echo "Installing Lustre client"
        sudo wget -O - https://fsx-lustre-client-repo-public-keys.s3.amazonaws.com/fsx-
ubuntu-public-key.asc | gpg --dearmor | sudo tee /usr/share/keyrings/fsx-ubuntu-public-
key.gpg > /dev/null
        sudo echo "deb [signed-by=/usr/share/keyrings/fsx-ubuntu-public-key.gpg]
 https://fsx-lustre-client-repo.s3.amazonaws.com/ubuntu jammy main" > /etc/apt/
sources.list.d/fsxlustreclientrepo.list
        sudo apt update | tail
        sudo apt install -y lustre-client-modules-$(uname -r) amazon-ec2-utils | tail
        modinfo lustre

        echo "Loading Lustre/EFA modules..."
        sudo /sbin/modprobe lnet
        sudo /sbin/modprobe kefalnd ipif_name="$eth_intf"
        sudo /sbin/modprobe ksocklnd
        sudo lnetctl lnet configure

        echo "Configuring TCP interface..."
        sudo lnetctl net del --net tcp 2> /dev/null
        sudo lnetctl net add --net tcp --if $eth_intf

        # For P5 instance type which supports 32 network cards,
        # by default add 8 EFA interfaces selecting every 4th device (1 per PCI bus)
        echo "Configuring EFA interface(s)..."
        instance_type="$(ec2-metadata --instance-type | awk '{ print $2 }')"
        num_efa_devices="$(ls -1 /sys/class/infiniband | wc -l)"
```

```
        echo "Found $num_efa_devices available EFA device(s)"

        if [[ "$instance_type" == "p5.48xlarge" || "$instance_type" ==
 "p5e.48xlarge" ]]; then
           for intf in $(ls -1 /sys/class/infiniband | awk 'NR % 4 == 1'); do
               sudo lnetctl net add --net efa --if $intf --peer-credits 32
           done
        else
        # Other instances: Configure 2 EFA interfaces by default if the instance
 supports multiple network cards,
        # or 1 interface for single network card instances
        # Can be modified to add more interfaces if instance type supports it
            sudo lnetctl net add --net efa --if $(ls -1 /sys/class/infiniband | head -
n1) --peer-credits 32
            if [[ $num_efa_devices -gt 1 ]]; then
               sudo lnetctl net add --net efa --if $(ls -1 /sys/class/infiniband | tail
 -n1) --peer-credits 32
            fi
        fi

        echo "Setting discovery and UDSP rule"
        sudo lnetctl set discovery 1
        sudo lnetctl udsp add --src efa --priority 0
        sudo /sbin/modprobe lustre

        sudo lnetctl net show
        echo "Added $(sudo lnetctl net show | grep -c '@efa') EFA interface(s)"
```

步驟 2. 建立節點群組

建立已啟用 EFA 的節點群組：

```
# Create node group using efa-ng.yaml
eksctl create nodegroup -f efa-ng.yaml
```

> **Important**
>
> 在 區段中調整您環境的這些值# 5. Mount FSx filesystem。
>
> ```
> FSX_DNS="<your-fsx-filesystem-dns>" # Needs to be adjusted.
> MOUNT_NAME="<your-mount-name>" # Needs to be adjusted.
> ```

```
MOUNT_POINT="</your/mount/point>" # Needs to be adjusted.
```

範例efa-ng.yaml：

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: final-efa
  region: us-east-1

managedNodeGroups:
  - name: ng-1
    instanceType: c6gn.16xlarge
    minSize: 1
    desiredCapacity: 1
    maxSize: 1
    availabilityZones: ["us-east-1a"]
    volumeSize: 300
    privateNetworking: true
    amiFamily: Ubuntu2204
    efaEnabled: true
    preBootstrapCommands:
      - |
        #!/bin/bash
        exec 1> >(logger -s -t $(basename $0)) 2>&1


 ############################################################################################
        #                                      Configuration Section
         #

 ############################################################################################

        # File System Configuration
        FSX_DNS="<your-fsx-filesystem-dns>" # Needs to be adjusted.
        MOUNT_NAME="<your-mount-name>" # Needs to be adjusted.
        MOUNT_POINT="</your/mount/point>" # Needs to be adjusted.

        # Lustre Tuning Parameters
        LUSTRE_LRU_MAX_AGE=600000
        LUSTRE_MAX_CACHED_MB=64
```

```
        LUSTRE_OST_MAX_RPC=32
        LUSTRE_MDC_MAX_RPC=64
        LUSTRE_MDC_MOD_RPC=50

        # File paths
        FUNCTIONS_SCRIPT="/usr/local/bin/lustre_functions.sh"
        TUNINGS_SCRIPT="/usr/local/bin/apply_lustre_tunings.sh"
        SERVICE_FILE="/etc/systemd/system/lustre-tunings.service"

        #EFA
        MIN_EFA_VERSION="2.12.1"

        # Function to check if a command was successful
        check_success() {
            if [ $? -eq 0 ]; then
                echo "SUCCESS: $1"
            else
                echo "FAILED: $1"
                return 1
            fi
        }

        echo "********Starting FSx for Lustre configuration********"

        # 1. Install Lustre client
        if grep -q '^ID=ubuntu' /etc/os-release; then
            echo "Detected Ubuntu, proceeding with Lustre setup..."
            # Add Lustre repository
            sudo wget -O - https://fsx-lustre-client-repo-public-keys.s3.amazonaws.com/
fsx-ubuntu-public-key.asc | sudo gpg --dearmor | sudo tee /usr/share/keyrings/fsx-
ubuntu-public-key.gpg > /dev/null

            echo "deb [signed-by=/usr/share/keyrings/fsx-ubuntu-public-key.gpg]
 https://fsx-lustre-client-repo.s3.amazonaws.com/ubuntu jammy main" | sudo tee /etc/
apt/sources.list.d/fsxlustreclientrepo.list

            sudo apt-get update
            sudo apt-get install -y lustre-client-modules-$(uname -r)
            sudo apt-get install -y lustre-client
        else
            echo "Not Ubuntu, exiting"
            exit 1
        fi
```

```
        check_success "Install Lustre client"

        # Ensure Lustre tools are in the PATH
        export PATH=$PATH:/usr/sbin

        # 2. Apply network and RPC tunings
        echo "********Applying network and RPC tunings********"
        if ! grep -q "options ptlrpc ptlrpcd_per_cpt_max" /etc/modprobe.d/
modprobe.conf; then
            echo "options ptlrpc ptlrpcd_per_cpt_max=64" | sudo tee -a /etc/modprobe.d/
modprobe.conf
            check_success "Set ptlrpcd_per_cpt_max"
        else
            echo "ptlrpcd_per_cpt_max already set in modprobe.conf"
        fi

        if ! grep -q "options ksocklnd credits" /etc/modprobe.d/modprobe.conf; then
            echo "options ksocklnd credits=2560" | sudo tee -a /etc/modprobe.d/
modprobe.conf
            check_success "Set ksocklnd credits"
        else
            echo "ksocklnd credits already set in modprobe.conf"
        fi

        # 3. Load Lustre modules
        manage_lustre_modules() {
            echo "Checking for existing Lustre modules..."
            if lsmod | grep -q lustre; then
                echo "Existing Lustre modules found."

                # Check for mounted Lustre filesystems
                echo "Checking for mounted Lustre filesystems..."
                if mount | grep -q "type lustre"; then
                    echo "Found mounted Lustre filesystems. Attempting to unmount..."
                    mounted_fs=$(mount | grep "type lustre" | awk '{print $3}')
                    for fs in $mounted_fs; do
                        echo "Unmounting $fs"
                        sudo umount $fs
                        check_success "Unmount filesystem $fs"
                    done
                else
                    echo "No Lustre filesystems mounted."
                fi
```

```
                # After unmounting, try to remove modules
                echo "Attempting to remove Lustre modules..."
                sudo lustre_rmmod
                if [ $? -eq 0 ]; then
                    echo "SUCCESS: Removed existing Lustre modules"
                else
                    echo "WARNING: Could not remove Lustre modules. They may still be
 in use."
                    echo "Please check for any remaining Lustre processes or mounts."
                    return 1
                fi
            else
                echo "No existing Lustre modules found."
            fi

            echo "Loading Lustre modules..."
            sudo modprobe lustre
            check_success "Load Lustre modules" || exit 1

            echo "Checking loaded Lustre modules:"
            lsmod | grep lustre
        }

        # Managing Lustre kernel modules
        echo "********Managing Lustre kernel modules********"
        manage_lustre_modules

        # 4. Initializing Lustre networking
        echo "********Initializing Lustre networking********"
        sudo lctl network up
        check_success "Initialize Lustre networking" || exit 1

        # 4.5 EFA Setup and Configuration
        setup_efa() {
            echo "********Starting EFA Setup********"

            # Get interface and version information
            eth_intf="$(/sbin/ip -br -4 a sh | grep $(hostname -i)/ | awk '{print
 $1}')"
            efa_version=$(modinfo efa | awk '/^version:/ {print $2}' | sed 's/[^0-9.]//
g')
            min_efa_version=$MIN_EFA_VERSION

            # Install or verify EFA driver
```

```
            if [[ "$(printf '%s\n' "$min_efa_version" "$efa_version" | sort -V | head -
n1)" != "$min_efa_version" ]]; then
                echo "Installing EFA driver..."
                sudo curl -O https://efa-installer.amazonaws.com/aws-efa-
installer-1.37.0.tar.gz
                tar -xf aws-efa-installer-1.37.0.tar.gz && cd aws-efa-installer

                # Install dependencies
                sudo apt-get update && apt-get upgrade -y
                sudo apt install -y pciutils environment-modules libnl-3-dev libnl-
route-3-200 libnl-route-3-dev dkms

                # Install EFA
                sudo ./efa_installer.sh -y
                modinfo efa
            else
                echo "Using existing EFA driver version $efa_version"
            fi
        }

        configure_efa_network() {
            echo "********Configuring EFA Network********"

            # Load required modules
            echo "Loading network modules..."
            sudo /sbin/modprobe lnet
            sudo /sbin/modprobe kefalnd ipif_name="$eth_intf"
            sudo /sbin/modprobe ksocklnd

            # Initialize LNet
            echo "Initializing LNet..."
            sudo lnetctl lnet configure

            # Configure TCP interface
            echo "Configuring TCP interface..."
            sudo lnetctl net del --net tcp 2> /dev/null
            sudo lnetctl net add --net tcp --if $eth_intf

            # For P5 instance type which supports 32 network cards,
            # by default add 8 EFA interfaces selecting every 4th device (1 per PCI
 bus)
            echo "Configuring EFA interface(s)..."
            instance_type="$(ec2-metadata --instance-type | awk '{ print $2 }')"
            num_efa_devices="$(ls -1 /sys/class/infiniband | wc -l)"
```

```
            echo "Found $num_efa_devices available EFA device(s)"

            if [[ "$instance_type" == "p5.48xlarge" || "$instance_type" ==
"p5e.48xlarge" ]]; then
                # P5 instance configuration
                for intf in $(ls -1 /sys/class/infiniband | awk 'NR % 4 == 1'); do
                    sudo lnetctl net add --net efa --if $intf --peer-credits 32
                done
            else
                # Standard configuration
                # Other instances: Configure 2 EFA interfaces by default if the
instance supports multiple network cards,
                # or 1 interface for single network card instances
                # Can be modified to add more interfaces if instance type supports it
                sudo lnetctl net add --net efa --if $(ls -1 /sys/class/infiniband |
head -n1) --peer-credits 32
                if [[ $num_efa_devices -gt 1 ]]; then
                    sudo lnetctl net add --net efa --if $(ls -1 /sys/class/infiniband |
tail -n1) --peer-credits 32
                fi
            fi

            # Configure discovery and UDSP
            echo "Setting up discovery and UDSP rules..."
            sudo lnetctl set discovery 1
            sudo lnetctl udsp add --src efa --priority 0
            sudo /sbin/modprobe lustre

            # Verify configuration
            echo "Verifying EFA network configuration..."
            sudo lnetctl net show
            echo "Added $(sudo lnetctl net show | grep -c '@efa') EFA interface(s)"
        }

        # Main execution
        setup_efa
        configure_efa_network

        # 5. Mount FSx filesystem
        if [ ! -z "$FSX_DNS" ] && [ ! -z "$MOUNT_NAME" ]; then
            echo "********Creating mount point********"
            sudo mkdir -p $MOUNT_POINT
            check_success "Create mount point"
```

```
        echo "Mounting FSx filesystem..."
        sudo mount -t lustre ${FSX_DNS}@tcp:/${MOUNT_NAME} ${MOUNT_POINT}
        check_success "Mount FSx filesystem"
    else
        echo "Skipping FSx mount as DNS or mount name is not provided"
    fi

    # 6. Applying Lustre performance tunings
    echo "********Applying Lustre performance tunings********"

    # Get number of CPUs
    NUM_CPUS=$(nproc)

    # Calculate LRU size (100 * number of CPUs)
    LRU_SIZE=$((100 * NUM_CPUS))

    #Apply LRU tunings
    echo "Apply LRU tunings"
    sudo lctl set_param ldlm.namespaces.*.lru_max_age=${LUSTRE_LRU_MAX_AGE}
    check_success "Set lru_max_age"
    sudo lctl set_param ldlm.namespaces.*.lru_size=$LRU_SIZE
    check_success "Set lru_size"

    # Client Cache Control
    sudo lctl set_param llite.*.max_cached_mb=${LUSTRE_MAX_CACHED_MB}
    check_success "Set max_cached_mb"

    # RPC Controls
    sudo lctl set_param osc.*OST*.max_rpcs_in_flight=${LUSTRE_OST_MAX_RPC}
    check_success "Set OST max_rpcs_in_flight"

    sudo lctl set_param mdc.*.max_rpcs_in_flight=${LUSTRE_MDC_MAX_RPC}
    check_success "Set MDC max_rpcs_in_flight"

    sudo lctl set_param mdc.*.max_mod_rpcs_in_flight=${LUSTRE_MDC_MOD_RPC}
    check_success "Set MDC max_mod_rpcs_in_flight"

    # 7. Verify all tunings
    echo "********Verifying all tunings********"

    # Function to verify parameter value
    verify_param() {
        local param=$1
        local expected=$2
```

```
            local actual=$3

            if [ "$actual" == "$expected" ]; then
                echo "SUCCESS: $param is correctly set to $expected"
            else
                echo "WARNING: $param is set to $actual (expected $expected)"
            fi
        }

        echo "Verifying all parameters:"

        # LRU tunings
        actual_lru_max_age=$(lctl get_param -n ldlm.namespaces.*.lru_max_age | head -1)
        verify_param "lru_max_age" "600000" "$actual_lru_max_age"

        actual_lru_size=$(lctl get_param -n ldlm.namespaces.*.lru_size | head -1)
        verify_param "lru_size" "$LRU_SIZE" "$actual_lru_size"

        # Client Cache
        actual_max_cached_mb=$(lctl get_param -n llite.*.max_cached_mb | grep
"max_cached_mb:" | awk '{print $2}')
        verify_param "max_cached_mb" "64" "$actual_max_cached_mb"

        # RPC Controls
        actual_ost_rpcs=$(lctl get_param -n osc.*OST*.max_rpcs_in_flight | head -1)
        verify_param "OST max_rpcs_in_flight" "32" "$actual_ost_rpcs"

        actual_mdc_rpcs=$(lctl get_param -n mdc.*.max_rpcs_in_flight | head -1)
        verify_param "MDC max_rpcs_in_flight" "64" "$actual_mdc_rpcs"

        actual_mdc_mod_rpcs=$(lctl get_param -n mdc.*.max_mod_rpcs_in_flight | head -1)
        verify_param "MDC max_mod_rpcs_in_flight" "50" "$actual_mdc_mod_rpcs"

        # Network and RPC configurations from modprobe.conf
        actual_ptlrpc=$(grep "ptlrpc ptlrpcd_per_cpt_max" /etc/modprobe.d/modprobe.conf
| awk '{print $3}')
        verify_param "ptlrpcd_per_cpt_max" "ptlrpcd_per_cpt_max=64" "$actual_ptlrpc"

        actual_ksocklnd=$(grep "ksocklnd credits" /etc/modprobe.d/modprobe.conf | awk
'{print $3}')
        verify_param "ksocklnd credits" "credits=2560" "$actual_ksocklnd"

        # 8. Setup persistence
        setup_persistence() {
```

```
            # Create functions file
            cat << EOF > $FUNCTIONS_SCRIPT
        #!/bin/bash

        apply_lustre_tunings() {
            local NUM_CPUS=\$(nproc)
            local LRU_SIZE=\$((100 * NUM_CPUS))

            echo "Applying Lustre performance tunings..."
            lctl set_param ldlm.namespaces.*.lru_max_age=$LUSTRE_LRU_MAX_AGE
            lctl set_param ldlm.namespaces.*.lru_size=\$LRU_SIZE
            lctl set_param llite.*.max_cached_mb=$LUSTRE_MAX_CACHED_MB
            lctl set_param osc.*OST*.max_rpcs_in_flight=$LUSTRE_OST_MAX_RPC
            lctl set_param mdc.*.max_rpcs_in_flight=$LUSTRE_MDC_MAX_RPC
            lctl set_param mdc.*.max_mod_rpcs_in_flight=$LUSTRE_MDC_MOD_RPC
        }
        EOF

            # Create tuning script
            cat << EOF > $TUNINGS_SCRIPT
        #!/bin/bash
        exec 1> >(logger -s -t \$(basename \$0)) 2>&1

        source $FUNCTIONS_SCRIPT

        # Function to check if Lustre is mounted
        is_lustre_mounted() {
            mount | grep -q "type lustre"
        }

        # Function to mount Lustre
        mount_lustre() {
            echo "Mounting Lustre filesystem..."
            mkdir -p $MOUNT_POINT
            mount -t lustre ${FSX_DNS}@tcp:/${MOUNT_NAME} $MOUNT_POINT
            return \$?
        }

        # Main execution
        # Try to mount if not already mounted
        if ! is_lustre_mounted; then
            echo "Lustre filesystem not mounted, attempting to mount..."
            mount_lustre
        fi
```

```
        # Wait for successful mount (up to 5 minutes)
        for i in {1..30}; do
            if is_lustre_mounted; then
                echo "Lustre filesystem mounted, applying tunings..."
                apply_lustre_tunings
                exit 0
            fi
            echo "Waiting for Lustre filesystem to be mounted... (attempt $i/30)"
            sleep 10
        done

        echo "Timeout waiting for Lustre filesystem mount"
        exit 1
        EOF

        # Create systemd service

        # Create systemd directory if it doesn't exist
        sudo mkdir -p /etc/systemd/system/

            # Create service file directly for Ubuntu
            cat << EOF > $SERVICE_FILE
        [Unit]
        Description=Apply Lustre Performance Tunings
        After=network.target remote-fs.target

        [Service]
        Type=oneshot
        ExecStart=/bin/bash -c 'source $FUNCTIONS_SCRIPT && $TUNINGS_SCRIPT'
        RemainAfterExit=yes

        [Install]
        WantedBy=multi-user.target
        EOF


            # Make scripts executable and enable service
            sudo chmod +x $FUNCTIONS_SCRIPT
            sudo chmod +x $TUNINGS_SCRIPT
            systemctl enable lustre-tunings.service
            systemctl start lustre-tunings.service
        }
```

```
        echo "********Setting up persistent tuning********"
        setup_persistence

        echo "FSx for Lustre configuration completed."
```

(選用) 步驟 3. 驗證 EFA 設定

對節點執行 SSH：

```
# Get instance ID from EKS console or {aws} CLI
ssh -i /path/to/your-key.pem ec2-user@<node-internal-ip>
```

驗證 EFA 組態：

```
sudo lnetctl net show
```

檢查設定日誌：

```
sudo cat /var/log/cloud-init-output.log
```

以下是 `lnetctl net show` 的預期輸出範例：

```
net:
    - net type: tcp
      ...
    - net type: efa
      local NI(s):
        - nid: xxx.xxx.xxx.xxx@efa
          status: up
```

範例部署

a. 建立 claim.yaml

```
#claim.yaml

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: fsx-claim-efa
```

```
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: ""
  resources:
    requests:
      storage: 4800Gi
  volumeName: fsx-pv
```

套用宣告：

```
kubectl apply -f claim.yaml
```

b. 建立 pv.yaml

更新 <replaceable-placeholders>：

```
#pv.yaml

apiVersion: v1
kind: PersistentVolume
metadata:
  name: fsx-pv
spec:
  capacity:
    storage: 4800Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteMany
  mountOptions:
    - flock
  persistentVolumeReclaimPolicy: Recycle
  csi:
    driver: fsx.csi.aws.com
    volumeHandle: fs-<1234567890abcdef0>
    volumeAttributes:
      dnsname: fs-<1234567890abcdef0>.fsx.us-east-1.amazonaws.com
      mountname: <abcdef01>
```

套用持久性磁碟區：

```
kubectl apply -f pv.yaml
```

c. 建立 pod.yaml

```
#pod.yaml

apiVersion: v1
kind: Pod
metadata:
  name: fsx-efa-app
spec:
  containers:
  - name: app
    image: amazonlinux:2
    command: ["/bin/sh"]
    args: ["-c", "while true; do dd if=/dev/urandom bs=100M count=20 > data/test_file;
 sleep 10; done"]
    resources:
      requests:
        vpc.amazonaws.com/efa: 1
      limits:
        vpc.amazonaws.com/efa: 1
    volumeMounts:
    - name: persistent-storage
      mountPath: /data
  volumes:
  - name: persistent-storage
    persistentVolumeClaim:
      claimName: fsx-claim-efa
```

套用 Pod：

```
kubectl apply -f pod.yaml
```

其他驗證命令

驗證 Pod 掛載並寫入檔案系統：

```
kubectl exec -ti fsx-efa-app -- df -h | grep data
# Expected output:
# <192.0.2.0>@tcp:/<abcdef01>  4.5T  1.2G  4.5T   1% /data

kubectl exec -ti fsx-efa-app -- ls /data
# Expected output:
```

```
# test_file
```

對節點執行 SSH，以驗證流量是否通過 EFA：

```
sudo lnetctl net show -v
```

預期輸出將顯示含有流量統計資料的 EFA 介面。

相關資訊

- the section called “部署驅動程式”
- the section called “最佳化 (非 EFA)”
- Amazon FSx for Lustre 效能
- Elastic Fabric Adapter

最佳化節點 (非 EFA) 上的 Amazon FSx for Lustre 效能

您可以使用啟動範本使用者資料，在節點初始化期間套用調校參數，以最佳化 Amazon FSx for Lustre 效能。

> **Note**
>
> - 如需建立和部署 FSx for Lustre CSI 驅動程式的資訊，請參閱 the section called “部署驅動程式”。
> - 如需使用已啟用 EFA 的節點最佳化效能，請參閱 the section called “最佳化 (EFA)”。

為什麼要使用啟動範本使用者資料？

- 在節點初始化期間自動套用調校。
- 請確保所有節點的組態一致。
- 無需手動設定節點組態。

範例指令碼概觀

本主題中定義的範例指令碼會執行這些操作：

 # 1. Install Lustre client

- 自動偵測您的 Amazon Linux (AL) 作業系統版本。
- 安裝適當的 Lustre 用戶端套件。

 # 2. Apply network and RPC tunings

- 設定 `ptlrpcd_per_cpt_max=64` 以進行平行 RPC 處理。
- 設定 `ksocklnd credits=2560` 以最佳化網路緩衝區。

 # 3. Load Lustre modules

- 安全地移除現有的 Lustre 模組 (如有)。
- 處理現有檔案系統的卸載。
- 載入全新的 Lustre 模組。

 # 4. Lustre Network Initialization

- 初始化 Lustre 網路組態。
- 設定必要的網路參數。

 # 5. Mount FSx filesystem

- 您必須在本區段中調整您環境的值。

 # 6. Apply tunings

- LRU (鎖定資源單位) 調校：
  - `lru_max_age=600000`
  - 根據 CPU 計數計算的 `lru_size`
- 用戶端快取控制：`max_cached_mb=64`
- RPC 控制：
  - OST `max_rpcs_in_flight=32`
  - MDC `max_rpcs_in_flight=64`

- MDC `max_mod_rpcs_in_flight=50`

`# 7. Verify tunings`

- 驗證所有套用的調校。
- 報告每個參數的成功或警告。

`# 8. Setup persistence`

- 您同樣必須在本區段中調整您環境的值。
- 自動偵測您的作業系統版本 (AL2023)，以決定要套用`Systemd`的服務。
- 系統啟動。
- `Systemd` 會啟動 `lustre-tunings` 服務 (原因是 `WantedBy=multi-user.target`)。
- 服務執行 `apply_lustre_tunings.sh`，其會：
  - 檢查檔案系統是否已掛載。
  - 如果尚未掛載，則掛載檔案系統。
  - 等待成功掛載 (最多五分鐘)。
  - 成功掛載後套用調校參數。
- 設定會保持作用中狀態，直到重新啟動為止。
- 指令碼完成後，服務即會結束。
  - Systemd 會將服務標記為「作用中 (已結束)」。
- 程序會在下次重新啟動開機時重複執行。

建立啟動範本

1. 前往 https://console.aws.amazon.com/ec2/ 開啟 Amazon EC2 主控台。
2. 選擇 Launch Templates (啟動範本)。
3. 選擇 Create launch template (建立啟動範本)。
4. 在 Advanced details (進階詳細資訊) 中，找到 User data (使用者資料) 區段。
5. 貼上以下指令碼，並視需要更新任何項目。

Important

在區段 `# 5. Mount FSx filesystem` 和區段 `# 8. Setup persistence` 中 `apply_lustre_tunings.sh` 的 `setup_persistence()` 函數中，為您的環境調整這些值：

```
FSX_DNS="<your-fsx-filesystem-dns>" # Needs to be adjusted.
MOUNT_NAME="<your-mount-name>" # Needs to be adjusted.
MOUNT_POINT="</your/mount/point>" # Needs to be adjusted.
```

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="==MYBOUNDARY=="
---==MYBOUNDARY==
Content-Type: text/x-shellscript; charset="us-ascii"
#!/bin/bash
exec 1> >(logger -s -t $(basename $0)) 2>&1
# Function definitions
check_success() {
    if [ $? -eq 0 ]; then
        echo "SUCCESS: $1"
    else
        echo "FAILED: $1"
        return 1
    fi
}
apply_tunings() {
    local NUM_CPUS=$(nproc)
    local LRU_SIZE=$((100 * NUM_CPUS))
    local params=(
        "ldlm.namespaces.*.lru_max_age=600000"
        "ldlm.namespaces.*.lru_size=$LRU_SIZE"
        "llite.*.max_cached_mb=64"
        "osc.*OST*.max_rpcs_in_flight=32"
        "mdc.*.max_rpcs_in_flight=64"
        "mdc.*.max_mod_rpcs_in_flight=50"
    )
    for param in "${params[@]}"; do
        lctl set_param $param
        check_success "Set ${param%%=*}"
```

```
    done
}
verify_param() {
    local param=$1
    local expected=$2
    local actual=$3

    if [ "$actual" == "$expected" ]; then
        echo "SUCCESS: $param is correctly set to $expected"
    else
        echo "WARNING: $param is set to $actual (expected $expected)"
    fi
}
verify_tunings() {
    local NUM_CPUS=$(nproc)
    local LRU_SIZE=$((100 * NUM_CPUS))
    local params=(
        "ldlm.namespaces.*.lru_max_age:600000"
        "ldlm.namespaces.*.lru_size:$LRU_SIZE"
        "llite.*.max_cached_mb:64"
        "osc.*OST*.max_rpcs_in_flight:32"
        "mdc.*.max_rpcs_in_flight:64"
        "mdc.*.max_mod_rpcs_in_flight:50"
    )
    echo "Verifying all parameters:"
    for param in "${params[@]}"; do
        name="${param%%:*}"
        expected="${param#*:}"
        actual=$(lctl get_param -n $name | head -1)
        verify_param "${name##*.}" "$expected" "$actual"
    done
}
setup_persistence() {
    # Create functions file
    cat << 'EOF' > /usr/local/bin/lustre_functions.sh
#!/bin/bash
apply_lustre_tunings() {
    local NUM_CPUS=$(nproc)
    local LRU_SIZE=$((100 * NUM_CPUS))

    echo "Applying Lustre performance tunings..."
    lctl set_param ldlm.namespaces.*.lru_max_age=600000
    lctl set_param ldlm.namespaces.*.lru_size=$LRU_SIZE
    lctl set_param llite.*.max_cached_mb=64
```

```
    lctl set_param osc.*OST*.max_rpcs_in_flight=32
    lctl set_param mdc.*.max_rpcs_in_flight=64
    lctl set_param mdc.*.max_mod_rpcs_in_flight=50
}
EOF
    # Create tuning script
    cat << 'EOF' > /usr/local/bin/apply_lustre_tunings.sh
#!/bin/bash
exec 1> >(logger -s -t $(basename $0)) 2>&1
# Source the functions
source /usr/local/bin/lustre_functions.sh
# FSx details
FSX_DNS="<your-fsx-filesystem-dns>" # Needs to be adjusted.
MOUNT_NAME="<your-mount-name>" # Needs to be adjusted.
MOUNT_POINT="</your/mount/point>" # Needs to be adjusted.
# Function to check if Lustre is mounted
is_lustre_mounted() {
    mount | grep -q "type lustre"
}
# Function to mount Lustre
mount_lustre() {
    echo "Mounting Lustre filesystem..."
    mkdir -p ${MOUNT_POINT}
    mount -t lustre ${FSX_DNS}@tcp:/${MOUNT_NAME} ${MOUNT_POINT}
    return $?
}
# Main execution
# Try to mount if not already mounted
if ! is_lustre_mounted; then
    echo "Lustre filesystem not mounted, attempting to mount..."
    mount_lustre
fi
# Wait for successful mount (up to 5 minutes)
for i in {1..30}; do
    if is_lustre_mounted; then
        echo "Lustre filesystem mounted, applying tunings..."
        apply_lustre_tunings
        exit 0
    fi
    echo "Waiting for Lustre filesystem to be mounted... (attempt $i/30)"
    sleep 10
done
echo "Timeout waiting for Lustre filesystem mount"
exit 1
```

```
EOF
    # Create systemd service
    cat << 'EOF' > /etc/systemd/system/lustre-tunings.service
[Unit]
Description=Apply Lustre Performance Tunings
After=network.target remote-fs.target
StartLimitIntervalSec=0
[Service]
Type=oneshot
ExecStart=/usr/local/bin/apply_lustre_tunings.sh
RemainAfterExit=yes
Restart=on-failure
RestartSec=30
[Install]
WantedBy=multi-user.target
EOF
    chmod +x /usr/local/bin/lustre_functions.sh
    chmod +x /usr/local/bin/apply_lustre_tunings.sh
    systemctl enable lustre-tunings.service
    systemctl start lustre-tunings.service
}
echo "Starting FSx for Lustre configuration..."
# 1. Install Lustre client
if grep -q 'VERSION="2"' /etc/os-release; then
    amazon-linux-extras install -y lustre
elif grep -q 'VERSION="2023"' /etc/os-release; then
    dnf install -y lustre-client
fi
check_success "Install Lustre client"
# 2. Apply network and RPC tunings
export PATH=$PATH:/usr/sbin
echo "Applying network and RPC tunings..."
if ! grep -q "options ptlrpc ptlrpcd_per_cpt_max" /etc/modprobe.d/modprobe.conf; then
    echo "options ptlrpc ptlrpcd_per_cpt_max=64" | tee -a /etc/modprobe.d/
modprobe.conf
    echo "options ksocklnd credits=2560" | tee -a /etc/modprobe.d/modprobe.conf
fi
# 3. Load Lustre modules
modprobe lustre
check_success "Load Lustre modules" || exit 1
# 4. Lustre Network Initialization
lctl network up
check_success "Initialize Lustre networking" || exit 1
# 5. Mount FSx filesystem
```

```
FSX_DNS="<your-fsx-filesystem-dns>" # Needs to be adjusted.
MOUNT_NAME="<your-mount-name>" # Needs to be adjusted.
MOUNT_POINT="</your/mount/point>" # Needs to be adjusted.
if [ ! -z "$FSX_DNS" ] && [ ! -z "$MOUNT_NAME" ]; then
    mkdir -p $MOUNT_POINT
    mount -t lustre ${FSX_DNS}@tcp:/${MOUNT_NAME} ${MOUNT_POINT}
    check_success "Mount FSx filesystem"
fi
# 6. Apply tunings
apply_tunings
# 7. Verify tunings
verify_tunings
# 8. Setup persistence
setup_persistence
echo "FSx for Lustre configuration completed."
--==MYBOUNDARY==--
```

6. 建立 Amazon EKS 節點群組時，選取此啟動範本。如需詳細資訊，請參閱the section called “建立”。

相關資訊

- the section called “部署驅動程式”
- the section called “最佳化 (EFA)”
- Amazon FSx for Lustre 效能

搭配 FSx for NetApp ONTAP 使用高效能應用程式儲存

NetApp Trident 提供使用容器儲存介面 (CSI) 合規驅動程式的動態儲存協同運作。這允許 Amazon EKS 叢集管理由 Amazon FSx for NetApp ONTAP 檔案系統支援的持久性磁碟區 (PV) 的生命週期。請注意，Amazon FSx for NetApp ONTAP CSI 驅動程式與 Amazon EKS 混合節點不相容。若要開始使用，請參閱 NetApp Trident 文件中的使用 Trident 與 Amazon FSx for NetApp ONTAP。

Amazon FSx for NetApp ONTAP 是一種儲存服務，允許您在雲端中啟動和執行全受管的 ONTAP 檔案系統。ONTAP 是 NetApp 的檔案系統技術，提供受到廣泛採用的資料存取和資料管理功能。FSx for ONTAP 提供內部部署 NetApp 檔案系統的功能、效能和 APIs，具有全受管 AWS 服務的靈活性、可擴展性和簡易性。如需詳細資訊，請參閱《FSx for ONTAP 使用者指南》。

Important

如果您搭配 Amazon EBS CSI 驅動程式使用 Amazon FSx for NetApp ONTAP 來佈建 EBS 磁碟區，則必須在 `multipath.conf` 檔案中指定不使用 EBS 裝置。如需支援的方法，請參閱組態檔案封鎖清單。請見此處範例。

```
defaults {
       user_friendly_names yes
       find_multipaths no
     }
     blacklist {
       device {
         vendor "NVME"
         product "Amazon Elastic Block Store"
       }
     }
```

搭配 Amazon FSx for OpenZFS 使用資料儲存

Amazon FSx for OpenZFS 是一種全受管檔案儲存服務，可讓您輕鬆地將資料 AWS 從內部部署 ZFS 或其他 Linux 型檔案伺服器移至 。您可以執行此動作，而無需變更應用程式程式碼或資料管理方式。這項服務提供建置在開放原始碼 OpenZFS 檔案系統上的高度可靠、可擴展、高效且功能豐富的檔案儲存。它結合了這些功能與全受管 AWS 服務的靈活性、可擴展性和簡單性。如需詳細資訊，請參閱《Amazon FSx for OpenZFS 使用者指南》。

FSx for OpenZFS 容器儲存介面 (CSI) 驅動程式提供 CSI 介面，允許 Amazon EKS 叢集管理 FSx for OpenZFS 磁碟區的生命週期。請注意，適用於 OpenZFS CSI 驅動程式的 Amazon FSx 與 Amazon EKS 混合節點不相容。若要將 FSx for OpenZFS CSI 驅動程式部署到您的 Amazon EKS 叢集，請參閱 GitHub 上的 aws-fsx-openzfs-csi-driver。

使用 Amazon File Cache 將延遲降到最低

Amazon File Cache 是全受管的高速快取， AWS 用於處理檔案資料，無論資料存放在何處。Amazon File Cache 會在資料第一次存取時自動將資料載入快取，並在未使用資料時將其釋放。如需詳細資訊，請參閱 Amazon File Cache 使用者指南。

Amazon File Cache Container Storage Interface (CSI) 驅動程式提供 CSI 介面，可讓 Amazon EKS 叢集管理 Amazon 檔案快取的生命週期。請注意，Amazon File Cache CSI 驅動程式與 Amazon EKS

混合節點不相容。若要將 Amazon File Cache CSI 驅動程式部署到您的 Amazon EKS 叢集，請參閱 GitHub 上的 aws-file-cache-csi-driver。

使用適用於 Amazon S3 的掛載點 CSI 驅動程式存取 Amazon S3 物件

使用適用於 Amazon S3 的掛載點容器儲存介面 (CSI) 驅動程式，您的 Kubernetes 應用程式可以透過檔案系統介面存取 Amazon S3 物件，在不變更任何應用程式碼的情況下實現高彙總輸送量。CSI 驅動程式以適用於 Amazon S3 的掛載點為建置基礎，將 Amazon S3 儲存貯體呈現為可透過在 Amazon EKS 中的容器和自我管理 Kubernetes 叢集存取的磁碟區。

考量事項

- 適用於 Amazon S3 的掛載點 CSI 驅動程式目前與以 Windows 為基礎的容器映像不相容。
- 適用於 Amazon S3 的掛載點 CSI 驅動程式目前與 Amazon EKS 混合節點不相容。
- 適用於 Amazon S3 的掛載點 CSI 驅動程式不支援 AWS Fargate。但是，系統支援在 Amazon EC2 (使用 Amazon EKS 或自訂 Kubernetes 安裝) 中執行的容器。
- 適用於 Amazon S3 的掛載點 CSI 驅動程式僅支援靜態佈建。不支援動態佈建或建立新儲存貯體。

> **Note**
>
> 靜態佈建是指使用在 `PersistentVolume` 物件中 `volumeAttributes` 內指定為 `bucketName` 的現有 Amazon S3 儲存貯體。如需詳細資訊，請參閱 GitHub 上的靜態佈建。

- 使用適用於 Amazon S3 的掛載點 CSI 驅動程式掛載的磁碟區不支援所有 POSIX 檔案系統功能。若要了解檔案系統行為的相關詳細資訊，請參閱 GitHub 上的適用於 Amazon S3 的掛載點檔案系統行為。

若要了解部署驅動程式的相關詳細資訊，請參閱 the section called “部署驅動程式”。若要了解移除驅動程式的相關詳細資訊，請參閱 the section called “移除驅動程式”。

部署適用於 Amazon S3 的掛載點驅動程式

使用適用於 Amazon S3 的掛載點容器儲存介面 (CSI) 驅動程式，您的 Kubernetes 應用程式可以透過檔案系統介面存取 Amazon S3 物件，在不變更任何應用程式碼的情況下實現高彙總輸送量。

此程序將向您展示如何部署適用於 Amazon S3 的掛載點 CSI Amazon EKS 驅動程式。在繼續之前，請檢閱考量事項。

先決條件

- 叢集的現有 AWS Identity and Access Management (IAM) OpenID Connect (OIDC) 提供者。若要判定您是否已經擁有一個，或是要建立一個，請參閱 the section called “IAM OIDC 提供商”。
- 裝置2.12.3或 AWS CloudShell 上安裝和設定的 AWS CLI 版本或更新版本。
- kubectl 命令列工具安裝在您的裝置或 AWS CloudShell 上。版本可以與您的叢集 Kubernetes 版本相同，或是為最多比該版本更舊一版或更新一版的次要版本。例如，如果您的叢集版本為 1.29，則可以搭配使用 kubectl 1.28、1.29 或 1.30 版。若要安裝或升級 kubectl，請參閱 the section called “設定 kubectl 和 eksctl”。

步驟 1：建立 IAM 政策

適用於 Amazon S3 的掛載點 CSI 驅動程式需要 Amazon S3 許可才能與檔案系統互動。此節將介紹如何建立可授予必要許可的 IAM 政策。

下列範例政策遵循掛載點的 IAM 許可建議。或者，您可以使用 AWS 受管政策 AmazonS3FullAccess，但此受管政策授予的許可超過掛載點所需的許可。

若要了解掛載點建議許可的相關詳細資訊，請參閱 GitHub 上的掛載點 IAM 許可。

1. 前往 https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。
2. 在左側導覽窗格中選擇 Policies (政策)。
3. 在政策頁面上，選擇建立政策。
4. 對於政策編輯器，選擇 JSON。
5. 在政策編輯器下，複製並貼上以下內容：

> **Important**
>
> 將 amzn-s3-demo-bucket1 取代為您自己 Amazon S3 儲存貯體的名稱。

```
{
   "Version":"2012-10-17",
   "Statement": [
        {
```

```
            "Sid": "MountpointFullBucketAccess",
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket1"
            ]
        },
        {
            "Sid": "MountpointFullObjectAccess",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:PutObject",
                "s3:AbortMultipartUpload",
                "s3:DeleteObject"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket1/*"
            ]
        }
    ]
}
```

Amazon S3 Express One Zone 儲存類別推出目錄儲存貯體，可使用與一般用途儲存貯體不同的驗證機制。您應使用 `s3express:CreateSession` 動作，而非 `s3:*` 動作。若要了解目錄儲存貯體的相關詳細資訊，請參閱Amazon S3 使用者指南中的目錄儲存貯體。

下面介紹了您將用於目錄儲存貯體的最低權限政策範例。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3express:CreateSession",
            "Resource": "arn:aws:s3express:us-west-2:111122223333:bucket/amzn-s3-
demo-bucket1--usw2-az1--x-s3"
        }
    ]
}
```

6. 選擇下一步。
7. 在檢閱與建立頁面上，為您的政策命名。本範例逐步教學使用的名稱是 `AmazonS3CSIDriverPolicy`。
8. 選擇建立政策。

步驟 2：建立 IAM 角色

適用於 Amazon S3 的掛載點 CSI 驅動程式需要 Amazon S3 許可才能與檔案系統互動。此節將介紹如何建立 IAM 角色以委派這些許可。如需建立此角色，您可使用以下其中一個工具：

- the section called “eksctl”
- the section called “AWS 管理主控台”
- the section called “AWS CLI”

> **Note**
>
> IAM 政策 `AmazonS3CSIDriverPolicy` 於上一節建立。

eksctl

使用 **`eksctl`** 建立適用於 Amazon S3 的掛載點 CSI 驅動程式 IAM 角色

執行以下命令來建立 IAM 角色和 Kubernetes 服務帳戶。這些命令也會將 `AmazonS3CSIDriverPolicy` IAM 政策連接至角色、使用 IAM 角色 Amazon Resource Name (ARN) 標註 Kubernetes 服務帳戶 (`s3-csi-controller-sa`)，並將 Kubernetes 服務帳戶名稱新增至 IAM 角色的信任政策。

```
CLUSTER_NAME=my-cluster
REGION=region-code
ROLE_NAME=AmazonEKS_S3_CSI_DriverRole
POLICY_ARN=AmazonEKS_S3_CSI_DriverRole_ARN
eksctl create iamserviceaccount \
    --name s3-csi-driver-sa \
    --namespace kube-system \
    --cluster $CLUSTER_NAME \
    --attach-policy-arn $POLICY_ARN \
    --approve \
```

```
    --role-name $ROLE_NAME \
    --region $REGION \
    --role-only
```

AWS 管理主控台

1. 前往 https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。
2. 在左側導覽窗格中，選擇 Roles (角色)。
3. 在 Roles (角色) 頁面上，選擇 Create role (建立角色)。
4. 在 Select trusted entity (選取信任的實體) 頁面上，執行以下作業：
   a. 在 Trusted entity type (信任的實體類型) 區段中，選擇 Web identity (Web 身分)。
   b. 對於 Identity provider (身分提供者)，為您的叢集選擇 OpenID Connect provider URL (OpenID Connect 供應商 URL)(如 Amazon EKS Overview (概觀) 下所示)。

      若 URL 未顯示，則檢閱先決條件。
   c. 針對 Audience (對象)，選擇 `sts.amazonaws.com`。
   d. 選擇下一步。
5. 在 Add permissions (新增許可) 頁面上，執行以下作業：
   a. 請在篩選政策方塊中，輸入 AmazonS3CSIDriverPolicy。

      > **Note**
      > 此政策於上一節建立。

   b. 勾選在搜尋中傳回的 `AmazonS3CSIDriverPolicy` 結果左側的核取方塊。
   c. 選擇下一步。
6. 在 Name, review, and create (命名、檢閱和建立) 頁面上，執行以下作業：
   a. 針對角色名稱，為您的角色輸入唯一名稱 (例如 AmazonEKS_S3_CSI_DriverRole)。
   b. 藉由連接標籤做為鍵值對，在新增標籤 (選用) 下將中繼資料新增至角色。如需有關在 IAM 中使用標籤的詳細資訊，請參閱《IAM 使用者指南》中的標記 IAM 資源。
   c. 選擇建立角色。
7. 建立角色之後，在主控台中選擇角色，以開啟角色進行編輯。
8. 選擇 Trust Relationships (信任關係) 標籤，然後選擇 Edit Trust Relationship (編輯信任政策)。
9. 查找與下列相似的行：

```
"oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE:aud":
 "sts.amazonaws.com"
```

在上一行的末尾新增一個逗號，然後在上一行之後新增下一行。將 *region-code* 取代為您的叢集所在的 AWS 區域。使用叢集的 OIDC 提供商 ID 取代 *EXAMPLED539D4633E53DE1B71EXAMPLE*。

```
"oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE:sub":
 "system:serviceaccount:kube-system:s3-csi-driver-sa"
```

10確認 Condition 運算子已設定為 "StringEquals"。

11選擇 Update policy (更新政策) 以完成操作。

AWS CLI

1. 檢視叢集的 OIDC 提供者 URL。使用您叢集的名稱取代 *my-cluster*。如果來自命令的輸出是 None，則請檢閱先決條件。

```
aws eks describe-cluster --name my-cluster --query "cluster.identity.oidc.issuer" --
output text
```

範例輸出如下。

```
https://oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE
```

2. 建立 IAM 角色，將 Kubernetes 服務帳戶授予 AssumeRoleWithWebIdentity 動作。
   a. 將下列內容複製到名為 aws-s3-csi-driver-trust-policy.json 的檔案。使用您的帳戶 ID 取代 *111122223333*。使用前一個步驟傳回的值取代 *EXAMPLED539D4633E53DE1B71EXAMPLE* 和 *region-code*。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::111122223333:oidc-provider/oidc.eks.us-
east-1.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
```

```
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "oidc.eks.us-east-1.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:sub": "system:serviceaccount:kube-system:s3-csi-
driver-sa",
          "oidc.eks.us-east-1.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:aud": "sts.amazonaws.com"
        }
      }
    }
  ]
}
```

b. 建立角色。您可以將 *AmazonEKS_S3_CSI_DriverRole* 變更為不同的名稱，但如果這樣做，請務必在稍後的步驟中進行變更。

```
aws iam create-role \
  --role-name AmazonEKS_S3_CSI_DriverRole \
  --assume-role-policy-document file://"aws-s3-csi-driver-trust-policy.json"
```

3. 使用以下命令將先前建立的 IAM 政策連接至角色。

```
aws iam attach-role-policy \
  --policy-arn arn:aws: iam::aws:policy/AmazonS3CSIDriverPolicy \
  --role-name AmazonEKS_S3_CSI_DriverRole
```

> **Note**
>
> IAM 政策 AmazonS3CSIDriverPolicy 於上一節建立。

4. 如果您要將驅動程式安裝為 Amazon EKS 附加元件，請略過此步驟。若為驅動程式的自我管理安裝，請建立使用您所建立的 IAM 角色之 ARN 予以註釋的 Kubernetes 服務帳戶。

a. 將下列內容儲存到名為 mountpoint-s3-service-account.yaml 的檔案中。使用您的帳戶 ID 取代 *111122223333*。

```
---
apiVersion: v1
kind: ServiceAccount
metadata:
```

```
  labels:
    app.kubernetes.io/name: aws-mountpoint-s3-csi-driver
  name: s3-csi-driver-sa
  namespace: kube-system
  annotations:
    eks.amazonaws.com/role-arn: arn:aws: iam::111122223333:role/
AmazonEKS_S3_CSI_DriverRole
```

b. 在您的叢集上建立 Kubernetes 服務帳戶。Kubernetes 服務帳戶 (`s3-csi-driver-sa`) 使用您建立的名稱為 *`AmazonEKS_S3_CSI_DriverRole`* 的 IAM 角色進行註解。

```
kubectl apply -f mountpoint-s3-service-account.yaml
```

> **Note**
>
> 當您在此程序中部署外掛程式時，其會建立並設定為使用名為 `s3-csi-driver-sa` 的服務帳戶。

步驟 3：安裝適用於 Amazon S3 的掛載點 CSI 驅動程式

您可透過 Amazon EKS 附加元件安裝適用於 Amazon S3 的掛載點 CSI 驅動程式。您可使用下面的工具，新增附加元件至您的叢集：

- the section called “eksctl”
- the section called “AWS 管理主控台”
- the section called “AWS CLI”

或者以自我管理安裝的方式，安裝適用於 Amazon S3 的掛載點 CSI 驅動程式。若要了解執行自我管理安裝的說明，請參閱 GitHub 上的安裝。

從 `v1.8.0` 開始，您可設定能夠容忍 CSI 驅動程式 Pod 的污點。若要這樣做，請指定一組自訂的污點，以使用 來容忍`node.tolerations`或容忍所有污點`node.tolerateAllTaints`。如需詳細資訊，請參閱 Kubernetes 文件中的污點和容差。

eksctl

使用 **`eksctl`** 來新增 Amazon S3 CSI 附加元件

執行下列命令。使用您的叢集名稱取代 *my-cluster*、使用您的帳戶 ID 取代 *111122223333*，再以先前建立的 IAM 角色名稱取代 *AmazonEKS_S3_CSI_DriverRole*。

```
eksctl create addon --name aws-mountpoint-s3-csi-driver --cluster my-cluster \
  --service-account-role-arn arn:aws: iam::111122223333:role/
AmazonEKS_S3_CSI_DriverRole --force
```

如果您移除 *--force* 選項，且任何 Amazon EKS 附加元件設定與現有設定衝突，然後更新 Amazon EKS 附加元件失敗，則您會收到錯誤訊息以協助您解決衝突。指定此選項之前，請確定 Amazon EKS 附加元件未管理您需要管理的設定，因為這些設定會被此選項覆寫。如需有關此設定之其他選項的詳細資訊，請參閱 eksctl 文件中的 Addons (附加元件)。如需有關 Amazon EKS Kubernetes 欄位管理的詳細資訊，請參閱 the section called “您可自訂的欄位”。

您可透過組態檔案來自訂 eksctl。如需詳細資訊，請參閱 eksctl 文件中的使用組態值。以下範例說明了如何容忍所有污點。

```
# config.yaml
...

addons:
- name: aws-mountpoint-s3-csi-driver
  serviceAccountRoleARN: arn:aws: iam::111122223333:role/AmazonEKS_S3_CSI_DriverRole
  configurationValues: |-
    node:
      tolerateAllTaints: true
```

AWS 管理主控台

1. 開啟 Amazon EKS 主控台。
2. 在左側導覽窗格中選擇叢集。
3. 選擇要為其設定適用於 Amazon S3 的掛載點 CSI 附加元件的叢集名稱。
4. 選擇附加元件索引標籤。
5. 選擇取得更多附加元件。
6. 在選取附加元件頁面上，執行下列動作：
   a. 在 Amazon EKS-外掛程式區段中，選取適用於 Amazon S3 的掛載點 CSI 驅動程式核取方塊。
   b. 選擇下一步。
7. 在設定選取的附加元件設定頁面上，執行以下操作：
   a. 選取您要使用的版本。

b. 針對選取 IAM 角色，選取已連接適用於 Amazon S3 的掛載點 CSI 驅動程式 IAM 政策的 IAM 角色名稱。

c. (選用) 展開選用的組態設定之後，更新衝突解決方法。如果您選取覆寫，就能使用 Amazon EKS 附加元件的設定來覆寫現有附加元件的一種或多種設定。若未啟用此選項，而且有設定與現有設定發生衝突，則操作會失敗。您可以使用產生的錯誤訊息對此衝突進行疑難排解。在選取此選項之前，請確認 Amazon EKS 附加元件未管理您需要自我管理的設定。

d. (選用) 展開選用的組態設定之後，在組態值欄位中設定容差。

e. 選擇下一步。

8. 在檢閱並新增頁面上，選擇建立。附加元件安裝完成後，您會看到已安裝的附加元件。

AWS CLI

使用 CLI 新增 Amazon S3 CSI AWS 附加元件的掛載點

執行下列命令。使用您的叢集名稱取代 *my-cluster*、使用您的帳戶 ID 取代 *111122223333*，再以先前建立的角色名稱取代 *AmazonEKS_S3_CSI_DriverRole*。

```
aws eks create-addon --cluster-name my-cluster --addon-name aws-mountpoint-s3-csi-
driver \
  --service-account-role-arn arn:aws: iam::111122223333:role/
AmazonEKS_S3_CSI_DriverRole
```

您可使用 `--configuration-values` 標誌來自訂命令。以下替代範例說明了如何容忍所有污點。

```
aws eks create-addon --cluster-name my-cluster --addon-name aws-mountpoint-s3-csi-
driver \
  --service-account-role-arn arn:aws: iam::111122223333:role/
AmazonEKS_S3_CSI_DriverRole \
  --configuration-values '{"node":{"tolerateAllTaints":true}}'
```

步驟 4：設定適用於 Amazon S3 的掛載點

在大多數情況下，您只能使用儲存貯體名稱來設定適用於 Amazon S3 的掛載點。若要了解設定適用於 Amazon S3 的掛載點的相關說明，請參閱 GitHub 上的設定適用於 Amazon S3 的掛載點。

步驟 5：部署範例應用程式

您可以將靜態佈建部署到現有 Amazon S3 儲存貯體上的驅動程式。如需詳細資訊，請參閱 GitHub 上的靜態佈建。

移除適用於 Amazon S3 的掛載點 Amazon EKS 附加元件

您可使用兩個選項來移除適用於 Amazon S3 的掛載點 CSI 驅動程式。

- Preserve add-on software on your cluster (在叢集上保留附加元件軟體)：此選項會移除任何設定的 Amazon EKS 管理。其也會移除 Amazon EKS 通知您更新的功能，並在您啟動更新後自動更新 Amazon EKS 附加元件。不過，該選項會保留您叢集上的附加元件軟體。此選項會使附加元件成為自我管理安裝，而不是 Amazon EKS 附加元件。如果啟用此選項，附加元件就無須停機。本程序中的命令使用此選項。
- Remove add-on software entirely from your cluster (從叢集中完全移除附加元件軟體)：只有叢集上沒有資源依賴於附加元件提供的功能時，我們才會建議您將 Amazon EKS 附加元件從叢集中移除。若要執行此選項，請從您在此程序中使用的命令刪除 `--preserve`。

若附加元件具有與其相關聯的 IAM 帳戶，則不會移除該 IAM 帳戶。

您可使用以下工具，來移除 Amazon S3 CSI 附加元件：

- the section called “eksctl”
- the section called “AWS 管理主控台”
- the section called “AWS CLI”

eksctl

使用 **`eksctl`** 移除 Amazon S3 CSI 附加元件

使用您叢集的名稱取代 *`my-cluster`*，然後執行以下命令。

```
eksctl delete addon --cluster my-cluster --name aws-mountpoint-s3-csi-driver --preserve
```

AWS 管理主控台

1. 開啟 Amazon EKS 主控台。
2. 在左側導覽窗格中選擇叢集。
3. 選擇您要移除 Amazon S3 CSI 附加元件掛載點的叢集名稱。
4. 選擇附加元件索引標籤。
5. 選擇適用於 Amazon S3 的掛載點 CSI 驅動程式。

6. 選擇移除。
7. 在移除：aws-mountpoint-s3-csi-driver 確認對話方塊中，請執行以下操作：
   a. 若希望 Amazon EKS 停止管理附加元件的設定，請選取在叢集上保留。若要在叢集上保留附加元件軟體，請執行此動作。如此一來，您就可以自行管理附加元件的所有設定。
   b. 輸入 `aws-mountpoint-s3-csi-driver`。
   c. 選取 Remove (移除)。

AWS CLI

使用 CLI 移除 Amazon S3 CSI AWS 附加元件

使用您叢集的名稱取代 *my-cluster*，然後執行以下命令。

```
aws eks delete-addon --cluster-name my-cluster --addon-name aws-mountpoint-s3-csi-driver --preserve
```

啟用 CSI 磁碟區的快照功能

快照功能可建立資料的即時複本。為了讓此功能在 Kubernetes 中運作，您需要同時具備支援快照的 CSI 驅動程式 (例如 Amazon EBS CSI 驅動程式) 和 CSI 快照控制器。快照控制器可作為 Amazon EKS 受管附加元件，或透過自行管理安裝提供。

以下是使用 CSI 快照控制器的一些注意事項。

- 快照控制器必須與具有快照拍攝功能的 CSI 驅動程式一起安裝。如需有關安裝 Amazon EBS CSI 驅動程式的說明，請參閱 the section called “Amazon EBS”。
- Kubernetes 不支援透過 CSI 移轉提供的磁碟區快照，例如使用具有佈建程式 `kubernetes.io/aws-ebs` 的 `StorageClass` 之 Amazon EBS 磁碟區。必須使用參照 CSI 驅動程式佈建程式 (`ebs.csi.aws.com`) 的 `StorageClass` 來建立磁碟區。
- Amazon EKS 自動模式不包含快照控制器。EKS 自動模式的儲存功能與快照控制器相容。

我們建議您透過 Amazon EKS 受管附加元件安裝 CSI 快照控制器。此附加元件包含在 Amazon EKS 上建立和管理快照所需的自訂資源定義 (CRD)。若要將 Amazon EKS 附加元件新增至叢集，請參閱 the section called “建立附加元件”。如需附加元件的詳細資訊，請參閱 the section called “Amazon EKS 附加元件”。

或者，如果您想要 CSI 快照控制器的自我管理安裝，請參閱 GitHub 上游 Kubernetes `external-snapshotter` 中的使用。

設定 Amazon EKS 叢集的聯網

您的 Amazon EKS 叢集是在 VPC 中建立。對於在 AWS 基礎結構上執行的節點，Pod 聯網功能由 Amazon VPC 容器網路介面 (CNI) 外掛程式提供。若您在自己的基礎結構上執行節點，請參閱 the section called “設定 CNI”。本章包括以下主題，可讓您進一步了解有關叢集聯網。

主題

- 從管理主控台將現有 VPC 子網路新增至 Amazon EKS 叢集
- 檢視 VPC 和子網路的 Amazon EKS 聯網需求
- 為您的 Amazon EKS 叢集建立 Amazon VPC
- 檢視叢集的 Amazon EKS 安全群組要求
- 管理 Amazon EKS 叢集的聯網附加元件

從管理主控台將現有 VPC 子網路新增至 Amazon EKS 叢集

1. 在管理主控台中導覽至您的叢集
2. 從聯網索引標籤，選取管理 VPC 資源
3. 在子網路下拉式清單中，從叢集的 VPC 選取其他子網路。

建立新的 VPC 子網路：

- 檢閱 EKS 子網路要求
- 請參閱《Amazon Virtual Private Cloud 使用者指南》中的 建立子網路。

檢視 VPC 和子網路的 Amazon EKS 聯網需求

當您建立叢集時，您可以指定 VPC 和至少兩個位於不同可用區域的子網。本主題提供您與叢集一起使用的 VPC 和子網的 Amazon EKS 特定需求和注意事項的概觀。如果您沒有可與 Amazon EKS 搭配使用的 VPC，請參閱 the section called “建立 VPC”。如果您要在 AWS Outposts 上建立本機或延伸叢集，請參閱 the section called “建立 VPC 和子網路”而非本主題。本主題中的內容適用於具有混合節點的 Amazon EKS 叢集。如需適用於混合節點的其他聯網需求，請參閱 the section called “準備聯網”。

VPC 要求和注意事項

當您建立叢集時，您指定的 VPC 必須符合下列要求和注意事項：

- 針對您要建立的叢集、任何節點和其他 Kubernetes 資源，VPC 必須有足夠數目的 IP 地址。如果要使用的 VPC 沒有足夠數目的 IP 位址，請嘗試增加可用 IP 位址的數目。

  您可以更新叢集組態來變更叢集使用的子網路和安全群組，藉此完成此操作。您可以從 AWS 管理主控台、最新版本的 AWS CLI、 AWS CloudFormation 和`eksctl`版本 `v0.164.0-rc.0` 或更新版本進行更新。這麼做才能為子網路提供更多可用 IP 地址，以成功升級叢集版本。

  > **Important**
  >
  > 您新增的所有子網路必須與建立叢集時原本提供的 AZ 位於同一組。新的子網路必須滿足所有其他要求，例如，它們必須具有足夠的 IP 地址。
  > 舉例來說，假設您建立叢集並指定四個子網路。依照您指定它們的順序，第一個子網路位於 `us-west-2a` 可用區域中，第二個和第三個子網路位於 `us-west-2b` 可用區域中，而第四個子網路位於 `us-west-2c` 可用區域中。如果要變更子網路，則必須在三個可用區域中各提供至少一個子網路，而且子網路必須與原始子網路位於相同的 VPC 中。

  如果您需要的 IP 地址超過 VPC 中的 CIDR 區塊，您可以將其他無類別域間路由 (CIDR) 區塊與 VPC 產生關聯，以新增其他 CIDR 區塊。您可以在建立叢集的之前或之後，將私有 (RFC 1918) 和公有 (非 RFC 1918) CIDR 區塊與 VPC 關聯。

  在新增新的 CIDR 區塊後，您可以立即新增使用該新區塊的節點。不過，由於控制平面僅在調解完成後才能辨識新的 CIDR 區塊，因此對於與 VPC 相關聯的 CIDR 區塊，可能需要最多一小時才能辨識叢集。然後，您可以對新 CIDR 區塊中的節點和 Pod `kubectl cp`執行 `kubectl attach`、`kubectl logs`、、 `kubectl exec`和 `kubectl port-forward`命令（這些命令使用 `kubelet API`)。此外，如果您有作為 Webhook 後端運作的 Pod，那麼您必須等待控制平面調解完成。

- 當您透過 Transit Gateway、VPC 對等互連或其他聯網組態將 EKS 叢集連線至其他 VPC 時，請避免 IP 位址範圍重疊。當您 EKS 叢集的服務 CIDR 與連接的 VPC 的 CIDR 重疊時，即會發生發生 CIDR 衝突。在這些情況下，服務 IP 地址會優先於具有相同 IP 地址VPCs 中的資源，但流量路由可能會變得無法預測，且應用程式可能無法連線至預期資源。

  為避免 CIDR 衝突，請確保您的 EKS 服務 CIDR 不會與任何已連接的 VPC CIDR 重疊，並會維護所有 CIDR 指派的集中記錄。如果您遇到 CIDR 重疊的情況，則您可以搭配傳輸閘道使用共用服務

VPC。如需詳細資訊，請參閱隔離 VPC 與共享服務和混合網路中的 Amazon EKS VPC 可路由 IP 地址保護模式。另外，請參閱《EKS 最佳實務指南》中 VPC 和子網路考量頁面的「跨 VPC 通訊」一節。

- 如果您希望 Kubernetes 指派 IPv6 地址至 Pod 和服務，請將 IPv6 CIDR 區塊與 VPC 建立關聯。如需詳細資訊，請參閱 Amazon VPC 使用者指南中的建立 IPv6 CIDR 區塊與 VPC 的關聯。您無法搭配使用 IPv6 地址與在混合節點上執行的 Pod 和服務，亦無法將混合節點與使用 IPv6 IP 位址系列設定的叢集搭配使用。
- VPC 必須支援 DNS 主機名稱和 DNS 解析。否則，節點無法註冊至您的叢集。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的 VPC 的 DNS 屬性。
- VPC 可能需要使用 AWS PrivateLink 的 VPC 端點。如需詳細資訊，請參閱the section called “子網需求和注意事項”。
- 如果叢集使用客戶路由控制平面輸出 (`controlPlaneEgressMode=CUSTOMER_ROUTED`)，則叢集使用的子網路必須能夠路由到控制平面呼叫的任何端點，例如許可 Webhook 伺服器和 OIDC 供應商。這需要一個輸出路徑（例如，NAT 閘道、NAT 執行個體、防火牆或傳輸閘道到集中式輸出 VPC)，以及允許流量的路由表、安全群組和網路 ACL 規則。如需詳細資訊，請參閱the section called “控制平面輸出”。

如果您使用 Kubernetes `1.14` 或舊版建立叢集，Amazon EKS 會將以下標籤新增至您的 VPC：

金鑰	值
`kubernetes.io/cluster/` *`my-cluster`*	`owned`

此標籤僅供 Amazon EKS 使用。您可以在不影響服務的情況下移除該標籤。其不會與版本 `1.15` 或更新版本的叢集搭配使用。

子網需求和注意事項

當您建立叢集時，Amazon EKS 會在您指定的子網中建立 2-4 個彈性網路介面。這些網路介面啟用您的叢集和 VPC 之間的通訊。這些網路介面也會啟用使用 的 Kubernetes 功能`kubelet API`。API 的連線`kubelet`用於 `kubectl attach`、`kubectl cp`、`kubectl logs`、 `kubectl exec`和 `kubectl port-forward`命令。每個 Amazon EKS 建立的網路介面都在其說明中包含 `Amazon EKS` *`cluster-name`* 字樣。

這些請求者受管網路介面也會在控制平面和工作者節點之間攜帶資料平面流量。Amazon EKS 會在這些網路介面的控制平面端吸收資料傳輸成本。客戶端流量的標準 AWS 資料傳輸率會向您收費，特別是

從控制平面傳入工作者節點，以及從工作者節點傳出至控制平面。如需詳細資訊，請參閱the section called “Amazon EKS 定價”。

Amazon EKS 可以在您建立叢集時指定的任何子網中建立其網路介面。建立叢集後，您可以變更 Amazon EKS 要在其中建立網路介面的子網路。當您更新叢集的 Kubernetes 版本時，Amazon EKS 會刪除其建立的原始網路介面，並建立新的網路介面。這些網路介面可能在與原始網路介面相同的子網中建立，或在與原始網路介面不同的子網中建立。若要控制在其中建立的子網路網路介面，可以將建立叢集時指定的子網路數量限制為僅有兩個子網路，或在建立叢集後更新子網路。

叢集的子網路要求

您建立或更新叢集時指定的子網路必須符合下列要求：

- 每個子網必須至少有六個 IP 地址，以供 Amazon EKS 使用。但是，我們建議至少使用 16 個 IP 地址。
- 子網路必須位於至少兩個不同的可用區域。
- 子網路不能位於 AWS Outposts 或 AWS Wavelength 中。但是，如果您的 VPC 中有它們，則可以部署自我管理的節點和 Kubernetes 資源到這些類型的子網。如需有關自我管理節點的詳細資訊，請參閱 the section called “自我管理的節點”。
- 子網路可以是公有或私有。不過，如果可能的話，我們建議您指定私有子網。公有子網路是一種子網路，其路由表包含到網際網路閘道的路由，而私有子網路也是一種子網路，其路由表不包括到網際網路閘道路由。
- 子網路不能位於下列可用區域：

AWS 區域	區域名稱	不允許的可用區域 ID
us-east-1	美國東部 (維吉尼亞北部)	use1-az3
us-west-1	美國西部 (加利佛尼亞北部)	usw1-az2
ca-central-1	加拿大 (中部)	cac1-az3

各元件的 IP 位址系列用量

下表包含 Amazon EKS 每個元件所使用的 IP 位址系列。您可以使用網路位址轉譯 (NAT) 或其他相容性系統，從具有資料表項目值為 "No" 的系列中的來源 IP 位址連接到這些元件。

功能可能會因叢集的 IP 系列 (`ipFamily`) 設定而異。此設定會變更 Kubernetes 指派給服務的 CIDR 區塊所使用的 IP 位址類型。設定值為 IPv4 的叢集稱為 IPv4 叢集，而設定值為 IPv6 的叢集則稱為 IPv6 叢集。

元件	IPv4 地址	IPv6 地址	雙堆疊地址
EKS API 公有端點	是[1、3]	是[1、3]	是[1、3]
EKS API VPC 端點	是	否	否
EKS 身分驗證 API 公有端點 (EKS Pod 身分識別)	是[1]	是[1]	是[1]
EKS 身分驗證 API VPC 端點 (EKS Pod 身分識別)	是[1]	是[1]	是[1]
IPv4 Kubernetes 叢集公有端點[2]	是	否	否
IPv4 Kubernetes 叢集私有端點[2]	是	否	否
IPv6 Kubernetes 叢集公有端點[2]	是[1、4]	是[1、4]	是[4]
IPv6 Kubernetes 叢集私有端點[2]	是[1、4]	是[1、4]	是[4]
Kubernetes 叢集子網路	是[2]	否	是[2]
節點主要 IP 位址	是[2]	否	是[2]
服務 IP 位址的叢集 CIDR 範圍	是[2]	是[2]	否
來自 VPC CNI 的 Pod IP 位址	是[2]	是[2]	否

元件	IPv4 地址	IPv6 地址	雙堆疊地址
IRSA OIDC 發行者 URL	是[1、3]	是[1、3]	是[1、3]

Note

[1] 端點是雙堆疊，具有 IPv4 和 IPv6 地址。外部的應用程式 AWS、叢集的節點，以及叢集內的 Pod 都可以透過 IPv4或 到達此端點IPv6。

[2] 當您建立叢集時，您可以在叢集的 IP 系列 (ipFamily) 設定中選擇 IPv4 叢集和 IPv6 叢集，並且這一點無法變更。相反地，在您建立另一個叢集並移轉工作負載時，您必須選擇不同的設定。

[3] 雙堆疊端點已於 2024 年 8 月推出。若要搭配 CLI AWS 使用雙堆疊端點，請參閱 SDK AWS SDKs 和工具參考指南中的雙堆疊和 FIPS 端點組態。下面列出了新的端點：

EKS API 公有端點

```
eks.region.api.aws
```

IRSA OIDC 發行者 URL

```
oidc-eks.region.api.aws
```

[4] 雙堆疊叢集端點已於 2024 年 10 月推出。EKS 會為新叢集建立下列端點，其中這些新叢集是在此日期之後建立的並且會在叢集的 IP 系列 (ipFamily) 設定中選取 IPv6：

EKS 叢集公有/私有端點

```
eks-cluster.region.api.aws
```

節點的子網路要求

您可以將節點和 Kubernetes 資源部署到您建立叢集時指定的相同子網。不過，這不是必要的。這是因為您還可以將節點和 Kubernetes 資源部署至建立叢集時未指定的子網路。如果將節點部署到不同的子網路，Amazon EKS 不會在這些子網路中建立叢集網路介面。您部署節點和 Kubernetes 資源的任何子網必須符合以下要求：

- 子網必須具有足夠的可用 IP 地址，以便將所有節點和 Kubernetes 資源部署至其中。

- 如果您希望 Kubernetes 指派 IPv6 地址至 Pod 和服務，則必須有一個 IPv6 CIDR 區塊和一個與子網路相關聯的 IPv4 CIDR 區塊。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的建立 IPv6 CIDR 區塊與子網路的關聯。與子網關聯的路由表必須包含傳送到 IPv4 和 IPv6 地址的路由。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的 Routes (路由)。Pod 僅指派一個 IPv6 地址。但是，Amazon EKS 為您的叢集和節點建立的網路介面受指派一個 IPv4 和一個 IPv6 地址。
- 如果您需要從網際網路對您的 Pod 進行傳入存取，請確保至少有一個公有子網路 (具有足夠可用的 IP 位址) 來為其部署負載平衡器和傳入。您可以將負載平衡器部署至公有子網。負載均衡器可以對私有子網路或公有子網路中的 Pod 進行負載平衡。如果可能的話，我們建議您將節點部署至私有子網。
- 如果您計畫將節點部署到公有子網，則該子網必須自動指派 IPv4 公有地址或 IPv6 地址。如果您將節點部署到具有相關聯 IPv6 CIDR 區塊的子網，則私有子網也必須自動指派 IPv6 地址。如果您在 2020 年 3 月 26 日之後使用 Amazon EKS 提供的 AWS CloudFormation 範本來部署 VPC，則會啟用此設定。如果在此日期之前使用範本部署 VPC，或者使用自己的 VPC，則必須手動啟用此設定。如需範本，請參閱 the section called “建立 VPC”。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的修改子網路的公有 IPv4 定址屬性及修改子網路的 IPv6 定址屬性。
- 如果您部署節點的子網路是私有子網路，且其路由表不包含網路位址轉譯 (NAT) 裝置 (IPv4) 或輸出限定閘道 (IPv6) 的路由，請使用 AWS PrivateLink將 VPC 端點新增至您的 VPC。您的節點和 Pod 需要與其通訊的所有 AWS 服務的 VPC 端點。範例包括 Amazon ECR、Elastic Load Balancing、Amazon CloudWatch、 AWS Security Token Service 和 Amazon Simple Storage Service (Amazon S3)。如果您將 IAM 角色用於服務帳戶 (IRSA)，您也可以使用端點私下連接叢集 OIDC 探索/JWKS `com.amazonaws.region-code.oidc-eks`端點。如需詳細資訊，請參閱the section called “使用 AWS PrivateLink 存取叢集 OIDC 端點”。並非所有 AWS 服務都支援 VPC 端點。如需詳細資訊，請參閱什麼是 is AWS PrivateLink？ AWS 與 AWS PrivateLink 整合的 和 服務。有關更多 Amazon EKS 要求的列表，請參閱 the section called “私有叢集”。
- 如果要將負載平衡器部署到子網，則子網必須具有以下標籤：
  - 私有子網路

金鑰	值
`kubernetes.io/role/internal-elb`	`1`

  - 公有子網路

金鑰	值
kubernetes.io/role/elb	1

如果建立的是版本 1.18 及更舊版本的 Kubernetes 叢集，Amazon EKS 會將下列標籤新增至指定的所有子網路。

金鑰	值
kubernetes.io/cluster/ *my-cluster*	shared

現在，當您建立新的 Kubernetes 叢集時，Amazon EKS 不會將標籤新增至您的子網路。如果叢集所使用的子網路具有標籤，而且該叢集此前的版本早於 1.19，則當叢集更新為較新版本時，系統不會自動從子網路移除該標籤。版本 2.1.1 AWS Load Balancer控制器需要此標籤。如果您使用較新版本的 Load Balancer Controller，則您無須中斷服務便可移除標籤。如需有關控制器的詳細資訊，請參閱 the section called “ AWS Load Balancer控制器”。

如果您使用 eksctl或任何 Amazon EKS AWS CloudFormation VPC 範本部署 VPC，則適用下列條件：

- 在 2020 年 3 月 26 日或之後：公有 IPv4 地址會由公有子網自動指派給此公有子網中已部署的新節點。
- 在 2020 年 3 月 26 日之前：公有 IPv4 地址不會由公有子網路自動指派給已部署到此公有子網路的新節點。

這項變更會影響部署於公有子網中的新節點群組的下列行為：

- 受管節點群組 ：如果節點群組在 2020 年 4 月 22 日當日或之後部署到公有子網路，則此公有子網路必須啟用公有 IP 位址的自動指派。如需詳細資訊，請參閱修改子網的公有 IPv4 定址屬性。
- Linux、 Windows 或 Arm 自我管理節點群組：如果節點群組在 2020 年 3 月 26 日當日或之後部署到公有子網路，此公有子網路必須啟用公有 IP 位址的自動指派。否則，必須改用公有 IP 地址啟動節點。如需詳細資訊，請參閱修改子網路的公有 IPv4 定址屬性或在執行個體啟動期間指派公有 IPv4 地址。

共用子網路需求和注意事項

您可以使用 VPC 共用，與相同 AWS 組織內的其他 AWS 帳戶共用子網路。您可以在共用子網路中建立 Amazon EKS 叢集，但會有下列考量：

- VPC 子網路的擁有者必須與參與者帳戶共用子網路，該帳戶才能在其中建立 Amazon EKS 叢集。
- 您無法使用 VPC 的預設安全群組來啟動資源，因為預設安全群組屬於擁有者。此外，參與者無法啟動使用由其他參與者或擁有者所擁有之安全群組的資源。
- 在共用子網路中，參與者和擁有者會分別控制每個個別帳戶中的安全群組。子網路擁有者可以查看由參與者建立的安全群組，但無法執行任何動作。如果子網路擁有者想要移除或修改這些安全群組，建立安全群組的參與者必須採取動作。
- 如果叢集是由參與者建立，則需要考量下列事項：
  - 必須在該帳戶中建立叢集 IAM 角色和節點 IAM 角色。如需詳細資訊，請參閱the section called “叢集 IAM 角色”及the section called “節點 IAM 角色”。
  - 所有節點都必須由相同的參與者建立，包括受管節點群組。
- 共用 VPC 擁有者無法檢視、更新或刪除參與者在共用子網路中建立的叢集。此為 VPC 資源 (每個帳戶具有不同存取權) 以外。如需更多資訊，請參閱《Amazon VPC 使用者指南》中的擁有者和參與者的責任與許可。
- 如果您使用 Kubernetes 專用 Amazon VPC CNI 外掛程式的自訂聯網功能，您需要使用擁有者帳戶中列出的可用區域 ID 映射來建立每個 `ENIConfig`。如需詳細資訊，請參閱the section called “自訂聯網”。

如需有關 VPC 子網路共用的詳細資訊，請參閱《Amazon VPC 使用者指南》中的與其他帳戶共用 VPC。

為您的 Amazon EKS 叢集建立 Amazon VPC

您可以使用 Amazon Virtual Private Cloud (Amazon VPC) 將 AWS 資源啟動至您定義的虛擬網路。此虛擬網路非常近似於您在自有資料中心內運作的傳統網路。但是，它帶來了使用 Amazon Web 服務的可擴展基礎架構的好處。部署生產 Amazon EKS 叢集前，建議您完整了解 Amazon VPC 服務。如需詳細資訊，請參閱 Amazon VPC 使用者指南。

Amazon EKS 叢集、節點和 Kubernetes 資源已部署到 VPC。如果您要將現有的 VPC 與 Amazon EKS 結合使用，則該 VPC 必須符合 the section called “VPC 和子網要求” 中描述的要求。本主題說明如何使用 Amazon EKS provided AWS CloudFormation 範本建立符合 Amazon EKS 要求的 VPC。部

署範本後，您可以檢視範本所建立的資源，以確切了解其建立的資源以及這些資源的組態。如果您使用混合節點，您的 VPC 必須在其路由表中包含用於內部部署網路的路由。有關混合節點的網路需求詳細資訊，請參閱 the section called “準備聯網”。

先決條件

若要為 Amazon EKS 建立 VPC，您必須擁有建立 Amazon VPC 資源所需的 IAM 許可。這些資源包括 VPC、子網、安全群組、路由表和路由，以及網際網路和 NAT 閘道。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的使用公有子網路範例政策建立 VPC，以及服務授權參考中的動作的完整清單。

您可以建立含公有和私有子網路、僅公有子網路或僅私有子網路的 VPC。

公有和私有子網路

此 VPC 具有兩個公用和兩個私有子網路。公有子網路與具有網際網路閘道路由的路由表相關聯。不過，私有子網路的關聯路由表沒有連至網際網路閘道的路由。一個公有子網路和一個私有子網路會部署到相同的可用區域。其他公有和私有子網路會部署到相同區域中的第二個可用區域 AWS 。我們建議大多數部署使用此選項。

使用此選項，您可以將節點部署到私有子網。此選項可讓 Kubernetes 將負載平衡器部署到公有子網路，以便將流量負載平衡到私有子網路中節點上執行的 Pod。公有 IPv4 位址會自動指派給部署到公有子網路的節點，但公有 IPv4 位址不會指派給部署到私有子網路的節點。

您可以將 IPv6 地址指派給公有和私有子網路中的節點。私有子網路中的節點可以與叢集和其他 AWS 服務通訊。可以透過部署在每個可用區域中的 NAT 閘道 (使用 IPv4 位址) 或僅限傳出之網際網路閘道 (使用 IPv6 位址) 與網際網路通訊。部署的安全群組包含拒絕來自叢集或節點之外的來源的所有傳入流量，但卻允許所有傳出流量的規則。子網路會加上標籤，以便 Kubernetes 可以將負載平衡器部署到它們。

a. 開啟 AWS CloudFormation 主控台。
b. 從導覽列中選取支援 Amazon EKS AWS 的區域。
c. 選擇 Create stack (建立堆疊)、With new resources (standard) (使用新資源 (標準))。
d. 在 Prerequisite - Prepare template (必要條件 - 準備範本) 下，請確定選擇 Template is ready (範本已就緒)，然後在 Specify template (指定範本) 下選取 Amazon S3 URL。
e. 您可以建立僅支援 IPv4 的 VPC，或支援 IPv4 和 IPv6 的 VPC。將下列 URL 之一貼入 Amazon S3 URL 下的文字區域並選擇 Next (下一步)：
   - IPv4

```
https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/amazon-eks-vpc-private-subnets.yaml
```

- IPv4 和 IPv6

```
https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/amazon-eks-ipv6-vpc-public-private-subnets.yaml
```

a. 在 Specify stack details (識別堆疊詳細資訊) 頁面上，輸入參數，然後選擇 Next (下一步)。
   - 堆疊名稱：為您的 AWS CloudFormation 堆疊選擇堆疊名稱。例如，您可以使用在先前步驟中使用的範本名稱。此名稱僅能使用英數字元 (區分大小寫) 和連字號。必須以英數字元開頭，且長度不可超過 100 個字元。名稱在您要建立叢集 AWS 的區域和 AWS 帳戶中必須是唯一的。
   - VpcBlock：選擇適用於您 VPC 的 IPv4 CIDR 範圍。您部署的每個節點、Pod 和負載平衡器都會從此區塊指派一個 IPv4 位址。預設 IPv4 值為大多數實作提供足夠的 IP 位址，但如果沒有，則可以進行變更。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的 VPC 和子網路規模調整。您也可以在 VPC 建立後將其他 CIDR 區塊新增至 VPC。如果您正在建立 IPv6 VPC，則會從 Amazon 的全域單點傳送位址空間自動為您指派 IPv6 CIDR 範圍。
   - PublicSubnet01Block：指定適用於公有子網路 1 的 IPv4 CIDR 區塊。預設值為大多數實作提供足夠的 IP 位址，但如果沒有，則可以進行變更。如果建立 IPv6 VPC，則會在範本中為您指定此區塊。
   - PublicSubnet02Block：指定適用於公有子網路 2 的 IPv4 CIDR 區塊。預設值為大多數實作提供足夠的 IP 位址，但如果沒有，則可以進行變更。如果建立 IPv6 VPC，則會在範本中為您指定此區塊。
   - PrivateSubnet01Block：指定適用於私有子網路 1 的 IPv4 CIDR 區塊。預設值為大多數實作提供足夠的 IP 位址，但如果沒有，則可以進行變更。如果建立 IPv6 VPC，則會在範本中為您指定此區塊。
   - PrivateSubnet02Block：指定適用於私有子網路 2 的 IPv4 CIDR 區塊。預設值為大多數實作提供足夠的 IP 位址，但如果沒有，則可以進行變更。如果建立 IPv6 VPC，則會在範本中為您指定此區塊。
b. (選用) 在 Configure stack options (設定堆疊選項) 頁面上，為堆疊資源加上標籤，然後選擇 Next (下一步)。
c. 在 Review (檢閱) 頁面上，選擇 Create stack (建立堆疊)。
d. 堆疊建立後，從主控台將其選取，然後選擇 Outputs (輸出)。
e. 為已建立的 VPC 記錄 VpcId。建立叢集和節點時，您需要此值。

f. 記錄已建立的子網路的 SubnetIds (子網路識別碼)，以及您是否將它們建立為公有或私有子網路的子網路。建立叢集和節點時，您至少需要其中兩個。

g. 如果您建立了 IPv4 VPC，請跳過此步驟。如果建立 IPv6 VPC，則必須為範本建立的公有子網路啟用自動指派 IPv6 地址選項。已為私有子網路啟用該設定。若要啟用此設定，請完成下列步驟：

   i. 在 https://console.aws.amazon.com/vpc/ 開啟 Amazon VPC 主控台。

   ii. 在左側導覽窗格中，選擇 Subnets (子網路)。

   iii. 選擇一個公有子網路 (***stack-name***/SubnetPublic01 或 ***stack-name***/SubnetPublic02，其中包含文字 public)，然後選擇動作、編輯子網路設定。

   iv. 勾選 Enable auto-assign IPv6 address (啟用自動指派 IPv6 地址) 核取方塊，然後選擇 Save (儲存)。

   v. 為您的其他公有子網路再次完成上述步驟。

僅公有子網路

此 VPC 有三個公有子網路，部署到 AWS 區域中的不同可用區域。所有節點都會自動指派公有 IPv4 地址，並且可以透過網際網路閘道傳送和接收網際網路流量。部署的安全群組會拒絕所有輸入流量，並允許所有輸出流量。子網路會加上標籤，以便 Kubernetes 可以將負載平衡器部署到它們。

a. 開啟 AWS CloudFormation 主控台。

b. 從導覽列中選取支援 Amazon EKS AWS 的區域。

c. 選擇 Create stack (建立堆疊)、With new resources (standard) (使用新資源 (標準))。

d. 在 Prepare template (準備範本) 下，請確定 Template is ready (範本已就緒)，然後在 Template source (範本來源) 下選取 Amazon S3 URL。

e. 將下列 URL 貼入 Amazon S3 URL 下的文字區域並選擇 Next (下一步)：

```
https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/amazon-eks-vpc-sample.yaml
```

a. 在 Specify Details (指定詳細資訊) 頁面上，輸入參數，然後選擇 Next (下一步)。

- 堆疊名稱：為您的 AWS CloudFormation 堆疊選擇堆疊名稱。例如，您可以稱它為 *amazon-eks-vpc-sample*。此名稱僅能使用英數字元 (區分大小寫) 和連字號。必須以英數字元開頭，且長度不可超過 100 個字元。名稱在您要建立叢集 AWS 的區域和 AWS 帳戶中必須是唯一的。
- VpcBlock：選擇適用於您 VPC 的 CIDR 區塊。您部署的每個節點、Pod 和負載平衡器都會從此區塊指派一個 IPv4 位址。預設 IPv4 值為大多數實作提供足夠的 IP 位址，但如果沒有，則可以

進行變更。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的 VPC 和子網路規模調整。您也可以在 VPC 建立後將其他 CIDR 區塊新增至 VPC。

- Subnet01Block：指定適用於子網路 1 的 CIDR 區塊。預設值為大多數實作提供足夠的 IP 位址，但如果沒有，則可以進行變更。
- Subnet02Block：指定適用於子網路 2 的 CIDR 區塊。預設值為大多數實作提供足夠的 IP 位址，但如果沒有，則可以進行變更。
- Subnet03Block：指定適用於子網路 3 的 CIDR 區塊。預設值為大多數實作提供足夠的 IP 位址，但如果沒有，則可以進行變更。

b. (選用) 在 Options (選項) 頁面上，為堆疊資源加上標籤。選擇下一步。

c. 在 Review (檢閱) 頁面上，選擇 Create (建立)。

d. 堆疊建立後，從主控台將其選取，然後選擇 Outputs (輸出)。

e. 為已建立的 VPC 記錄 VpcId。建立叢集和節點時，您需要此值。

f. 為已建立的子網路記錄 SubnetIds。建立叢集和節點時，您至少需要其中兩個。

g. (選用) 叢集部署到此 VPC 的任何叢集都能夠為您的 Pod 和服務指派私有 `IPv4` 位址。如果要將叢集部署到此 VPC，以指派私有 `IPv6` 位址給您的 Pod 和 服務，則必須對 VPC、子網路、路由表和安全群組進行更新。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的將IPv4 中現有的 VPC 遷移至IPv6。Amazon EKS 要求您的子網路啟用 `Auto-assign IPv6` 地址選項。預設為停用狀態。

一個私有子網路

此 VPC 有三個私有子網路，部署到 AWS 區域中的不同可用區域。部署到子網路的資源無法存取網際網路，網際網路也無法存取子網路中的資源。範本會使用 AWS PrivateLink 為節點通常需要存取的多項 AWS 服務建立 VPC 端點。如果您的節點需要傳出網際網路存取權，您可以在建立 VPC 後的每個子網的可用區域中新增公有 NAT 閘道。建立的安全群組會拒絕所有傳入流量，部署到子網中的資源除外。安全群組亦允許所有傳出流量。子網路會加上標籤，以便 Kubernetes 可以對它們部署負載平衡器。如果您使用此組態來建立 VPC，請參閱 the section called “私有叢集” 了解其他要求和注意事項。

a. 開啟 AWS CloudFormation 主控台。

b. 從導覽列中選取支援 Amazon EKS AWS 的區域。

c. 選擇 Create stack (建立堆疊)、With new resources (standard) (使用新資源 (標準))。

d. 在 Prepare template (準備範本) 下，請確定 Template is ready (範本已就緒)，然後在 Template source (範本來源) 下選取 Amazon S3 URL。

e. 將下列 URL 貼入 Amazon S3 URL 下的文字區域並選擇 Next (下一步)：

```
https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/amazon-eks-fully-private-vpc.yaml
```

a. 在 Specify Details (指定詳細資訊) 頁面上，輸入參數，然後選擇 Next (下一步)。

- 堆疊名稱：為您的 AWS CloudFormation 堆疊選擇堆疊名稱。例如，您可以稱它為 *amazon-eks-fully-private-vpc*。此名稱僅能使用英數字元 (區分大小寫) 和連字號。必須以英數字元開頭，且長度不可超過 100 個字元。名稱在您要建立叢集 AWS 的區域和 AWS 帳戶中必須是唯一的。
- VpcBlock：選擇適用於您 VPC 的 CIDR 區塊。您部署的每個節點、Pod 和負載平衡器都會從此區塊指派一個 IPv4 位址。預設 IPv4 值為大多數實作提供足夠的 IP 位址，但如果沒有，則可以進行變更。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的 VPC 和子網路規模調整。您也可以在 VPC 建立後將其他 CIDR 區塊新增至 VPC。
- PrivateSubnet01Block：指定適用於子網路 1 的 CIDR 區塊。預設值為大多數實作提供足夠的 IP 位址，但如果沒有，則可以進行變更。
- PrivateSubnet02Block：指定適用於子網路 2 的 CIDR 區塊。預設值為大多數實作提供足夠的 IP 位址，但如果沒有，則可以進行變更。
- PrivateSubnet03Block：指定適用於子網路 3 的 CIDR 區塊。預設值為大多數實作提供足夠的 IP 位址，但如果沒有，則可以進行變更。

b. (選用) 在 Options (選項) 頁面上，為堆疊資源加上標籤。選擇下一步。

c. 在 Review (檢閱) 頁面上，選擇 Create (建立)。

d. 堆疊建立後，從主控台將其選取，然後選擇 Outputs (輸出)。

e. 為已建立的 VPC 記錄 VpcId。建立叢集和節點時，您需要此值。

f. 為已建立的子網路記錄 SubnetIds。建立叢集和節點時，您至少需要其中兩個。

g. (選用) 叢集部署到此 VPC 的任何叢集都能夠為您的 Pod 和服務指派私有 IPv4 位址。如果要將叢集部署到此 VPC，以指派私有 IPv6 位址給您的 Pod 和 服務，則必須對 VPC、子網路、路由表和安全群組進行更新。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的將IPv4 中現有的VPC遷移至IPv6。Amazon EKS 要求您的子網路啟用 Auto-assign IPv6 位址選項 (預設情況下處於停用狀態)。

檢視叢集的 Amazon EKS 安全群組要求

本主題說明 Amazon EKS 叢集的安全群組要求。

預設的叢集安全群組

當您建立叢集時，Amazon EKS 會建立名稱為 eks-cluster-sg-*my-cluster-uniqueID* 的安全群組。安全性群組具有以下預設規則：

規則類型	通訊協定	連接埠	來源	目標
傳入	全部	全部	Self	
傳出	全部	全部		0.0.0.0/0(IPv4) 或 ::/0 (IPv6)
傳出	全部	全部		自我 (適用於 EFA 流量)

預設的安全群組包括傳出規則，該規則允許具有相同安全群組目的地的 Elastic Fabric Adapter (EFA) 流量。這樣一來，叢集內的 EFA 流量可從 AI/ML 與高效能運算 (HPC) 工作負載中受益。如需詳細資訊，請參閱 Amazon Elastic Compute Cloud 使用者指南中的 Amazon EC2 上適用於 AI/ML 與 HPC 工作負載的 Elastic Fabric Adapter。

> **Important**
>
> 如果您的叢集不需要傳出規則到 0.0.0.0/0(IPv4)、 ::/0(IPv6)，您可以將其移除。如果將其移除，您仍必須具有限制叢集流量中列出的最低規則。如果您移除允許流量進出叢集安全群組本身的傳入或傳出規則，Amazon EKS 會在叢集更新時重新建立。

Amazon EKS 會將下列標籤新增至安全群組。如果您移除標籤，則 Amazon EKS 會在叢集更新時將其新增回安全群組。

金鑰	值
kubernetes.io/cluster/ *my-cluster*	owned
aws:eks:cluster-name	*my-cluster*
Name	eks-cluster-sg- *my-cluster* -*uniqueid*

Amazon EKS 會自動將此安全群組與以下資源關聯，其資源也會建立：

- 2—4 個彈性網路界面（本文件的其餘部分稱為網路界面），網路界面於叢集建立時建立。
- 您建立的任何受管節點群組中節點的網路界面。

此安全群組的設計目的是允許來自控制平面和 受管節點群組的所有流量彼此之間可以自由流動，以及允許所有的輸出流量流往任何目的地。當您建立叢集時，您可以（選用）指定自己的安全群組。如果您指定自己的安全群組，則 Amazon EKS 還會將您指定的安全群組與為叢集建立的網路界面建立關聯。但是，其不會與您建立的任何節點群組相關聯。

您可以在叢集的聯網區段 AWS 管理主控台 下的 中，判斷叢集安全群組的 ID。或者，您可以執行下列 CLI AWS 命令來執行此操作。

```
aws eks describe-cluster --name my-cluster --query
 cluster.resourcesVpcConfig.clusterSecurityGroupId
```

限制叢集流量

如果您需要限制 EKS 控制平面和節點之間的開放連接埠，您可以將預設傳出規則移除為 `0.0.0.0/0`(IPv4)/`::/0` (IPv6)，並新增叢集所需的下列最低規則。

如果您移除允許來源本身（來自叢集安全群組的流量）的所有流量的預設傳入規則，Amazon EKS 會在叢集更新時重新建立該規則。

如果您移除允許目的地本身（叢集安全群組的流量）的所有流量的預設傳出規則，Amazon EKS 會在叢集更新時重新建立該規則。

規則類型	通訊協定	站點	目標
傳出	TCP	443	叢集安全群組
傳出	TCP	10250	叢集安全群組
傳出 (DNS)	TCP 和 UDP	53	叢集安全群組

您還必須為以下流量新增規則：

- 您預期節點用於節點間通訊的任何通訊協定和連接埠。

- 需要對外網際網路存取，以便叢集在啟動時存取 Amazon EKS API，進行叢集自我檢查和節點註冊。若節點不可接入網際網路，則檢閱部署網際網路存取受到限制的私有叢集，了解其他考量。
- 需要虛擬節點存取以從 Amazon ECR 或是從提取映像所需的其他容器登錄檔 API 提取映像，例如 DockerHub。如需詳細資訊，請參閱《 AWS 一般參考》中的 AWS IP 地址範圍。
- 節點對 Amazon S3 進行存取。
- 需要 IPv4 和 IPv6 地址的個別規則。
- 若您使用混合式節點，則必須新增其他安全群組至叢集，以便允許與內部部署節點及 Pod 通訊。如需詳細資訊，請參閱the section called “準備聯網”。

如果您正在考慮限制規則，建議您先徹底測試 Pod，之後再將變更的規則套用到生產叢集。

如果您最初使用 Kubernetes 1.14 和 eks.3 平台版本或更舊版本部署叢集，則請考慮下列事項：

- 您可能還擁有控制平面和節點安全群組。建立這些群組時，群組包括之前的表格中列出的受限制的規則。不再需要這些安全群組，可以進行移除。但是，您需要確定您的叢集安全群組包含那些群組包含的規則。
- 如果您直接使用 API 部署叢集，或使用 CLI AWS 或 AWS CloudFormation 等工具來建立叢集，而且未在叢集建立時指定安全群組，則 VPC 的預設安全群組會套用至 Amazon EKS 建立的叢集網路介面。

共用安全群組

Amazon EKS 支援共用安全群組。

- 安全群組 VPC 關聯可關聯相同帳戶及區域中的安全群組與 VPC。
  - 了解 Amazon VPC 使用者指南中的如何關聯安全群組與多個 VPC。
- 共用安全群組可讓您與其他 AWS 帳戶共用安全群組。帳戶必須位於相同的 AWS 組織中。
  - 了解 Amazon VPC 使用者指南中的如何與組織共用安全群組。
- 安全群組一律僅限於單一 AWS 區域。

Amazon EKS 的考量事項

- EKS 對共用或多個 VPC 安全群組的要求與標準安全群組相同。

管理 Amazon EKS 叢集的聯網附加元件

數個聯網附加元件可用於 Amazon EKS 叢集。

內建附加元件

> **Note**
>
> 建立 EKS 叢集時：
>
> - 使用 AWS 主控台：內建附加元件（例如 CoreDNS、kube-proxy 等）會自動安裝為 Amazon EKS 附加元件。您可以透過 AWS 主控台、CLI 或 SDKs 輕鬆設定和更新這些項目。
> - 使用其他方法 (CLI、SDK 等)：相同的內建附加元件會以自我管理版本形式安裝，並作為標準 Kubernetes 部署執行。這些需要手動組態和更新，因為它們無法透過 AWS 工具進行管理。
>
> 建議使用 Amazon EKS 附加元件而非自我管理版本，以簡化附加元件管理，並透過 AWS 服務啟用集中式組態和更新。

適用於 Kubernetes 的 Amazon VPC CNI 外掛程式

此 CNI 附加元件會建立彈性網路介面並將其連接至 Amazon EC2 節點。附加元件還會將 VPC 中的私有 `IPv4` 或 `IPv6` 位址指派給每個 Pod 和服務。您的叢集預設會安裝此附加元件。如需詳細資訊，請參閱the section called “Amazon VPC CNI”。若使用混合節點，VPC CNI 仍會預設安裝，但會透過反親和性規則防止其在混合節點上執行。如需有關混合節點 CNI 選項的詳細資訊，請參閱 the section called “設定 CNI”。

CoreDNS

CoreDNS 是一個靈活、可擴展的 DNS 伺服器，可作為 Kubernetes 集群 DNS。CoreDNS Pod 為叢集中的所有 Pod 提供名稱解析。您的叢集預設會安裝此附加元件。如需詳細資訊，請參閱the section called “CoreDNS”。

`kube-proxy`

此附加元件會維護 Amazon EC2 節點上的網路規則，並啟用與 Pod 的網路通訊。您的叢集預設會安裝此附加元件。如需詳細資訊，請參閱the section called “kube-proxy”。

選用 AWS 的聯網附加元件

AWS Load Balancer 控制器

當您部署類型為 的 Kubernetes 服務物件時`loadbalancer`，控制器會建立 AWS Network Load Balancer。當您建立 Kubernetes 輸入物件時，控制器會建立 AWS Application Load Balancer。我們建議您使用此控制器來佈建 Network Load Balancer，而不是使用內建至 Kubernetes 的舊式雲端供應商控制器。如需詳細資訊，請參閱 AWS Load Balancer 控制器說明文件。

AWS 閘道器 API 控制器

此控制器可讓您使用 Kubernetes 閘道 API 在多重 Kubernetes 叢集之間連接服務。控制器使用 Amazon VPC Lattice 服務，連接在 Amazon EC2 執行個體、容器和無伺服器功能上執行的 Kubernetes 服務。如需詳細資訊，請參閱 AWS 閘道 API 控制器文件。

如需附加元件的詳細資訊，請參閱 the section called “Amazon EKS 附加元件”。

使用 Amazon VPC CNI 將 IP 指派給 Pod

> **Tip**
>
> 註冊即將舉行的 Amazon EKS 研討會。

> **Tip**
>
> 使用 Amazon EKS 自動模式，您無需安裝或升級聯網附加元件。自動模式包括 Pod 聯網和負載平衡功能。
> 如需詳細資訊，請參閱EKS 自動模式。

Kubernetes 專用 Amazon VPC CNI 外掛程式的附加元件會部署在 Amazon EKS 叢集中的每個 Amazon EC2 節點上。附加元件會建立彈性網路界面並將其連接到 Amazon EC2 節點。附加元件還會將 VPC 中的私有 `IPv4` 或 `IPv6` 地址指派給每個 Pod。

附加元件的版本會隨叢集中的每個 Fargate 節點一起部署，但您不會在 Fargate 節點上進行更新。其他相容的 CNI 外掛程式可用於 Amazon EKS 叢集，但這是 Amazon EKS 針對在 AWS 基礎設施上執行的節點支援的唯一 CNI 外掛程式。如需有關其他相容 CNI 外掛程式的詳細資訊，請參閱 the section

called “替代 CNI 外掛程式”。VPC CNI 不支援與混合節點搭配使用。如需有關混合節點 CNI 選項的詳細資訊，請參閱 the section called “設定 CNI”。

下表列出每個 Kubernetes 版本可用的 Amazon EKS 附加元件類型的最新版本。

Amazon VPC CNI 版本

Kubernetes 版本	Amazon EKS 類型的 VPC CNI 版本
1.36	v1.22.4-eksbuild.3
1.35	v1.22.4-eksbuild.3
1.34	v1.22.4-eksbuild.3
1.33	v1.22.4-eksbuild.3
1.32	v1.22.4-eksbuild.3
1.31	v1.22.4-eksbuild.3

Important

如果您要自行管理此附加元件，資料表中的版本可能與可用的自我管理版本不同。如需更新此附加元件之自我管理類型的詳細資訊，請參閱 the section called “更新 (自我管理)”。

Important

若要升級至 VPC CNI v1.12.0 或更新版本，您必須先升級至 VPC CNI v1.7.0。我們建議您一次更新一個次要版本。

考量事項

以下是使用該功能的考量事項。

- 版本指定為 `major-version.minor-version.patch-version-eksbuild.build-number`。

- 檢查每個功能的版本相容性。Kubernetes 專用 Amazon VPC CNI 外掛程式每個版本的某些功能都需要特定的 Kubernetes 版本。使用不同的 Amazon EKS 功能時，如果需要特定版本的附加元件，則會備註於功能文件中。除非您有執行舊版本的特定理由，否則建議您執行最新版本。

建立 Amazon VPC CNI (Amazon EKS 附加元件)

使用下面的步驟，來建立適用於 Kubernetes Amazon EKS 附加元件的 Amazon VPC CNI 外掛程式。

請檢閱考量之後再開始。如需詳細資訊，請參閱the section called “考量事項”。

先決條件

下面說明了適用於 Kubernetes Amazon EKS 附加元件的 Amazon VPC CNI 外掛程式的先決條件。

- 現有 Amazon EKS 叢集。若要部署叢集，請參閱 開始使用。
- 叢集的現有 AWS Identity and Access Management (IAM) OpenID Connect (OIDC) 提供者。若要判定您是否已經擁有一個，或是要建立一個，請參閱 the section called “IAM OIDC 提供商”。
- 連接有 AmazonEKS_CNI_Policy IAM 政策 (如果您的叢集使用 IPv4 系列) 或 IPv6 政策 (如果您的叢集使用 IPv6 系列) 的 IAM 角色。若要了解 VPC CNI 角色的相關詳細資訊，請參閱 the section called “為 IRSA 設定”。若要了解 IPv6 政策的相關詳細資訊，請參閱 the section called “為使用 IPv6 系列的叢集建立 IAM 政策”。

> **Important**
>
> 藉助適用於 Kubernetes 版本的 Amazon VPC CNI 外掛程式 `v1.16.0` 至 `v1.16.1` 來實作 CNI 規範版本 `v1.0.0`。若要了解 CNI 規範 `v1.0.0` 的相關詳細資訊，請參閱 GitHub 上的容器網路介面 (CNI) 規範。

程序

完成先決條件之後，使用下面的步驟來建立附加元件。

1. 查看叢集上目前安裝了哪些附加元件版本。

```
kubectl describe daemonset aws-node --namespace kube-system | grep amazon-k8s-cni: |
 cut -d : -f 3
```

範例輸出如下。

```
v1.16.4-eksbuild.2
```

2. 查看叢集上安裝的附加元件類型。視您用來建立叢集的工具而定，您的叢集上目前可能沒有安裝 Amazon EKS 附加元件類型。使用您叢集的名稱取代 *my-cluster*。

```
aws eks describe-addon --cluster-name my-cluster --addon-name vpc-cni --query
 addon.addonVersion --output text
```

如果傳回版本編號，則表明已在叢集上安裝 Amazon EKS 類型的附加元件，並且無需完成此程序中的剩餘步驟。如果傳回錯誤，則表明沒有在叢集上安裝 Amazon EKS 類型的附加元件。完成此程序的剩餘步驟以安裝該類型。

3. 儲存您目前安裝的附加元件。

```
kubectl get daemonset aws-node -n kube-system -o yaml > aws-k8s-cni-old.yaml
```

4. 使用 CLI AWS 建立附加元件。如果您想要使用 AWS 管理主控台 或 `eksctl` 建立附加元件，請參閱 the section called “建立附加元件”並`vpc-cni`指定附加元件名稱。將隨後的命令複製到您的裝置。視需要對命令進行下列修改，然後執行修改後的命令。
   - 使用您叢集的名稱取代 *my-cluster*。
   - 將 *v1.20.3-eksbuild.1* 取代為叢集版本最新版本資料表中列出的最新版本。若要查看最新版本資料表，請參閱 the section called “Amazon VPC CNI 版本”。
   - 使用帳戶 ID 取代 *111122223333*，並且使用所建立現有 IAM 角色的名稱取代 *AmazonEKSVPCCNIRole*。若要指定角色，您的叢集需要具有 IAM OpenID Connect (OIDC) 提供者。若要判定您的叢集是否已經擁有一個提供者，或是要建立一個提供者，請參閱 the section called “IAM OIDC 提供商”。

```
aws eks create-addon --cluster-name my-cluster --addon-name vpc-cni --addon-version
 v1.20.3-eksbuild.1 \
    --service-account-role-arn arn:aws: iam::111122223333:role/AmazonEKSVPCCNIRole
```

如果您已將自訂設定套用至與 Amazon EKS 附加元件的預設設定衝突的目前附加元件，建立動作可能會失敗。若建立失敗，您會收到錯誤，其中的訊息有助於您解決問題。或者，您可以將 `--resolve-conflicts OVERWRITE` 新增至上一條命令。這可讓附加元件覆寫任何現有的自訂設定。建立附加元件後，可使用自訂設定來更新該附加元件。

5. 確認叢集 Kubernetes 版本的附加元件的最新版本已新增至叢集。使用您叢集的名稱取代 *my-cluster*。

```
aws eks describe-addon --cluster-name my-cluster --addon-name vpc-cni --query
 addon.addonVersion --output text
```

建立附加元件的動作可能需要幾秒鐘的時間才能完成。

範例輸出如下。

```
v1.20.3-eksbuild.1
```

6. 如果您對原始附加元件制定自訂設定，請在建立 Amazon EKS 附加元件之前，使用在上一步中儲存的組態，以您的自訂設定更新 EKS 附加元件。請遵循 the section called “更新 (EKS 附加元件)” 中的步驟。
7. (選用) 安裝 `cni-metrics-helper` 到您的叢集。它湊集彈性網路介面和 IP 地址資訊，匯總叢集層級的指標，並將指標發佈至 Amazon CloudWatch。如需詳細資訊，請參閱 GitHub 上的 cni-metrics-helper。

更新 Amazon VPC CNI (Amazon EKS 附件元件)

更新適用於 Kubernetes 的 Amazon VPC CNI 外掛程式附加元件的 Amazon EKS 類型。如果您尚未將 Amazon EKS 類型的附加元件新增至叢集，您可遵循 the section called “建立” 來安裝。或遵循 the section called “更新 (自我管理)” 來更新其他類型的 VPC CNI 安裝。

1. 查看叢集上目前安裝了哪些附加元件版本。使用您的叢集名稱取代 *my-cluster*。

```
aws eks describe-addon --cluster-name my-cluster --addon-name vpc-cni --query
 "addon.addonVersion" --output text
```

範例輸出如下。

```
v1.20.0-eksbuild.1
```

參照 the section called “Amazon VPC CNI 版本” 中的最新版本資料表來比較版本。如果傳回的版本與最新版本資料表中的叢集 Kubernetes 版本相同，則表明您已經在叢集上安裝最新版本，不需要完成此程序的剩餘步驟。若您收到錯誤而非版本編號，則表明叢集上沒有安裝 Amazon EKS 類型的附加元件。您必須先建立附加元件，才能使用此程序進行更新。如需建立 VPC CNI 附加元件的 Amazon EKS 類型，您可遵循 the section called “建立”。

2. 儲存您目前安裝的附加元件。

```
kubectl get daemonset aws-node -n kube-system -o yaml > aws-k8s-cni-old.yaml
```

3. 使用 CLI AWS 更新您的附加元件。如果您想要使用 AWS 管理主控台 或 eksctl 來更新附加元件，請參閱 the section called “更新附加元件”。將隨後的命令複製到您的裝置。視需要對命令進行下列修改，然後執行修改後的命令。
   - 使用您叢集的名稱取代 *my-cluster*。
   - 使用叢集版本的最新版本資料表中列出的最新版本來取代 *v1.20.0-eksbuild.1*。
   - 使用帳戶 ID 取代 *111122223333*，並且使用所建立現有 IAM 角色的名稱取代 *AmazonEKSVPCCNIRole*。如需針對 VPC CNI 建立 IAM 角色，請參閱 the section called “步驟 1：建立 Kubernetes 專用 Amazon VPC CNI 外掛程式 IAM 角色”。若要指定角色，您的叢集需要具有 IAM OpenID Connect (OIDC) 提供者。若要判定您的叢集是否已經擁有一個提供者，或是要建立一個提供者，請參閱 the section called “IAM OIDC 提供商”。
   - `--resolve-conflicts PRESERVE` 選項會保留附加元件的現有組態值。如果對於附加元件設定，您已設定自訂值，但未使用此選項，Amazon EKS 會以其預設值覆寫您的值。如果您使用此選項，建議您在更新生產叢集上的附加元件之前，測試非生產叢集上的任何欄位和值變更。如果您將此值變更為 OVERWRITE，則所有設定都會變更為 Amazon EKS 預設值。如果您已設定任何設定的自訂值，則可能會使用 Amazon EKS 預設值覆寫這些值。如果您將此值變更為 none，Amazon EKS 不會變更任何設定的值，但更新可能會失敗。若更新失敗，您會收到錯誤訊息，以協助您解決衝突。
   - 如果您不更新組態設定，請從命令中移除 `--configuration-values '{`*`"env":{"AWS_VPC_K8S_CNI_EXTERNALSNAT":"true"}`*`}'`。如果您要更新組態設定，請將 *"env":{"AWS_VPC_K8S_CNI_EXTERNALSNAT":"true"}* 取代為您要設定的設定值。在此範例中，AWS_VPC_K8S_CNI_EXTERNALSNAT 環境變數設定為 true。您指定的值必須對組態結構描述有效。若您不知道組態結構描述，請執行 `aws eks describe-addon-configuration --addon-name vpc-cni --addon-version` *v1.20.0-eksbuild.1* ，其中請用要查看組態的附加元件版本編號取代 *v1.20.0-eksbuild.1*。結構描述會在輸出中傳回。如果您有任何現有的自訂組態，並且想要全部移除，同時將所有設定的值設定回 Amazon EKS 預設值，請從命令中移除 *"env":{"AWS_VPC_K8S_CNI_EXTERNALSNAT":"true"}*，這樣就會得到空白的 {}。如需每個設定的說明，請參閱 GitHub 上的 CNI 組態變數。

```
aws eks update-addon --cluster-name my-cluster --addon-name vpc-cni --addon-version
 v1.20.3-eksbuild.1 \
    --service-account-role-arn arn:aws: iam::111122223333:role/AmazonEKSVPCCNIRole
 \
```

```
    --resolve-conflicts PRESERVE --configuration-values '{"env":
{"AWS_VPC_K8S_CNI_EXTERNALSNAT":"true"}}'
```

更新動作可能需要幾秒鐘的時間才能完成。

4. 確認附加元件版本已更新。使用您叢集的名稱取代 *my-cluster*。

```
aws eks describe-addon --cluster-name my-cluster --addon-name vpc-cni
```

更新動作可能需要幾秒鐘的時間才能完成。

範例輸出如下。

```
{
    "addon": {
        "addonName": "vpc-cni",
        "clusterName": "my-cluster",
        "status": "ACTIVE",
        "addonVersion": "v1.20.3-eksbuild.1",
        "health": {
            "issues": []
        },
        "addonArn": "arn:aws: eks:region:111122223333:addon/my-cluster/vpc-
cni/74c33d2f-b4dc-8718-56e7-9fdfa65d14a9",
        "createdAt": "2023-04-12T18:25:19.319000+00:00",
        "modifiedAt": "2023-04-12T18:40:28.683000+00:00",
        "serviceAccountRoleArn": "arn:aws: iam::111122223333:role/
AmazonEKSVPCCNIRole",
        "tags": {},
        "configurationValues": "{\"env\":{\"AWS_VPC_K8S_CNI_EXTERNALSNAT\":\"true
\"}}"
    }
}
```

疑難排解

從早於 v1.13.2 的版本升級 VPC CNI 時，您必須在更新後取代叢集中的所有節點。v1.13.2 之前的版本使用 iptables-legacy 後端來插入適當功能所需的 iptables 規則，例如來源 NAT (SNAT)。

1.13.2 版是引進 iptables-wrapper 的重要版本，可自動偵測適當的 iptables 後端 (iptables-legacy 或 iptables-nft) 以插入鏈結和規則。由於效能限制，此變更符合上游 Kubernetes 決定移離舊版後端。

從 VPC CNI 1.13.2 版之前的版本升級之後需要替換節點，因為將規則引入 iptables-legacy 和 iptables-nft 後端可能會導致來自非主要 ENIs的流量發生意外行為。

更新 Amazon VPC CNI (自我管理的附件元件)

> **Important**
>
> 建議將附加元件的 Amazon EKS 類型新增到叢集，而不是使用附加元件的自我管理類型。如果您不熟悉類型之間的差異，則請參閱 the section called “Amazon EKS 附加元件”。如需將 Amazon EKS 附加元件新增至叢集的詳細資訊，請參閱 the section called “建立附加元件”。如果您無法使用 Amazon EKS 附加元件，我們建議您提交有關為何無法存取容器藍圖 GitHub 儲存庫的問題。

1. 確認您的叢集上已安裝 Amazon EKS 類型的附加元件。使用您叢集的名稱取代 *my-cluster*。

   ```
   aws eks describe-addon --cluster-name my-cluster --addon-name vpc-cni --query
    addon.addonVersion --output text
   ```

   如果傳回錯誤訊息，則表明沒有在叢集上安裝 Amazon EKS 類型的附加元件。若要自行管理附加元件，請完成此程序中的剩餘步驟來更新附加元件。如果傳回版本編號，則表明已在叢集上安裝附加元件的 Amazon EKS 類型。若要將其更新，請使用 the section called “更新附加元件” 中的程序，而不是使用此程序。如果您不熟悉附加元件類型之間的差異，請參閱 the section called “Amazon EKS 附加元件”。
2. 查看叢集上目前安裝了哪些容器映像版本。

   ```
   kubectl describe daemonset aws-node --namespace kube-system | grep amazon-k8s-cni: |
    cut -d : -f 3
   ```

   範例輸出如下。

   ```
   v1.20.0-eksbuild.1
   ```

   您的輸出可能不包含建置編號。
3. 備份您目前的設定，以便在更新版本後進行相同的設定。

   ```
   kubectl get daemonset aws-node -n kube-system -o yaml > aws-k8s-cni-old.yaml
   ```

若要檢閱可用版本並熟悉更新版本中的變更，請參閱 GitHub 上的發布版本。請注意，建議您更新最新版本資料表中列出的相同 major.minor.patch 版本，即使 GitHub 上有更新的版本也是如此。若要查看最新提供的版本資料表，請參閱 the section called “Amazon VPC CNI 版本”。資料表中列出的建置版本並未在 GitHub 上列出的自我管理版本中指定。透過以下列其中一種選項完成任務來更新您的版本：

- 如果沒有附加元件的任何自訂設定，則請執行 GitHub 上 `To apply this release:` 標題下的命令，以獲取正在更新到的發行版本。
- 如果有自訂設定，則請使用以下命令下載清單檔案。將 *https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/v1.20.0/config/master/aws-k8s-cni.yaml* 變更為您要更新之 GitHub 發行版本的 URL。

```
curl -O https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/v1.20.3/config/
master/aws-k8s-cni.yaml
```

如有必要，請使用您在前一步所建立備份中的自訂設定修改清單檔案，然後將修改後的清單檔案套用至叢集。如果您的節點無法存取從中提取映像的私有 Amazon EKS Amazon ECR 儲存庫 (請參閱資訊清單中開頭為 `image:` 的資料行)，則您必須下載映像，將其複製到您自己的儲存庫，然後修改資訊清單以便從儲存庫中提取映像。如需詳細資訊，請參閱the section called “將映像複製到儲存庫”。

```
kubectl apply -f aws-k8s-cni.yaml
```

4. 確認您的叢集上現在已安裝新版本。

```
kubectl describe daemonset aws-node --namespace kube-system | grep amazon-k8s-cni: |
 cut -d : -f 3
```

範例輸出如下。

```
v1.20.3
```

5. (選用) 安裝 `cni-metrics-helper` 到您的叢集。它湊集彈性網路介面和 IP 地址資訊，匯總叢集層級的指標，並將指標發佈至 Amazon CloudWatch。如需詳細資訊，請參閱 GitHub 上的 cni-metrics-helper。

設定 Amazon VPC CNI 外掛程式以使用 IRSA

Kubernetes 專用 Amazon VPC CNI 外掛程式是 Amazon EKS 叢集中供 Pod 聯網的聯網外掛程式。外掛程式負責將 VPC IP 地址配置給 Kubernetes Pod，並為每個節點上的 Pod 設定必要的聯網。

> **Note**
>
> Kubernetes 專用 Amazon VPC CNI 外掛程式也支援 Amazon EKS Pod 身分識別。如需詳細資訊，請參閱the section called “指派 IAM 角色”。

外掛程式：

- 需要 AWS Identity and Access Management (IAM) 許可。如果您的叢集使用 `IPv4` 系列，則會在 `AmazonEKS_CNI_Policy` AWS 受管政策中指定許可。如果您的叢集使用 `IPv6` 系列，則必須將許可新增至您建立的 IAM 政策；如需說明，請參閱 the section called “為使用 IPv6 系列的叢集建立 IAM 政策”。您可附加政策至 Amazon EKS 節點 IAM 角色，或附加至單獨 IAM 角色。有關將政策附加到 Amazon EKS 節點 IAM 角色的說明，請參閱 the section called “節點 IAM 角色”。建議如本主題所述，將其指派給不同的角色。
- 部署時，建立並設定為使用名為 `aws-node` 的 Kubernetes 服務帳戶。服務帳戶綁定到一個名為 `aws-node` 的 Kubernetes `clusterrole`，它被指派了所需的 Kubernetes 許可。

> **Note**
>
> 除非您封鎖對 IMDS 的存取，否則 Kubernetes 專用 Amazon VPC CNI 外掛程式的 Pod 有權存取指派給 Amazon EKS 節點 IAM 角色的許可。如需詳細資訊，請參閱限制存取指派給工作節點的執行個體設定檔。

- 需要現有的 Amazon EKS 叢集。若要部署叢集，請參閱 開始使用。
- 需要您叢集的現有 AWS Identity and Access Management (IAM) OpenID Connect (OIDC) 供應商。若要判定您是否已經擁有一個，或是要建立一個，請參閱 the section called “IAM OIDC 提供商”。

步驟 1：建立 Kubernetes 專用 Amazon VPC CNI 外掛程式 IAM 角色

1. 確定您叢集的 IP 系列。

```
aws eks describe-cluster --name my-cluster | grep ipFamily
```

範例輸出如下。

```
"ipFamily": "ipv4"
```

輸出可能會傳回 ipv6。

2. 建立 IAM 角色。您可以使用 eksctl或 kubectl和 AWS CLI 來建立 IAM 角色。

   eksctl

   - 建立 IAM 角色，並使用與您叢集 IP 系列相符的命令將 IAM 政策連接至此角色。命令會建立和部署建立 IAM 角色的 AWS CloudFormation 堆疊、連接您指定的政策，並使用所建立 aws-node IAM 角色的 ARN 註釋現有的 Kubernetes 服務帳戶。

     - IPv4

       用您的值取代*my-cluster*。

       ```
       eksctl create iamserviceaccount \
           --name aws-node \
           --namespace kube-system \
           --cluster my-cluster \
           --role-name AmazonEKSVPCCNIRole \
           --attach-policy-arn arn:aws: iam::aws:policy/AmazonEKS_CNI_Policy \
           --override-existing-serviceaccounts \
           --approve
       ```

     - IPv6

       用您的值取代*my-cluster*。將 *111122223333* 取代為您的帳戶 ID，以及將 *AmazonEKS_CNI_IPv6_Policy* 取代為您的 IPv6 策略名稱。如果沒有 IPv6 政策，請參閱 the section called “為使用 IPv6 系列的叢集建立 IAM 政策” 來建立一個。若要將 IPv6 與您的叢集一起使用，其必須滿足幾個要求。如需詳細資訊，請參閱the section called “IPv6”。

       ```
       eksctl create iamserviceaccount \
           --name aws-node \
           --namespace kube-system \
           --cluster my-cluster \
           --role-name AmazonEKSVPCCNIRole \
       ```

```
    --attach-policy-arn arn:aws: iam::111122223333:policy/
AmazonEKS_CNI_IPv6_Policy \
    --override-existing-serviceaccounts \
    --approve
```

kubectl 和 AWS CLI

i. 檢視叢集的 OIDC 提供商 URL。

```
aws eks describe-cluster --name my-cluster --query
 "cluster.identity.oidc.issuer" --output text
```

範例輸出如下。

```
https://oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE
```

如果未傳回任何輸出，則您必須為叢集建立 IAM OIDC 提供商。

ii. 將下列內容複製到名為 *vpc-cni-trust-policy.json* 的檔案。將 *111122223333* 取代為您的帳戶 ID，以及將 *EXAMPLED539D4633E53DE1B71EXAMPLE* 取代為前一個步驟中傳回的輸出。將 *region-code* 取代為您的叢集所在的 AWS 區域。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::111122223333:oidc-provider/
oidc.eks.us-east-1.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "oidc.eks.us-east-1.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:aud": "sts.amazonaws.com",
                    "oidc.eks.us-east-1.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:sub": "system:serviceaccount:kube-system:aws-
node"
                }
            }
        }
    ]
```

```
}
```

iii. 建立角色。您可以將 *AmazonEKSVPCCNIRole* 取代為選擇的任何名稱。

```
aws iam create-role \
  --role-name AmazonEKSVPCCNIRole \
  --assume-role-policy-document file://"vpc-cni-trust-policy.json"
```

iv. 將所需的 IAM 政策連接至角色。執行與叢集的 IP 系列相符的命令。

- IPv4

```
aws iam attach-role-policy \
  --policy-arn arn:aws: iam::aws:policy/AmazonEKS_CNI_Policy \
  --role-name AmazonEKSVPCCNIRole
```

- IPv6

  將 *111122223333* 取代為您的帳戶 ID，以及將 *AmazonEKS_CNI_IPv6_Policy* 取代為您的 IPv6 策略名稱。如果沒有 IPv6 政策，請參閱 the section called “為使用 IPv6 系列的叢集建立 IAM 政策” 來建立一個。若要將 IPv6 與您的叢集一起使用，其必須滿足幾個要求。如需詳細資訊，請參閱the section called “IPv6”。

```
aws iam attach-role-policy \
  --policy-arn arn:aws: iam::111122223333:policy/AmazonEKS_CNI_IPv6_Policy \
  --role-name AmazonEKSVPCCNIRole
```

v. 執行下列命令以使用您之前建立之 IAM 角色的 ARN 來標註 aws-node 服務帳戶。以您自己的值取代範例值。

```
kubectl annotate serviceaccount \
    -n kube-system aws-node \
    eks.amazonaws.com/role-arn=arn:aws: iam::111122223333:role/
AmazonEKSVPCCNIRole
```

3. （選用）設定 Kubernetes 服務帳戶所使用的 AWS Security Token Service 端點類型。如需詳細資訊，請參閱the section called “STS 端點”。

步驟 2：重新部署 Kubernetes 專用 Amazon VPC CNI 外掛程式 Pod

1. 刪除並重新建立任何與服務帳戶相關聯的現有 Pod，以套用憑證環境變數。註釋不適用於目前在沒有註釋的情況下執行的 Pod。下列命令會刪除現有的 aws-node DaemonSet Pod，並使用服務帳戶註釋來部署它們。

```
kubectl delete Pods -n kube-system -l k8s-app=aws-node
```

2. 確認 Pod 全部重新啟動。

```
kubectl get pods -n kube-system -l k8s-app=aws-node
```

3. 描述其中一個 Pod 並確認 AWS_WEB_IDENTITY_TOKEN_FILE 和 AWS_ROLE_ARN 環境變數存在。使用前一個步驟輸出中傳回的其中一個 Pod 名稱取代 *cpjw7*。

```
kubectl describe pod -n kube-system aws-node-cpjw7 | grep 'AWS_ROLE_ARN:\|
AWS_WEB_IDENTITY_TOKEN_FILE:'
```

   範例輸出如下。

```
AWS_ROLE_ARN: arn:aws: iam::111122223333:role/AmazonEKSVPCCNIRole
      AWS_WEB_IDENTITY_TOKEN_FILE:  /var/run/secrets/eks.amazonaws.com/
serviceaccount/token
      AWS_ROLE_ARN: arn:aws: iam::111122223333:role/AmazonEKSVPCCNIRole
      AWS_WEB_IDENTITY_TOKEN_FILE:            /var/run/secrets/eks.amazonaws.com/
serviceaccount/token
```

   Pod 包含兩個容器，因此會傳回兩組重複結果。兩個容器具有相同的值。

   如果您的 Pod 使用 AWS 區域端點，則上一個輸出也會傳回以下行。

```
AWS_STS_REGIONAL_ENDPOINTS=regional
```

步驟 3：從節點 IAM 角色移除 CNI 政策

如果您的 Amazon EKS 節點 IAM 角色目前已連接 AmazonEKS_CNI_Policy IAM (IPv4) 政策或 IPv6 政策，且您已建立個別的 IAM 角色、改為連接政策，並將其指派給 Kubernetes aws-node 服務帳戶，則建議您使用符合叢集 IP 系列的 AWS CLI 命令，從節點角色中移除政策。將 *AmazonEKSNodeRole* 取代為您的節點角色名稱。

- IPv4

```
aws iam detach-role-policy --role-name AmazonEKSNodeRole --policy-arn arn:aws:
 iam::aws:policy/AmazonEKS_CNI_Policy
```

- IPv6

  將 *111122223333* 取代為您的帳戶 ID，以及將 *AmazonEKS_CNI_IPv6_Policy* 取代為您的 IPv6 策略名稱。

```
aws iam detach-role-policy --role-name AmazonEKSNodeRole --policy-arn arn:aws:
 iam::111122223333:policy/AmazonEKS_CNI_IPv6_Policy
```

為使用 **IPv6** 系列的叢集建立 IAM 政策

如果您建立了使用 IPv6 系列的叢集，並且該叢集設定了版本 1.10.1 或更高版本的 Kubernetes 專用 Amazon VPC CNI 外掛程式附加元件，則需要建立可以指派給 IAM 角色的 IAM 政策。如果您現有的叢集在建立時未使用 IPv6 系列進行設定，則為了使用 IPv6，必須建立新的叢集。如需搭配使用 IPv6 與叢集的詳細資訊，請參閱 the section called “IPv6”。

1. 複製下列文字並將它儲存至名為 vpc-cni-ipv6-policy.json 的檔案。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:AssignIpv6Addresses",
                "ec2:DescribeInstances",
                "ec2:DescribeTags",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeInstanceTypes"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateTags"
            ],
```

```
            "Resource": [
                "arn:aws:ec2:*:*:network-interface/*"
            ]
        }
    ]
}
```

2. 建立 IAM 政策。

```
aws iam create-policy --policy-name AmazonEKS_CNI_IPv6_Policy --policy-document
 file://vpc-cni-ipv6-policy.json
```

了解 VPC CNI 模式與組態

適用於 Kubernetes 的 Amazon VPC CNI 外掛程式為 Pod 提供聯網功能。使用下面的資料，了解可用聯網功能的詳細資訊。

聯網功能	進一步了解
設定叢集指派 IPv6 位址到叢集、Pod 與服務	the section called “IPv6”
使用 Pod 的 IPv4 來源網路位址轉譯	the section called “傳出流量”
限制傳入與傳出 Pod 的網路流量	the section called “限制流量”
設定 Pod IP 地址的子網路選擇	the section called “子網路選擇”
自訂節點中的次要網路介面	the section called “自訂聯網”
增加節點的 IP 位址	the section called “增加 IP 位址”
使用適用於 Pod 網路流量的安全群組	the section called “Pod 的安全群組”
使用適用於 Pod 的多個網路介面	the section called “多個介面”

了解叢集、Pod 與服務的 IPv6 位址

適用於：具有 Amazon EC2 執行個體的 Pod 和 Fargate Pod

預設情況下，Kubernetes 會為您的 Pod 和服務指派 IPv4 位址。您可以將叢集設定為對 Pod 和服務指派 IPv6 位址，而不是指派 IPv4 位址。即使 Kubernetes 本身支援，Amazon EKS 也不支援雙重堆疊 Pod 或服務。因此，您無法同時將 IPv4 和 IPv6 位址指派給 Pod 和服務。

您可以在建立叢集時選擇要用於此叢集的 IP 系列。建立叢集後無法變更系列。

有關部署 Amazon EKS IPv6 叢集的教學，請參閱 the section called “部署”。

以下是使用該功能的考量事項：

IPv6 功能支援

- 不支援 Windows：不支援 Windows Pod 與服務。
- 需要 Nitro 型 EC2 節點：您只能IPv6搭配 AWS Nitro 型 Amazon EC2 或 Fargate 節點使用 。
- 支援 EC2 與 Fargate 節點：在 Amazon EC2 節點與 Fargate 節點上，皆可搭配 the section called “Pod 的安全群組” 使用 IPv6。
- 不支援 Outpost：無法在 AWS Outposts 上的 Amazon EKS 上使用 IPv6。
- 不支援 FSx for Lustre：不支援 the section called “Amazon FSx for Lustre”。
- 不支援自訂聯網：如果您以前使用 the section called “自訂聯網” 來幫助緩解 IP 位址耗盡情況，則可以改用 IPv6。您無法將自訂聯網與 IPv6 搭配使用。如果您使用自訂聯絡進行網路隔離，則可能需要繼續為叢集使用自訂聯網和 IPv4 系列。

IP 位址指派

- Kubernetes 服務：Kubernetes 服務僅獲指派IPv6地址。它們不會被指派 IPv4 位址。
- Pod：Pod 會被指派一個 IPv6 位址和一個主機本機的 IPv4 位址。主機本機 IPv4 位址透過使用與 VPC CNI 鏈結的主機本機 CNI 外掛程式指派，且位址不會報告給 Kubernetes 控制平面。只有在 Pod 需要與其他 Amazon VPC 或網際網路中的外部 IPv4 資源通訊時，才會使用此資源。主機本機 IPv4 位址會透過 VPC CNI 進行來源網路位址轉換 (SNAT)，對應到工作節點主要 ENI 的主要 IPv4 位址。
- Pod 和服務：Pod 和服務只會接收 IPv6 位址，而非 IPv4 位址。當 Pod 需要與外部 IPv4 端點通訊時，會使用節點本身的 NAT。此內建的 NAT 功能消除了對 DNS64 和 NAT64 的需求。對於需要公有網際網路存取的流量，Pod 的流量是轉換為公有 IP 位址的來源網路位址。
- 路由位址：當 Pod 在 VPC 外部通訊時，會保留其原始 IPv6 位址 (不會轉譯為節點的 IPv6 位址)。此流量會直接透過網際網路閘道或僅限輸出的網際網路閘道進行路由。
- 節點：為所有節點指派 IPv4 和 IPv6 位址。

- Fargate Pod：每個 Fargate Pod 接收來自 CIDR 的 IPv6 位址，為其部署在其中的子網路指定此位址。執行 Fargate Pod 的基礎硬體單位從 CIDR 取得唯一的 IPv4 和 IPv6 位址，這些位址指派給部署硬體單位的子網路。

如何搭配 EKS 使用 **IPv6**

- 建立新叢集：您必須建立新叢集，並指定要為該叢集使用 IPv6 系列。無法為從早期版本更新的叢集啟用 IPv6 系列。如需如何建立新叢集的指示，請參閱 the section called “部署”。
- 使用最新的 VPC CNI：部署 Amazon VPC CNI 版本 1.10.1 或更新版本。依預設，將部署此版本或更新版本。部署附加元件後，您就無法將 Amazon VPC CNI 附加元件降級為低於 1.10.1 的版本，且不先移除叢集中所有節點群組中的所有節點。
- 設定 **IPv6** 的 VPC CNI：如果您使用 Amazon EC2 節點，則必須使用 IP 字首委派和 IPv6 設定 Amazon VPC CNI 附加元件。如果您在建立叢集時選擇 IPv6 系列，則附加元件的 1.10.1 版本預設為此設定。自我管理或 Amazon EKS 附加元件都是這種情況。如需 IP 字首委派的詳細資訊，請參閱 the section called “增加 IP 位址”。
- 設定 **IPv4** 和 **IPv6** 位址：建立叢集時，您指定的 VPC 和子網路必須具有指派給所指定 VPC 和子網路的 IPv6 CIDR 區塊。此外，還必須為其指派 IPv4 CIDR 區塊。這是因為，即使您只想使用 IPv6，VPC 仍然需要 IPv4 CIDR 區塊才能正常工作。如需詳細資訊，請參閱 Amazon VPC 使用者指南中的建立 IPv6 CIDR 區塊與 VPC 的關聯。
- 將 IPv6 位址自動指派給節點：建立叢集和節點時，必須指定設定為自動指派 IPv6 位址的子網路。否則，您無法部署您的節點。根據預設，會停用此設定。如需詳細資訊，請參閱 Amazon VPC 使用者指南中的修改子網路的 IPv6 定址屬性。
- 設定路由表以使用 **IPv6**：指派給子網路的路由表必須具有 IPv6 位址的路由。如需詳細資訊，請參閱 Amazon VPC 使用者指南中的遷移至 IPv6。
- 設定 **IPv6** 的安全群組：您的安全群組必須允許 IPv6 位址。如需詳細資訊，請參閱 Amazon VPC 使用者指南中的遷移至 IPv6。
- 設定負載平衡器：使用版本 2.3.1或更新版本的 AWS Load Balancer控制器，在 IP 模式下使用 the section called “應用程式負載平衡”或 網路流量將 HTTP 應用程式負載平衡the section called “網路負載平衡”至具有任一負載平衡器的 IPv6 Pod，但不使用執行個體模式。如需詳細資訊，請參閱the section called “ AWS Load Balancer控制器”。
- 新增 **IPv6** IAM 政策：您必須將 IPv6 IAM 政策連接至節點 IAM 或 CNI IAM 角色。在兩者之間，建議您將其連接至 CNI IAM 角色。如需詳細資訊，請參閱the section called “為使用 IPv6 系列的叢集建立 IAM 政策”及the section called “步驟 1：建立 Kubernetes 專用 Amazon VPC CNI 外掛程式 IAM 角色”。

- 評估所有元件：在部署IPv6叢集之前，執行與您整合的應用程式、Amazon EKS 附加元件 AWS 和服務的完整評估。這是為了確保使用 IPv6 時一切都能正常工作。

部署 Amazon EKS **IPv6** 叢集和受管 Amazon Linux 節點

在本教學課程中，您將部署 IPv6 Amazon VPC、含 IPv6 系列的 Amazon EKS 叢集，以及含有 Amazon EC2 Amazon Linux 節點的受管節點群組。您無法在 IPv6 叢集中部署 Amazon EC2 Windows 節點。您也可以將 Fargate 節點部署到自己的叢集，但為了簡單起見，本主題中未提供這些指示。

先決條件

開始本教學之前，先完成下列動作：

安裝和設定下列所需工具和資源，以便建立和管理 Amazon EKS 叢集。

- 建議您先熟悉所有設定，再使用符合需求的設定來部署叢集。如需詳細資訊，請參閱此主題的 the section called “建立 叢集”、the section called “受管節點群組” 與考量。您只可在建立叢集時啟用某些設定。
- kubectl 命令列工具安裝在您的裝置或 AWS CloudShell 上。版本可以與您的叢集 Kubernetes 版本相同，或是為最多比該版本更舊一版或更新一版的次要版本。例如，如果您的叢集版本為 1.29，則可以搭配使用 kubectl 1.28、1.29 或 1.30 版。若要安裝或升級 kubectl，請參閱 the section called “設定 kubectl 和 eksctl”。
- 您使用的 IAM 安全主體必須具有使用 Amazon EKS IAM 角色、服務連結角色、 AWS CloudFormation、VPC 和相關資源的許可。如需詳細資訊，請參閱《IAM 使用者指南》中的動作和使用服務連結角色。
- 如果您使用 eksctl，請在電腦上安裝版本 0.215.0或更新版本。如需有關安裝或更新的指示，請參閱 eksctl 文件中的 Installation 一節。
- 在您的裝置或 AWS CloudShell 上安裝和設定的 AWS 命令列界面 (AWS CLI) 版本 1.27.160 2.12.3或更新版本。若要檢查您目前的版本，請使用 aws --version | cut -d / -f2 | cut -d ' ' -f1。適用於 macOS 的 yum、 apt-get或 Homebrew 等套件管理員通常是最新版本 CLI AWS 後面的幾個版本。若要安裝最新版本，請參閱《 AWS 命令列界面使用者指南》中的使用 aws 設定安裝 和 快速組態。 https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-quickstart.html#cli-configure-quickstart-config安裝在 AWS CloudShell 中的 AWS CLI 版本也可能是最新版本後面的幾個版本。若要更新它，請參閱《CloudShell AWS 使用者指南》中的將 CLI 安裝到您的主目錄。 AWS CloudShell 如果您使用 AWS CloudShell，您可能需要安裝 2.12.3 版或更新版

本或 1.27.160 版或更新版本的 AWS CLI，因為安裝在 AWS CloudShell 中的預設 AWS CLI 版本可能是舊版。

您可使用 eksctl 或 CLI 來部署 IPv6 叢集。

使用 eksctl 部署 IPv6 叢集

a. 建立 ipv6-cluster.yaml 檔案。將隨後的命令複製到您的裝置。視需要對命令進行下列修改，然後執行修改後的命令：

- 使用您的叢集名稱取代 *my-cluster*。此名稱僅能使用英數字元 (區分大小寫) 和連字號。必須以英數字元開頭，且長度不可超過 100 個字元。名稱在您要建立叢集 AWS 的區域和 AWS 帳戶中必須是唯一的。
- 將 *region-code* 取代為 Amazon EKS 支援的任何 AWS 區域。如需 AWS 區域清單，請參閱《AWS 一般參考指南》中的 Amazon EKS 端點和配額。
- 包含叢集版本的 version 值。如需詳細資訊，請參閱 Amazon EKS 支援的版本。
- 將 *my-nodegroup* 取代為您的節點群組名稱。節點群組名稱不可超過 63 個字元。它必須以字母或數字開頭，但也可以包含其餘字元的連字號和底線。
- 將 *t3.medium* 取代為任何 AWS Nitro 系統執行個體類型。

```
cat >ipv6-cluster.yaml <<EOF
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: my-cluster
  region: region-code
  version: "X.XX"

kubernetesNetworkConfig:
  ipFamily: IPv6

addons:
  - name: vpc-cni
    version: latest
  - name: coredns
    version: latest
  - name: kube-proxy
    version: latest
```

```
iam:
  withOIDC: true

managedNodeGroups:
  - name: my-nodegroup
    instanceType: t3.medium
EOF
```

b. 建立叢集。

```
eksctl create cluster -f ipv6-cluster.yaml
```

叢集建立需要幾分鐘的時間。在看到最後一行輸出之前，請勿繼續操作，該行看起來類似於以下輸出。

```
[...]
[#]  EKS cluster "my-cluster" in "region-code" region is ready
```

c. 確認已為預設 Pod 指派 IPv6 位址。

```
kubectl get pods -n kube-system -o wide
```

範例輸出如下。

```
NAME                       READY   STATUS    RESTARTS   AGE     IP
                    NODE                                                NOMINATED NODE
 READINESS GATES
aws-node-rslts             1/1     Running   1          5m36s
 2600:1f13:b66:8200:11a5:ade0:c590:6ac8   ip-192-168-34-75.region-
code.compute.internal   <none>           <none>
aws-node-t74jh             1/1     Running   0          5m32s
 2600:1f13:b66:8203:4516:2080:8ced:1ca9   ip-192-168-253-70.region-
code.compute.internal  <none>           <none>
coredns-85d5b4454c-cw7w2   1/1     Running   0          56m
 2600:1f13:b66:8203:34e5::                ip-192-168-253-70.region-
code.compute.internal  <none>           <none>
coredns-85d5b4454c-tx6n8   1/1     Running   0          56m
 2600:1f13:b66:8203:34e5::1               ip-192-168-253-70.region-
code.compute.internal  <none>           <none>
```

```
kube-proxy-btpbk            1/1     Running   0          5m36s
 2600:1f13:b66:8200:11a5:ade0:c590:6ac8   ip-192-168-34-75.region-
code.compute.internal   <none>           <none>
kube-proxy-jjk2g            1/1     Running   0          5m33s
 2600:1f13:b66:8203:4516:2080:8ced:1ca9   ip-192-168-253-70.region-
code.compute.internal   <none>           <none>
```

d. 確認已為預設服務指派 IPv6 地址。

```
kubectl get services -n kube-system -o wide
```

範例輸出如下。

```
NAME       TYPE        CLUSTER-IP          EXTERNAL-IP   PORT(S)         AGE
 SELECTOR
kube-dns   ClusterIP   fd30:3087:b6c2::a   <none>        53/UDP,53/TCP   57m   k8s-
app=kube-dns
```

e. (選用) 部署範例應用程式或部署 AWS Load Balancer 控制器和範例應用程式，以透過 the section called “應用程式負載平衡” 對 HTTP 應用程式進行負載平衡，或透過 the section called “網路負載平衡” 對 IPv6 Pod 的網路流量進行負載平衡。

f. 在完成為本教學建立的叢集和節點後，您應該使用如下命令清除所建立的資源。

```
eksctl delete cluster my-cluster
```

使用 CLI 部署 IPv6 AWS 叢集

Important

- 您必須以同一位使用者的身分完成本程序中的所有步驟。若要檢查目前使用者，請執行以下命令：

  ```
  aws sts get-caller-identity
  ```

- 必須在同一 shell 中完成此程序中的所有步驟。若干步驟使用前面步驟中設定的變數。如果在其他 shell 中設定變數值，則使用此變數的步驟將無法正常工作。如果您使用 AWS CloudShell 完成以下程序，請記住，如果您在大約 20—30 分鐘內沒有使用鍵盤或指標與其互動，則 shell 工作階段將結束。正在執行的進程不算作互動。

- 這些指示是針對 Bash shell 編寫的，在其他 shell 中可能需要進行調整。

將程序此步驟中的所有範例值取代為您自己的值。

a. 執行下列命令以設定稍後步驟中使用的某些變數。將 *region-code* 取代為您要部署資源 AWS 的區域。此值可以是 Amazon EKS 支援的任何 AWS 區域。如需 AWS 區域清單，請參閱《 AWS 一般參考指南》中的 Amazon EKS 端點和配額。使用您的叢集名稱取代 *my-cluster*。此名稱僅能使用英數字元 (區分大小寫) 和連字號。必須以英數字元開頭，且長度不可超過 100 個字元。名稱在您要建立叢集 AWS 的區域和 AWS 帳戶中必須是唯一的。將 *my-nodegroup* 取代為您的節點群組名稱。節點群組名稱不可超過 63 個字元。它必須以字母或數字開頭，但也可以包含其餘字元的連字號和底線。使用您的帳戶 ID 取代 *111122223333*。

```
export region_code=region-code
export cluster_name=my-cluster
export nodegroup_name=my-nodegroup
export account_id=111122223333
```

b. 使用符合 Amazon EKS 和 IPv6 要求的公有和私有子網建立 Amazon VPC。

i. 執行下列命令來設定 AWS CloudFormation 堆疊名稱的變數。您可以使用選擇的任何名稱取代 *my-eks-ipv6-vpc*。

```
export vpc_stack_name=my-eks-ipv6-vpc
```

ii. 使用 AWS CloudFormation 範本建立 IPv6 VPC。

```
aws cloudformation create-stack --region $region_code --stack-name $vpc_stack_name
 \
  --template-url https://s3.us-west-2.amazonaws.com/amazon-eks/
cloudformation/2020-10-29/amazon-eks-ipv6-vpc-public-private-subnets.yaml
```

堆疊需要幾分鐘的時間建立。執行下列命令。在命令輸出為 CREATE_COMPLETE 之前，請勿繼續下一步。

```
aws cloudformation describe-stacks --region $region_code --stack-name
 $vpc_stack_name --query Stacks[].StackStatus --output text
```

iii. 檢索已建立的公有子網路的 ID。

```
aws cloudformation describe-stacks --region $region_code --stack-name
 $vpc_stack_name \
    --query='Stacks[].Outputs[?OutputKey==`SubnetsPublic`].OutputValue' --output
 text
```

範例輸出如下。

```
subnet-0a1a56c486EXAMPLE,subnet-099e6ca77aEXAMPLE
```

iv. 為已建立的公有子網啟用自動指派 IPv6 地址選項。

```
aws ec2 modify-subnet-attribute --region $region_code --subnet-id
 subnet-0a1a56c486EXAMPLE --assign-ipv6-address-on-creation
aws ec2 modify-subnet-attribute --region $region_code --subnet-id
 subnet-099e6ca77aEXAMPLE --assign-ipv6-address-on-creation
```

v. 從部署的 AWS CloudFormation 堆疊擷取範本建立的子網路和安全群組名稱，並將其存放在變數中，以供後續步驟使用。

```
security_groups=$(aws cloudformation describe-stacks --region $region_code --
stack-name $vpc_stack_name \
    --query='Stacks[].Outputs[?OutputKey==`SecurityGroups`].OutputValue' --output
 text)

public_subnets=$(aws cloudformation describe-stacks --region $region_code --stack-
name $vpc_stack_name \
    --query='Stacks[].Outputs[?OutputKey==`SubnetsPublic`].OutputValue' --output
 text)

private_subnets=$(aws cloudformation describe-stacks --region $region_code --
stack-name $vpc_stack_name \
    --query='Stacks[].Outputs[?OutputKey==`SubnetsPrivate`].OutputValue' --output
 text)

subnets=${public_subnets},${private_subnets}
```

c. 建立叢集 IAM 角色，並將所需的 Amazon EKS IAM 受管政策連接到該角色。Amazon EKS 管理的 Kubernetes 叢集會代表您呼叫其他 AWS 服務，以管理您與服務搭配使用的資源。

i. 執行下列命令以建立 `eks-cluster-role-trust-policy.json` 檔案。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

ii. 執行下列命令，以設定角色名稱的變數。您可以將 *myAmazonEKSClusterRole* 取代為選擇的任何名稱。

```
export cluster_role_name=myAmazonEKSClusterRole
```

iii. 建立角色。

```
aws iam create-role --role-name $cluster_role_name --assume-role-policy-document
 file://"eks-cluster-role-trust-policy.json"
```

iv. 擷取 IAM 角色的 ARN 並將其儲存在變數中，以便在稍後的步驟中使用。

```
CLUSTER_IAM_ROLE=$(aws iam get-role --role-name $cluster_role_name --
query="Role.Arn" --output text)
```

v. 將必要的 Amazon EKS 受管 IAM 政策連接到角色。

```
aws iam attach-role-policy --policy-arn arn:aws: iam::aws:policy/
AmazonEKSClusterPolicy --role-name $cluster_role_name
```

d. 建立叢集。

```
aws eks create-cluster --region $region_code --name $cluster_name --kubernetes-
version 1.XX \
   --role-arn $CLUSTER_IAM_ROLE --resources-vpc-config subnetIds=
$subnets,securityGroupIds=$security_groups \
   --kubernetes-network-config ipFamily=ipv6
```

i. 請注意：您可能會收到錯誤，表示在請求中的其中一個可用區域沒有足夠的容量可建立 Amazon EKS 叢集。如果發生這種情況，錯誤輸出包含的可用區域可支援新的叢集。使用至少兩個位於帳戶的支援可用區域子網路來建立您的叢集。如需詳細資訊，請參閱the section called “容量不足”。

   建立叢集需要幾分鐘才能完成。執行下列命令。在命令輸出為 ACTIVE 之前，請勿繼續下一步。

```
aws eks describe-cluster --region $region_code --name $cluster_name --query
 cluster.status
```

e. 為叢集建立或更新 kubeconfig 檔案，以便能夠與您的叢集通訊。

```
aws eks update-kubeconfig --region $region_code --name $cluster_name
```

   根據預設，已在 ~/.kube 建立了 config 檔案，或者已將新叢集的組態新增至 ~/.kube 中的現有 config 檔案。

f. 建立節點 IAM 角色。

i. 執行下列命令以建立 vpc-cni-ipv6-policy.json 檔案。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:AssignIpv6Addresses",
                "ec2:DescribeInstances",
                "ec2:DescribeTags",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeInstanceTypes"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateTags"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:network-interface/*"
```

```
            ]
        }
    ]
}
```

ii. 建立 IAM 政策。

```
aws iam create-policy --policy-name AmazonEKS_CNI_IPv6_Policy --policy-document
 file://vpc-cni-ipv6-policy.json
```

iii. 執行下列命令以建立 node-role-trust-relationship.json 檔案。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

iv. 執行下列命令，以設定角色名稱的變數。您可以將 *AmazonEKSNodeRole* 取代為選擇的任何名稱。

```
export node_role_name=AmazonEKSNodeRole
```

v. 建立 IAM 角色。

```
aws iam create-role --role-name $node_role_name --assume-role-policy-document
 file://"node-role-trust-relationship.json"
```

vi. 將 IAM 政策連接至 IAM 角色。

```
aws iam attach-role-policy --policy-arn arn:aws: iam::$account_id:policy/
AmazonEKS_CNI_IPv6_Policy \
    --role-name $node_role_name
```

> **Important**
>
> 為實現本教學的簡單性，將政策連接至此 IAM 角色。但是，在生產叢集中，我們建議將政策連接至單獨的 IAM 角色。如需詳細資訊，請參閱the section called “為 IRSA 設定”。

vii.將兩個所需的 IAM 受管政策連接到 IAM 角色。

```
aws iam attach-role-policy --policy-arn arn:aws: iam::aws:policy/
AmazonEKSWorkerNodePolicy \
  --role-name $node_role_name
aws iam attach-role-policy --policy-arn arn:aws: iam::aws:policy/
AmazonEC2ContainerRegistryReadOnly \
  --role-name $node_role_name
```

viii擷取 IAM 角色的 ARN 並將其儲存在變數中，以便在稍後的步驟中使用。

```
node_iam_role=$(aws iam get-role --role-name $node_role_name --query="Role.Arn" --
output text)
```

g. 建立受管節點群組。

i. 查看您在上一個步驟中建立的子網路的 ID。

```
echo $subnets
```

範例輸出如下。

```
subnet-0a1a56c486EXAMPLE,subnet-099e6ca77aEXAMPLE,subnet-0377963d69EXAMPLE,subnet-0c05f819
```

ii. 建立節點群組。將 *0a1a56c486EXAMPLE*、*099e6ca77aEXAMPLE*、*0377963d69EXAMPLE* 和 *0c05f819d5EXAMPLE* 取代為上一個步驟輸出中傳回的值。請確保從以下命令中的上一個輸出刪除子網路 ID 之間的逗號。您可以將 *t3.medium* 取代為任何 AWS Nitro 系統執行個體類型。

```
aws eks create-nodegroup --region $region_code --cluster-name $cluster_name --
nodegroup-name $nodegroup_name \
    --subnets subnet-0a1a56c486EXAMPLE subnet-099e6ca77aEXAMPLE
 subnet-0377963d69EXAMPLE subnet-0c05f819d5EXAMPLE \
    --instance-types t3.medium --node-role $node_iam_role
```

此節點群組需要幾分鐘的時間建立。執行下列命令。在傳回的輸出為 ACTIVE 之前，請勿進行下一個步驟。

```
aws eks describe-nodegroup --region $region_code --cluster-name $cluster_name --
nodegroup-name $nodegroup_name \
    --query nodegroup.status --output text
```

h. 確認在 IP 欄中已為預設 Pod 指派 IPv6 位址。

```
kubectl get pods -n kube-system -o wide
```

範例輸出如下。

```
NAME                       READY   STATUS    RESTARTS   AGE     IP
                    NODE                                              NOMINATED NODE
 READINESS GATES
aws-node-rslts             1/1     Running   1          5m36s
 2600:1f13:b66:8200:11a5:ade0:c590:6ac8   ip-192-168-34-75.region-
code.compute.internal   <none>           <none>
aws-node-t74jh             1/1     Running   0          5m32s
 2600:1f13:b66:8203:4516:2080:8ced:1ca9   ip-192-168-253-70.region-
code.compute.internal  <none>           <none>
coredns-85d5b4454c-cw7w2   1/1     Running   0          56m
 2600:1f13:b66:8203:34e5::                ip-192-168-253-70.region-
code.compute.internal  <none>           <none>
coredns-85d5b4454c-tx6n8   1/1     Running   0          56m
 2600:1f13:b66:8203:34e5::1               ip-192-168-253-70.region-
code.compute.internal  <none>           <none>
kube-proxy-btpbk           1/1     Running   0          5m36s
 2600:1f13:b66:8200:11a5:ade0:c590:6ac8   ip-192-168-34-75.region-
code.compute.internal   <none>           <none>
kube-proxy-jjk2g           1/1     Running   0          5m33s
 2600:1f13:b66:8203:4516:2080:8ced:1ca9   ip-192-168-253-70.region-
code.compute.internal  <none>           <none>
```

i. 確認在 IP 欄中已為預設服務指派 IPv6 地址。

```
kubectl get services -n kube-system -o wide
```

範例輸出如下。

```
NAME       TYPE        CLUSTER-IP          EXTERNAL-IP   PORT(S)         AGE
 SELECTOR
kube-dns   ClusterIP   fd30:3087:b6c2::a   <none>        53/UDP,53/TCP   57m   k8s-
app=kube-dns
```

j. (選用) 部署範例應用程式或部署 AWS Load Balancer 控制器和範例應用程式，以透過 the section called “應用程式負載平衡” 對 HTTP 應用程式進行負載平衡，或透過 the section called “網路負載平衡” 對 IPv6 Pod 的網路流量進行負載平衡。

k. 在完成為本教學建立的叢集和節點後，您應該使用如下命令清除所建立的資源。在刪除之前，請確保您沒有使用本教程以外的任何資源。

i. 如果您在與完成之前步驟不同的 shell 中完成此步驟，請設定之前步驟中使用的所有變數的值，並將範例值取代為在完成之前步驟時指定的值。如果您要在與完成之前步驟相同的 shell 中完成此步驟，請跳至下一步。

```
export region_code=region-code
export vpc_stack_name=my-eks-ipv6-vpc
export cluster_name=my-cluster
export nodegroup_name=my-nodegroup
export account_id=111122223333
export node_role_name=AmazonEKSNodeRole
export cluster_role_name=myAmazonEKSClusterRole
```

ii. 刪除節點群組。

```
aws eks delete-nodegroup --region $region_code --cluster-name $cluster_name --
nodegroup-name $nodegroup_name
```

刪除需要幾分鐘的時間。執行下列命令。如果傳回任何輸出，請勿繼續下一步。

```
aws eks list-nodegroups --region $region_code --cluster-name $cluster_name --query
 nodegroups --output text
```

iii. 刪除叢集。

```
aws eks delete-cluster --region $region_code --name $cluster_name
```

叢集需要幾分鐘的時間刪除。在繼續之前，請務必使用下列命令來確定叢集已被刪除。

```
aws eks describe-cluster --region $region_code --name $cluster_name
```

在輸出與下列輸出類似之前，請不要進行下一個步驟。

```
An error occurred (ResourceNotFoundException) when calling the DescribeCluster
 operation: No cluster found for name: my-cluster.
```

iv. 刪除您建立的 IAM 資源。將 *AmazonEKS_CNI_IPv6_Policy* 取代為您選擇的名稱 (如果您選擇的名稱與之前步驟中使用的名稱不同)。

```
aws iam detach-role-policy --role-name $cluster_role_name --policy-arn arn:aws:
 iam::aws:policy/AmazonEKSClusterPolicy
aws iam detach-role-policy --role-name $node_role_name --policy-arn arn:aws:
 iam::aws:policy/AmazonEKSWorkerNodePolicy
aws iam detach-role-policy --role-name $node_role_name --policy-arn arn:aws:
 iam::aws:policy/AmazonEC2ContainerRegistryReadOnly
aws iam detach-role-policy --role-name $node_role_name --policy-arn arn:aws: iam::
$account_id:policy/AmazonEKS_CNI_IPv6_Policy
aws iam delete-policy --policy-arn arn:aws: iam::$account_id:policy/
AmazonEKS_CNI_IPv6_Policy
aws iam delete-role --role-name $cluster_role_name
aws iam delete-role --role-name $node_role_name
```

v. 刪除建立 VPC 的 AWS CloudFormation 堆疊。

```
aws cloudformation delete-stack --region $region_code --stack-name $vpc_stack_name
```

啟用 Pod 的傳出網際網路存取

適用於：包含 Amazon EC2 執行個體的 Linux IPv4 Fargate 節點、Linux 節點

如果您使用 IPv6 系列部署叢集，則本主題中的資訊並不適用於您的叢集，因為 IPv6 位址未經過網路轉換。如需搭配使用 IPv6 與叢集的詳細資訊，請參閱 the section called “IPv6”。

預設情況下，叢集中的每個 Pod 均從與部署 Pod 的 VPC 關聯的無類別域間路由 (CIDR) 區塊指派一個私有 IPv4 位址。同一 VPC 中的 Pod 使用這些私有 IP 位址作為端點相互通訊。當 Pod 與任何不在與您的 VPC 關聯的 CIDR 區塊內的 IPv4 位址通訊時，預設情況下，Amazon VPC CNI 外掛程式 (適用於 Linux 或 Windows) 會將 Pod 的 IPv4 位址轉譯為正在執行 Pod 的節點的主要彈性網路介面的主要私有 IPv4 位址 *。

> **Note**
>
> 對於 Windows 節點，還有其他詳細資訊需要考慮。依預設，適用於 Windows 的 VPC CNI 外掛程式為使用網路組態定義，其中，通往相同 VPC 內目的地的流量會從 SNAT 中排除。這表示內部 VPC 通訊已停用 SNAT，且配置給 Pod 的 IP 位址可在 VPC 內路由。但是通往 VPC 外目的地的流量會將來源 Pod IP SNAT 到執行個體 ENI 的主要 IP 位址。Windows 的此預設組態可確保 Pod 可以使用與主機執行個體相同的方式存取 VPC 外部的網路。

由於此行為：

- 如果您的 Pod 在 上執行的節點具有公有或彈性 IP 地址，且位於路由表具有路由至網際網路閘道的公有子網路中，或位於路由表具有路由至 NAT 閘道的私有子網路中，則您的 Pod 可以與網際網路資源通訊。 https://docs.aws.amazon.com/vpc/latest/userguide/VPC_Internet_Gateway.html我們建議儘可能將節點部署到私有子網。
- 對於版本比 `1.8.0` 更早的外掛程式，使用 VPC 對等互連、傳輸 VPC 或 AWS Direct Connect 連線到叢集 VPC 的網路或 VPC 的資源，無法針對輔助彈性網路介面背後的 Pod 啟動通訊。您的 Pod 可以啟動與這些資源的通訊並接收來自資源的回應。

如果在您的環境中下列任一陳述式成立，請使用下列命令變更預設組態。

- 您在網路或 VPC 擁有資源，且其使用 VPC 對等互連、傳輸 VPC 或 AWS Direct Connect 連線到叢集，且其需使用 `IPv4` 位址與您的 Pod 啟動通訊，而您的外掛程式版本早於 `1.8.0`。
- 您希望在流量離開節點時保留 Pod 的來源 IP 地址，並由 NAT 閘道而非節點執行 SNAT，例如，網路設備或日誌會觀察 Pod 的 IP 地址。您的節點必須位於具有 NAT 閘道路由的私有子網路中，因為 Pod 來源的流量無法透過網際網路閘道路由。

```
kubectl set env daemonset -n kube-system aws-node AWS_VPC_K8S_CNI_EXTERNALSNAT=true
```

> **Note**
>
> `AWS_VPC_K8S_CNI_EXTERNALSNAT` 和 `AWS_VPC_K8S_CNI_EXCLUDE_SNAT_CIDRS` CNI 組態變數不適用於 Windows 節點。Windows 不支援停用 SNAT。對於從 SNAT 排除 `IPv4` CIDR，您可透過在 Windows 引導指令碼中指定 `ExcludedSnatCIDRs` 參數來進行定義。如需使用此參數的詳細資訊，請參閱 the section called “引導指令碼組態參數”。

主機聯網

* 如果 Pod 規格包含 `hostNetwork=true` (預設為 `false`)，則其 IP 位址不會被轉譯為其他位址。根據預設，在您叢集上執行的 `kube-proxy` 和 Kubernetes 專用 Amazon VPC CNI 外掛程式 Pod 屬於這種情況。對於這些 Pod，IP 位址與節點的主 IP 位址相同，因此未轉譯 Pod 的 IP 位址。如需有關 Pod `hostNetwork` 設定的詳細資訊，請參閱 Kubernetes API 參考中的 PodSpec v1 核心。

使用 Kubernetes 網路政策限制 Pod 流量

概觀

根據預設，Kubernetes 中沒有任何限制施加於 IP 位址、連接埠，或叢集中任何 Pod 之間或您的 Pod 與任何其他網路中的資源之間的連結。您可以使用 Kubernetes 網路政策來限制往返 Pod 之間的網路流量。如需詳細資訊，請參閱 Kubernetes 文件的網路政策。

標準網路政策

您可以使用 標準`NetworkPolicy`來分割叢集中的 pod-to-pod 流量。這些網路政策會在 OSI 網路模型的第 3 層和第 4 層運作，可讓您控制 Amazon EKS 叢集內 IP 地址或連接埠層級的流量。標準網路政策的範圍是命名空間層級。

使用案例

- 分割工作負載之間的網路流量，以確保只有相關的應用程式可以互相通訊。
- 使用 政策在命名空間層級隔離租用戶，以強制執行網路分離。

範例

在下面的政策中，來自太陽命名空間中 Webapp Pod 的輸出流量受到限制。

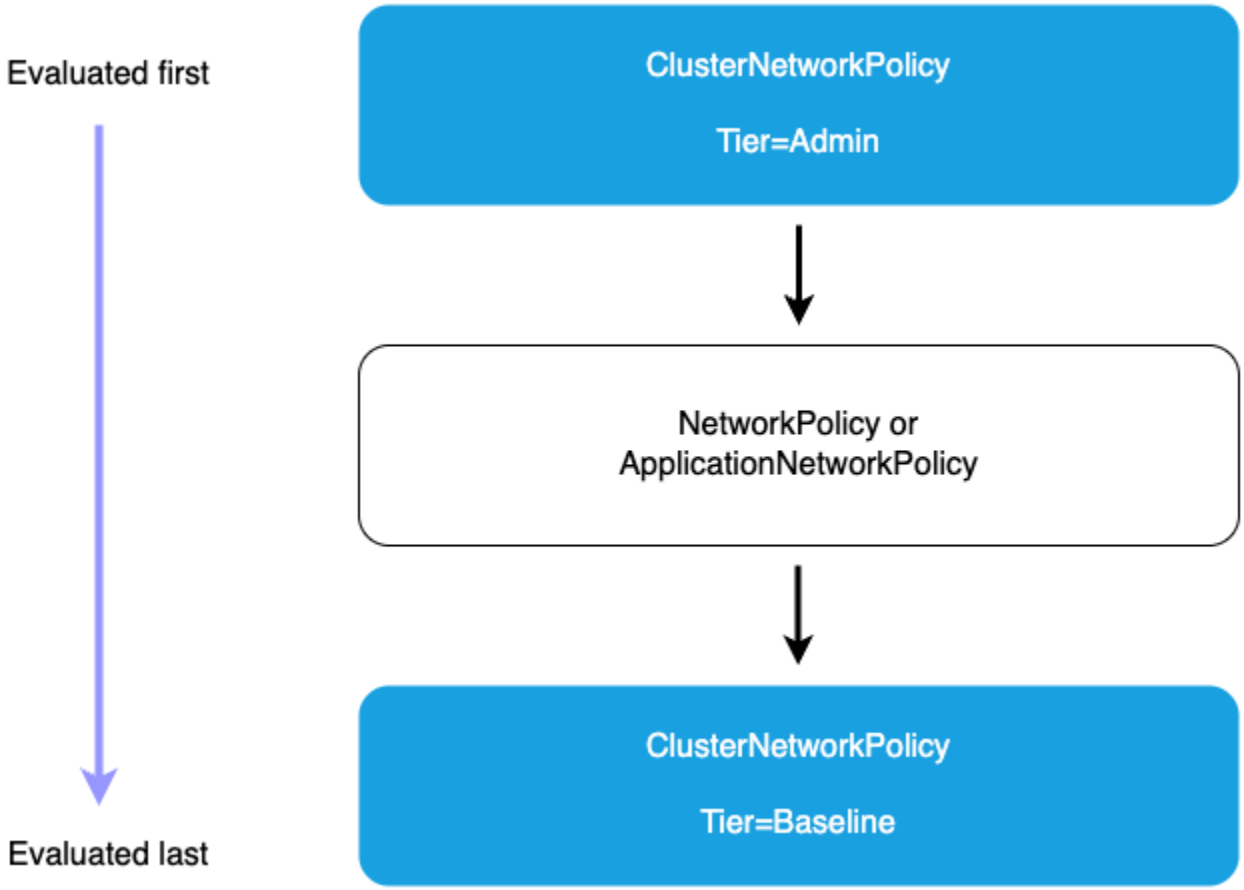
```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: webapp-egress-policy
  namespace: sun
spec:
  podSelector:
    matchLabels:
      role: webapp
  policyTypes:
```

```
  - Egress
  egress:
  - to:
    - namespaceSelector:
        matchLabels:
          name: moon
      podSelector:
        matchLabels:
          role: frontend
    ports:
    - protocol: TCP
      port: 8080
  - to:
    - namespaceSelector:
        matchLabels:
          name: stars
      podSelector:
        matchLabels:
          role: frontend
    ports:
    - protocol: TCP
      port: 8080
```

此政策適用於sun命名空間role: webapp中具有 標籤的 Pod。

- 允許流量：TCP 連接埠上moon命名空間role: frontend中具有 標籤的 Pod 8080
- 允許流量：TCP 連接埠上stars命名空間role: frontend中具有 標籤的 Pod 8080
- 封鎖流量：來自 Pod webapp 的所有其他傳出流量都會隱含拒絕

管理員（或叢集）網路政策



您可以使用 ClusterNetworkPolicy強制執行適用於整個叢集的網路安全標準。您可以使用單一政策來集中管理叢集中不同工作負載的網路存取控制，而不重複定義和維護每個命名空間的不同政策，無論其命名空間為何。

使用案例

- 集中管理 EKS 叢集中所有（或一部分）工作負載的網路存取控制。
- 定義整個叢集的預設網路安全狀態。
- 以更具營運效率的方式，將組織安全標準擴展到叢集的範圍。

範例

在下列政策中，您可以明確封鎖來自其他命名空間的叢集流量，以防止網路存取敏感工作負載命名空間。

```
apiVersion: networking.k8s.aws/v1alpha1
```

```
kind: ClusterNetworkPolicy
metadata:
  name: protect-sensitive-workload
spec:
  tier: Admin
  priority: 10
  subject:
    namespaces:
      matchLabels:
        kubernetes.io/metadata.name: earth
  ingress:
    - action: Deny
      from:
      - namespaces:
          matchLabels: {} # Match all namespaces.
      name: select-all-deny-all
```

重要說明

下列組態支援適用於 Kubernetes 的 Amazon VPC CNI 外掛程式中的網路政策。

- 適用於標準和管理網路政策的 Amazon VPC CNI 外掛程式 1.21.0 版 （或更新版本）。
- 設定為 IPv4 或 IPv6 地址的叢集。
- 您可以搭配 Pod 的安全群組來使用網路政策。有了網路政策，您就可以控制所有叢集內通訊。透過 Pod 的安全群組，您可以控制從 Pod 內的應用程式存取 AWS 服務。
- 您可以搭配自訂聯網和字首委派來使用網路政策。

考量事項

架構

- 使用 Kubernetes 專用 Amazon VPC CNI 外掛程式將 Kubernetes 專用 Amazon VPC CNI 外掛程式網路政策套用至您的叢集時，您只能將這些政策套用至 Amazon EC2 Linux 節點。您無法將這些政策套用至 Fargate 或 Windows 節點。
- 網路政策僅適用於 IPv4或 IPv6地址，但不適用於兩者。在 IPv4叢集中，VPC CNI 會將IPv4地址指派給 Pod 並套用IPv4政策。在 IPv6叢集中，VPC CNI 會將IPv6地址指派給 Pod 並套用IPv6政策。套用至 IPv6 叢集的任何 IPv4 網路政策規則都會遭到忽略。套用至 IPv4 叢集的任何 IPv6 網路政策規則都會遭到忽略。

網路政策

- Amazon EKS 會針對具有`metadata.ownerReferences`欄位集的 Pod 最佳化網路政策強制執行。這包括由 Deployments、StatefulSets、DaemonSets、Jobs 和 CronJobs 等控制器管理的 Pod。在沒有控制器的情況下直接建立的獨立 Pod 未`metadata.ownerReferences`設定，而且這些 Pod 的網路政策強制執行可能無法可靠地運作。
- 您可以將多個網路政策套用至相同的 Pod。當設定了兩個以上選取相同 Pod 的政策，所有政策皆會套用至 Pod。
- 在所有網路政策中，單一 IP 位址範圍 (CIDR) 的連接埠和通訊協定組合數目上限為 24。選取器，如 `namespaceSelector` 解析為一或多個 CIDRs。如果多個選擇器解析為單一 CIDR，或者您在相同或不同的網路政策中多次指定相同的直接 CIDR，這些都會計入此限制。
- 對於任何一種您的 Kubernetes 服務，服務連接埠必須與容器連接埠相同。如果您使用的是已命名連接埠，請也要在服務規格中使用相同的名稱。

管理員網路政策

1. 管理員層政策（先評估）：所有管理員層 ClusterNetworkPolicies 都會在任何其他政策之前進行評估。在管理員層中，政策會依優先順序處理（優先順序最低的數字優先）。動作類型會決定接下來會發生的情況。
   - 拒絕動作（最高優先順序）：當具有拒絕動作的管理員政策符合流量時，無論任何其他政策為何，都會立即封鎖該流量。不會再處理 ClusterNetworkPolicy 或 NetworkPolicy 規則。這可確保命名空間層級政策不會覆寫整個組織的安全性控制項。
   - 允許動作：評估拒絕規則之後，具有允許動作的管理員政策會依優先順序處理（優先順序最低的數字優先）。當允許動作相符時，系統會接受流量，而不會進一步評估政策。這些政策可以根據標籤選擇器授予多個命名空間的存取權，從而集中控制哪些工作負載可以存取特定資源。
   - 傳遞動作：在管理層政策中傳遞動作，將決策委派給較低的層。當流量符合通過規則時，評估會略過該流量的所有剩餘管理員層規則，並直接進入 NetworkPolicy 層。這可讓管理員將特定流量模式的控制明確委派給應用程式團隊。例如，您可以使用 Pass 規則將命名空間內流量管理委派給命名空間管理員，同時對外部存取維持嚴格的控制。
2. 網路政策層：如果沒有管理員層政策與拒絕或允許相符，或者如果通過動作相符，則會評估命名空間範圍的 NetworkPolicy 資源。這些政策可在個別命名空間內提供精細的控制，並由應用程式團隊管理。命名空間範圍政策只能比管理員政策更嚴格。他們無法覆寫管理員政策的拒絕決策，但可以進一步限制管理員政策允許或傳遞的流量。
3. 基準層管理員政策：如果沒有管理員或命名空間範圍的政策符合流量，則會評估基準層 ClusterNetworkPolicies。這些提供可由命名空間範圍政策覆寫的預設安全狀態，允許管理員設定整

個組織的預設值，同時為團隊提供視需要自訂的彈性。基準政策會依優先順序（優先順序最低的數字優先）進行評估。

4. 預設拒絕（如果沒有政策相符）：此deny-by-default行為可確保只允許明確允許的連線，並維持強大的安全狀態。

移轉

- 如果您的叢集目前正在使用第三方解決方案來管理 Kubernetes 網路政策，您可以搭配 Kubernetes 專用 Amazon VPC CNI 外掛程式來使用那些相同的政策。然而，您必須移除現有的解決方案，如此才不會管理相同的政策。

> **Warning**
>
> 我們建議在移除網路政策解決方案後，更換所有曾套用該網路政策解決方案的節點。這是因為如果解決方案的 Pod 突然退出，流量規則可能會殘留。

安裝

- 網路政策功能會建立且需要稱為 `policyendpoints.networking.k8s.aws` 的 `PolicyEndpoint` 自訂資源定義 (CRD)。自訂資源的 `PolicyEndpoint` 物件是由 Amazon EKS 管理。您不應修改或刪除這些資源。
- 如果您執行使用執行個體角色 IAM 憑證的 Pod 或連線至 EC2 IMDS，請小心檢查是否有會封鎖 EC2 IMDS 存取的網路政策。您可能需要新增網路政策以允許 EC2 IMDS 的存取權。如需詳細資訊，請參閱《Amazon EC2 使用者指南》中的執行個體中繼資料和使用者資料。

  使用服務帳戶的 IAM 角色或 EKS Pod 身分識別的 Pod 不會存取 EC2 IMDS。
- Kubernetes 專用 Amazon VPC CNI 外掛程式不會將網路政策套用至每個 Pod 的其他網路介面，而只會套用每個 Pod 的主要介面 (`eth0`)。這會影響下列架構：
  - `ENABLE_V4_EGRESS` 變數設定為 `true` 的 `IPv6` Pod。此變數可讓 `IPv4` 輸出功能將 IPv6 Pod 連線到 `IPv4` 端點，例如叢集外部的端點。`IPv4` 輸出功能的運作方式是使用本機迴路 IPv4 地址建立額外的網路介面。
  - 使用鏈結的網絡外掛程式（例如 Multus）時。由於這些外掛程式會將網路介面新增至每個 Pod，因此網路政策不會套用至鏈結的網路外掛程式。

使用 Kubernetes 網路政策來限制 Pod 網路流量

您可以使用 Kubernetes 網路政策來限制往返 Pod 之間的網路流量。如需詳細資訊，請參閱 Kubernetes 文件的網路政策。

要使用此功能，您必須設定以下內容：

1. 設定政策在 Pod 啟動時強制執行。您可以在 VPC CNI `DaemonSet` 的 `aws-node` 容器中執行此操作。
2. 啟用附加元件的網路政策參數。
3. 將叢集設定為使用 Kubernetes 網路政策。

請檢閱考量之後再開始。如需詳細資訊，請參閱the section called “考量事項”。

先決條件

以下是此功能的先決條件：

最低叢集版本

現有 Amazon EKS 叢集。若要部署叢集，請參閱 開始使用。叢集必須執行下表所列的其中一種 Kubernetes 版本和平台版本，請注意，也支援比上列任何 Kubernetes 與平台版本更新的版本。您可使用叢集名稱取代下列命令的 *my-cluster*，然後執行修改的命令來檢查目前 Kubernetes 版本：

```
aws eks describe-cluster --name my-cluster --query cluster.version --output text
```

Kubernetes 版本	平台版本
1.27.4	eks.5
1.26.7	eks.6

最低 VPC CNI 版本

若要建立標準 Kubernetes 網路政策和管理網路政策，您需要執行 VPC CNI 外掛程式1.21的版本。您可以使用下列命令來查看您目前擁有哪個版本。

```
kubectl describe daemonset aws-node --namespace kube-system | grep amazon-k8s-cni: |
 cut -d : -f 3
```

如果您的版本早於 1.21，請查看 the section called “更新 (EKS 附加元件)” 以升級到版本 1.21 或更高版本。

最低 Linux 核心版本

您的節點必須具有 Linux 核心版本 5.10 或更高版本。您可以使用 uname -r 來檢查您的核心版本。如果您使用的是最新版本的 Amazon EKS 最佳化 Amazon Linux、Amazon EKS 最佳化加速 Amazon Linux AMI 和 Bottlerocket AMI，則它們已經具有所需的核心版本。

Amazon EKS 最佳化加速 Amazon Linux AMI 版本 v20231116 或具有核心版本 5.10 的更新版本。

步驟 1：設定政策在 Pod 啟動時強制執行

Kubernetes 專用 Amazon VPC CNI 外掛程式 會在 Pod 佈建的同時設定 Pod 的網路政策。在為新 Pod 設定所有政策之前，新 Pod 中的容器會以預設允許政策啟動。這稱為標準模式。預設允許政策意味著允許進出新 Pod 的所有輸入和輸出流量。例如，在使用作用中的政策更新新 Pod 之前，Pod 不會強制執行任何防火牆規則 (允許所有流量)。

將 NETWORK_POLICY_ENFORCING_MODE 變數設定為 strict 時，使用 VPC CNI 的 Pod 會從預設拒絕政策開始，然後設定政策。這稱為嚴格模式。在嚴格模式下，您必須為 Pod 需要存取的叢集中的每個端點都設定網路政策。請注意，此要求適用於 CoreDNS Pod。預設拒絕政策不會針對具有主機聯網的 Pod 進行設定。

您可以透過在 VPC CNI DaemonSet 的 aws-node 容器中將環境變數 strict 設定為 NETWORK_POLICY_ENFORCING_MODE 來變更預設網路政策。

```
env:
  - name: NETWORK_POLICY_ENFORCING_MODE
    value: "strict"
```

步驟 2：啟用附加元件的網路政策參數

依預設，網路政策功能會使用節點的連接埠 8162 做為指標。此外，此功能會使用連接埠8163進行運作狀態探查。如果您在需要使用這些連接埠的節點或 Pod 內部執行另一應用程式，則該應用程式將無法執行。從 VPC CNI 版本 v1.14.1 或更新版本，您可以變更這些連接埠。

使用以下程序來啟用附加元件的網路政策參數。

AWS 管理主控台

1. 開啟 Amazon EKS 主控台。

2. 在左側導覽窗格中，選取叢集，然後選取您要為其設定 Amazon VPC CNI 附加元件的叢集名稱。
3. 選擇附加元件索引標籤。
4. 選取附加元件方塊右上方的方塊，然後選擇 Edit (編輯)。
5. 在設定 **Amazon VPC CNI** 頁面上：
   a. 在版本清單中，選取 v1.14.0-eksbuild.3 或更高版本。
   b. 展開選用組態設定。
   c. 在組態值中輸入 JSON 金鑰 "enableNetworkPolicy": 和值 "true"。產生的文字必須是有效的 JSON 物件。如果此金鑰和值是文字方塊中唯一的資料，請以大括號 { } 括住該金鑰和值。

      下列範例已啟用網路政策功能，以及指標與運作狀態探查已設定為預設連接埠號碼：

```
{
    "enableNetworkPolicy": "true",
    "nodeAgent": {
        "healthProbeBindAddr": "8163",
        "metricsBindAddr": "8162"
    }
}
```

Helm

如果您已透過 helm 安裝 Kubernetes 專用 Amazon VPC CNI 外掛程式，您可更新組態來變更連接埠。

1. 執行下列命令來變更連接埠。分別在金鑰 nodeAgent.metricsBindAddr 或金鑰 nodeAgent.healthProbeBindAddr 值設定連接埠號碼。

```
helm upgrade --set nodeAgent.metricsBindAddr=8162 --set
 nodeAgent.healthProbeBindAddr=8163 aws-vpc-cni --namespace kube-system eks/aws-vpc-
cni
```

kubectl

1. 在您的編輯器中開啟 aws-node DaemonSet。

```
kubectl edit daemonset -n kube-system aws-node
```

2. 在 VPC CNI aws-node daemonset 清單檔案的 aws-network-policy-agent 容器，取代 args: 中下列命令引數的連接埠號碼。

```
    - args:
            - --metrics-bind-addr=:8162
            - --health-probe-bind-addr=:8163
```

步驟 3：設定叢集以使用 Kubernetes 網路政策

您可為 Amazon EKS 附加元件或自我管理附加元件設定此值。

Amazon EKS 附加元件

使用 AWS CLI，您可以執行下列命令，將叢集設定為使用 Kubernetes 網路政策。將 my-cluster 取代為您的叢集名稱，並將 IAM 角色 ARN 取代為您正在使用的角色。

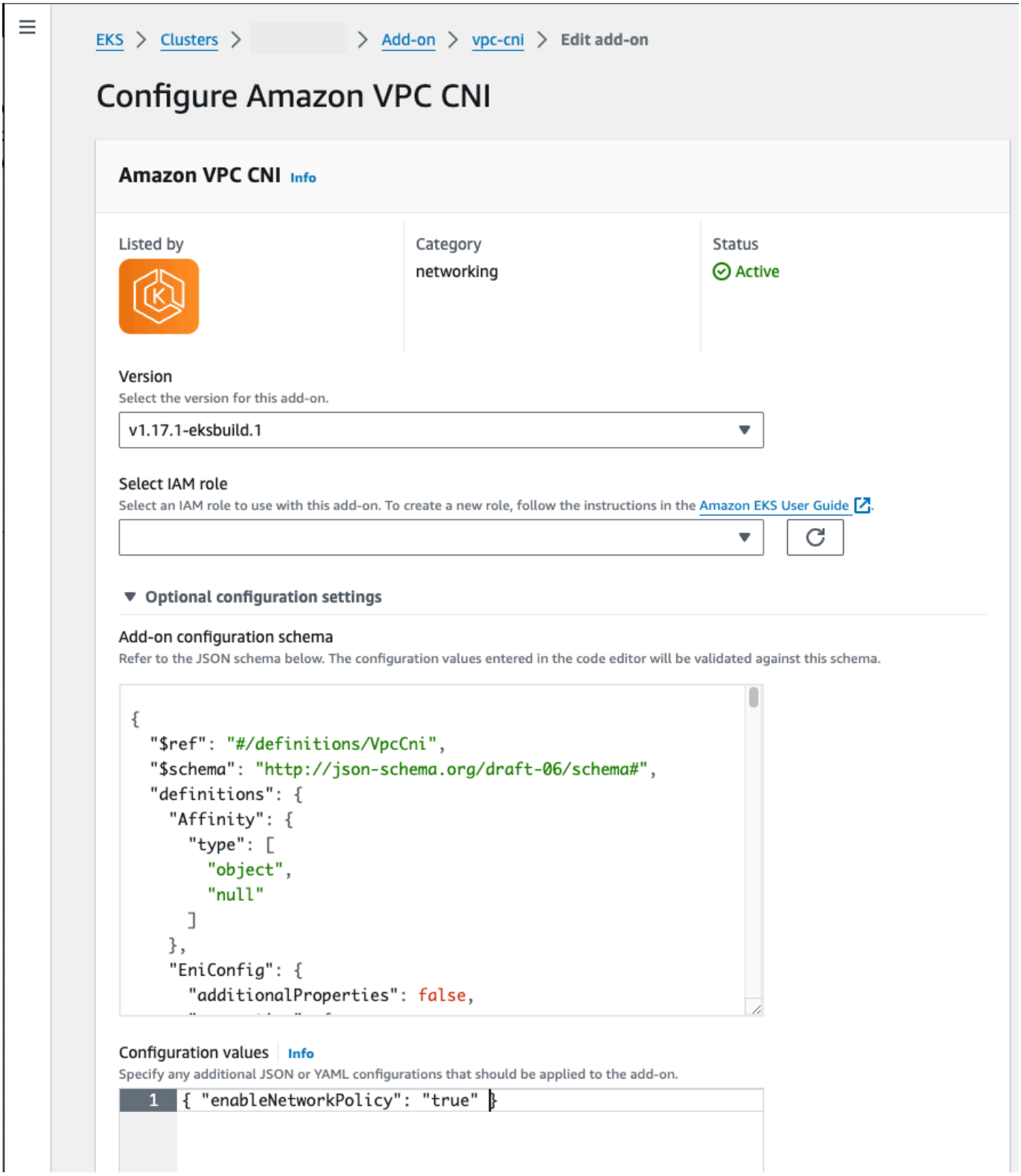
```
aws eks update-addon --cluster-name my-cluster --addon-name vpc-cni --addon-version
 v1.14.0-eksbuild.3 \
    --service-account-role-arn arn:aws: iam::123456789012:role/AmazonEKSVPCCNIRole \
    --resolve-conflicts PRESERVE --configuration-values '{"enableNetworkPolicy":
 "true"}'
```

若要使用 AWS 管理主控台設定此項目，請依照下列步驟進行：

1. 開啟 Amazon EKS 主控台。
2. 在左側導覽窗格中，選取叢集，然後選取您要為其設定 Amazon VPC CNI 附加元件的叢集名稱。
3. 選擇附加元件索引標籤。
4. 選取附加元件方塊右上方的方塊，然後選擇 Edit (編輯)。
5. 在設定 **Amazon VPC CNI** 頁面上：
   a. 在版本清單中，選取 v1.14.0-eksbuild.3 或更高版本。
   b. 展開選用組態設定。
   c. 在組態值中輸入 JSON 金鑰 "enableNetworkPolicy": 和值 "true"。產生的文字必須是有效的 JSON 物件。如果此金鑰和值是文字方塊中唯一的資料，請以大括號 { } 括住該金鑰和值。以下範例顯示已啟用網路政策：

```
{ "enableNetworkPolicy": "true" }
```

下列螢幕擷取畫面展示了案例的範例。



自我管理附加元件

Helm

如果您已透過 helm 安裝 Kubernetes 專用 Amazon VPC CNI 外掛程式，您可以更新組態以啟用網路政策。

1. 執行下列命令以啟用網路政策。

```
helm upgrade --set enableNetworkPolicy=true aws-vpc-cni --namespace kube-system eks/
aws-vpc-cni
```

kubectl

1. 在您的編輯器中開啟 amazon-vpc-cni ConfigMap。

```
kubectl edit configmap -n kube-system amazon-vpc-cni -o yaml
```

2. 在 ConfigMap 中，加入下列行至 data。

```
enable-network-policy-controller: "true"
```

   一旦您新增此行，您的 ConfigMap 看起來應該會像下列範例。

```
apiVersion: v1
 kind: ConfigMap
 metadata:
  name: amazon-vpc-cni
  namespace: kube-system
 data:
  enable-network-policy-controller: "true"
```

3. 在您的編輯器中開啟 aws-node DaemonSet。

```
kubectl edit daemonset -n kube-system aws-node
```

   a. 在 VPC CNI aws-node daemonset 清單檔案的 aws-network-policy-agent 容器，以 true 取代 args: 中命令引數 --enable-network-policy=false 裡的 false。

```
    - args:
       - --enable-network-policy=true
```

步驟 4. 後續步驟

完成組態後，請確認 aws-node Pod 正在叢集上執行。

```
kubectl get pods -n kube-system | grep 'aws-node\|amazon'
```

範例輸出如下。

```
aws-node-gmqp7                                          2/2     Running   1 (24h ago)
 24h
aws-node-prnsh                                          2/2     Running   1 (24h ago)
 24h
```

版本 1.14 和更新版本的 aws-node Pod 中有 2 個容器。在先前版本中，如果網路政策已停用，則 aws-node Pod 中只會有單一容器。

您現在可以將 Kubernetes 網路政策部署到您叢集。

若要實作 Kubernetes 網路政策，您可以建立 Kubernetes NetworkPolicy或ClusterNetworkPolicy物件並將其部署到您的叢集。 NetworkPolicy 物件的範圍是命名空間，而ClusterNetworkPolicy物件的範圍可以是整個叢集或多個命名空間。您實施政策以允許或拒絕根據標籤選取工具、命名空間及 IP 位址範圍的 Pod 之間的流量。如需有關建立 NetworkPolicy 物件的詳細資訊，請參閱 Kubernetes 文件中的網路政策。

Kubernetes NetworkPolicy 物件的強制執行是使用延伸式 Berkeley 封包篩選 (eBPF) 實作的。相對於 iptables 型實作，它提供了更低的延遲和效能特性，包含降低 CPU 使用率與避免循序查詢。此外，eBPF 探查可讓您存取內容豐富的資料，協助除錯複雜的核心層級問題並改善可觀測性。Amazon EKS 支援 eBPF 型匯出工具，此工具利用探查來記錄每個節點上的政策結果，並將資料匯出到外部日誌收集器以協助除錯。如需詳細資訊，請參閱 eBPF 文件。

停用 Amazon EKS Pod 網路流量的 Kubernetes 網路政策

停用 Kubernetes 網路政策以停止限制 Amazon EKS Pod 網路流量。

1. 列出所有 Kubernetes 網路政策。

```
kubectl get netpol -A
```

2. 刪除每個 Kubernetes 網路政策。您必須先刪除所有網路政策，才能停用網路政策。

```
kubectl delete netpol <policy-name>
```

3. 在編輯器中開啟 aws-node DaemonSet。

```
kubectl edit daemonset -n kube-system aws-node
```

4. 在 VPC CNI `aws-node` daemonset 清單檔案的 `aws-network-policy-agent` 容器，以 `false` 取代 `args:` 中命令引數 `--enable-network-policy` 裡的 `true`。

```
    - args:
       - --enable-network-policy=false
```

Amazon EKS 的 Kubernetes 網路政策疑難排解

這是 Amazon VPC CNI 網路政策功能的疑難排解指南。

本指南涵蓋：

- 安裝資訊、CRD 和 RBAC 許可 the section called “新的 policyendpoints CRD 和許可”
- 診斷網路政策問題時要檢查的日誌 the section called “網路政策日誌”
- 執行 eBPF SDK 工具集以進行故障診斷
- 已知問題與解決方案 the section called “已知問題與解決方案”

> **Note**
>
> 請注意，網路政策僅會套用至由 Kubernetes 部署建立的 Pod。如需 VPC CNI 中網路政策的更多限制，請參閱 the section called “考量事項”。

閱讀網路政策日誌並透過執行來自 eBPF SDK 的工具，藉此對使用網路政策的網路連線進行故障診斷與調查。

新的 **policyendpoints** CRD 和許可

- CRD：policyendpoints.networking.k8s.aws
- Kubernetes API：稱為 v1.networking.k8s.io 的 apiservice
- Kubernetes 資源：Kind: NetworkPolicy
- RBAC：稱為 aws-node 的 ClusterRole (VPC CNI)、稱為 eks:network-policy-controller 的 ClusterRole (EKS 叢集控制平面中的網路政策控制器)

對於網路政策，VPC CNI 會建立稱為 policyendpoints.networking.k8s.aws 的新 CustomResourceDefinition (CRD)。VPC CNI 必須擁有許可，才能建立 CRD，並為此 CRD 及由 VPC CNI (eniconfigs.crd.k8s.amazonaws.com) 安裝的其他 CRD 建立 CustomResources (CR)。這兩個 CRD 都可以在 GitHub 上的 crds.yaml 檔案中找到。具體而言，VPC CNI 必須具有 policyendpoints 的 "get"、"list" 和 "watch" 動詞許可。

Kubernetes 網路政策是稱為 v1.networking.k8s.io 的 apiservice 一部分，並且這是您的政策 YAML 檔案中的 apiversion: networking.k8s.io/v1。VPC CNI DaemonSet 必須擁有許可才能使用 Kubernetes API 的這一部分。

VPC CNI 許可位於稱為 aws-node 的 ClusterRole 中。請注意，ClusterRole 物件不會在命名空間中進行分組。下面顯示了叢集的 aws-node：

```
kubectl get clusterrole aws-node -o yaml
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    app.kubernetes.io/instance: aws-vpc-cni
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/name: aws-node
    app.kubernetes.io/version: v1.19.4
    helm.sh/chart: aws-vpc-cni-1.19.4
    k8s-app: aws-node
  name: aws-node
rules:
- apiGroups:
  - crd.k8s.amazonaws.com
  resources:
  - eniconfigs
```

```
  verbs:
  - list
  - watch
  - get
- apiGroups:
  - ""
  resources:
  - namespaces
  verbs:
  - list
  - watch
  - get
- apiGroups:
  - ""
  resources:
  - pods
  verbs:
  - list
  - watch
  - get
- apiGroups:
  - ""
  resources:
  - nodes
  verbs:
  - list
  - watch
  - get
- apiGroups:
  - ""
  - events.k8s.io
  resources:
  - events
  verbs:
  - create
  - patch
  - list
- apiGroups:
  - networking.k8s.aws
  resources:
  - policyendpoints
  verbs:
  - get
  - list
```

```
  - watch
- apiGroups:
  - networking.k8s.aws
  resources:
  - policyendpoints/status
  verbs:
  - get
- apiGroups:
  - vpcresources.k8s.aws
  resources:
  - cninodes
  verbs:
  - get
  - list
  - watch
  - patch
```

此外，新的控制器會在每個 EKS 叢集的控制平面中執行。控制器會使用稱為 eks:network-policy-controller 的 ClusterRole 的許可。下面顯示了叢集的 eks:network-policy-controller：

```
kubectl get clusterrole eks:network-policy-controller -o yaml
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    app.kubernetes.io/name: amazon-network-policy-controller-k8s
  name: eks:network-policy-controller
rules:
- apiGroups:
  - ""
  resources:
  - namespaces
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - ""
  resources:
  - pods
```

```
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - ""
  resources:
  - services
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - networking.k8s.aws
  resources:
  - policyendpoints
  verbs:
  - create
  - delete
  - get
  - list
  - patch
  - update
  - watch
- apiGroups:
  - networking.k8s.aws
  resources:
  - policyendpoints/finalizers
  verbs:
  - update
- apiGroups:
  - networking.k8s.aws
  resources:
  - policyendpoints/status
  verbs:
  - get
  - patch
  - update
- apiGroups:
  - networking.k8s.io
  resources:
  - networkpolicies
  verbs:
  - get
```

```
  - list
  - patch
  - update
  - watch
```

網路政策日誌

VPC CNI 的每個決策，是否允許或拒絕網路政策的連線都會記錄在流程日誌中。每個節點上的網路政策日誌均包含具有網路政策的每個 Pod 之流程日誌。網路政策日誌會儲存於 `/var/log/aws-routed-eni/network-policy-agent.log`。以下範例來自於 `network-policy-agent.log` 檔案：

```
{"level":"info","timestamp":"2023-05-30T16:05:32.573Z","logger":"ebpf-
client","msg":"Flow Info: ","Src
IP":"192.168.87.155","Src Port":38971,"Dest IP":"64.6.160","Dest
Port":53,"Proto":"UDP","Verdict":"ACCEPT"}
```

根據預設，網路政策日誌已停用。若要啟用網路政策日誌，請遵循下列步驟：

> **Note**
>
> 網路政策日誌需要為 VPC CNI `aws-node` DaemonSet 資訊清單中的 `aws-network-policy-agent` 容器額外配備 1 個 vCPU。

Amazon EKS 附加元件

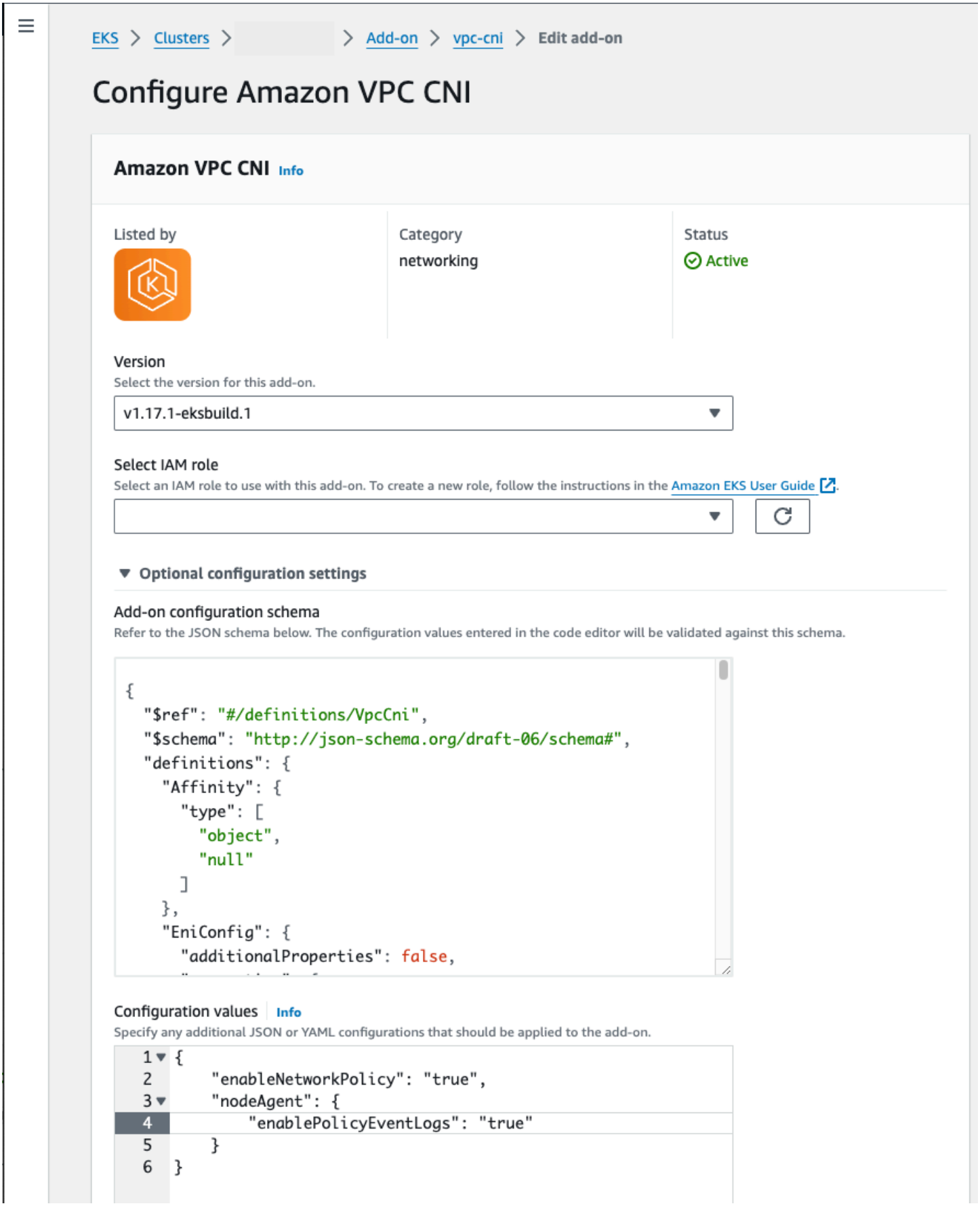
AWS 管理主控台

a. 開啟 Amazon EKS 主控台。
b. 在左側導覽窗格中，選取叢集，然後選取您要為其設定 Amazon VPC CNI 附加元件的叢集名稱。
c. 選擇附加元件索引標籤。
d. 選取附加元件方塊右上方的方塊，然後選擇 Edit (編輯)。
e. 在設定 ***Amazon VPC CNI*** 頁面上：
   i. 在版本下拉式清單中，選取 `v1.14.0-eksbuild.3` 或更高版本。
   ii. 展開選用組態設定。
   iii. 輸入最上層 JSON 金鑰 `"nodeAgent":`，且值為具有組態值中的金鑰 `"enablePolicyEventLogs":` 與 `"true"` 的值之物件。產生的文字必須是有效的 JSON 物

件。下列範例顯示網路政策和網路政策日誌已啟用，並且網路政策日誌會傳送至 CloudWatch Logs：

```
{
    "enableNetworkPolicy": "true",
    "nodeAgent": {
        "enablePolicyEventLogs": "true"
    }
}
```

下列螢幕擷取畫面展示了案例的範例。


EKS > Clusters > > Add-on > vpc-cni > Edit add-on
Configure Amazon VPC CNI
Amazon VPC CNI Info
Listed by
Category
networking
Status
Active
Version
Select the version for this add-on.
v1.17.1-eksbuild.1
Select IAM role
Select an IAM role to use with this add-on. To create a new role, follow the instructions in the Amazon EKS User Guide.
Optional configuration settings
Add-on configuration schema
Refer to the JSON schema below. The configuration values entered in the code editor will be validated against this schema.
{
"$ref": "#/definitions/VpcCni",
"$schema": "http://json-schema.org/draft-06/schema#",
"definitions": {
"Affinity": {
"type": [
"object",
"null"
]
},
"EniConfig": {
"additionalProperties": false,
Configuration values Info
Specify any additional JSON or YAML configurations that should be applied to the add-on.
1 {
2 "enableNetworkPolicy": "true",
3 "nodeAgent": {
4 "enablePolicyEventLogs": "true"
5 }
6 }

AWS CLI

a. 執行下列 AWS CLI 命令。將 my-cluster 取代為您的叢集名稱，並將 IAM 角色 ARN 取代為您正在使用的角色。

```
aws eks update-addon --cluster-name my-cluster --addon-name vpc-cni --addon-
version v1.14.0-eksbuild.3 \
    --service-account-role-arn arn:aws: iam::123456789012:role/AmazonEKSVPCCNIRole
 \
    --resolve-conflicts PRESERVE --configuration-values '{"nodeAgent":
 {"enablePolicyEventLogs": "true"}}'
```

自我管理附加元件

Helm

如果您已透過 helm 安裝 Kubernetes 專用 Amazon VPC CNI 外掛程式，您可以更新組態，以寫入網路政策日誌。

a. 執行下列命令以啟用網路政策。

```
helm upgrade --set nodeAgent.enablePolicyEventLogs=true aws-vpc-cni --namespace
 kube-system eks/aws-vpc-cni
```

kubectl

如果您已透過 kubectl 安裝 Kubernetes 專用 Amazon VPC CNI 外掛程式，您可以更新組態，以寫入網路政策日誌。

a. 在您的編輯器中開啟 aws-node DaemonSet。

```
kubectl edit daemonset -n kube-system aws-node
```

b. 在 VPC CNI aws-node DaemonSet 資訊清單的 aws-network-policy-agent 容器，使用 args: 中命令引數 --enable-policy-event-logs=false 中的 true 取代 false。

```
    - args:
       - --enable-policy-event-logs=true
```

將網路政策日誌傳送到 Amazon CloudWatch Logs

您可以使用 Amazon CloudWatch Logs 這一類服務來監控網路政策日誌。您可以使用下列方法將網路政策日誌傳送到 CloudWatch Logs。

若為 EKS 叢集，政策日誌會放置在 /aws/eks/*cluster-name*/cluster/ 下；若為自我管理的 K8S 群集，日誌會放置在 /aws/k8s-cluster/cluster/ 下。

使用 Kubernetes 專用 Amazon VPC CNI 外掛程式傳送網路政策日誌

如果您啟用網路政策，則會將第二個容器新增至節點代理程式的 aws-node Pod。此節點代理程式可以將網路政策日誌傳送到 CloudWatch Logs。

> **Note**
>
> 網路政策日誌僅會由節點代理程式傳送。其他由 VPC CNI 製作的日誌不包含在內。

先決條件

- 將下列許可作為一節或個別政策新增至您用於 VPC CNI 的 IAM 角色。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups",
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": "*"
        }
    ]
}
```

Amazon EKS 附加元件

AWS 管理主控台

a. 開啟 Amazon EKS 主控台。

b. 在左側導覽窗格中，選取叢集，然後選取您要為其設定 Amazon VPC CNI 附加元件的叢集名稱。

c. 選擇附加元件索引標籤。

d. 選取附加元件方塊右上方的方塊，然後選擇 Edit (編輯)。

e. 在設定 ***Amazon VPC CNI*** 頁面上：

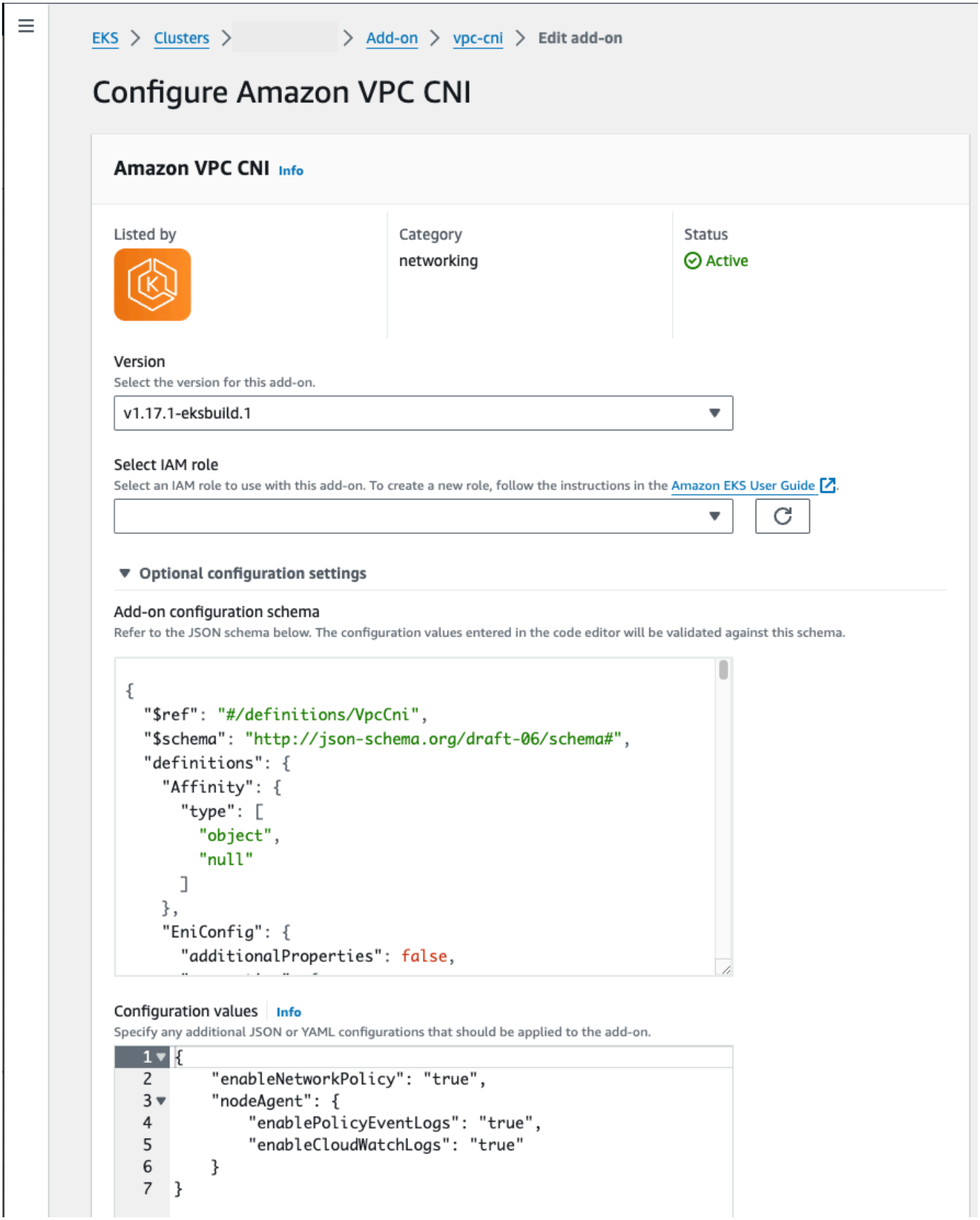
i. 在版本下拉式清單中，選取 `v1.14.0-eksbuild.3` 或更高版本。

ii. 展開選用組態設定。

iii. 輸入最上層 JSON 金鑰 `"nodeAgent":`，且值為具有組態值中的金鑰 `"enableCloudWatchLogs":` 與 `"true"` 的值之物件。產生的文字必須是有效的 JSON 物件。下列範例顯示網路政策和網路政策日誌已啟用，並且日誌會傳送至 CloudWatch Logs：

```
{
    "enableNetworkPolicy": "true",
    "nodeAgent": {
        "enablePolicyEventLogs": "true",
        "enableCloudWatchLogs": "true",
    }
}
```

下列螢幕擷取畫面展示了案例的範例。


EKS > Clusters > > Add-on > vpc-cni > Edit add-on
Configure Amazon VPC CNI
Amazon VPC CNI Info
Listed by
Category
networking
Status
Active
Version
Select the version for this add-on.
v1.17.1-eksbuild.1
Select IAM role
Select an IAM role to use with this add-on. To create a new role, follow the instructions in the Amazon EKS User Guide.
Optional configuration settings
Add-on configuration schema
Refer to the JSON schema below. The configuration values entered in the code editor will be validated against this schema.
{
"$ref": "#/definitions/VpcCni",
"$schema": "http://json-schema.org/draft-06/schema#",
"definitions": {
"Affinity": {
"type": [
"object",
"null"
]
},
"EniConfig": {
"additionalProperties": false,
Configuration values Info
Specify any additional JSON or YAML configurations that should be applied to the add-on.
1 {
2 "enableNetworkPolicy": "true",
3 "nodeAgent": {
4 "enablePolicyEventLogs": "true",
5 "enableCloudWatchLogs": "true"
6 }
7 }

AWS CLI

a. 執行下列 AWS CLI 命令。將 my-cluster 取代為您的叢集名稱，並將 IAM 角色 ARN 取代為您正在使用的角色。

```
aws eks update-addon --cluster-name my-cluster --addon-name vpc-cni --addon-
version v1.14.0-eksbuild.3 \
    --service-account-role-arn arn:aws: iam::123456789012:role/AmazonEKSVPCCNIRole
 \
    --resolve-conflicts PRESERVE --configuration-values '{"nodeAgent":
 {"enablePolicyEventLogs": "true", "enableCloudWatchLogs": "true"}}'
```

自我管理附加元件

Helm

如果您已透過 helm 安裝 Kubernetes 專用 Amazon VPC CNI 外掛程式，您可以更新組態，以傳送網路政策日誌至 CloudWatch 日誌。

a. 執行下列命令以啟用網路政策日誌，並將其傳送至 CloudWatch Logs。

```
helm upgrade --set nodeAgent.enablePolicyEventLogs=true --set
 nodeAgent.enableCloudWatchLogs=true aws-vpc-cni --namespace kube-system eks/aws-
vpc-cni
```

kubectl

a. 在您的編輯器中開啟 aws-node DaemonSet。

```
kubectl edit daemonset -n kube-system aws-node
```

b. 在 VPC CNI aws-node DaemonSet 資訊清單的 aws-network-policy-agent 容器，使用 args: 中的兩個命令引數 --enable-policy-event-logs=false 和 --enable-cloudwatch-logs=false 中的 true 取代 false。

```
    - args:
       - --enable-policy-event-logs=true
       - --enable-cloudwatch-logs=true
```

透過 Fluent Bit **DaemonSet** 傳送網路政策日誌

如果您正在 DaemonSet 使用 Fluent Bit 從節點傳送日誌，則可新增組態以便包括來自網路政策的網路政策日誌。您可以使用下列範例組態：

```
    [INPUT]
        Name              tail
        Tag               eksnp.*
        Path              /var/log/aws-routed-eni/network-policy-agent*.log
        Parser            json
        DB                /var/log/aws-routed-eni/flb_npagent.db
        Mem_Buf_Limit     5MB
        Skip_Long_Lines   On
        Refresh_Interval  10
```

已包含 eBPF SDK

Kubernetes 專用 Amazon VPC CNI 外掛程式會在節點上安裝 eBPF SDK 工具集。您可以使用 eBPF SDK 來找出網路政策的問題。例如，下列命令會列出目前在節點上執行的程式。

```
sudo /opt/cni/bin/aws-eks-na-cli ebpf progs
```

若要執行此命令，您可以使用任何方法連接到節點。

已知問題與解決方案

下列各節說明 Amazon VPC CNI 網路政策功能及其解決方案的已知問題。

儘管 enable-policy-event-logs 設定為 false，但仍會產生網路政策日誌

問題：即使 `enable-policy-event-logs` 設定為 `false`，EKS VPC CNI 也會產生網路政策日誌。

解決方案：`enable-policy-event-logs` 設定僅會停用政策「決策」日誌，但不會停用所有網路政策代理程式記錄。GitHub 上的 aws-network-policy-agent README 記載了這種行為。若要完全停用記錄，您可能需要調整其他記錄組態。

網路政策映射清除問題

問題：刪除 Pod 後，網路 `policyendpoint` 仍然存在並且無法清除的問題。

解決方案：此問題是因 VPC CNI 附加元件版本 1.19.3-eksbuild.1 的問題所造成。更新至較新版本的 VPC CNI 附加元件來解決此問題。

未套用網路政策

問題：Amazon VPC CNI 外掛程式中已啟用網路政策功能，但網路政策未正確套用。

如果您建立網路政策`kind: NetworkPolicy`且不會影響 Pod，請檢查政策端點物件是否在與 Pod 相同的命名空間中建立。如果命名空間中沒有 `policyendpoint` 物件，則網路政策控制器 (EKS 叢集的一部分) 無法為要套用的網路政策代理程式 (VPC CNI 的一部分) 建立網路政策規則。

解決方案：解決方案是修正 VPC CNI (`ClusterRole：aws-node`) 和網路政策控制站 (`ClusterRole：eks:network-policy-controller`) 的許可，並在任何政策強制執行工具 (例如 Kyverno) 中允許這些動作。確保 Kyverno 政策不會封鎖 `policyendpoint` 物件的建立。如需 中的必要許可，請參閱上一節the section called “新的 policyendpoints CRD 和許可”。

在嚴格模式下刪除政策後，Pod 不會返回預設拒絕狀態

問題：在嚴格模式下啟用網路政策時，Pod 會從預設拒絕政策開始。套用政策後，允許流量到達指定的端點。不過，刪除政策時，Pod 不會返回預設拒絕狀態，而是進入預設允許狀態。

解決方案：此問題已在 VPC CNI 1.19.3 版中進行修正，其中包括網路政策代理程式 1.2.0 版。修正後，在啟用嚴格模式的情況下，只要移除政策，Pod 即會如預期返回到預設拒絕狀態。

Pod 啟動延遲的安全群組

問題：在 EKS 中使用 Pod 的安全群組功能時，Pod 啟動延遲會增加。

解決方案：延遲是因資源控制器中來自 `CreateNetworkInterface` API 上的 API 限流的速率限制所致，其中 VPC 資源控制器會用於為 Pod 建立分支 ENI。檢查您帳戶的此操作的 API 限制，並考慮視需要請求提高限制。

由於 vpc.amazonaws.com/pod-eni 不足而導致 FailedScheduling

問題：Pod 無法排程並出現錯誤：`FailedScheduling 2m53s (x28 over 137m) default-scheduler 0/5 nodes are available: 5 Insufficient vpc.amazonaws.com/pod-eni. preemption: 0/5 nodes are available: 5 No preemption victims found for incoming pod.`

解決方案：與上一個問題一樣，將安全群組指派給 Pod 會增加 Pod 排程延遲，並且新增每個 ENI 的時間可能會超出 CNI 閾值，進而導致啟動 Pod 的失敗。這是使用 Pod 的安全群組時的預期行為。設計工作負載架構時，請考慮排程影響。

IPAM 連線問題和分段錯誤

問題：出現多個錯誤，包括 IPAM 連線問題、限流請求和分段錯誤：

- Checking for IPAM connectivity …
- Throttling request took 1.047064274s
- Retrying waiting for IPAM-D
- panic: runtime error: invalid memory address or nil pointer dereference

解決方案：如果您在 AL2023 上安裝 systemd-udev，就會出現此問題，因為該檔案會根據中斷政策重新寫入。當更新至具有更新的套件的不同 releasever 或手動更新套件本身時，可能會發生這種情況。避免在 AL2023 節點上安裝或更新 systemd-udev。

「無法依名稱尋找裝置」錯誤

問題：錯誤訊息：{"level":"error","ts":"2025-02-05T20:27:18.669Z","caller":"ebpf/bpf_client.go:578","msg":"failed to find device by name eni9ea69618bf0: %!w(netlink.LinkNotFoundError={0xc000115310})"}

解決方案：此問題已在最新版本的 Amazon VPC CNI 網路政策代理程式 (v1.2.0) 中進行識別和修正。更新至最新版本的 VPC CNI 來解決此問題。

Multus CNI 映像中的 CVE 漏洞

問題：增強型 EKS ImageScan CVE 報告可識別 Multus CNI 映像版本 v4.1.4-eksbuild.2_thick 中的漏洞。

解決方案：更新至新版本的 Multus CNI 映像和新的網路政策控制器映像，它們沒有任何漏洞。可以更新掃描器，以解決先前版本中找到的漏洞。

日誌中的流程資訊 DENY 判定結果

問題：網路政策日誌顯示 DENY 判定結果：{"level":"info","ts":"2024-11-25T13:34:24.808Z","logger":"ebpf-client","caller":"events/events.go:193","msg":"Flow Info: ","Src IP":"","Src Port":9096,"Dest IP":"","Dest Port":56830,"Proto":"TCP","Verdict":"DENY"}

解決方案：此問題已在新版本的網路政策控制器中解決。更新至最新的 EKS 平台版本，以解決記錄問題。

從 Calico 移轉後的 Pod 至 pod 通訊問題

問題：將 EKS 叢集升級至版本 1.30 並針對網路政策從 Calico 切換到 Amazon VPC CNI 後，套用網路政策時 Pod 至 Pod 通訊即會失敗。刪除網路政策時，即會還原通訊。

解決方案：VPC CNI 中的網路政策代理程式指定的連接埠數量無法與 Calico 相比。請在網路政策中改用連接埠範圍。網路政策中每個通訊協定`ingress:`或`egress:`選擇器中每個通訊協定的唯一連接埠組合數目上限為 24。使用連接埠範圍來減少唯一的連接埠的數量，並避免此限制。

網路政策代理程式不支援獨立 Pod

問題：套用至獨立 Pod 的網路政策可能會有不一致的行為。

解決方案：網路政策代理程式目前僅支援部署為 deployment/replicaset 一部分的 Pod。如果將網路政策套用至獨立 Pod，則行為中可能會出現一些不一致。此頁面頂端、the section called “考量事項”中和 GitHub 上的 aws-network-policy-agent GitHub 問題 #327 中會記載這一情況。將 Pod 部署為 deployment 或 replicaset 的一部分，以實現一致的網路政策行為。

Amazon EKS 的網路政策的星示範

此示範會在 Amazon EKS 叢集上建立前端、後端和用戶端服務。示範中亦將建立管理圖形使用者介面，其顯示各服務之間可用的輸入和輸出路徑。我們建議您在不執行生產工作負載的叢集上完成示範。

在您建立任何網路政策前，所有服務都可以雙向通訊。在您套用網路政策後，便可看到用戶端只能與前端服務通訊，同時後端只能接受來自前端的流量。

1. 套用前端、後端、用戶端和管理使用者介面服務：

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/eks-
workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-
policies/calico/stars_policy_demo/create_resources.files/namespace.yaml
kubectl apply -f https://raw.githubusercontent.com/aws-samples/eks-
workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-
policies/calico/stars_policy_demo/create_resources.files/management-ui.yaml
kubectl apply -f https://raw.githubusercontent.com/aws-samples/eks-
workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-
policies/calico/stars_policy_demo/create_resources.files/backend.yaml
kubectl apply -f https://raw.githubusercontent.com/aws-samples/eks-
workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-
policies/calico/stars_policy_demo/create_resources.files/frontend.yaml
kubectl apply -f https://raw.githubusercontent.com/aws-samples/eks-
workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-
policies/calico/stars_policy_demo/create_resources.files/client.yaml
```

2. 檢視叢集上的所有 Pod。

```
kubectl get pods -A
```

範例輸出如下。

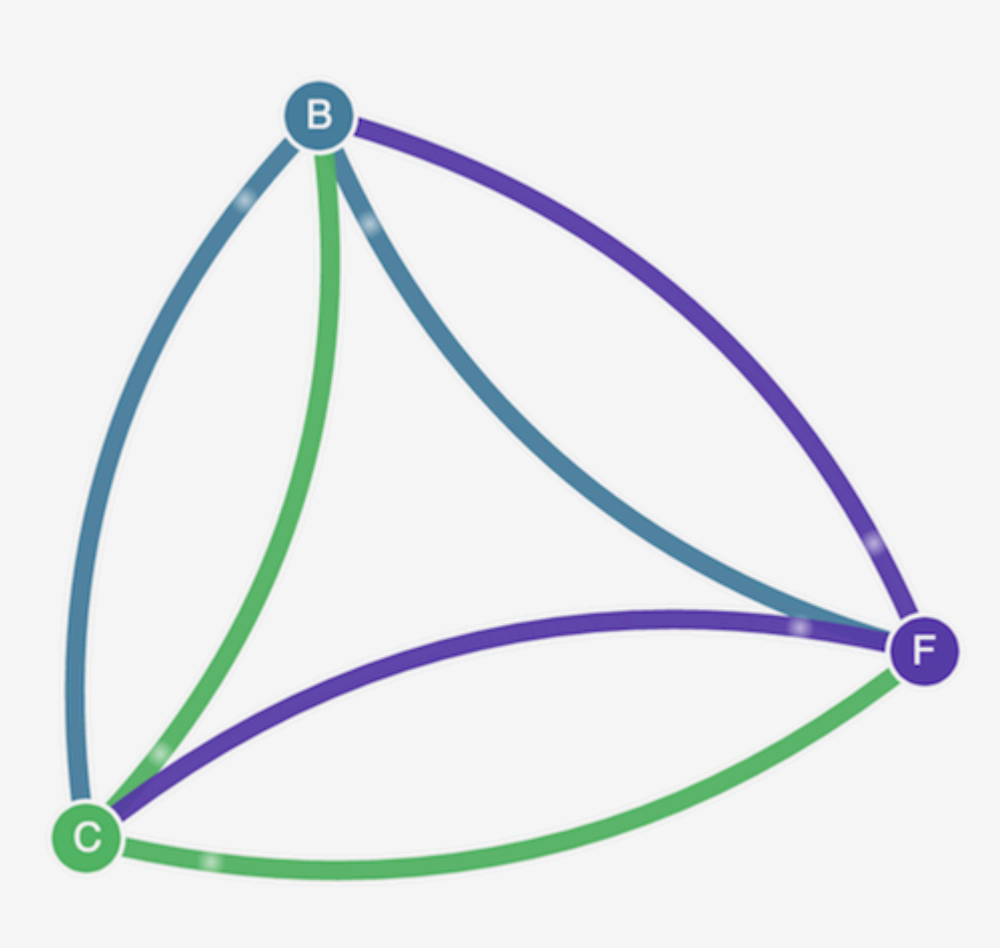
在輸出中，您應該在以下輸出中顯示的命名空間中看到 pod。*NAMES* 和 READY 直欄中的 Pod 數量與下列輸出中的數量不同。不要繼續，直到您看到具有相似名稱的 Pod 並且其在 STATUS 資料欄都有 Running 狀態。

```
NAMESPACE         NAME                                  READY   STATUS
 RESTARTS   AGE
[...]
client            client-xlffc                          1/1     Running   0
     5m19s
[...]
management-ui     management-ui-qrb2g                   1/1     Running   0
     5m24s
stars             backend-sz87q                         1/1     Running   0
     5m23s
stars             frontend-cscnf                        1/1     Running   0
     5m21s
[...]
```

3. 若要連接到管理使用者介面，請連接到在叢集上執行之服務的 EXTERNAL-IP：

```
kubectl get service/management-ui -n management-ui
```

4. 開啟瀏覽器至上一個步驟中的位置。您應該會看到管理使用者介面。C 節點是用戶端服務，而 F 節點是前端服務，且 B 節點是後端服務。每個節點都有對所有其他節點的完整通訊存取權 (如粗體、上顏色行的文字所指示)。



5. 在 stars 和 client 命名空間中套用以下網路政策來將服務彼此隔離：

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: default-deny
spec:
  podSelector:
    matchLabels: {}
```

您可以使用下列命令將政策同時套用至兩個命名空間：

```
kubectl apply -n stars -f https://raw.githubusercontent.com/aws-samples/eks-
workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-
policies/calico/stars_policy_demo/apply_network_policies.files/default-deny.yaml
kubectl apply -n client -f https://raw.githubusercontent.com/aws-samples/eks-
workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-
policies/calico/stars_policy_demo/apply_network_policies.files/default-deny.yaml
```

6. 重新整理您的瀏覽器。您將看到管理使用者介面不再到達任何節點，所以各節點均不會顯示在使用者介面中。
7. 套用下列不同的網路政策，以允許管理使用者介面存取服務。套用此政策以允許 UI：

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  namespace: stars
  name: allow-ui
spec:
  podSelector:
    matchLabels: {}
  ingress:
    - from:
        - namespaceSelector:
            matchLabels:
              role: management-ui
```

套用此政策以允許用戶端：

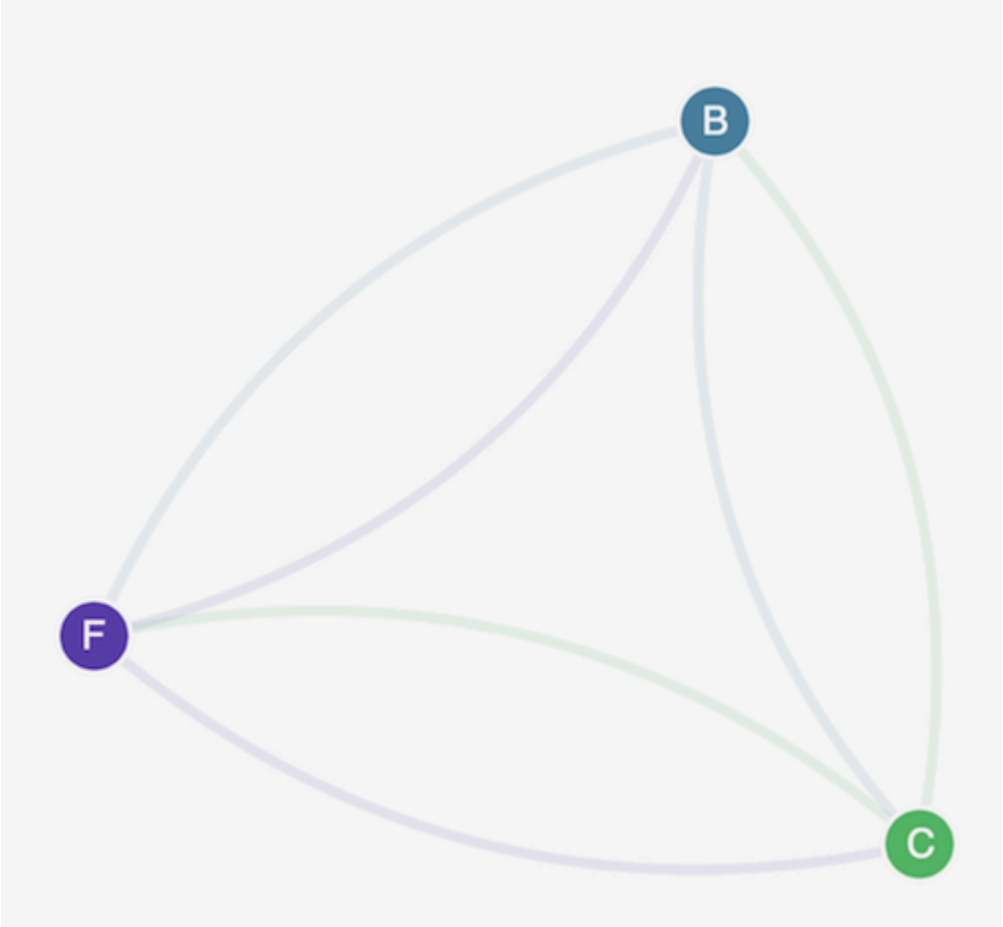
```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  namespace: client
  name: allow-ui
spec:
  podSelector:
    matchLabels: {}
  ingress:
    - from:
        - namespaceSelector:
            matchLabels:
              role: management-ui
```

您可以使用下列命令來同時套用兩個政策：

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/eks-
workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-
policies/calico/stars_policy_demo/apply_network_policies.files/allow-ui.yaml
```

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/eks-
workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-
policies/calico/stars_policy_demo/apply_network_policies.files/allow-ui-client.yaml
```

8. 重新整理您的瀏覽器。您將看到管理使用者介面再次可到達各節點，但各節點無法互相通訊。

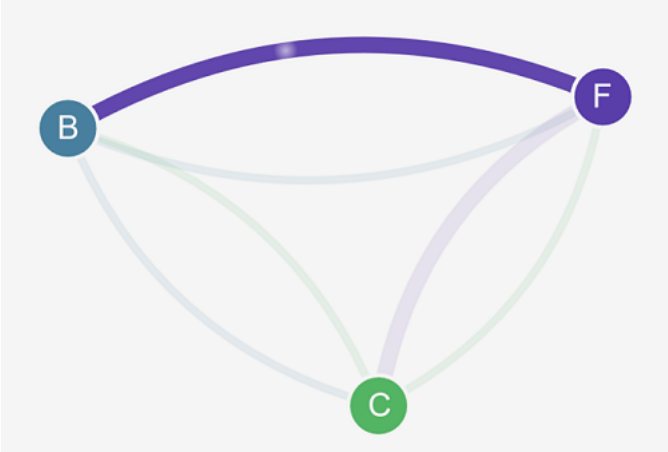


9. 套用以下網路政策以允許流量從前端服務流向後端服務：

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  namespace: stars
  name: backend-policy
spec:
  podSelector:
    matchLabels:
      role: backend
  ingress:
    - from:
        - podSelector:
```

```
            matchLabels:
              role: frontend
      ports:
        - protocol: TCP
          port: 6379
```

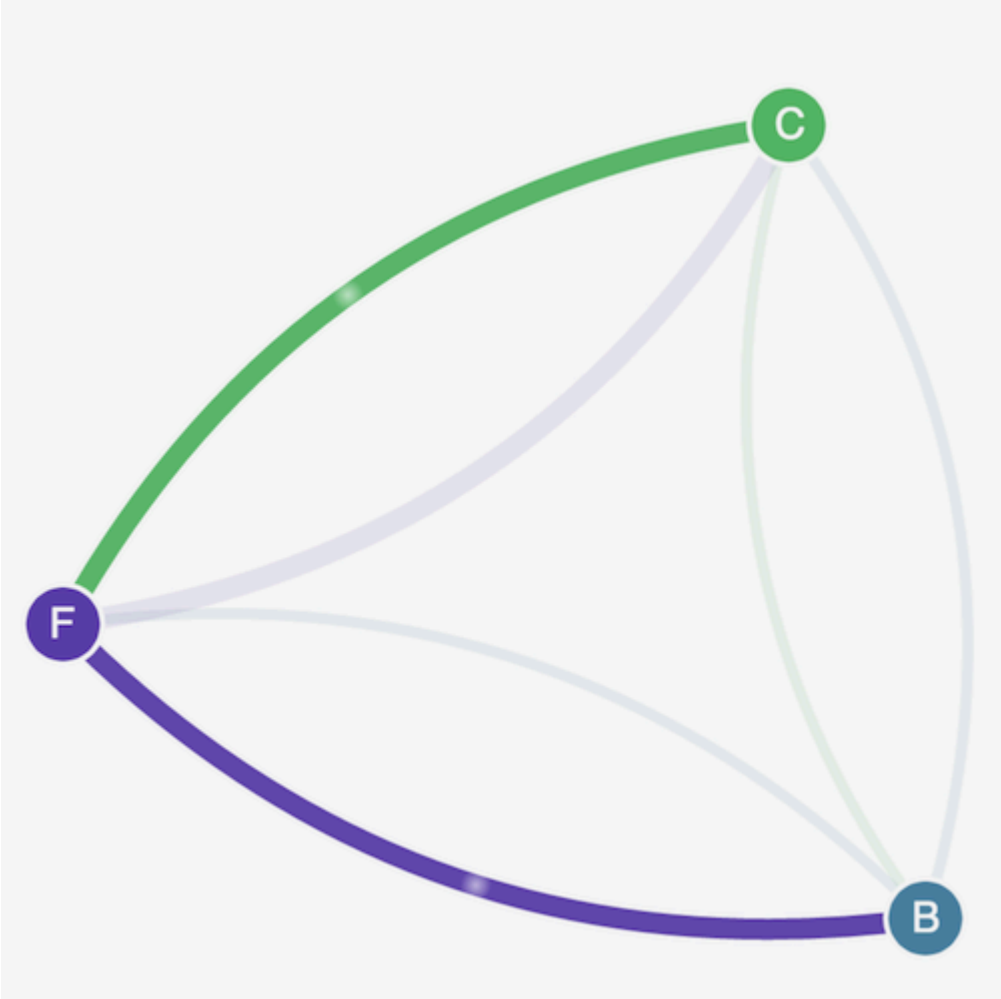
10重新整理您的瀏覽器。您可以看到前端服務可以與後端服務通訊。



11套用以下網路政策以允許流量從用戶端流向後端服務：

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  namespace: stars
  name: frontend-policy
spec:
  podSelector:
    matchLabels:
      role: frontend
  ingress:
    - from:
        - namespaceSelector:
            matchLabels:
              role: client
      ports:
        - protocol: TCP
          port: 80
```

12重新整理您的瀏覽器。您可以看到用戶端可以與前端服務通訊。前端服務仍然可以與後端服務通訊。



13(選用) 在完成示範後，您可以刪除其資源。

```
kubectl delete -f https://raw.githubusercontent.com/aws-samples/eks-
workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-
policies/calico/stars_policy_demo/create_resources.files/client.yaml
kubectl delete -f https://raw.githubusercontent.com/aws-samples/eks-
workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-
policies/calico/stars_policy_demo/create_resources.files/frontend.yaml
kubectl delete -f https://raw.githubusercontent.com/aws-samples/eks-
workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-
policies/calico/stars_policy_demo/create_resources.files/backend.yaml
kubectl delete -f https://raw.githubusercontent.com/aws-samples/eks-
workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-
policies/calico/stars_policy_demo/create_resources.files/management-ui.yaml
kubectl delete -f https://raw.githubusercontent.com/aws-samples/eks-
workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-
policies/calico/stars_policy_demo/create_resources.files/namespace.yaml
```

即使刪除資源之後，節點上仍可能有網路政策端點，這些端點可能會以非預期的方式干擾叢集中的聯網。移除這些規則的唯一確定方法，是重新啟動節點或終止所有節點並將其回收。若要終止所有節點，請將 Auto Scaling 群組所需的計數設定為 0，然後備份到所需的數字，或是只終止節點。

設定 Pod IP 地址的子網路選擇

適用於：包含 Amazon EC2 執行個體的 Linux 節點

適用於 Kubernetes 的 Amazon VPC CNI 外掛程式會在節點上建立次要彈性網路介面 (ENIs)，並將這些 ENIs IP 地址指派給 Pod。根據預設，VPC CNI 會在與節點主要網路界面相同的子網路中建立次要 ENIs。您可以透過下列方法控制 VPC CNI 用於 Pod IP 地址的子網路：

- 增強型子網路探索 – VPC CNI 會自動探索並使用在相同 VPC 和可用`kubernetes.io/role/cni`區域中以 標記的子網路。需要 VPC CNI 1.18.0 版或更新版本。對於大多數使用案例，我們建議使用此方法。
- 自訂聯網 – 使用`ENIConfig`自訂資源，為每個可用區域手動指定子網路和安全群組。如需詳細資訊，請參閱the section called “自訂聯網”。

> **Note**
>
> 啟用這兩個功能時，以自訂聯網為優先。

增強型子網路探索

VPC CNI 1.18.0 版及更新版本預設會啟用增強型子網路探索 (`ENABLE_SUBNET_DISCOVERY=true`)。VPC CNI 會自動探索與節點位於相同 VPC 和可用區域中的子網路，然後使用它們來建立次要 ENIs並配置 Pod IP 地址。這可展開可用的 IP 地址空間，無需手動`ENIConfig`設定。

若要確認已啟用此功能：

```
kubectl describe ds aws-node -n kube-system | grep ENABLE_SUBNET_DISCOVERY
```

若要停用此功能，請在 `aws-node` DaemonSet `ENABLE_SUBNET_DISCOVERY=false`上設定 。

子網路標籤行為 (**kubernetes.io/role/cni**)

kubernetes.io/role/cni 標籤控制 VPC CNI 如何處理 ENI 操作和 Pod IP 配置的子網路。標籤具有不同的效果，取決於 VPC CNI 是建立新的 ENI 還是協調現有的 ENI。

> **Important**
>
> 叢集範圍和排除性標記機制只能從 VPC CNI v1.22.2 版本開始提供。在此版本之前，只有標準選擇加入行為可透過標籤 使用kubernetes.io/role/cni=1。

標籤值

下表摘要說明每個標籤值如何影響子網路行為：

標籤值	建立新的 ENI	現有的 ENI 對帳
1	選擇加入：VPC CNI 會在此子網路中建立新的 ENIs。	ENI 仍可用於 Pod IP 配置。
0	排除：VPC CNI 不會在此子網路中建立新的 ENIs。	排除：VPC CNI 會從 Pod IP 配置中排除此子網路中的現有 ENIs。此 ENI 未指派新的 Pod IPs。
不存在（無標籤）	不用於新的 ENIs：VPC CNI 不會選取沒有新 ENI 建立標籤的次要子網路。主要子網路（節點啟動的子網路）仍會用於新的 ENI 建立，即使沒有 標籤也適用於回溯相容性。	不中斷：未標記子網路中的現有 ENIs 仍可用於 Pod IP 配置。VPC CNI 不會移除或排除這些 ENIs。

建立與對帳行為

VPC CNI 會在建立新 ENIs 和協調現有 ENIs 時刻意套用不同的政策：

- 建立（次要子網路關閉失敗）：當 VPC CNI 需要建立新的次要 ENI 時，它只會使用以 明確標記的次要子網路kubernetes.io/role/cni=1。從未選取未標記的次要子網路來建立新的 ENI。這可確保新的網路界面僅放置在管理員明確核准的子網路中。

- 調校（未標記的子網路無法開啟）：VPC CNI 啟動或調校已連接到節點的現有 ENIs 時，不會排除 ENIs因為其子網路缺少kubernetes.io/role/cni標籤。這可防止中斷已使用這些 ENIs IP 地址的執行中 Pod。

VPC CNI 刻意使用此設計。從未標記的子網路強制排除已連接的 ENI 會中斷目前使用該 ENI IP 地址的 Pod。

> **Important**
>
> 為了防止子網路提供新的 Pod IPs，包括來自現有 ENIs 的 IP，請使用 標記子網路kubernetes.io/role/cni=0。缺少標籤只會防止在該子網路中建立新的 ENI。它不會從配置中排除現有的 ENIs。

主要子網路處理

VPC CNI 一律包含節點的主要子網路（節點啟動的子網路）以進行 ENI 建立，即使沒有kubernetes.io/role/cni標籤也一樣。這可維持與現有叢集的回溯相容性。主要子網路行為：

- 包含用於建立 ENI，無論標籤是否存在（除非已標記 0)。
- 如果使用 標記kubernetes.io/role/cni=0，VPC CNI 會從新的 ENI 建立和現有的 ENI 配置中排除主要子網路。

叢集範圍子網路篩選

當子網路使用 標記時kubernetes.io/role/cni=1，VPC CNI 還會使用金鑰格式 檢查叢集特定的標籤cni.networking.k8s.aws/cluster/<cluster-name>。如果子網路具有此格式的任何叢集標籤，則只有名稱相符的叢集會使用該子網路。具有 kubernetes.io/role/cni=1且沒有叢集特定標籤的子網路可供 VPC 中的所有叢集使用。

例如，若要將子網路限制在特定叢集：

```
aws ec2 create-tags --resources subnet-example \
  --tags Key=kubernetes.io/role/cni,Value=1 Key=cni.networking.k8s.aws/cluster/my-
cluster,Value=shared
```

當多個 EKS 叢集共用 VPC 且您希望每個叢集針對 Pod IP 地址使用不同的子網路時，這會很有用。

建議的工作流程

若要為 Pod IP 地址新增子網路：

1. 在與節點相同的 VPC 和可用區域中建立新的子網路。
2. 使用 標記子網路`kubernetes.io/role/cni=1`。
3. 確保子網路具有適當的路由表和網路 ACLs。
4. 確認 VPC CNI 發現並開始使用新的子網路。

若要從 Pod IP 配置中移除子網路：

1. 使用 標記子網路`kubernetes.io/role/cni=0`。
2. 等待使用該子網路 IPs Pod 自然終止或重新排程。
3. 確認 VPC CNI 停止從該子網路中的 ENIs 配置新的 Pod IPs。

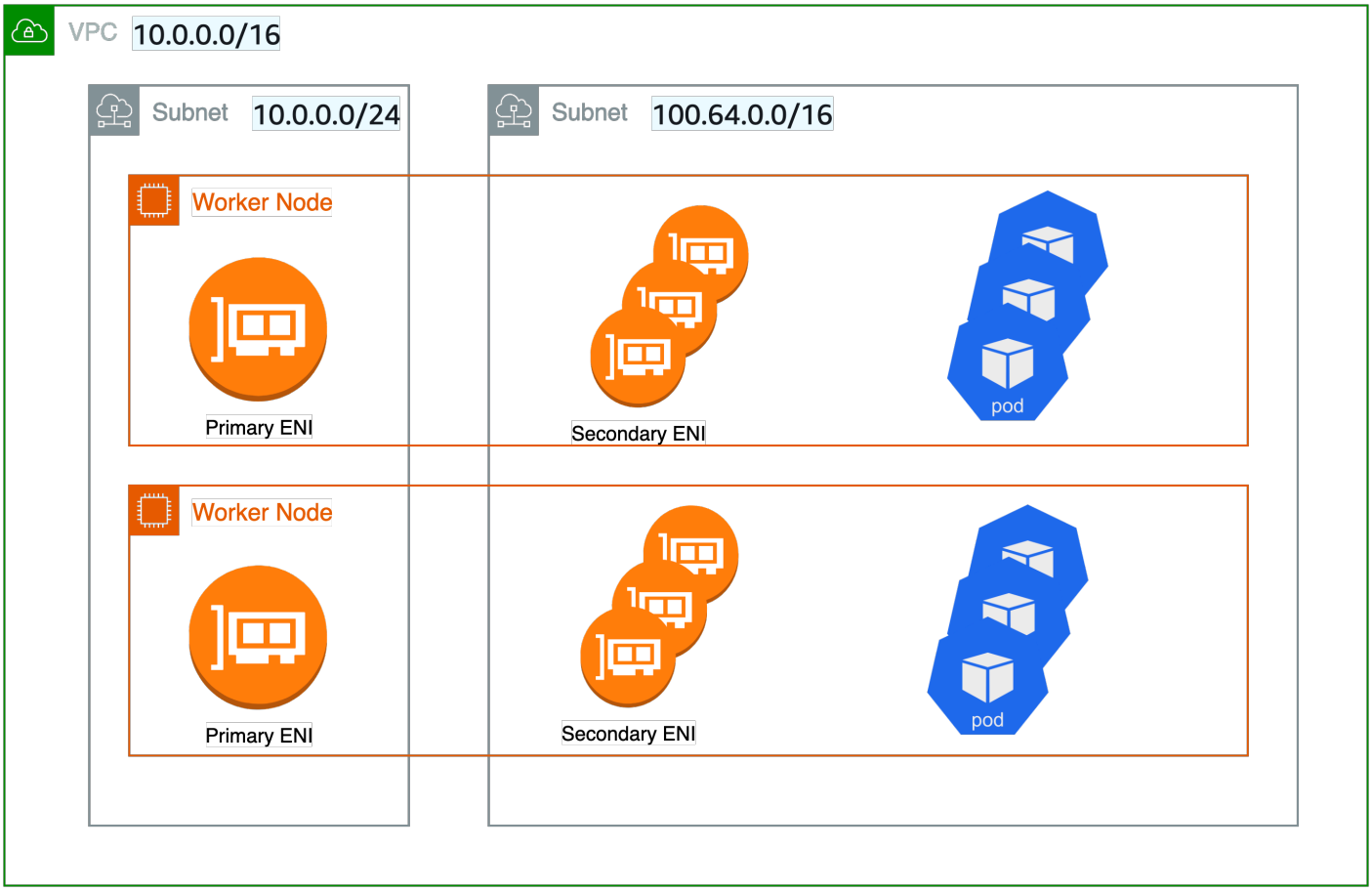
> **Important**
>
> 請勿移除`kubernetes.io/role/cni`標籤以停止使用子網路。移除標籤可防止建立新的 ENI，但不會將現有的 ENIs 從配置中排除。若要主動排除子網路，請使用 標記子網路`kubernetes.io/role/cni=0`。

考量事項

- 增強型子網路探索需要 Amazon VPC CNI 1.18.0 版或更新版本。
- 此功能需要 VPC CNI IAM 角色中的`ec2:DescribeSubnets`許可。`AmazonEKS_CNI_Policy` 受管政策包含此許可。IPv6 自我管理 IAM 政策不包含在內。如果您使用自我管理的 IAM 政策（例如 IPv6 叢集），請`ec2:DescribeSubnets`手動新增 以啟用子網路探索。
- 此功能適用於次要 IP 地址模式和字首委派模式。
- 所有探索到的子網路都必須與節點位於相同的 VPC 中。
- VPC CNI 只會在與節點位於相同可用區域的子網路中建立 ENIs。
- 無論標籤變更為何，VPC CNI 都不會取消配置仍有指派給 Pod 之 IP 地址的 ENIs。
- 使用共用 VPCs（跨帳戶子網路）時，請在啟動叢集的參與者帳戶中標記子網路。
- 您可以搭配 Pod、網路政策、字首委派和 SNAT 的安全群組使用增強型子網路探索。

使用自訂聯網在替代子網路中部署 Pod

適用於：包含 Amazon EC2 執行個體的 Linux IPv4 Fargate 節點、Linux 節點



預設情況下，當 Kubernetes 專用 Amazon VPC CNI 外掛程式為您的 Amazon EC2 節點建立次要彈性網路介面 (網路介面) 時，會在與節點的主網路介面相同的子網路中建立這些介面。它還會將與主網路介面關聯的相同安全群組關聯至次要網路介面。由於以下一或多個原因，您可能希望外掛程式在不同子網中建立次要網路介面，或者希望將不同的安全群組與次要網路介面關聯 (或者兩者兼具)：

- 主網路介面所在的子網路中可用的 IPv4 位址數量有限。這可能會限制您可以在子網路中建立的 Pod 數量。如果為次要網路介面使用不同的子網路，您可以增加可用於 Pod 的 IPv4 位址的數量。
- 基於安全考量，您的 Pod 可能需使用節點之主要網路介面以外的其他子網路或安全群組。
- 將節點設定在公有子網路，並將 Pod 放置在私有子網路。與公有子網相關聯的路由表包括網際網路閘道的路由。與私有子網路相關聯的路由表不會包含網際網路閘道的路由。

> **Tip**
>
> 您也可以直接將新的或現有的子網路新增到 Amazon EKS 叢集，而無需使用自訂聯網。如需詳細資訊，請參閱 the section called “從管理主控台將現有 VPC 子網路新增至 Amazon EKS 叢集”。

考量事項

以下是使用該功能的考量事項。

- 啟用自訂聯網後，不會將指派給主網路介面的 IP 位址指派給 Pod。僅有來自次要網路介面的 IP 位址會指派給 Pod。
- 如果您的叢集使用 IPv6 系列，您無法使用自訂聯網。
- 如果您計劃僅使用自訂聯網來協助緩解 IPv4 地址耗盡的情況，則可以改為使用 IPv6 系列來建立叢集。如需詳細資訊，請參閱 the section called “IPv6”。
- 即使部署到為次要網路介面指定的子網路的 Pod 可以使用與節點的主網路介面不同的子網路和安全群組，但子網路和安全群組仍須與節點位於同一 VPC 中。
- 對於 Fargate，子網路是透過 Fargate 設定檔控制的。如需詳細資訊，請參閱 the section called “定義設定檔”。

在 Amazon EKS 節點中自訂次要網路介面

開始本教學之前，先完成下列動作：

- 檢閱考量事項
- 熟悉 Kubernetes 專用 Amazon VPC CNI 外掛程式如何建立次要網路介面並將 IP 位址指派給 Pod。如需詳細資訊，請參閱 GitHub 上的 ENI Allocation (ENI 分配)。
- 裝置或 AWS CloudShell 上安裝和設定的 AWS 命令列界面 (AWS CLI) 版本 1.27.160 2.12.3或更新版本。若要檢查您目前的版本，請使用 aws --version | cut -d / -f2 | cut -d ' ' -f1。適用於 macOS 的 yum、 apt-get或 Homebrew 等套件管理員通常是最新版本 CLI AWS 後面的幾個版本。若要安裝最新版本，請參閱《 AWS 命令列界面使用者指南》中的使用 aws 設定安裝 和快速組態。 https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-quickstart.html#cli-configure-quickstart-config安裝在 AWS CloudShell 中的 AWS CLI 版本也可能是最新版本後面的幾個版本。若要更新它，請參閱《CloudShell AWS 使用者指南》中的將 CLI 安裝到您的主目錄。 AWS CloudShell

- kubectl 命令列工具安裝在您的裝置或 AWS CloudShell 上。若要安裝或升級 kubectl，請參閱 the section called “設定 kubectl 和 eksctl”。
- 建議您在 Bash shell 中完成本主題中的步驟。如果您不使用 Bash shell，則某些指令碼命令 (如行接續字元以及設定和使用變數的方式) 需要針對 shell 進行調整。此外，您的 Shell 的引用及轉義規則可能會有所不同。如需詳細資訊，請參閱《 AWS 命令列界面使用者指南》中的使用引號搭配 AWS CLI 中的字串。

在本教學課程中，我們建議您使用範例值，除非其註明要取代。完成生產叢集的步驟時，您可以取代任何範例值。我們建議您在同一終端中完成所有步驟。這是因為變數是在整個步驟中設定和使用，並且不會存在於不同的終端中。

本主題中的命令使用 AWS CLI 範例中列出的慣例進行格式化。如果您從命令列執行的命令，與您正在使用的 AWS CLI AWS 描述檔中定義的預設 AWS 區域不同，則需要將 --region us-west-2新增至命令，us-west-2以您的 AWS 區域取代 。

如果要將自訂聯網部署到生產叢集，請跳至 the section called “步驟 2：設定 VPC”。

步驟 1：建立測試 VPC 和叢集

下列程序可幫助您建立測試 VPC 和叢集，並為該叢集設定自訂聯網。我們不建議對生產工作負載使用測試叢集，因為本主題未涵蓋您可能在生產叢集上使用的幾個不相關功能。如需詳細資訊，請參閱the section called “建立 叢集”。

1. 執行下列命令來定義 account_id 變數。

```
account_id=$(aws sts get-caller-identity --query Account --output text)
```

2. 建立 VPC。

   a. 如果您要部署到測試系統，請使用 Amazon EKS AWS CloudFormation 範本建立 VPC。

```
aws cloudformation create-stack --stack-name my-eks-custom-networking-vpc \
  --template-url https://s3.us-west-2.amazonaws.com/amazon-eks/
cloudformation/2020-10-29/amazon-eks-vpc-private-subnets.yaml \
  --parameters ParameterKey=VpcBlock,ParameterValue=192.168.0.0/24 \
  ParameterKey=PrivateSubnet01Block,ParameterValue=192.168.0.64/27 \
  ParameterKey=PrivateSubnet02Block,ParameterValue=192.168.0.96/27 \
  ParameterKey=PublicSubnet01Block,ParameterValue=192.168.0.0/27 \
  ParameterKey=PublicSubnet02Block,ParameterValue=192.168.0.32/27
```

b. AWS CloudFormation 堆疊需要幾分鐘的時間才能建立。若要檢查堆疊的部署狀態，請執行下列命令。

```
aws cloudformation describe-stacks --stack-name my-eks-custom-networking-vpc --
query Stacks\[\].StackStatus  --output text
```

在命令輸出為 CREATE_COMPLETE 之前，請勿繼續下一步。

c. 使用以範本建立的私有子網 ID 的值來定義變數。

```
subnet_id_1=$(aws cloudformation describe-stack-resources --stack-name my-eks-
custom-networking-vpc \
    --query "StackResources[?
LogicalResourceId=='PrivateSubnet01'].PhysicalResourceId" --output text)
subnet_id_2=$(aws cloudformation describe-stack-resources --stack-name my-eks-
custom-networking-vpc \
    --query "StackResources[?
LogicalResourceId=='PrivateSubnet02'].PhysicalResourceId" --output text)
```

d. 使用上一步中擷取的子網的可用區域來定義變數。

```
az_1=$(aws ec2 describe-subnets --subnet-ids $subnet_id_1 --query
 'Subnets[*].AvailabilityZone' --output text)
az_2=$(aws ec2 describe-subnets --subnet-ids $subnet_id_2 --query
 'Subnets[*].AvailabilityZone' --output text)
```

3. 建立叢集 IAM 角色。

a. 執行下列命令以建立 IAM 信任政策 JSON 檔案。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

b. 建立 Amazon EKS 叢集 IAM 角色。如有必要，請在 eks-cluster-role-trust-policy.json 前面加上您在上一步中寫入檔案的電腦的路徑。命令會將您在上一步驟中建立的信任策略與角色相關聯。若要建立 IAM 角色，必須為建立角色的 IAM 主體指派以下 iam:CreateRole 動作 (許可)。

```
aws iam create-role --role-name myCustomNetworkingAmazonEKSClusterRole --assume-
role-policy-document file://"eks-cluster-role-trust-policy.json"
```

c. 將名為 AmazonEKSClusterPolicy 的 Amazon EKS 受管 IAM 政策連接到角色。若要將 IAM 政策連接至 IAM 主體，必須為連接政策的 IAM 實體指派以下 IAM 動作之一 (許可)：iam:AttachUserPolicy 或 iam:AttachRolePolicy。

```
aws iam attach-role-policy --policy-arn arn:aws: iam::aws:policy/
AmazonEKSClusterPolicy --role-name myCustomNetworkingAmazonEKSClusterRole
```

4. 建立 Amazon EKS 叢集，並設定您的裝置以便與其進行通訊。

a. 建立叢集。

```
aws eks create-cluster --name my-custom-networking-cluster \
   --role-arn arn:aws: iam::$account_id:role/
myCustomNetworkingAmazonEKSClusterRole \
   --resources-vpc-config subnetIds="$subnet_id_1","$subnet_id_2"
```

> **Note**
>
> 您可能會收到錯誤，表示在請求中的其中一個可用區域沒有足夠的容量可建立 Amazon EKS 叢集。如果發生這種情況，錯誤輸出包含的可用區域可支援新的叢集。使用至少兩個位於帳戶的支援可用區域子網路來建立您的叢集。如需詳細資訊，請參閱the section called “容量不足”。

b. 建立叢集需要幾分鐘才能完成。若要檢查叢集的部署狀態，請執行下列命令。

```
aws eks describe-cluster --name my-custom-networking-cluster --query
 cluster.status
```

在命令輸出為 "ACTIVE" 之前，請勿繼續下一步。

c. 設定 kubectl 以便與叢集通訊。

```
aws eks update-kubeconfig --name my-custom-networking-cluster
```

步驟 2：設定 VPC

本教學課程需要在 the section called “步驟 1：建立測試 VPC 和叢集” 中建立的 VPC。對於生產叢集，透過將所有範例值取代為您自己的值，為您的 VPC 相應地調整步驟。

1. 確認目前安裝的 Kubernetes 專用 Amazon VPC CNI 外掛程式是最新版本。若要確定 Amazon EKS 附加元件類型的最新版本並更新您的版本，請參閱 the section called “更新附加元件”。若要確定自我管理附加元件類型的最新版本並更新您的版本，請參閱 the section called “Amazon VPC CNI”。
2. 擷取叢集 VPC 的 ID，並將其存放在變數中以供稍後步驟使用。

```
vpc_id=$(aws eks describe-cluster --name my-custom-networking-cluster --query
 "cluster.resourcesVpcConfig.vpcId" --output text)
```

3. 將額外的無類別域間路由 (CIDR) 區塊與叢集的 VPC 相關聯。CIDR 區塊不能與任何現有關聯的 CIDR 區塊重疊。

   a. 檢視目前與您的 VPC 關聯的 CIDR 區塊。

```
aws ec2 describe-vpcs --vpc-ids $vpc_id \
    --query 'Vpcs[*].CidrBlockAssociationSet[*].{CIDRBlock: CidrBlock, State:
 CidrBlockState.State}' --out table
```

   範例輸出如下。

```
---------------------------------
|          DescribeVpcs         |
+-----------------+-------------+
|    CIDRBlock    |    State    |
+-----------------+-------------+
|  192.168.0.0/24 |  associated |
+-----------------+-------------+
```

   b. 將其他 CIDR 區塊與 VPC 建立關聯。在以下命令中取代 CIDR 區塊值。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的將 IPv4 CIDR 區塊與 VPC 建立關聯。

```
aws ec2 associate-vpc-cidr-block --vpc-id $vpc_id --cidr-block 192.168.1.0/24
```

c. 確認新區塊已關聯。

```
aws ec2 describe-vpcs --vpc-ids $vpc_id --query
 'Vpcs[*].CidrBlockAssociationSet[*].{CIDRBlock: CidrBlock, State:
 CidrBlockState.State}' --out table
```

範例輸出如下。

```
----------------------------------
|          DescribeVpcs          |
+-----------------+--------------+
|    CIDRBlock    |    State     |
+-----------------+--------------+
|  192.168.0.0/24 |  associated  |
|  192.168.1.0/24 |  associated  |
+-----------------+--------------+
```

在新 CIDR 區塊 State 為 associated 之前，請勿進行下一個步驟。

4. 在現有子網所在的每個可用區域中建立任意數量的子網。指定在上一步中與 VPC 關聯的 CIDR 區塊內的 CIDR 區塊。

a. 建立新子網。在以下命令中取代 CIDR 區塊值。子網必須在不同於現有子網所在的 VPC CIDR 區塊中建立，但與現有子網位於相同的可用區域中。在此範例中，會在目前私有子網所在的每個可用區域的新 CIDR 區塊中，建立一個子網。建立的子網 ID 會儲存在變數中，以便在稍後步驟中使用。Name 值與指派給在上一步中使用 Amazon EKS VPC 範本建立的子網的值相符。名稱並非必要資訊。您可以使用不同的名稱。

```
new_subnet_id_1=$(aws ec2 create-subnet --vpc-id $vpc_id --availability-zone $az_1
 --cidr-block 192.168.1.0/27 \
    --tag-specifications 'ResourceType=subnet,Tags=[{Key=Name,Value=my-eks-custom-
networking-vpc-PrivateSubnet01},{Key=kubernetes.io/role/internal-elb,Value=1}]' \
    --query Subnet.SubnetId --output text)
new_subnet_id_2=$(aws ec2 create-subnet --vpc-id $vpc_id --availability-zone $az_2
 --cidr-block 192.168.1.32/27 \
    --tag-specifications 'ResourceType=subnet,Tags=[{Key=Name,Value=my-eks-custom-
networking-vpc-PrivateSubnet02},{Key=kubernetes.io/role/internal-elb,Value=1}]' \
    --query Subnet.SubnetId --output text)
```

> **Important**
>
> 預設情況下，您的新子網路與 VPC 的主路由表具有隱式關聯。此路由表允許在 VPC 中部署的所有資源之間進行通訊。但是，不允許與其 IP 位址在與 VPC 關聯的 CIDR 區塊之外的資源進行通訊。您可以將自己的路由表與子網相關聯以變更此行為。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的子網由表。

b. 檢視 VPC 中的目前子網。

```
aws ec2 describe-subnets --filters "Name=vpc-id,Values=$vpc_id" \
    --query 'Subnets[*].{SubnetId: SubnetId,AvailabilityZone:
 AvailabilityZone,CidrBlock: CidrBlock}' \
    --output table
```

範例輸出如下。

```
-------------------------------------------------------------------
|                          DescribeSubnets                        |
+------------------+--------------------+---------------------------+
| AvailabilityZone |     CidrBlock      |         SubnetId          |
+------------------+--------------------+---------------------------+
|  us-west-2d      |  192.168.0.0/27    |     subnet-example1       |
|  us-west-2a      |  192.168.0.32/27   |     subnet-example2       |
|  us-west-2a      |  192.168.0.64/27   |     subnet-example3       |
|  us-west-2d      |  192.168.0.96/27   |     subnet-example4       |
|  us-west-2a      |  192.168.1.0/27    |     subnet-example5       |
|  us-west-2d      |  192.168.1.32/27   |     subnet-example6       |
+------------------+--------------------+---------------------------+
```

您可以看到您建立的 `192.168.1.0` CIDR 區塊中的子網，與 `192.168.0.0` CIDR 區塊中的子網位於相同的可用區域中。

步驟 3：設定 Kubernetes 資源

1. 在 `aws-node` DaemonSet 中，將 `AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG` 環境變數設定為 `true`。

```
kubectl set env daemonset aws-node -n kube-system
 AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG=true
```

2. 擷取叢集安全群組的 ID，並將其存放在變數中以供下一步驟使用。在您建立叢集時，Amazon EKS 會自動建立此安全群組。

```
cluster_security_group_id=$(aws eks describe-cluster --name my-custom-networking-
cluster --query cluster.resourcesVpcConfig.clusterSecurityGroupId --output text)
```

3. 為每個您要為其部署 Pod 的子網路建立 ENIConfig 自訂資源。

   a. 為每個網路介面組態建立唯一的檔案。

   以下命令為在上一步中建立的兩個子網建立單獨的 ENIConfig 檔案。name 的值必須是唯一的值。此名稱與子網所在的可用區域相同。叢集安全群組指派給 ENIConfig。

```
cat >$az_1.yaml <<EOF
apiVersion: crd.k8s.amazonaws.com/v1alpha1
kind: ENIConfig
metadata:
  name: $az_1
spec:
  securityGroups:
    - $cluster_security_group_id
  subnet: $new_subnet_id_1
EOF
```

```
cat >$az_2.yaml <<EOF
apiVersion: crd.k8s.amazonaws.com/v1alpha1
kind: ENIConfig
metadata:
  name: $az_2
spec:
  securityGroups:
    - $cluster_security_group_id
  subnet: $new_subnet_id_2
EOF
```

   對於生產叢集，您可以對前述命令進行以下變更：

   - 將 $cluster_security_group_id 取代為您想要用於每個 ENIConfig 的現有安全群組的 ID。

- 我們建議盡可能將您的 ENIConfigs 命名為與您將使用 ENIConfig 的可用區域相同的名稱。由於各種原因，您可能需要為您的 ENIConfigs 使用與可用區域名稱不同的名稱。例如，如果您在同一可用區域中有兩個以上的子網，並且希望將其與自訂聯網一起使用，則需要多個 ENIConfigs 用於相同的可用區域。由於每個 ENIConfig 都需要一個唯一的名稱，因此您不能使用可用區域名稱命名多個 ENIConfigs。

  如果您的 ENIConfig 名稱與可用區域名稱並不完全相同，則將 $az_1 和 $az_2 取代為在前面命令中您自己的名稱，並使用在本教學課程後的 ENIConfig 註釋您的節點。

  > **Note**
  >
  > 如果您未指定用於生產叢集的有效安全群組，並且您正在使用：

- 1.8.0 版或更高版本的 Kubernetes 專用 Amazon VPC CNI 外掛程式，則會使用與節點的主要彈性網路介面關聯的安全群組。
- Kubernetes 專用 Amazon VPC CNI 外掛程式的版本低於 1.8.0，則 VPC 的預設安全群組將指派給次要網路介面。

  > **Important**
  >
  > - AWS_VPC_K8S_CNI_EXTERNALSNAT=false 是適用於 Kubernetes 的 Amazon VPC CNI 外掛程式組態中的預設設定。如果您使用預設設定，則送往不在與 VPC 關聯的其中一個 CIDR 區塊內之 IP 位址的流量，將使用節點主網路介面的安全群組和子網路。在您的 ENIConfigs 中定義、用於建立次要網路介面的子網路和安全群組不用於此流量。如需有關此設定的詳細資訊，請參閱 the section called “傳出流量”。
> - 如果您還為 Pod 使用安全群組，則會使用 SecurityGroupPolicy 中指定的安全群組，而不是 ENIConfigs 中指定的安全群組。如需詳細資訊，請參閱the section called “Pod 的安全群組”。

b. 使用下列命令，將您建立的每個自訂資源檔案套用到叢集。

```
kubectl apply -f $az_1.yaml
kubectl apply -f $az_2.yaml
```

4. 確認您的 ENIConfigs 已建立。

```
kubectl get ENIConfigs
```

範例輸出如下。

```
NAME         AGE
us-west-2a   117s
us-west-2d   105s
```

5. 如果要在生產叢集上啟用自訂聯網，並將您的 ENIConfigs 命名為所用可用區域以外的名稱，則請跳到下一步以部署 Amazon EC2 節點。

   啟用 Kubernetes 以將可用區域的 ENIConfig 自動套用至叢集中建立的任何新 Amazon EC2 節點。

   a. 對於本教學課程中的測試叢集，請跳至下一步。

      對於生產叢集，請檢查 aws-node DaemonSet 的容器規格中是否存在含有 ENI_CONFIG_ANNOTATION_DEF 環境變數的金鑰 k8s.amazonaws.com/eniConfig 的注釋。

      ```
      kubectl describe daemonset aws-node -n kube-system | grep
       ENI_CONFIG_ANNOTATION_DEF
      ```

      如果傳回輸出，則表示註釋已存在。如果未傳回輸出，表示變數尚未設定。對於生產叢集，您可以使用此設定或以下步驟中的設定。如果使用此設定，將會覆寫以下步驟中的設定。在本教學課程中，將使用下一步中的設定。

   b. 更新您的 aws-node DaemonSet 以將可用區域的 ENIConfig 自動應用至叢集中建立的任何新 Amazon EC2 節點。

      ```
      kubectl set env daemonset aws-node -n kube-system
       ENI_CONFIG_LABEL_DEF=topology.kubernetes.io/zone
      ```

步驟 4：部署 Amazon EC2 節點

1. 建立節點 IAM 角色。

   a. 執行下列命令以建立 IAM 信任政策 JSON 檔案。

      ```
      {
        "Version":"2012-10-17",
        "Statement": [
      ```

```
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

b. 建立 IAM 角色，並將其傳回的 Amazon Resource Name (ARN) 儲存在變數中，以便在稍後步驟中使用。

```
node_role_arn=$(aws iam create-role --role-name myCustomNetworkingNodeRole --
assume-role-policy-document file://"node-role-trust-relationship.json" \
    --query Role.Arn --output text)
```

c. 將三個所需的 IAM 受管政策連接到 IAM 角色。

```
aws iam attach-role-policy \
  --policy-arn arn:aws: iam::aws:policy/AmazonEKSWorkerNodePolicy \
  --role-name myCustomNetworkingNodeRole
aws iam attach-role-policy \
  --policy-arn arn:aws: iam::aws:policy/AmazonEC2ContainerRegistryReadOnly \
  --role-name myCustomNetworkingNodeRole
aws iam attach-role-policy \
    --policy-arn arn:aws: iam::aws:policy/AmazonEKS_CNI_Policy \
    --role-name myCustomNetworkingNodeRole
```

> **Important**
>
> 在本教學課程中，為簡單起見，將 AmazonEKS_CNI_Policy 政策連接至節點 IAM 角色。但是，在生產叢集中，我們建議將政策連接至僅與 Kubernetes 專用 Amazon VPC CNI 外掛程式一起使用的單獨 IAM 角色。如需詳細資訊，請參閱the section called “為 IRSA 設定”。

2. 建立下列其中一種節點群組類型。若要確定您想要部署的執行個體類型，請參閱 the section called “Amazon EC2 執行個體類型”。在本教學課程中，請完成受管、沒有啟動範本或未指定 AMI ID 的啟動範本選項。如果要將節點群組用於生產工作負載，我們建議您熟悉所有受管節點群組和自我管理節點群組選項，然後再部署節點群組。

- 受管：使用下列其中一個選項，部署節點群組：
  - 沒有啟動範本，或有啟動範本，但沒有指定 AMI ID：執行下列命令。對於本教學課程，請使用範例值。對於生產節點群組，請將所有 example-values 取代為您自己的值。節點群組名稱不可超過 63 個字元。它必須以字母或數字開頭，但也可以包含其餘字元的連字號和底線。

```
aws eks create-nodegroup --cluster-name my-custom-networking-cluster --nodegroup-
name my-nodegroup \
    --subnets $subnet_id_1 $subnet_id_2 --instance-types t3.medium --node-role
 $node_role_arn
```

  - 使用具有指定 AMI ID 的啟動範本

    A. 根據您的執行個體類型，決定節點的 Pod 數量上限。如需詳細資訊，請參閱the section called “maxPods的判斷方式”。請注意用於下一個步驟的值。

    B. 在啟動範本中，指定 Amazon EKS 最佳化 AMI ID 或基於 Amazon EKS 最佳化 AMI 的自訂 AMI，然後使用啟動範本部署節點群組，並在啟動範本中提供下列使用者資料。此使用者資料會將引數傳遞至NodeConfig規格。如需 NodeConfig 的詳細資訊，請參閱 NodeConfig API 參考。您可將 20 取代為上一個步驟的值 (建議) 或您自己的值。

```
---
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="BOUNDARY"
--BOUNDARY
Content-Type: application/node.eks.aws

---
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name: my-cluster
    ...
    kubelet:
      config:
        maxPods: 20
```

    如果您建立的自訂 AMI 並非基於 Amazon EKS 最佳化 AMI，則需要自行自訂建立組態。

- 自我管理

  i. 根據您的執行個體類型，決定節點的 Pod 數量上限。如需詳細資訊，請參閱the section called “maxPods的判斷方式”。請注意用於下一個步驟的值。

ii. 使用 the section called “Amazon Linux” 中的指示部署節點群組。

> **Note**
>
> 如果您希望生產叢集中的節點支援大量 Pod，您可以啟用字首委派。例如：m5.large 執行個體類型會傳回 110。如需有關如何啟用此功能的說明，請參閱 the section called “增加 IP 位址”。您可將此功能與自訂聯網搭配使用。

3. 建立節點群組需要幾分鐘的時間。您可以使用以下命令檢查受管節點群組的建立狀態。

```
aws eks describe-nodegroup --cluster-name my-custom-networking-cluster --nodegroup-
name my-nodegroup --query nodegroup.status --output text
```

在傳回的輸出為 ACTIVE 之前，請勿繼續進行下一個步驟。

4. 針對本教學課程，您可以略過此步驟。

對於生產叢集，如果您未將 ENIConfigs 命名為與您使用它們的可用區域相同的名稱，則必須使用應與節點一起使用的 ENIConfig 名稱來註釋您的節點。如果您在每個可用區域中只有一個子網路，且將 ENIConfigs 命名為與可用區域相同的名稱，則無需執行此步驟。這是因為當您在上一步中啟用時，Kubernetes 專用 Amazon VPC CNI 外掛程式會自動為您將正確的 ENIConfig 與節點關聯起來。

a. 取得您叢集中的節點清單。

```
kubectl get nodes
```

範例輸出如下。

```
NAME                                          STATUS   ROLES    AGE     VERSION
ip-192-168-0-126.us-west-2.compute.internal   Ready    <none>   8m49s   v1.22.9-
eks-810597c
ip-192-168-0-92.us-west-2.compute.internal    Ready    <none>   8m34s   v1.22.9-
eks-810597c
```

b. 確定每個節點所在的可用區域。針對先前步驟傳回的每個節點執行下列命令，並根據先前的輸出取代 IP 位址。

```
aws ec2 describe-instances --filters Name=network-interface.private-dns-
name,Values=ip-192-168-0-126.us-west-2.compute.internal \
```

```
--query 'Reservations[].Instances[].{AvailabilityZone: Placement.AvailabilityZone,
 SubnetId: SubnetId}'
```

範例輸出如下。

```
[
    {
        "AvailabilityZone": "us-west-2d",
        "SubnetId": "subnet-Example5"
    }
]
```

c. 使用您為子網 ID 和可用區域建立的 ENIConfig 註釋每個節點。您只能使用一個 ENIConfig 註釋一個節點，但可以使用相同的 ENIConfig 註釋多個節點。使用自己的取代範例值。

```
kubectl annotate node ip-192-168-0-126.us-west-2.compute.internal
 k8s.amazonaws.com/eniConfig=EniConfigName1
kubectl annotate node ip-192-168-0-92.us-west-2.compute.internal
 k8s.amazonaws.com/eniConfig=EniConfigName2
```

5. 如果在切換到使用自訂聯網功能之前，生產叢集中的節點正在執行 Pod，請先完成以下任務：

   a. 確保您具有使用自訂聯網功能的可用節點。

   b. 封鎖並排空節點以正常關閉 Pod。如需詳細資訊，請參閱 Kubernetes 文件中的安全地耗盡節點。

   c. 終止節點。如果節點位於現有的受管節點群組中，則可以刪除該節點群組。執行下列命令。

```
aws eks delete-nodegroup --cluster-name my-custom-networking-cluster --nodegroup-
name my-nodegroup
```

   只有使用 k8s.amazonaws.com/eniConfig 標籤註冊的新節點能使用自訂聯網功能。

6. 確認 Pod 從 CIDR 區塊 (與您在上一步中建立的子網路之一相關聯) 指派 IP 位址。

```
kubectl get pods -A -o wide
```

範例輸出如下。

```
NAMESPACE     NAME                          READY   STATUS    RESTARTS   AGE     IP
           NODE                                          NOMINATED NODE   READINESS
 GATES
```

```
kube-system   aws-node-2rkn4             1/1     Running   0          7m19s
 192.168.0.92    ip-192-168-0-92.us-west-2.compute.internal    <none>
 <none>
kube-system   aws-node-k96wp             1/1     Running   0          7m15s
 192.168.0.126   ip-192-168-0-126.us-west-2.compute.internal   <none>
 <none>
kube-system   coredns-657694c6f4-smcgr   1/1     Running   0          56m
 192.168.1.23    ip-192-168-0-92.us-west-2.compute.internal    <none>
 <none>
kube-system   coredns-657694c6f4-stwv9   1/1     Running   0          56m
 192.168.1.28    ip-192-168-0-92.us-west-2.compute.internal    <none>
 <none>
kube-system   kube-proxy-jgshq           1/1     Running   0          7m19s
 192.168.0.92    ip-192-168-0-92.us-west-2.compute.internal    <none>
 <none>
kube-system   kube-proxy-wx9vk           1/1     Running   0          7m15s
 192.168.0.126   ip-192-168-0-126.us-west-2.compute.internal   <none>
 <none>
```

您可以看到 CoreDNS Pod 被指派了 IP 位址來自您新增至 VPC 的 `192.168.1.0` CIDR 區塊的 IP 位址。如果沒有自訂網路，他們將被指派來自 `192.168.0.0` CIDR 區塊的地址，因為其是最初與 VPC 關聯的唯一 CIDR 區塊。

如果 Pod `spec` 包含 `hostNetwork=true`，則會為其指派節點的主 IP 位址。它不會從您新增的子網路中指派位址。依預設，此值是設為 `false`。此值針對在您叢集上執行的 `kube-proxy` 和 Kubernetes 專用 Amazon VPC CNI 外掛程式 (`aws-node`) Pod 設定為 `true`。這就是為什麼在前面的輸出中沒有為 `kube-proxy` 和外掛程式的 `aws-node` Pod 指派 192.168.1.x 位址的原因。如需有關 Pod `hostNetwork` 設定的詳細資訊，請參閱 Kubernetes API 參考中的 PodSpec v1 核心。

步驟 5：刪除教學課程資源

完成本教學課程後，建議您刪除您建立的資源。然後，您可以調整步驟以啟用生產叢集的自訂聯網。

1. 如果您建立的節點群組僅用於測試，請將其刪除。

```
aws eks delete-nodegroup --cluster-name my-custom-networking-cluster --nodegroup-name
 my-nodegroup
```

2. 即使 CLI AWS 輸出顯示叢集已刪除後，刪除程序仍可能實際上未完成。刪除程序需要幾分鐘。執行下列命令來確認已完成。

```
aws eks describe-nodegroup --cluster-name my-custom-networking-cluster --nodegroup-
name my-nodegroup --query nodegroup.status --output text
```

在傳回與下列輸出類似的輸出之前，請勿繼續執行。

```
An error occurred (ResourceNotFoundException) when calling the DescribeNodegroup
 operation: No node group found for name: my-nodegroup.
```

3. 如果您建立的節點群組僅用於測試，則請刪除節點 IAM 角色。

   a. 將策略與角色分離。

   ```
   aws iam detach-role-policy --role-name myCustomNetworkingNodeRole --policy-arn
    arn:aws: iam::aws:policy/AmazonEKSWorkerNodePolicy
   aws iam detach-role-policy --role-name myCustomNetworkingNodeRole --policy-arn
    arn:aws: iam::aws:policy/AmazonEC2ContainerRegistryReadOnly
   aws iam detach-role-policy --role-name myCustomNetworkingNodeRole --policy-arn
    arn:aws: iam::aws:policy/AmazonEKS_CNI_Policy
   ```

   b. 刪除角色。

   ```
   aws iam delete-role --role-name myCustomNetworkingNodeRole
   ```

4. 刪除叢集。

   ```
   aws eks delete-cluster --name my-custom-networking-cluster
   ```

   請使用下列命令來確認刪除叢集。

   ```
   aws eks describe-cluster --name my-custom-networking-cluster --query cluster.status
    --output text
   ```

   如果傳回類似以下的輸出，則已成功刪除叢集。

   ```
   An error occurred (ResourceNotFoundException) when calling the DescribeCluster
    operation: No cluster found for name: my-custom-networking-cluster.
   ```

5. 刪除叢集 IAM 角色。

   a. 將策略與角色分離。

```
aws iam detach-role-policy --role-name myCustomNetworkingAmazonEKSClusterRole --
policy-arn arn:aws: iam::aws:policy/AmazonEKSClusterPolicy
```

b. 刪除角色。

```
aws iam delete-role --role-name myCustomNetworkingAmazonEKSClusterRole
```

6. 刪除您在上一個步驟中建立的子網。

```
aws ec2 delete-subnet --subnet-id $new_subnet_id_1
aws ec2 delete-subnet --subnet-id $new_subnet_id_2
```

7. 刪除您建立的 VPC。

```
aws cloudformation delete-stack --stack-name my-eks-custom-networking-vpc
```

將更多 IP 位址指派給具有字首的 Amazon EKS 節點

適用於：包含 Amazon EC2 執行個體的 Linux 和 Windows 節點

適用於：公有和私有子網路

每個 Amazon EC2 執行個體都支援最大數量的彈性網路介面，以及可指派給每個網路介面的最大數量的 IP 地址。每個節點的每個網路介面都需要一個 IP 地址。所有其他可用的 IP 地址都可以指派給 Pods。每個 Pod 都需要專屬的 IP 地址。因此，您的節點可能具有可用的運算和記憶體資源，但無法容納其他 Pods，因為節點已耗盡要指派給 Pods 的 IP 位址。

您可以透過指派 IP 字首 (而不是將個別的次要 IP 位址指派給節點)，大幅增加節點可指派給 Pods 的 IP 位址數量。每個字首都包含多個 IP 地址。如果您未針對 IP 字首指派設定叢集，則叢集必須進行更多 Amazon EC2 應用程式介面 (API) 呼叫，才能設定 Pod 連線所需的網路介面和 IP 位址。隨著叢集規模不斷擴大，這些 API 呼叫的頻率可能會導致 Pod 和執行個體的啟動時間變得更長。這樣一來，會導致擴展延遲以滿足大型和高峰工作負載的需求，並增加成本和管理負荷，因為您需要佈建額外的叢集和 VPC 以符合擴展需求。如需詳細資訊，請參閱 GitHub 上的 Kubernetes 可擴展性閾值。

與適用於 Kubernetes 功能的 Amazon VPC CNI 外掛程式相容

您可以搭配 IP 字首使用下列功能：

- IPv4 來源網路位址轉譯 - 如需詳細資訊，請參閱 the section called “傳出流量”。

- 叢集、Pod 及服務的 IPv6 位址 - 如需詳細資訊，請參閱 the section called “IPv6”。
- 使用 Kubernetes 網路政策來限制流量 - 如需詳細資訊，請參閱 the section called “Kubernetes 政策”。

以下清單提供了適用的 Amazon VPC CNI 外掛程式設定的相關資訊。有關每個設定的更多資訊，請參閱 GitHub 上的 amazon-vpc-cni-k8s。

- `WARM_IP_TARGET`
- `MINIMUM_IP_TARGET`
- `WARM_PREFIX_TARGET`

考量事項

使用此功能時請考慮以下事項：

- 每種 Amazon EC2 執行個體類型都支援最大數量的 Pod。如果受管節點群組包含多個執行個體類型，叢集中執行個體的最小 Pod 數量上限會套用至叢集中的所有節點。
- 依預設，您最多可在節點上執行 110 個 Pods，但您可以變更該數量。如果您變更該數量且擁有現有受管節點群組，則節點群組的下一個 AMI 或啟動範本更新會導致新的工作節點出現變更後的值。
- 從指派 IP 地址轉換為指派 IP 字首時，建議您建立新的節點群組來增加可用 IP 地址的數量，而不是對現有節點執行輪流取代。在同時指派 IP 位址和字首的節點上執行 Pod，可能會導致公告的 IP 位址容量不一致，進而影響節點日後的工作負載。有關執行轉換的建議方式，請參閱《Amazon EKS 最佳實務指南》中的 Linux 的字首委派模式。
- 安全群組範圍為節點層級 - 如需詳細資訊，請參閱安全群組。
- 指派給網路介面的 IP 字首支援每個節點的高 Pod 密度，且具有最佳的啟動時間。
- IP 字首和 IP 地址與標準 Amazon EC2 彈性網路介面相關聯。需要特定安全群組的 Pod 會被指派分支網路介面的主要 IP 地址。您可以把取得 IP 位址的 Pod，或從 IP 字首的 IP 位址與在相同節點上取得分支網路介面的 Pod 混合。
- 僅適用於具有 Linux 節點的叢集。
  - 設定附加元件以指派字首給網路介面後，您就無法將 Kubernetes 專用 Amazon VPC CNI 外掛程式附加元件降級為低於 `1.9.0` (或 `1.10.1`) 的版本，且不移除叢集中所有節點群組中的所有節點。
  - 如果您也使用 Pod 的安全群組，且 `POD_SECURITY_GROUP_ENFORCING_MODE = standard`、`AWS_VPC_K8S_CNI_EXTERNALSNAT = false`，則當 Pod 與 VPC 外部端點通訊時，將使用節點的安全群組，而不會使用您指派給 Pod 的任何安全群組。

如果您也使用 Pod 的安全群組，且 POD_SECURITY_GROUP_ENFORCING_MODE = strict，則當 Pods 與 VPC 外部端點通訊時，將使用 Pod's 安全群組。

增加 Amazon EKS 節點的可用 IP 位址數量

您可以透過指派 IP 字首 (而不是將個別的次要 IP 位址指派給節點)，大幅增加節點可指派給 Pod 的 IP 位址數量。

先決條件

- 您需要現有的叢集。若要部署叢集，請參閱 the section called "建立 叢集"。
- Amazon EKS 節點所在的子網路必須具有足夠的連續 /28 (針對 IPv4 叢集) 或 /80 (針對 IPv6 叢集) 無類別域間路由 (CIDR) 區塊。IPv6 叢集中只能包含 Linux 節點。如果 IP 地址分散在子網路 CIDR 中，則使用 IP 字首可能會失敗。我們建議下列作法：
  - 使用子網路 CIDR 保留，如此一來，即使保留範圍內的任何 IP 位址仍在使用中，發布後也不會重新指派 IP 位址。此舉可確保字首無需分割，即可用於配置。
  - 使用專門用於執行指派 IP 字首之工作負載的新子網路。指派 IP 字首時，Windows 和 Linux 工作負載可在同一子網路中同時執行。
- 若要將 IP 字首指派給節點，您的節點必須是 AWS Nitro 型。非 Nitro 型執行個體會繼續分配個別的次要 IP 位址，但要指派給 Pod 的 IP 位址數量遠低於 Nitro 型執行個體。
- 僅適用於具有節點的叢集：如果您的叢集針對 IPv4 系列設定，則必須安裝 Kubernetes 專用 Amazon VPC CNI 外掛程式附加元件的版本 1.9.0 或更新版本。您可以使用下列命令來檢查目前版本：

  ```
  kubectl describe daemonset aws-node --namespace kube-system | grep Image | cut -d "/"
   -f 2
  ```

  如果您的叢集針對 IPv6 系列設定，則必須安裝附加元件的版本 1.10.1。如果您的外掛程式版本早於所需版本，則必須對其進行更新。如需詳細資訊，請參閱使用 Amazon VPC CNI 將 IP 指派給 Pod的更新章節。
- 僅適用於具有 Windows 節點的叢集
  - 您必須為您的叢集啟用 Windows 支援。如需詳細資訊，請參閱the section called "啟用 Windows 支援"。

將 IP 位址字首指派給節點

設定叢集以將 IP 地址字首指派給節點。完成與節點作業系統相符的程序。

Linux

1. 啟用參數以為 Amazon VPC CNI DaemonSet 的網路介面指派字首。當您部署叢集時，Kubernetes 專用 Amazon VPC CNI 外掛程式附加元件的版本 `1.10.1` 或更高版本將與其一起部署。如果您使用 `IPv6` 系列建立叢集，則此設定會預設設為 `true`。如果您使用 `IPv4` 系列建立叢集，則此設定會預設設為 `false`。

```
kubectl set env daemonset aws-node -n kube-system ENABLE_PREFIX_DELEGATION=true
```

> **Important**
>
> 即使子網路有可用的 IP 位址，如果子網路沒有任何可用的連續 /28 區塊，您也會在 Kubernetes 專用 Amazon VPC CNI 外掛程式日誌中看到下列錯誤。
>
> ```
> InsufficientCidrBlocks: The specified subnet does not have enough free cidr
>  blocks to satisfy the request
> ```
>
> 這可能是由分散在子網路上的現有次要 IP 地址的分段造成。若要解決此錯誤，請建立新的子網路並在該處啟動 Pod，或使用 Amazon EC2 子網路 CIDR 保留來保留子網路內的空間，以便與字首指派搭配使用。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的子網路 CIDR 保留。

2. 如果計劃在沒有啟動範本的情況下或使用未在其中指定 AMI ID 的啟動範本來部署受管節點群組，且您使用的是先決條件中列出的 Kubernetes 專用 Amazon VPC CNI 外掛程式版本或更新版本，則請跳至下一個步驟。受管節點群組會自動為您計算 Pod 的數量上限。

   如果您要使用已在其中指定 AMI ID 的啟動範本來部署自我管理節點群組或受管節點群組，則必須設定節點的 Pod 數目上限。如需如何判斷適當值的詳細資訊，請參閱 the section called “maxPods的判斷方式”。

> **Important**
>
> 受管節點群組會對 的值強制執行最大數量maxPods。對於少於 30 個 vCPU 的執行個體，數量上限為 110；對於所有其他執行個體，數量上限為 250。無論是否啟用字首委派，都會套用此數量上限。

3. 如果您使用針對 IPv6 設定叢集，請跳到下一個步驟。

   在下列其中一個選項中指定參數。要判斷哪個選項適合您，以及提供哪個值，請參閱 GitHub 上的 WARM_PREFIX_TARGET、WARM_IP_TARGET 和 MINIMUM_IP_TARGET。

   您可以用大於零的值取代 example values。

   - WARM_PREFIX_TARGET

     ```
     kubectl set env ds aws-node -n kube-system WARM_PREFIX_TARGET=1
     ```

   - WARM_IP_TARGET 或 MINIMUM_IP_TARGET：如果此值已設定，則其會覆寫為 WARM_PREFIX_TARGET 設定的任何值。

     ```
     kubectl set env ds aws-node -n kube-system WARM_IP_TARGET=5
     ```

     ```
     kubectl set env ds aws-node -n kube-system MINIMUM_IP_TARGET=2
     ```

4. 使用至少一個 Amazon EC2 Nitro Amazon Linux 2023 執行個體類型，建立下列其中一個節點群組類型。如需 Nitro 執行個體類型的清單，請參閱《Amazon EC2 使用者指南》中的建置在 Nitro 系統上的執行個體。Windows 不支援這項功能。對於包含 *110* 的選項，使用步驟 3 的值 (建議) 或您自己的值將其取代。

   - 自我管理 – 使用建立自我管理的 Amazon Linux 節點中的指示來部署節點群組。建立 CloudFormation 堆疊之前，請開啟範本檔案，並將 UserData中的 調整NodeLaunchTemplate為如下。

     ```
     ...
                 apiVersion: node.eks.aws/v1alpha1
                 kind: NodeConfig
                 spec:
                   cluster:
                     name: ${ClusterName}
                     apiServerEndpoint: ${ApiServerEndpoint}
     ```

```
                certificateAuthority: ${CertificateAuthorityData}
                cidr: ${ServiceCidr}
              kubelet:
                config:
                  maxPods: 110
...
```

如果使用 eksctl 來建立節點群組，便可使用以下命令。

```
eksctl create nodegroup --cluster my-cluster --managed=false --max-pods-per-node
 110
```

- 受管：使用下列其中一個選項，部署節點群組：
  - 沒有啟動範本，或有啟動範本，但沒有指定 AMI ID：完成建立叢集的受管節點群組中的程序。受管節點群組會自動為您計算 Amazon EKS 建議max-pods值。
  - 使用具有指定 AMI ID 的啟動範本：在啟動範本中，指定 Amazon EKS 最佳化 AMI ID 或基於 Amazon EKS 最佳化 AMI 的自訂 AMI，然後使用啟動範本部署節點群組，並在啟動範本中提供下列使用者資料。此使用者資料會傳遞NodeConfig物件，以供節點上的nodeadm工具讀取。如需 的詳細資訊nodeadm，請參閱 nodeadm 文件。

    ```
    MIME-Version: 1.0
    Content-Type: multipart/mixed; boundary="//"

    --//
    Content-Type: application/node.eks.aws

    ---
    apiVersion: node.eks.aws/v1alpha1
    kind: NodeConfig
    spec:
     cluster:
       apiServerEndpoint: my-cluster
       certificateAuthority: LS0t...
       cidr: 10.100.0.0/16
       name: my-cluster
     kubelet:
       config:
         maxPods: 110
    --//--
    ```

    如果使用 eksctl 來建立節點群組，便可使用以下命令。

```
eksctl create nodegroup --cluster my-cluster --max-pods-per-node 110
```

如果您建立的自訂 AMI 並非基於 Amazon EKS 最佳化 AMI，則需要自行自訂建立組態。

> **Note**
>
> 如果您還想要將 IP 位址指派給來自與執行個體不同子網路的 Pod，則需要在此步驟中啟用該功能。如需詳細資訊，請參閱the section called “自訂聯網”。

Windows

1. 啟用 IP 字首指派。
   a. 開啟 `amazon-vpc-cni` `ConfigMap` 進行編輯。

   ```
   kubectl edit configmap -n kube-system amazon-vpc-cni -o yaml
   ```

   b. 將下行新增至 `data` 區段：

   ```
     enable-windows-prefix-delegation: "true"
   ```

   c. 儲存檔案並關閉編輯器。
   d. 確認行已新增至 `ConfigMap`。

   ```
   kubectl get configmap -n kube-system amazon-vpc-cni -o "jsonpath={.data.enable-
   windows-prefix-delegation}"
   ```

   如果傳回的輸出不是 `true`，則可能會出現錯誤。請嘗試再次完成該步驟。

   > **Important**
   >
   > 即使子網路有可用的 IP 位址，如果子網路沒有任何可用的連續 /28 區塊，您也會在 Kubernetes 專用 Amazon VPC CNI 外掛程式日誌中看到下列錯誤。
   >
   > ```
   > InsufficientCidrBlocks: The specified subnet does not have enough free cidr
   >  blocks to satisfy the request
   > ```

這可能是由分散在子網路上的現有次要 IP 地址的分段造成。若要解決此錯誤，請建立新的子網路並在該處啟動 Pod，或使用 Amazon EC2 子網路 CIDR 保留來保留子網路內的空間，以便與字首指派搭配使用。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的子網路 CIDR 保留。

2. (選用) 指定其他組態來控制叢集的預先擴展和動態擴展行為。如需詳細資訊，請參閱 GitHub 上的 Windows 上包含字首委派模式的組態選項。

   a. 開啟 `amazon-vpc-cni ConfigMap` 進行編輯。

   ```
   kubectl edit configmap -n kube-system amazon-vpc-cni -o yaml
   ```

   b. 將範例值取代為大於零的值，並將您需要的項目新增至 的 `data`區段`ConfigMap`。如果您為 `warm-ip-target` 或 `minimum-ip-target` 設定了值，則該值會覆寫任何為 `warm-prefix-target` 設定的值。

   ```
   warm-prefix-target: "1"
   warm-ip-target: "5"
   minimum-ip-target: "2"
   ```

   c. 儲存檔案並關閉編輯器。

3. 建立 Windows 節點群組，其中至少包含一個 Amazon EC2 Nitro 執行個體類型。如需 Nitro 執行個體類型的清單，請參閱《Amazon EC2 使用者指南》中的建置在 Nitro 系統上的執行個體。依預設，您最多可將 110 個 Pod 部署到節點。如果您要增加或減少該數量，請在引導組態的使用者資料中指定以下內容：將 *`max-pods-quantity`* 取代為您的 max Pod 值。

   ```
   -KubeletExtraArgs '--max-pods=max-pods-quantity'
   ```

   如果您要部署受管節點群組，則需要在啟動範本中新增此組態。如需詳細資訊，請參閱the section called “啟動範本”。如需有關 Windows 引導指令碼組態參數的詳細資訊，請參閱 the section called “引導指令碼組態參數”。

確定最大 Pod 和可用的 IP 位址

1. 部署您的節點後，請檢視叢集中的節點。

   ```
   kubectl get nodes
   ```

範例輸出如下。

```
NAME                                               STATUS     ROLES    AGE   VERSION
ip-192-168-22-103.region-code.compute.internal   Ready      <none>   19m   v1.XX.X-
eks-6b7464
ip-192-168-97-94.region-code.compute.internal    Ready      <none>   19m   v1.XX.X-
eks-6b7464
```

2. 描述其中一個節點，判斷該節點的 max-pods 值以及可用 IP 地址的數量。使用上一個輸出中傳回的其中一個節點名稱中的 IPv4 位址取代 *192.168.30.193*。

```
kubectl describe node ip-192-168-30-193.region-code.compute.internal | grep 'pods\|
PrivateIPv4Address'
```

範例輸出如下。

```
pods:                                   110
vpc.amazonaws.com/PrivateIPv4Address:  144
```

在先前的輸出中，110 是 Kubernetes 將部署至節點的 Pod 數量上限，即使有 *144* 個可用的 IP 位址。

將安全群組指派至個別 Pod

適用於：包含 Amazon EC2 執行個體的 Linux 節點

適用於：私有子網路

Pod 的安全群組整合了 Amazon EC2 安全群組與 Kubernetes Pod。您可以使用 Amazon EC2 安全群組定義規則，允許您部署的 Pod 到許多 Amazon EC2 執行個體類型和 Fargate 上執行的節點之間的對內和對外網路流量。如需此功能的詳細說明，請參閱 Pod 的安全群組簡介部落格文章。

與適用於 Kubernetes 功能的 Amazon VPC CNI 外掛程式相容

您可針對具備以下功能的 Pod 使用安全群組：

- IPv4 來源網路位址轉譯 - 如需詳細資訊，請參閱 the section called “傳出流量”。
- 叢集、Pod 及服務的 IPv6 位址 - 如需詳細資訊，請參閱 the section called “IPv6”。

- 使用 Kubernetes 網路政策來限制流量 - 如需詳細資訊，請參閱 the section called “Kubernetes 政策”。

考量事項

部署 Pod 的安全群組之前，請考慮下列限制和條件：

- Pod 的安全群組無法與 Windows 節點或 EKS Auto 模式搭配使用。
- Pod 的安全群組可與為包含 Amazon EC2 節點的 IPv6 系列設定的叢集搭配使用，方法是使用 1.16.0 版或更新的 Amazon VPC CNI 外掛程式。您可以使用 Amazon VPC CNI 外掛程式的 1.7.7 版或更新版本，針對僅包含 Fargate 節點的 IPv6 系列設定叢集，使用 Pod 的安全群組。如需詳細資訊，請參閱the section called “IPv6”。
- Pod 的安全群組受大多數 Nitro 型 Amazon EC2 執行個體系列支援，但並非受所有系列世代皆支援。例如，支援 m5、c5、r5、m6g、c6g 和 r6g 執行個體系列和世代。不支援 t 系列中的執行個體類型。如需完整的支援執行個體類型清單，請參閱 GitHub 上的 limits.go 檔案。您的節點必須為該檔案列出的具有 IsTrunkingCompatible: true 的執行個體類型之一。
- 如果您使用的是自訂聯網和 Pod 的安全群組，則會使用 Pod 安全群組指定的安全群組，而不是 ENIConfig 指定的安全群組。
- 如果您使用的是版本 1.10.2 或更新版本的 Amazon VPC CNI 外掛程式，且 Pod 規格包括 terminationGracePeriodSeconds 設定，則該設置的值不能為零。
- 如果您使用的是版本 1.10 或更舊版本的 Amazon VPC CNI 外掛程式或預設設定為 POD_SECURITY_GROUP_ENFORCING_MODE=strict 的版本 1.11，則使用具有設定為 externalTrafficPolicy 的 Local 的 NodePort 和 LoadBalancer 類型的 Kubernetes 服務不支援指派安全群組至其的 Pod。如需將負載平衡器與執行個體目標搭配使用的詳細資訊，請參閱 the section called “網路負載平衡”。
- 如果您使用的是版本 1.10 或更舊版本的 Amazon VPC CNI 外掛程式或預設設定為 POD_SECURITY_GROUP_ENFORCING_MODE=strict 的版本 1.11，來自具有指派安全群組之 Pod 的對外流量會停用來源 NAT，以便套用輸出安全群組規則。若要存取網際網路，必須在部署在使用 NAT 閘道或執行個體設定的私有子網路中的節點上啟動具有指派安全群組的 Pod。已指派安全群組部署至公有子網路的 Pod 無法存取網際網路。

  如果您使用的是版本 1.11 或更新版本的搭配 POD_SECURITY_GROUP_ENFORCING_MODE=standard 的外掛程式，則以 VPC 外部為目標的 Pod 流量會轉換為執行個體主要網路介面的 IP 位址。對於此流量，將使用主要網路介面的安全群組中的規則，而不是使用 Pod 安全群組中的規則。

- 若要將 Calico 網路政策與關聯安全群組的 Pod 搭配使用，則必須使用 `1.11.0` 版或更新版本的 Amazon VPC CNI 外掛程式，並設定為 `POD_SECURITY_GROUP_ENFORCING_MODE=standard`。否則，具有關聯的安全群組之 Pod 的流量流出和流向不受 Calico 網路政策強制執行的約束，且僅限於 Amazon EC2 安全群組強制執行。若要更新 Amazon VPC CNI 版本，請參閱 the section called "Amazon VPC CNI"。
- Pod 在 Amazon EC2 節點上執行，這些節點使用的安全群組位於使用 NodeLocal DNSCache 的叢集，僅受 Amazon VPC CNI 外掛程式 `1.11.0` 版或更新版本支援，且設定為 `POD_SECURITY_GROUP_ENFORCING_MODE=standard`。若要更新您的 Amazon VPC CNI 外掛程式版本，請參閱 the section called "Amazon VPC CNI"。
- Pod 的安全群組可能會導致具有高流失率的 Pod 產生更高的 Pod 啟動延遲。原因是資源控制器的速率限制。
- EC2 安全群組範圍為 Pod 層級 - 如需詳細資訊，請參閱安全群組。

  若您設定 `POD_SECURITY_GROUP_ENFORCING_MODE=standard` 和 `AWS_VPC_K8S_CNI_EXTERNALSNAT=false`，則目的地為 VPC 外部端點的流量會使用節點的安全群組，而不是 Pod 的安全群組。

針對適用於 Amazon EKS Pod 的安全群組設定適用於 Kubernetes 的 Amazon VPC CNI 外掛程式

若您將 Pod 與 Amazon EC2 執行個體配合使用，則需針對安全群組設定適用於 Kubernetes 的 Amazon VPC CNI 外掛程式

若您僅使用 Fargate Pod，且叢集中沒有任何 Amazon EC2 節點，請參閱 the section called "SecurityGroupPolicy"。

1. 使用下列命令檢查您目前適用於 Kubernetes 的 Amazon VPC CNI 外掛程式版本：

```
kubectl describe daemonset aws-node --namespace kube-system | grep amazon-k8s-cni: |
 cut -d : -f 3
```

   範例輸出如下。

```
v1.7.6
```

   若適用於 Kubernetes 的 Amazon VPC CNI 外掛程式版本早於 `1.7.7`，則請將外掛程式更新至 `1.7.7` 版或更新版本。如需詳細資訊，請參閱the section called "Amazon VPC CNI"。

2. 新增 AmazonEKSVPCResourceController 受管 IAM 政策至與您的 Amazon EKS 叢集關聯的叢集角色。此政策允許 角色管理網路介面及其私有 IP 地址，以及將網路介面連接至執行個體和從執行個體分離。

   a. 擷取叢集 IAM 角色的名稱，並存放在變數中。使用您叢集的名稱取代 *my-cluster*。

   ```
   cluster_role=$(aws eks describe-cluster --name my-cluster --query cluster.roleArn
    --output text | cut -d / -f 2)
   ```

   b. 將政策連接到角色。

   ```
   aws iam attach-role-policy --policy-arn arn:aws: iam::aws:policy/
   AmazonEKSVPCResourceController --role-name $cluster_role
   ```

3. 藉由將 ENABLE_POD_ENI 變數設定為 aws-node DaemonSet 中的 true，啟用 Amazon VPC CNI 附加程式來管理 Pod 的網路介面。一旦此設定設定為 true，對於叢集中的每個節點，該附加元件會建立一個 cninode 自訂資源。VPC 資源控制器會建立並連接一個稱為幹線網路介面與描述 aws-k8s-trunk-eni 的特殊網路介面。

   ```
   kubectl set env daemonset aws-node -n kube-system ENABLE_POD_ENI=true
   ```

   > **Note**
   >
   > 幹線網路介面包含在執行個體類型所支援的最大網路介面數量中。如需每種執行個體類型所支援的介面數量上限的清單，請參閱 Amazon EC2 使用者指南中的每種執行個體類型每個網路介面的 IP 位址。如果您的節點已經連接到標準網路介面的最大數量，那麼 VPC 資源控制器將保留一個空間。您將不得不縮減正在執行的 Pod 的規模，以便控制器分開並刪除標準網路介面，建立幹線網路介面，並將其連接到執行個體。

4. 您可以使用以下命令查看您的哪些節點擁有 CNINode 自訂資源。如果傳回 No resources found，則等待幾秒鐘後再重試一次。上一步驟要求重新啟動適用於 Kubernetes Pod 的 Amazon VPC CNI 外掛程式，這需要幾秒鐘的時間。

   ```
   kubectl get cninode -A
        NAME FEATURES
        ip-192-168-64-141.us-west-2.compute.internal [{"name":"SecurityGroupsForPods"}]
        ip-192-168-7-203.us-west-2.compute.internal [{"name":"SecurityGroupsForPods"}]
   ```

如果您使用的 VPC CNI 版本低於 1.15，則會使用節點標籤而非 CNINode 自訂資源。您可以使用以下命令查看您的哪些節點已將節點標籤 aws-k8s-trunk-eni 設定為 true。如果傳回 No resources found，則等待幾秒鐘後再重試一次。上一步驟要求重新啟動適用於 Kubernetes Pod 的 Amazon VPC CNI 外掛程式，這需要幾秒鐘的時間。

```
kubectl get nodes -o wide -l vpc.amazonaws.com/has-trunk-attached=true
```

一旦建立幹線網路介面，Pod 就可以從幹線或標準網路介面指派次要 IP 位址。如果刪除節點，則會自動刪除幹線介面。

當您在稍後的步驟中為 Pod 部署安全群組時，VPC 資源控制器會建立稱為分支網路介面與 aws-k8s-branch-eni 的描述，並將安全群組與其關聯。除了連接至節點的標準和幹線網路介面之外，還會建立分支網路介面。

如果您使用的是存活或整備探查，則還需要停用 TCP 早期的 demux，以便 kubelet 可以透過 TCP 連接到分支網路介面上的 Pod。若要停用 TCP 早期的 demux，請執行下列命令：

```
kubectl patch daemonset aws-node -n kube-system \
  -p '{"spec": {"template": {"spec": {"initContainers": [{"env":
[{"name":"DISABLE_TCP_EARLY_DEMUX","value":"true"}],"name":"aws-vpc-cni-init"}]}}}}'
```

> **Note**
>
> 若您使用的是適用於 Kubernetes 附加元件的 Amazon VPC CNI 外掛程式 1.11.0 版或更新版本，並設定 POD_SECURITY_GROUP_ENFORCING_MODE=standard，如下一個步驟所述，則不需要執行之前的命令。

5. 若您的叢集使用 NodeLocal DNSCache，或者您希望將 Calico 網路政策與具有自己的安全群組的 Pod 搭配使用，或者您的 Kubernetes 服務類型為 NodePort 和 LoadBalancer，其使用的執行個體目標中您欲指派安全群組的 Pod 的 externalTrafficPolicy 設定為 Local，則必須使用適用於 Kubernetes 附加元件的 Amazon VPC CNI 外掛程式 1.11.0 版或更新版本，且必須啟用下列設定：

```
kubectl set env daemonset aws-node -n kube-system
 POD_SECURITY_GROUP_ENFORCING_MODE=standard
```

重要事項：Pod 安全群組規則不會套用至介於 Pod 之間或介於 Pod 與服務之間的流量，例如位於同一個節點上的 **kubelet** 或 **nodeLocalDNS**。在相同節點上使用不同安全群組的 Pod 無法進行通訊，這是因為它們在不同子網路中設定，且這些子網路之間已停用路由。來自 Pod 的傳出流量傳輸到 VPC 外部的地址是轉換為執行個體主要網路介面的 IP 位址的網路地址 (除非您還設定了 AWS_VPC_K8S_CNI_EXTERNALSNAT=true)。對於此流量，將使用主要網路介面的安全群組中的規則，而不是使用 Pod 安全群組中的規則。** 若要將此設定套用至現有的 Pod，您必須重新啟動 Pod 或 Pod 正在執行之節點。

6. 如需了解如何使用適用於 Pod 的安全群組政策，請參閱 the section called “SecurityGroupPolicy”。

使用適用於 Amazon EKS Pod 的安全群組政策

如需使用適用於 Pod 的安全群組，必須設定現有的安全群組。下列步驟介紹如何使用 Pod 的安全群組政策。除非另有備註，請從同一終端機完成所有步驟，因為變數用於不會保留跨終端機的以下步驟。

若您有包含 Amazon EC2 執行個體的 Pod，使用此程序之前，必須設定外掛程式。如需詳細資訊，請參閱the section called “設定”。

1. 建立 Kubernetes 命名空間以部署 資源。您可以將 *my-namespace* 取代為要使用的命名空間的名稱。

```
kubectl create namespace my-namespace
```

2. 將 Amazon EKS SecurityGroupPolicy 部署至您的叢集。

   a. 將以下內容複製到您的裝置。若您希望根據服務帳戶標籤選取 Pod，則您可以使用 serviceAccountSelector 取代 *podSelector*。您必須指定其中一個選取器或其他工具。空白 podSelector (例如：podSelector: {}) 選取命名空間中的所有 Pood。您可以將 *my-role* 變更為您的角色名稱。空白 serviceAccountSelector 會選取命名空間中的所有服務帳戶。您可以將 *my-security-group-policy* 取代為您的 SecurityGroupPolicy 的名稱，將 *my-namespace* 取代為要在其中建立 SecurityGroupPolicy 的命名空間。

   您必須將 *my_pod_security_group_id* 取代為現有安全群組的 ID。如果您沒有現有的安全群組，則必須先建立一個。如需詳細資訊，請參閱《Amazon EC2 使用者指南》https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/中的適用於 Linux 執行個體的 Amazon EC2 安全群組一節。您可以指定 1-5 個安全群組 ID。如果您指定多個 ID，則所有安全群組中的所有規則的組合對選取的 Pod 都有效。

```
cat >my-security-group-policy.yaml <<EOF
```

```
apiVersion: vpcresources.k8s.aws/v1beta1
kind: SecurityGroupPolicy
metadata:
  name: my-security-group-policy
  namespace: my-namespace
spec:
  podSelector:
    matchLabels:
      role: my-role
  securityGroups:
    groupIds:
      - my_pod_security_group_id
EOF
```

> **Important**
>
> 您指定的 Pod 的一個或多個安全群組 必須符合下列條件：
>
> - 它們必須存在。如果它們不存在，那麼當您部署符合選取器的 Pod 時，您的 Pod 仍會停留在建立程序中。如果您描述 Pod，您會看到類似下列錯誤訊息：`An error occurred (InvalidSecurityGroupID.NotFound) when calling the CreateNetworkInterface operation: The securityGroup ID 'sg-05b1d815d1EXAMPLE' does not exist`。
> - 安全群組必須允許透過您已設定偵測的任何連接埠，而來自套用於您節點的安全群組的傳入通訊 (適用於 kubelet)。
> - 其必須允許透過 TCP 和 UDP 連接埠 53 與指派給執行 CoreDNS 的 Pod 的安全群組 (或 Pod 在其上執行的節點) 進行對外通訊。您的 CoreDNS Pod 的安全群組必須允許來自您所指定安全群組的傳入 TCP 和 UDP 連接埠 53 流量。
> - 其必須具備必要的傳入和傳出規則，從而和其需要與之溝通的其他 Pod 溝通。
> - 如果您使用具有 Fargate 的安全群組，其必須具有允許 Pod 與 Kubernetes 控制平面通訊的規則。執行此動作最簡單的方法是指定叢集安全群組為其中一個安全群組。
>
> 安全群組政策僅適用於新排程的 Pod。它們不會影響執行中的 Pod。

b. 部署政策。

```
kubectl apply -f my-security-group-policy.yaml
```

3. 部署範例應用程式，其中標籤符合您在上一步驟指定的 *podSelector* 的 *my-role* 值。

a. 將以下內容複製到您的裝置。使用您自己的值取代範例值，然後執行修改後的命令。如果您取代 *my-role*，請確保其與您在上一步中為選取器指定的值相同。

```
cat >sample-application.yaml <<EOF
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment
  namespace: my-namespace
  labels:
    app: my-app
spec:
  replicas: 4
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
        role: my-role
    spec:
      terminationGracePeriodSeconds: 120
      containers:
      - name: nginx
        image: public.ecr.aws/nginx/nginx:1.23
        ports:
        - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: my-app
  namespace: my-namespace
  labels:
    app: my-app
spec:
  selector:
    app: my-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

```
EOF
```

b. 使用下列命令部署應用程式。當您部署應用程式時，適用於 Kubernetes 的 Amazon VPC CNI 外掛程式會符合您在上一步驟中指定的 `role` 標籤和安全群組，且可套用至 Pod。

```
kubectl apply -f sample-application.yaml
```

4. 檢視與範例應用程式一起部署的 Pod。對於本主題的餘數，該終端機稱為 `TerminalA`。

```
kubectl get pods -n my-namespace -o wide
```

範例輸出如下。

```
NAME                             READY   STATUS    RESTARTS   AGE     IP
  NODE                                                NOMINATED NODE   READINESS GATES
my-deployment-5df6f7687b-4fbjm   1/1     Running   0          7m51s   192.168.53.48
  ip-192-168-33-28.region-code.compute.internal   <none>           <none>
my-deployment-5df6f7687b-j9fl4   1/1     Running   0          7m51s   192.168.70.145
  ip-192-168-92-33.region-code.compute.internal   <none>           <none>
my-deployment-5df6f7687b-rjxcz   1/1     Running   0          7m51s   192.168.73.207
  ip-192-168-92-33.region-code.compute.internal   <none>           <none>
my-deployment-5df6f7687b-zmb42   1/1     Running   0          7m51s   192.168.63.27
  ip-192-168-33-28.region-code.compute.internal   <none>           <none>
```

> **Note**
>
> 若任何 Pod 停滯，則嘗試這些小貼士。
>
> - 若任何 Pod 停滯在 `Waiting` 狀態，則請執行 `kubectl describe pod my-deployment-xxxxxxxxxx-xxxxx -n my-namespace` 。若您看到 `Insufficient permissions: Unable to create Elastic Network Interface.`，請確認您已在上一步驟中將 IAM 政策新增至 IAM 叢集角色。
> - 若任何 Pod 停滯在 `Pending` 狀態，請確認您的節點執行個體類型已列在 limits.go 中，並且尚未達到該執行個體類型支援的分支網路介面數量上限乘以節點群組中節點數量的乘積。例如，`m5.large` 執行個體支援九個分支網路介面。如果您的節點群組有五個節點，則最多可以為節點群組建立 45 個分支網路介面。您嘗試部署的第 46 個 Pod 將位於 `Pending` 狀態，直到刪除具有相關聯安全群組的另一個 Pod 為止。

如果您執行 kubectl describe pod *my-deployment-xxxxxxxxxx-xxxxx* -n *my-namespace* ，並看到類似下列訊息的訊息，則可以放心地將其忽略。當適用於 Kubernetes 的 Amazon VPC CNI 外掛程式嘗試設定主機聯網並在建立網路介面時失敗時，可能會出現此訊息。外掛程式會記錄此事件，直到網路介面建立為止。

```
Failed to create Pod sandbox: rpc error: code = Unknown desc = failed to set up
 sandbox container "e24268322e55c8185721f52df6493684f6c2c3bf4fd59c9c121fd4cdc894579f"
 network for Pod "my-deployment-5df6f7687b-4fbjm": networkPlugin
cni failed to set up Pod "my-deployment-5df6f7687b-4fbjm-c89wx_my-namespace" network:
 add cmd: failed to assign an IP address to container
```

您不能超過執行個體類型上可以執行的 Pod 數量上限。關於每種執行個體類型上可執行的 Pod 數目上限清單，請參閱 GitHub 上的 eni-max-pods.txt。當您刪除具有關聯安全群組的 Pod，或刪除 Pod 執行所在的節點時，VPC 資源控制器會刪除分支網路介面。如果您使用安全群組的 Pod 來刪除具有 Pod 的叢集，則控制器不會刪除分支網路介面，因此您需要自行刪除它們。如需有關如何刪除網路介面的資訊，請參閱《Amazon EC2 使用者指南》中的刪除網路介面。

5. 在單獨的終端機中，shell 轉換為 Pod。對於本主題的餘數，該終端機稱為 TerminalB。使用從上一步驟傳回的輸出中的其中一個 Pod 的 ID 取代 *5df6f7687b-4fbjm*。

```
kubectl exec -it -n my-namespace my-deployment-5df6f7687b-4fbjm -- /bin/bash
```

6. 從 TerminalB 中的 shell，確認範例應用程式是否正常運作。

```
curl my-app
```

範例輸出如下。

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
[...]
```

您收到了輸出，因為所有執行該應用程式的 Pod 都與您建立的安全群組關聯。該群組包含一個規則，其允許與安全群組關聯的所有 Pod 之間的流量。允許從該安全群組傳出 DNS 流量到與節點關聯的叢集安全群組。這些節點正在執行 CoreDNS Pod，您的 Pod 對此進行了名稱查詢。

7. 從 TerminalA 中，移除允許從安全群組與叢集進行 DNS 通訊的安全群組規則。如果在上一步驟並未新增 DNS 規則至叢集安全群組，則用您在其中建立規則的安全群組的 ID 取代 *$my_cluster_security_group_id*。

```
aws ec2 revoke-security-group-ingress --group-id $my_cluster_security_group_id --
security-group-rule-ids $my_tcp_rule_id
aws ec2 revoke-security-group-ingress --group-id $my_cluster_security_group_id --
security-group-rule-ids $my_udp_rule_id
```

8. 從 TerminalB，再次嘗試存取該應用程式。

```
curl my-app
```

範例輸出如下。

```
curl: (6) Could not resolve host: my-app
```

嘗試失敗，因為 Pod 不再能存取 CoreDNS Pod，其具有與之關聯的叢集安全群組。叢集安全群組不再具有允許從與 Pod 關聯的安全群組進行 DNS 通訊的安全群組規則。

如果您在上一步驟中，嘗試存取使用其中一個 Pod 傳回的 IP 位址的應用程式，您仍會接收到回應，因為已允許具有與之關聯的安全群組的 Pod 之間的所有連接埠，並且不需要進行名稱查詢。

9. 在實驗完成後，您可以移除自己建立的範例安全群組政策、應用程式和安全群組。從 TerminalA 執行下列命令。

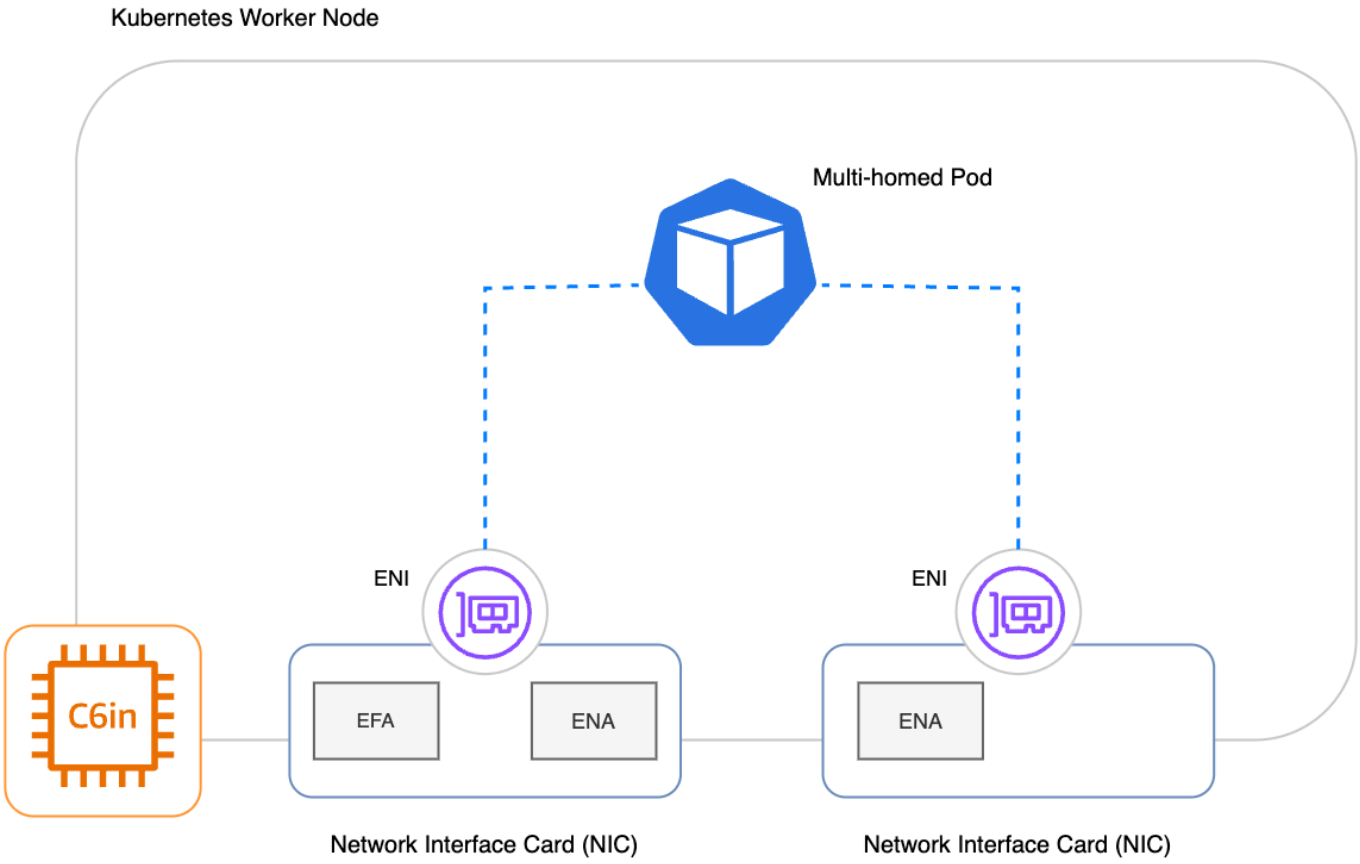
```
kubectl delete namespace my-namespace
aws ec2 revoke-security-group-ingress --group-id $my_pod_security_group_id --
security-group-rule-ids $my_inbound_self_rule_id
wait
sleep 45s
aws ec2 delete-security-group --group-id $my_pod_security_group_id
```

連接多個網路介面至 Pod

依預設，Amazon VPC CNI 外掛程式將一個 IP 位址指派給每個 Pod。此 IP 位址連接至彈性網路介面，以便處理 Pod 的所有傳入與傳出流量。如需增加頻寬及每秒封包速率效能，您可利用 VPC CNI 的多重 NIC 功能來設定多重主目錄 Pod。多重主目錄 Pod 是使用多個網路介面 (及多個 IP 位址) 的單

一 Kubernetes Pod。您可藉由執行多重主目錄 Pod，利用同時連線將其應用程式流量分佈到多個網路介面。對於人工智慧 (AI)、機器學習 (ML) 與高效能運算 (HPC) 使用案例，這樣做尤其有用。

下面的圖表顯示了在工作節點上執行的多重主目錄 Pod，其中使用了多個網路介面卡 (NIC)。



背景介紹

在 Amazon EC2 上，彈性網路介面是代表虛擬網路卡之 VPC 中的邏輯網路元件。針對多種 EC2 執行個體類型，網路介面會在硬體中共用單一網路介面卡 (NIC)。此單一 NIC 具有最大頻寬與每秒封包速率。

若啟用多重 NIC 功能，VPC CNI 預設不會大量指派 IP 位址。取而代之的是，VPC CNI 在新的 Pod 啟動時，會隨需向每個網路卡上的網路介面指派一個 IP 位址。此行為會降低 IP 位址耗盡的速率，透過使用多重主目錄 Pod 而增加。由於 VPC CNI 將隨需指派 IP 位址，在啟用多重 NIC 功能的執行個體上，啟用 Pod 可能需要更長時間。

考量事項

- 請確認 Kubernetes 叢集在執行 VPC CNI `1.20.0` 版及更新版本。多重 NIC 功能僅適用於 VPC CNI `1.20.0` 版或更新版本。
- 在 VPC CNI 外掛程式中啟用 `ENABLE_MULTI_NIC` 環境變數。您可執行以下命令來設定變數，以及啟動 DaemonSet 部署。
  - `kubectl set env daemonset aws-node -n kube-system ENABLE_MULTI_NIC=true`
- 請確認建立的工作節點具有多個網路介面卡 (NIC)。若要了解具有多個網路介面卡的 EC2 執行個體清單，請參閱 Amazon EC2 使用者指南中的網路卡。
- 若啟用多重 NIC 功能，VPC CNI 預設不會大量指派 IP 位址。由於 VPC CNI 將隨需指派 IP 位址，在啟用多重 NIC 功能的執行個體上，啟用 Pod 可能需要更長時間。如需詳細資訊，請參閱前一 the section called “背景介紹” 節。
- 啟用多重 NIC 功能後，依預設，Pod 沒有多個網路介面。必須設定每個工作負載使用多重 NIC。新增 `k8s.amazonaws.com/nicConfig: multi-nic-attachment` 註解至應擁有多個網路介面的工作負載。

IPv6 考量

- 自訂 IAM 政策 - 若是 `IPv6` 叢集，針對 VPC CNI 建立及使用以下自訂 IAM 政策。此政策專門針對多重 NIC 設定。若要了解 VPC CNI 與 `IPv6` 叢集配合的更多一般資訊，請參閱 the section called “IPv6”。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AmazonEKSCNIPolicyIPv6MultiNIC",
            "Effect": "Allow",
            "Action": [
                "ec2:CreateNetworkInterface",
                "ec2:DescribeInstances",
                "ec2:AssignIpv6Addresses",
                "ec2:DetachNetworkInterface",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeTags",
                "ec2:ModifyNetworkInterfaceAttribute",
                "ec2:DeleteNetworkInterface",
                "ec2:DescribeInstanceTypes",
                "ec2:UnassignIpv6Addresses",
```

```
                "ec2:AttachNetworkInterface",
                "ec2:DescribeSubnets"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AmazonEKSCNIPolicyENITagIPv6MultiNIC",
            "Effect": "Allow",
            "Action": "ec2:CreateTags",
            "Resource": "arn:aws:ec2:*:*:network-interface/*"
        }
    ]
}
```

- IPv6 轉換機制無法使用 - 若使用多重 NIC 功能，VPC CNI 不會指派 IPv4 位址給 IPv6 叢集上的 Pod。VPC CNI 而是將主機本機 IPv4 位址指派給每個 Pod，以便 Pod 與其他 Amazon VPC 或網際網路的外部 IPv4 資源通訊。

Usage

在 VPC CNI 中啟用多重 NIC aws-node 功能，以及 Pod 重新啟動之後，您可將每個工作負載設定為多重主目錄。以下是具有所需註釋的 YAML 組態範例：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: orders-deployment
  namespace: ecommerce
  labels:
    app: orders
spec:
  replicas: 3
  selector:
    matchLabels:
      app: orders
  template:
    metadata:
      annotations:
         k8s.amazonaws.com/nicConfig: multi-nic-attachment
      labels:
        app: orders
    spec:
```

```
...
```

常見問答集

1。什麼是網路介面卡 (NIC)？

網路介面卡 (NIC) 亦稱為網路卡，這是一種可針對底層雲端運算硬體啟用網路連線的實體裝置。在現代 EC2 伺服器中，網路介面卡是指 Nitro 網路卡。彈性網路介面 (ENI) 是這種底層網路卡的虛擬呈現方式。

某些 EC2 執行個體類型具有多重 NIC，能夠增加頻寬及封包速率效能。針對此類執行個體，您可指派次要 ENI 給額外的網路卡。舉例來說，ENI #1 能夠用做 NIC 連接至網路卡索引 0 的介面；而 ENI #2 則可用做 NIC 連接至單獨網路卡索引的介面。

2. 什麼是多重主目錄 Pod？

多重主目錄 Pod 是使用多個網路介面 (及隱含多個 IP 位址) 的單一 Kubernetes Pod。每個 Pod 網路介面與彈性網路介面 (ENI) 關聯，並且這些 ENI 是底層工作節點上單獨 NIC 的邏輯呈現方式。藉助多個網路介面，多重主目錄 Pod 可擁有額外的資料傳輸容量，這亦可提高其資料傳輸率。

> **Important**
>
> VPC CNI 僅可在擁有多重 NIC 的執行個體類型上設定多重主目錄 Pod。

3. 這項功能有什麼用？

若您需要在 Kubernetes 型工作負載中擴展網路效能，可使用多重 NIC 功能來執行多重主目錄 Pod，這些 Pod 能夠與所有連接 ENA 裝置的底層 NIC 互動。藉助額外的網路卡，將應用程式流量分佈至多個同時連線，藉此來增加應用程式的頻寬容量與封包速率效能。對於人工智慧 (AI)、機器學習 (ML) 與高效能運算 (HPC) 使用案例，這樣做尤其有用。

4. 如何使用此功能？

1. 首先，必須確認 Kubernetes 叢集使用 VPC CNI 1.20 版或更新版本。若要了解作為 EKS 附加元件來更新 VPC CNI 的步驟，請參閱 the section called “更新 (EKS 附加元件)”。
2. 接著，必須使用 `ENABLE_MULTI_NIC` 環境變數，在 VPC CNI 中啟用多重 NIC 支援。
3. 然後，必須確認建立並聯結擁有多個網路卡的節點。若要了解擁有多個網路卡的 EC2 執行個體類型清單，請參閱 Amazon EC2 使用者指南中的網路卡。
4. 最後，設定每個工作負載使用多個網路介面 (多重主目錄 Pod) 或單一網路介面。

5. 如何設定我的工作負載在支援的工作節點上使用多重 NIC？

如需使用多重主目錄 Pod，需要新增以下註解：`k8s.amazonaws.com/nicConfig: multi-nic-attachment`。這會連接底層執行個體中每個 NIC 的 ENI 至 Pod (Pod 與 NIC 間的一至多映射)。

若缺少此註解，VPC CNI 則會假設 Pod 僅需 1 個網路介面，並且在任何可用的 NIC 上將其指派給 ENI 中的 IP。

6. 這項功能支援哪些網路介面卡？

若您至少有一個 ENA 連接至底層網路卡用於處理 IP 流量，則可使用任何網路介面卡。若要了解詳細詳細，請參閱 Amazon EC2 使用者指南中的彈性網路介面卡 (ENA)。

受支援的網路裝置組態：

- ENA 介面提供支援 VPC IP 聯網所需的所有傳統 IP 聯網和路由功能。如需詳細資訊，請參閱在 EC2 執行個體上使用 ENA 啟用增強型聯網。
- EFA (EFA 搭配 ENA) 介面提供用於 IP 聯網的 ENA 裝置和用於低延遲、高輸送量通訊的 EFA 裝置。

> **Important**
>
> 若網路卡僅連接 EFA 限定介面卡，VPC CNI 會在佈建多重主目錄 Pod 的網路連線時將其略過。然而，若您結合使用 EFA 限定介面卡與網路卡上的 ENA 介面卡，則 VPC CNI 亦會管理此裝置上的 ENI。如需將 EFA 限定介面卡與 EKS 叢集配合使用，請參閱 the section called “使用 EFA 設定訓練叢集”。

7. 能否查看我的叢集中的節點是否支援 ENA？

是，您可以使用 AWS CLI 或 EC2 API 來擷取叢集中 EC2 執行個體的網路資訊。其中提供了執行個體是否具有 ENA 支援的相關詳細資訊。在以下範例中，使用節點的 EC2 執行個體 ID 來取代 `<your-instance-id>`。

AWS CLI 範例：

```
aws ec2 describe-instances --instance-ids <your-instance-id> --query
 "Reservations[].Instances[].EnaSupport"
```

輸出範例：

```
[ true ]
```

8. 能否查看與 Pod 關聯的不同 IP 位址？

否，不容易查看。但是，您可使用節點的 `nsenter` 來執行 `ip route show` 等常用網路工具，以及查看額外的 IP 位址與介面。

9. 能否控制 Pod 的網路介面數目？

否。設定工作負載在支援的執行個體上使用多重 NIC 後，單一 Pod 將自動擁有執行個體上每個網路卡的 IP 位址。或者，單一主目錄 Pod 將一個網路介面連接至執行個體上的一個 NIC。

> **Important**
>
> VPC CNI 會略過僅連接 EFA 限定裝置的網路卡。

10. 能否設定 Pod 使用特定 NIC？

否，不支援。若 Pod 擁有相關註解，VPC CNI 則自動將其設定為讓工作節點上的每一個 NIC 與 ENA 介面卡配合使用。

11. 這項功能是否適用於其他 VPC CNI 聯網功能？

是，VPC CNI 中的多重 NIC 功能適用於自訂聯網與增強型子網路探索。然而，多重主目錄 Pod 不使用自訂子網路或安全群組。取而代之的是，VPC CNI 會將 IP 位址與網路介面指派到多重主目錄 Pod，其具有與節點相同的子網路及安全群組組態。如需自訂聯網的詳細資訊，請參閱 the section called “自訂聯網”。

VPC CNI 中的多重 NIC 功能不適用於 Pod 的安全群組，亦不能與其結合使用。

12. 能否搭配此功能使用網路政策？

是，您可搭配多重 NIC 使用 Kubernetes 網路政策。Kubernetes 網路政策限制傳入與傳出 Pod 的網路流量。若要了解藉助 VPC CNI 套用網路政策的相關詳細資訊，請參閱 the section called “Kubernetes 政策”。

13. 在 EKS 自動模式中是否啟用多 NIC 支援？

EKS 自動模式叢集不支援多 NIC。

適用於 Amazon EKS 叢集的替代 CNI 外掛程式

Kubernetes 專用 Amazon VPC CNI 外掛程式是具有 Amazon EC2 節點的 Amazon EKS 唯一支援的 CNI 外掛程式。Amazon EKS 支援 Cilium 與 Calico 的核心功能，可用於 Amazon EKS 混合節點。Amazon EKS 會執行上游 Kubernetes，因此您可以將替代相容 CNI 外掛程式安裝到叢集中的 Amazon EC2 節點。如果您的叢集中有 Fargate 節點，則表示 Kubernetes 專用 Amazon VPC CNI 外掛程式已在 Fargate 節點上。這是唯一可以與 Fargate 節點搭配使用的 CNI 外掛程式。嘗試在 Fargate 節點上安裝替代 CNI 外掛程式失敗。

如果您打算在 Amazon EC2 節點中使用替代 CNI 外掛程式，那麼建議您取得外掛程式的商業支援，或擁有內部專業知識來排除問題，並為 CNI 外掛程式專案提供修正方法。

Amazon EKS 會持續與提供替代相容 CNI 外掛程式支援的合作夥伴網路建立關係。請參閱下列合作夥伴文件，獲取有關版本、資格和所執行測試的詳細資訊。

合作夥伴	產品	文件
Tigera	Calico	安裝說明
Isovalent	Cilium	安裝說明
Juniper	雲端原生 Contrail Networking (CN2)	安裝說明
VMware	Antrea	安裝說明

Amazon EKS 旨在為您提供各種選擇來涵蓋所有使用案例。

替代相容網路政策外掛程式

Calico 是廣泛採用的容器聯網與安全性解決方案。在 EKS 上使用 Calico 可為您的 EKS 叢集提供完全符合標準的網路政策強制執行。此外，您可以選擇使用 Calico 的聯網，這會保留基礎 VPC 的 IP 地址。Calico Cloud 增強了 Calico Open Source 的功能，提供進階的安全性與可觀測性能力。

具有關聯安全群組的 Pod 的流量不受 Calico 網路政策強制執行的約束，且僅限於 Amazon VPC 安全群組強制執行。

若您使用 Calico 網路政策強制執行，建議您將環境變數 ANNOTATE_POD_IP 設定為 `true`，以避免 Kubernetes 的已知問題。若要使用此功能，您必須將 Pod 的 `patch` 許可新增至 `aws-node`

ClusterRole。請注意，向 aws-node DaemonSet 新增修補程式許可會增加此外掛程式的安全範圍。如需詳細資訊，請參閱 GitHub 上 VPC CNI 儲存庫中的 ANNOTATE_POD_IP。

Amazon EKS 自動模式的考量事項

Amazon EKS 自動模式不支援替代的 CNI 外掛程式或網路政策外掛程式。如需詳細資訊，請參閱EKS 自動模式。

藉助 Multus 連接多個網路介面至 Pod

Multus CNI 是適用於 Amazon EKS 的容器網路介面 (CNI) 外掛程式，可將多個網路介面連接到 Pod。如需詳細資訊，請參閱 GitHub 上的 Multus-CNI文檔。

在 Amazon EKS 中，每個 Pod 都有一個由 Amazon VPC CNI 外掛程式指派的網路介面。使用 Multus，您可以建立具有多個介面的多重主目錄 Pod。這由做為「中繼外掛程式」的 Multus 完成；可呼叫多個其他 CNI 外掛程式的 CNI 外掛程式。Multus 的 AWS 支援使用 Amazon VPC CNI 外掛程式設定為預設委派外掛程式。

- Amazon EKS 將無法建置和發布單一根 I/O 虛擬化 (SR-IOV) 和資料平面開發套件 (DPDK) CNI 外掛程式。不過，您可以透過 Multus 受管主機裝置和 ipvlan 外掛程式直接連接至 Amazon EC2 彈性網路介面 (ENA)，進而達成封包加速。
- Amazon EKS 支援 Multus，它提供了一個通用的程序，可以簡單地鏈接其他 CNI 外掛程式。支援 Multus 和鏈結程序，但 AWS 不支援所有可鏈結的相容 CNI 外掛程式，或與鏈結組態無關的這些 CNI 外掛程式中可能出現的問題。
- Amazon EKS 為 Multus 外掛程式提供支援和生命週期管理，但不負責與其他網路介面相關聯的任何 IP 位址或其他管理。使用 Amazon VPC CNI 外掛程式之預設網路介面的 IP 地址和管理維持不變。
- mazon VPC CNI 外掛程式僅獲預設委派外掛程式的正式支援。如果您選擇不將 Amazon VPC CNI 外掛程式用於主要聯網，則需要修改已發佈的 Multus 安裝清單檔案，以將預設委派外掛程式重新設定為替代 CNI。
- 只有在使用 Amazon VPC CNI 作為主要 CNI 時，才支援 Multus。用於高階、次要或其他介面時，我們不支援 Amazon VPC CNI。
- 若要防止 Amazon VPC CNI 外掛程式嘗試管理指派給 Pod 的其他網路介面，請將下列標籤新增至網路介面：

  金鑰

  ```
  : node.k8s.amazonaws.com/no_manage
  ```

value

: `true`

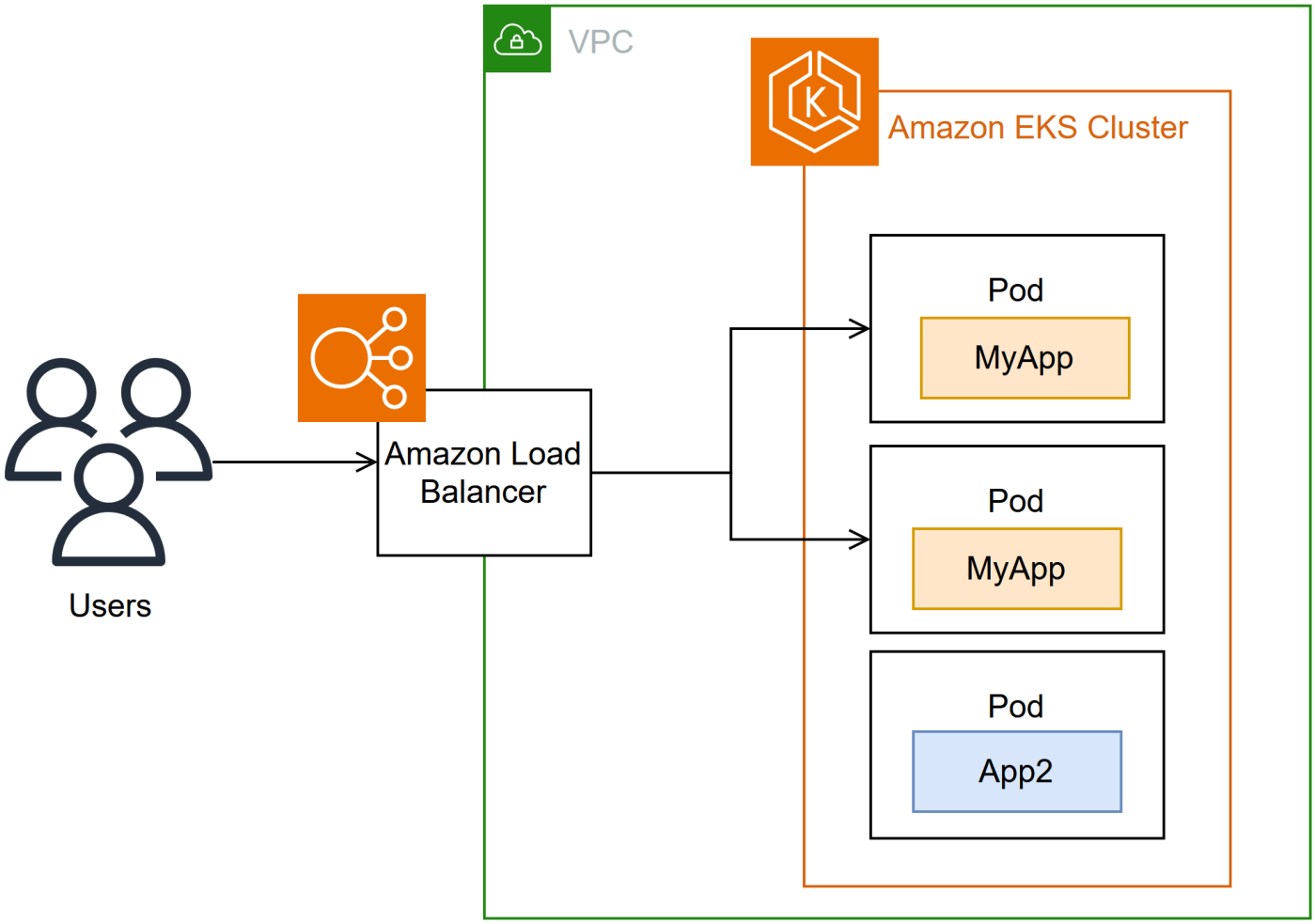
- Multus 與網路政策相容，但必須加強政策才能包含連接埠和 IP 位址，其中這些連接埠和 IP 位址可能是連接至 Pod 的其他網路介面的一部分。

如需實作演練，請參閱 GitHub 上的 Multus 設定指南。

使用 AWS Load Balancer控制器路由網際網路流量

> **Tip**
>
> 註冊即將舉行的 Amazon EKS 研討會。

The AWS Load Balancer 控制器會管理 Kubernetes 叢集的 AWS Elastic Load Balancer。您可透過此控制器將叢集應用程式開放至網際網路。控制器會佈建指向叢集服務或輸入資源的 AWS 負載平衡器。換言之，控制器會建立單一 IP 位址或 DNS 名稱，並將其指向叢集中的多個 Pod。



控制器會持續監控 Kubernetes Ingress 或 Service 資源。為了回應，它會建立適當的 AWS Elastic Load Balancing資源。您可透過在 Kubernetes 資源上套用註解，設定負載平衡器的特定行為。例如，您可以使用註釋將 AWS 安全群組連接至負載平衡器。

控制器會佈建以下資源：

Kubernetes **`Ingress`**

LBC 會在您建立 `Ingress` 時，建立 AWS Application Load Balancer (ALB)。請檢閱可套用至 Ingress 資源的註釋。

類型為 **`LoadBalancer`** 的 Kubernetes 服務

當您建立類型為 的 Kubernetes 服務時，LBC 會建立 AWS Network Load Balancer (NLB)`LoadBalancer`。請檢閱可套用至 Service 資源的註釋。

在過去，Kubernetes Network Load Balancer 用於執行個體目標，但 LBC 用於 IP 目標。使用 AWS Load Balancer控制器版本 `2.3.0` 或更新版本，您可以使用任一目標類型建立 NLBs。如需 NLB 目標類型的詳細資訊，請參閱《Network Load Balancer 使用者指南》中的目標類型。

此控制器是託管於 GitHub 上的 開源專案。

在部署控制器之前，我們建議您先檢閱透過 Application Load Balancer 路由應用程式與 HTTP 流量與 the section called “網路負載平衡” 中的先決條件和考量事項。在這些主題中，您將部署包含 AWS 負載平衡器的範例應用程式。

Kubernetes **`Gateway`** API

使用 AWS Load Balancer控制器版本 `2.14.0`或更新版本，LBC 會在您建立 Kubernetes 時建立 AWS Application Load Balancer (ALB)`Gateway`。Kubernetes Gateway 會將比輸入更多的組態標準化，這需要許多常見選項的自訂註釋。檢閱您可以套用至閘道資源的組態。 如需 `Gateway` API 的詳細資訊，請參閱 Kubernetes 文件中的閘道 API。

安裝控制器

您可以使用下列其中一個程序來安裝 AWS Load Balancer控制器：

- 如果您是初次使用 Amazon EKS，我們建議您使用 Helm 進行安裝，因為它可簡化 AWS Load Balancer控制器安裝。如需詳細資訊，請參閱the section called “使用 Helm 安裝”。
- 若需進階設定，如對公有容器登錄庫的網路存取受限的叢集，請使用 Kubernetes 資訊清單。如需詳細資訊，請參閱the section called “使用資訊清單安裝”。

從已淘汰的控制器版本移轉

- 如果您已安裝已棄用版本的 AWS Load Balancer控制器，請參閱 the section called “從已棄用的移轉”。
- 已棄用的版本無法升級。必須移除它們，並安裝目前版本的 AWS Load Balancer控制器。
- 已棄用的版本包含：
  - AWS 適用於 Kubernetes 的 ALB 輸入控制器 (「輸入控制器」)，是 AWS Load Balancer控制器的前身。
  - 任何`0.1.`*`x`* 版本的 AWS Load Balancer控制器

舊版雲端供應商

Kubernetes 包含 AWS的舊版雲端供應商。舊版雲端提供者能夠佈建 AWS 負載平衡器，類似於 AWS Load Balancer控制器。舊版雲端供應商建立的是 Classic Load Balancer。如果您未安裝 AWS Load Balancer控制器，Kubernetes 會預設為使用舊版雲端提供者。您應該安裝 AWS Load Balancer控制器，並避免使用舊版雲端提供者。

> **Important**
>
> 在 2.5 版和更新版本中， AWS Load Balancer控制器會使用 成為 Kubernetes 服務資源的預設控制器，`type: LoadBalancer`並為每個服務建立 AWS Network Load Balancer (NLB)。它透過為服務製作變異的 webhook 來做到這一點，此 webhook 會為 `type: LoadBalancer` 的新服務將 `spec.loadBalancerClass` 欄位設定至 `service.k8s.aws/nlb`。您可以關閉此功能並恢復為使用舊式雲端供應商作為預設控制器，方法是將 Helm Chart 值 `enableServiceMutatorWebhook` 設定為 `false`。除非您關閉此功能，否則叢集不會為您的服務佈建新的 Classic Load Balancer。現有的 Classic Load Balancer 會繼續運作。

搭配 Helm 的 Install AWS Load Balancer 控制器

> **Tip**
>
> 註冊即將舉行的 Amazon EKS 研討會。

> **Tip**
>
> 使用 Amazon EKS 自動模式，您無需安裝或升級聯網附加元件。自動模式包括 Pod 聯網和負載平衡功能。
> 如需詳細資訊，請參閱EKS 自動模式。

本主題說明如何使用 Helm、Kubernetes 套件管理員 和 安裝 AWS Load Balancer控制器`eksctl`。已使用預設選項安裝控制器。如需有關控制器的詳細資訊，包括使用註釋設定控制器的詳細資訊，請參閱 GitHub 上的AWS Load Balancer 控制器文件。

在下列步驟中，使用您自己的值取代範例值。

先決條件

開始本教學之前，您必須完成下列步驟：

- 建立 Amazon EKS 叢集。若要建立服務角色，請參閱開始使用。
- 在 Amazon EKS 叢集上安裝閘道 API CRDs。
- 在您的本機電腦上安裝 Helm。
- 請確保您的 Kubernetes 專用 Amazon VPC CNI 外掛程式、`kube-proxy` 及 CoreDNS 附加元件為服務帳戶字符中列出的最低版本。
- 了解 AWS Elastic Load Balancing概念。如需詳細資訊，請參閱《Elastic Load Balancing 使用者指南》。
- 了解 Kubernetes 服務和傳入資源。

考量事項

在繼續此頁面上的組態步驟之前，請考量下列情況：

- IAM 政策和角色 (`AmazonEKSLoadBalancerControllerRole`) 可以在相同 AWS 帳戶中的多個 EKS 叢集之間重複使用。
- 如果您要在最初建立角色 (`AmazonEKSLoadBalancerControllerRole`) 的相同叢集上安裝控制器，請在驗證角色存在後，前往步驟 2：安裝 Load Balancer 控制器。
- 如果您使用服務帳戶的 IAM 角色 (IRSA)，則必須為每個叢集設定 IRSA，而且角色信任政策中的 OpenID Connect (OIDC) 提供商 ARN 專屬於每個 EKS 叢集。此外，如果您要在具有現有 `AmazonEKSLoadBalancerControllerRole` 的新叢集上安裝控制器，請更新角色的信任政策，以包含新叢集的 OIDC 提供商，並使用適當的角色註釋來建立新的服務帳戶。若要判定您是否已經擁有 OIDC 提供商，或是要建立一個，則請參閱 the section called “IAM OIDC 提供商”。

步驟 1：使用 **`eksctl`** 建立 IAM 角色

下列步驟參考 AWS Load Balancer控制器 v2.14.1 發行版本。如需有關所有發行版本的詳細資訊，請參閱 GitHub 上的 AWS Load Balancer 控制器發行頁面。

1. 下載 AWS Load Balancer控制器的 IAM 政策，允許其代表您呼叫 AWS APIs。

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-
controller/v2.14.1/docs/install/iam_policy.json
```

- 如果您位於非標準 AWS 分割區，例如政府或中國區域，請檢閱 GitHub 上的政策，並下載適用於您區域的適當政策。

2. 使用上一個步驟中下載的政策，建立 IAM 政策。

```
aws iam create-policy \
    --policy-name AWSLoadBalancerControllerIAMPolicy \
    --policy-document file://iam_policy.json
```

> **Note**
>
> 如果您在 中檢視政策 AWS 管理主控台，主控台會顯示 ELB 服務的警告，但不會顯示 ELB v2 服務的警告。發生這種情況是由於政策中存在適用於 ELB v2 的某些動作，但不適用於 ELB。您可以忽略這些對 ELB 發出的警告。

3. 取代叢集名稱、區域代碼和帳戶 ID 的值。

```
eksctl create iamserviceaccount \
    --cluster=<cluster-name> \
    --namespace=kube-system \
    --name=aws-load-balancer-controller \
    --attach-policy-arn=arn:aws: iam::<AWS_ACCOUNT_ID>:policy/
AWSLoadBalancerControllerIAMPolicy \
    --override-existing-serviceaccounts \
    --region <aws-region-code> \
    --approve
```

步驟 2：Install AWS Load Balancer 控制器

1. 新增 `eks-charts` Helm Chart 儲存庫。 會在 GitHub 上 AWS 維護此儲存庫。

```
helm repo add eks https://aws.github.io/eks-charts
```

2. 更新您的本機儲存庫，以確定您擁有最新的圖表。

```
helm repo update eks
```

3. 安裝 AWS Load Balancer控制器。

如果您要將控制器部署到限制存取 Amazon EC2 執行個體中繼資料服務 (IMDS) 的 Amazon EC2 節點，或者如果您正在部署到 Fargate 或 Amazon EKS 混合節點，則請將以下旗標新增到以下 helm 命令中：

- --set region=*region-code*
- --set vpcId=*vpc-xxxxxxxx*

使用您叢集的名稱取代 *my-cluster*。在下列命令中，aws-load-balancer-controller 是您在上一個步驟中建立的 Kubernetes 服務帳戶。

如需設定 Helm Chart 的詳細資訊，請參閱 GitHub 上的 values.yaml。

```
helm install aws-load-balancer-controller eks/aws-load-balancer-controller \
  -n kube-system \
  --set clusterName=my-cluster \
  --set serviceAccount.create=false \
  --set serviceAccount.name=aws-load-balancer-controller \
  --version 1.14.0
```

> **Important**
>
> 部署的圖表不會自動接收安全更新。當圖表可用時，您需要手動升級到較新的圖表。升級時，請將先前命令中的##變更為 upgrade。

helm install 命令會自動安裝控制器的自訂資源定義 (CRD)。helm upgrade 命令則不會。如果您使用 helm upgrade,，則必須手動安裝 CRD。執行下列命令以安裝 CRD：

```
wget https://raw.githubusercontent.com/aws/eks-charts/master/stable/aws-load-balancer-
controller/crds/crds.yaml
kubectl apply -f crds.yaml
```

步驟 3：確認控制器已安裝

1. 確認控制器已安裝。

```
kubectl get deployment -n kube-system aws-load-balancer-controller
```

範例輸出如下。

```
NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
aws-load-balancer-controller   2/2     2            2           84s
```

如果使用 Helm 進行部署，則會收到先前的輸出。如果使用 Kubernetes 清單檔案進行部署，則您只有一個複本。

2. 使用控制器佈建 AWS 資源之前，您的叢集必須符合特定需求。如需詳細資訊，請參閱the section called “應用程式負載平衡”及the section called “網路負載平衡”。

具有資訊清單的 Install AWS Load Balancer 控制器

> **Tip**
>
> 使用 Amazon EKS 自動模式，您無需安裝或升級聯網附加元件。自動模式包括 Pod 聯網和負載平衡功能。
> 如需詳細資訊，請參閱EKS 自動模式。

本主題會說明如何透過下載和套用 Kubernetes 資訊清單來安裝控制器。您可以檢視 GitHub 上的完整控制器文件。

在下列步驟中，使用您自己的值取代範例值。

先決條件

開始本教學之前，您必須完成下列步驟：

- 建立 Amazon EKS 叢集。若要建立服務角色，請參閱開始使用。
- 在您的本機電腦上安裝 Helm。
- 請確保您的 Kubernetes 專用 Amazon VPC CNI 外掛程式、`kube-proxy` 及 CoreDNS 附加元件為服務帳戶字符中列出的最低版本。
- 了解 AWS Elastic Load Balancing概念。如需詳細資訊，請參閱《Elastic Load Balancing 使用者指南》。
- 了解 Kubernetes 服務和傳入資源。

考量事項

在繼續此頁面上的組態步驟之前，請考量下列情況：

- IAM 政策和角色 (AmazonEKSLoadBalancerControllerRole) 可以在相同 AWS 帳戶中的多個 EKS 叢集之間重複使用。
- 如果您要在最初建立角色 (AmazonEKSLoadBalancerControllerRole) 的相同叢集上安裝控制器，請在驗證角色存在後，前往步驟 2：安裝 cert-manager。
- 如果您使用服務帳戶的 IAM 角色 (IRSA)，則必須為每個叢集設定 IRSA，而且角色信任政策中的 OpenID Connect (OIDC) 提供商 ARN 專屬於每個 EKS 叢集。此外，如果您要在具有現有 AmazonEKSLoadBalancerControllerRole 的新叢集上安裝控制器，請更新角色的信任政策，以包含新叢集的 OIDC 提供商，並使用適當的角色註釋來建立新的服務帳戶。若要判定您是否已經擁有 OIDC 提供商，或是要建立一個，則請參閱 the section called “IAM OIDC 提供商”。

步驟 1：設定 IAM

下列步驟參考 AWS Load Balancer控制器 v2.14.1 發行版本。如需有關所有發行版本的詳細資訊，請參閱 GitHub 上的 AWS Load Balancer 控制器發行頁面。

1. 下載 AWS Load Balancer控制器的 IAM 政策，允許其代表您呼叫 AWS APIs。

   Example

   AWS

   ```
   curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.14.1/docs/install/iam_policy.json
   ```

   AWS GovCloud (US)

   ```
   curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.14.1/docs/install/iam_policy_us-gov.json
   ```

   ```
   mv iam_policy_us-gov.json iam_policy.json
   ```

2. 使用上一個步驟中下載的政策，建立 IAM 政策。

   ```
   aws iam create-policy \
       --policy-name AWSLoadBalancerControllerIAMPolicy \
   ```

```
    --policy-document file://iam_policy.json
```

> **Note**
>
> 如果您在 中檢視政策 AWS 管理主控台，主控台會顯示 ELB 服務的警告，但不會顯示 ELB v2 服務的警告。發生這種情況是由於政策中存在適用於 ELB v2 的某些動作，但不適用於 ELB。您可以忽略這些對 ELB 發出的警告。

Example

eksctl

a. 使用叢集名稱取代 *my-cluster*，並使用帳戶 ID 取代 *111122223333*，然後執行命令。

```
eksctl create iamserviceaccount \
  --cluster=my-cluster \
  --namespace=kube-system \
  --name=aws-load-balancer-controller \
  --role-name AmazonEKSLoadBalancerControllerRole \
  --attach-policy-arn=arn:aws: iam::111122223333:policy/
AWSLoadBalancerControllerIAMPolicy \
  --approve
```

AWS CLI and kubectl

a. 擷取叢集的 OIDC 提供商 ID 並將其存放在變數中。

```
oidc_id=$(aws eks describe-cluster --name my-cluster --query
 "cluster.identity.oidc.issuer" --output text | cut -d '/' -f 5)
```

b. 判斷您的帳戶中是否已經有擁有叢集 ID 的 IAM OIDC 提供商。您需要為叢集和 IAM 設定 OIDC。

```
aws iam list-open-id-connect-providers | grep $oidc_id | cut -d "/" -f4
```

如果有輸出傳回，則表示叢集已經擁有 IAM OIDC 提供商。如果未傳回任何輸出，則您必須為叢集建立 IAM OIDC 提供商。如需詳細資訊，請參閱the section called “IAM OIDC 提供商”。

c. 將以下內容複製到您的裝置。使用您的帳戶 ID 取代 *111122223333*。將 *region-code* 取代為您的叢集所在的 AWS 區域。使用前一個步驟傳回的輸出值取代 *EXAMPLED539D4633E53DE1B71EXAMPLE*。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::111122223333:oidc-provider/oidc.eks.us-
east-1.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "oidc.eks.us-east-1.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:aud": "sts.amazonaws.com",
                    "oidc.eks.us-east-1.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:sub": "system:serviceaccount:kube-system:aws-
load-balancer-controller"
                }
            }
        }
    ]
}
```

d. 建立 IAM 角色。

```
aws iam create-role \
  --role-name AmazonEKSLoadBalancerControllerRole \
  --assume-role-policy-document file://"load-balancer-role-trust-policy.json"
```

e. 將必要的 Amazon EKS 受管 IAM 政策連接到 IAM 角色。使用您的帳戶 ID 取代 *111122223333*。

```
aws iam attach-role-policy \
  --policy-arn arn:aws: iam::111122223333:policy/
AWSLoadBalancerControllerIAMPolicy \
  --role-name AmazonEKSLoadBalancerControllerRole
```

f. 將以下內容複製到您的裝置。使用您的帳戶 ID 取代 *111122223333*。取代文字後，請執行已修改的命令以建立 aws-load-balancer-controller-service-account.yaml 檔案。

```
cat >aws-load-balancer-controller-service-account.yaml <<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    app.kubernetes.io/component: controller
    app.kubernetes.io/name: aws-load-balancer-controller
  name: aws-load-balancer-controller
  namespace: kube-system
  annotations:
    eks.amazonaws.com/role-arn: arn:aws: iam::111122223333:role/
AmazonEKSLoadBalancerControllerRole
EOF
```

g. 在您的叢集上建立 Kubernetes 服務帳戶。名為 aws-load-balancer-controller 的 Kubernetes 服務帳戶使用您建立名為 *AmazonEKSLoadBalancerControllerRole* 的 IAM 角色進行註釋。

```
kubectl apply -f aws-load-balancer-controller-service-account.yaml
```

步驟 2：安裝 **cert-manager**

使用以下其中一個方法安裝 cert-manager，將憑證組態注入 Webhook。如需詳細資訊，請參閱 cert-manager 文件中的入門。

建議使用 quay.io 容器登錄檔來安裝 cert-manager。如果您的節點無法存取 quay.io 容器登錄檔，則請使用 Amazon ECR 安裝 cert-manager (請參閱下文)。

Example

Quay.io

a. 如果您的節點可以存取 quay.io 容器登錄檔，請安裝 cert-manager 以將憑證組態注入 Webhook。

```
kubectl apply \
    --validate=false \
```

```
    -f https://github.com/jetstack/cert-manager/releases/download/v1.13.5/cert-
manager.yaml
```

Amazon ECR

a. 使用以下其中一個方法安裝 `cert-manager`，將憑證組態注入 Webhook。如需詳細資訊，請參閱 cert-manager 文件中的入門。

b. 下載清單檔案。

```
curl -Lo cert-manager.yaml https://github.com/jetstack/cert-manager/releases/
download/v1.13.5/cert-manager.yaml
```

c. 提取以下映像並將其推送到節點可存取的儲存庫。有關如何提取、標記映像並將其推送到您儲存庫的詳細資訊，請參閱 the section called “將映像複製到儲存庫”。

```
quay.io/jetstack/cert-manager-cainjector:v1.13.5
quay.io/jetstack/cert-manager-controller:v1.13.5
quay.io/jetstack/cert-manager-webhook:v1.13.5
```

d. 使用您的登錄檔名稱取代清單檔案中三個映像的 `quay.io`。下列命令假定私有儲存庫的名稱與來源儲存庫的名稱相同。使用私有登錄檔取代 *111122223333.dkr.ecr.region-code.amazonaws.com*。

```
sed -i.bak -e 's|quay.io|111122223333.dkr.ecr.region-code.amazonaws.com|' ./cert-
manager.yaml
```

e. 套用清單檔案。

```
kubectl apply \
    --validate=false \
    -f ./cert-manager.yaml
```

步驟 3：Install AWS Load Balancer 控制器

1. 下載控制器規格。如需控制器的詳細資訊，請參閱 GitHub 上的文件。

```
curl -Lo v2_14_1_full.yaml https://github.com/kubernetes-sigs/aws-load-balancer-
controller/releases/download/v2.14.1/v2_14_1_full.yaml
```

2. 對檔案進行以下編輯。

   a. 如果您已下載 v2_14_1_full.yaml 檔案，請執行下列命令以移除清單檔案中的 ServiceAccount 區段。如果您未移除此區段，則會覆寫您在前一個步驟中對服務帳戶所做的必要註解。如果您刪除控制器，移除此區段還可保留您在上一個步驟中建立的服務帳戶。

   ```
   sed -i.bak -e '764,772d' ./v2_14_1_full.yaml
   ```

   如果您下載不同的檔案版本，請在編輯器中開啟檔案並移除下列各行。

   ```
   apiVersion: v1
   kind: ServiceAccount
   metadata:
     labels:
       app.kubernetes.io/component: controller
       app.kubernetes.io/name: aws-load-balancer-controller
     name: aws-load-balancer-controller
     namespace: kube-system
   ---
   ```

   b. 使用叢集名稱取代 *my-cluster*，從而使用叢集名稱取代檔案的 Deployment spec 區段中的 your-cluster-name。

   ```
   sed -i.bak -e 's|your-cluster-name|my-cluster|' ./v2_14_1_full.yaml
   ```

   c. 如果您的節點無法存取 Amazon EKS Amazon ECR 映像儲存庫，則需要提取下列映像並將其推送到節點可存取的儲存庫。有關如何提取、標記映像並將其推送到您儲存庫的詳細資訊，請參閱 the section called “將映像複製到儲存庫”。

   ```
   public.ecr.aws/eks/aws-load-balancer-controller:v2.14.1
   ```

   將您的登錄檔名稱新增至資訊清單。下列命令假定私有儲存庫的名稱與來源儲存庫的名稱相同，並將您的私有登錄檔名稱新增至檔案中。使用您的登錄檔取代 *111122223333.dkr.ecr.region-code.amazonaws.com*。此行假定您將私有儲存庫命名為與來源儲存庫相同的名稱。如果沒有，請將私有登錄檔名稱後的 eks/aws-load-balancer-controller 文字變更為儲存庫名稱。

   ```
   sed -i.bak -e 's|public.ecr.aws/eks/aws-load-balancer-controller|
   111122223333.dkr.ecr.region-code.amazonaws.com/eks/aws-load-balancer-
   controller|' ./v2_14_1_full.yaml
   ```

d. (僅 Fargate 或受限制的 IMDS 需要)

如果您要將控制器部署到限制存取 Amazon EC2 執行個體中繼資料服務 (IMDS) 的 Amazon EC2 節點，或者如果您正在部署到 Fargate 或 Amazon EKS 混合節點，那麼請在 - args: 下新增 following parameters。

```
[...]
spec:
      containers:
        - args:
            - --cluster-name=your-cluster-name
            - --ingress-class=alb
            - --aws-vpc-id=vpc-xxxxxxxx
            - --aws-region=region-code


[...]
```

3. 套用檔案。

```
kubectl apply -f v2_14_1_full.yaml
```

4. 將 IngressClass 和 IngressClassParams 清單檔案下載至叢集。

```
curl -Lo v2.14.1_ingclass.yaml https://github.com/kubernetes-sigs/aws-load-balancer-controller/releases/download/v2.14.1/v2_14_1_ingclass.yaml
```

5. 將清單檔案套用至叢集。

```
kubectl apply -f v2.14.1_ingclass.yaml
```

步驟 4：確認控制器已安裝

1. 確認控制器已安裝。

```
kubectl get deployment -n kube-system aws-load-balancer-controller
```

範例輸出如下。

```
NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
```

```
aws-load-balancer-controller   2/2     2            2           84s
```

如果使用 Helm 進行部署，則會收到先前的輸出。如果使用 Kubernetes 清單檔案進行部署，則您只有一個複本。

2. 使用控制器佈建 AWS 資源之前，您的叢集必須符合特定需求。如需詳細資訊，請參閱the section called “應用程式負載平衡”及the section called “網路負載平衡”。

從已棄用的 ALB 傳入控制器移轉應用程式

本主題會說明如何從已棄用的控制器版本移轉。更具體地說，它描述了如何移除已棄用版本的 AWS Load Balancer控制器。

- 已棄用的版本無法升級。您必須先將其移除，然後再安裝目前版本。
- 已棄用的版本包含：
  - AWS 適用於 Kubernetes 的 ALB 傳入控制器（「傳入控制器」），是 AWS Load Balancer控制器的前身。
  - 任何`0.1.x`版本的 AWS Load Balancer控制器

移除已棄用的控制器版本

> **Note**
>
> 您可能已經使用 Helm 或手動使用 Kubernetes 資訊清單安裝已棄用的版本。請使用最初使用安裝的工具來完成此程序。

1. 如果您安裝了 `incubator/aws-alb-ingress-controller` Helm Chart，請將其解除安裝。

```
helm delete aws-alb-ingress-controller -n kube-system
```

2. 如果您安裝了第 `0.1.x` 版的 `eks-charts/aws-load-balancer-controller` 圖表，請將其解除安裝。由於與 Webhook API 版本不相容，無法從 `0.1.x` 升級至版本 `1.0.0`。

```
helm delete aws-load-balancer-controller -n kube-system
```

3. 檢查目前是否已安裝控制器。

```
kubectl get deployment -n kube-system alb-ingress-controller
```

如果未安裝控制器，則此為輸出。

```
Error from server (NotFound): deployments.apps "alb-ingress-controller" not found
```

如果已安裝控制器，則此為輸出。

```
NAME                      READY UP-TO-DATE AVAILABLE AGE
alb-ingress-controller 1/1   1          1         122d
```

4. 輸入下列命令以移除控制器。

```
kubectl delete -f https://raw.githubusercontent.com/kubernetes-sigs/aws-alb-ingress-
controller/v1.1.8/docs/examples/alb-ingress-controller.yaml
kubectl delete -f https://raw.githubusercontent.com/kubernetes-sigs/aws-alb-ingress-
controller/v1.1.8/docs/examples/rbac-role.yaml
```

遷移至 AWS Load Balancer控制器

若要從 Kubernetes 的 ALB 傳入控制器遷移至 AWS Load Balancer控制器，您需要：

1. 移除 ALB 傳入控制器 (請參閱上文)。
2. 安裝 AWS Load Balancer控制器。
3. 將其他政策新增至 AWS Load Balancer控制器所使用的 IAM 角色。此政策允許 LBC 管理適用於 Kubernetes 的 ALB 傳入控制器建立的資源。
4. 下載 IAM 政策。此政策允許 AWS Load Balancer控制器管理由適用於 Kubernetes 的 ALB 傳入控制器建立的資源。您也可以檢視政策。

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-
controller/v2.14.1/docs/install/iam_policy_v1_to_v2_additional.json
```

5. 如果您的叢集位於 AWS GovCloud （美國東部） 或 AWS GovCloud （美國西部） AWS 區域，請將 取代 `arn:aws:` 為 `arn:aws-us-gov:`。

```
sed -i.bak -e 's|arn:aws:|arn:aws-us-gov:|' iam_policy_v1_to_v2_additional.json
```

6. 建立 IAM 政策並記下傳回的 ARN。

```
aws iam create-policy \
  --policy-name AWSLoadBalancerControllerAdditionalIAMPolicy \
  --policy-document file://iam_policy_v1_to_v2_additional.json
```

7. 將 IAM 政策連接至 AWS Load Balancer控制器所使用的 IAM 角色。將 *your-role-name* 取代為角色的名稱，例如 AmazonEKSLoadBalancerControllerRole。

   如果使用 eksctl 建立角色，然後要尋找已建立的角色名稱，請開啟 AWS CloudFormation 主控台，然後選取 eksctl-***my-cluster***-addon-iamserviceaccount-kube-system-aws-load-balancer-controller 堆疊。選取 Resources (資源) 標籤。角色名稱位於 Physical ID (實體 ID) 欄。

```
aws iam attach-role-policy \
  --role-name your-role-name \
  --policy-arn arn:aws: iam::111122223333:policy/
AWSLoadBalancerControllerAdditionalIAMPolicy
```

管理 Amazon EKS 叢集中的 CoreDNS for DNS

> **Tip**
>
> 使用 Amazon EKS 自動模式，您無需安裝或升級聯網附加元件。自動模式包括 Pod 聯網和負載平衡功能。
> 如需詳細資訊，請參閱EKS 自動模式。

CoreDNS 是一個靈活、可擴展的 DNS 伺服器，可作為 Kubernetes 集群 DNS。當您啟動具有至少一個節點的 Amazon EKS 叢集時，依預設會部署兩個 CoreDNS 映像複本，而不論叢集中部署的節點數量為何。CoreDNS Pod 為叢集中的所有 Pod 提供名稱解析。如果您的叢集包含 Fargate Profile，且該設定檔的命名空間與 CoreDNS deployment 的命名空間相符，則可將 CoreDNS Pod 部署至 Fargate 節點。如需 CoreDNS 的詳細資訊，請參閱 Kubernetes 文件中的使用 CoreDNS 進行服務探索。

CoreDNS 版本

下表列出適用於每個 Kubernetes 版本的 Amazon EKS 附加元件類型的最新版本。

Kubernetes 版本	CoreDNS 版本
1.36	v1.14.3-eksbuild.16
1.35	v1.14.3-eksbuild.16
1.34	v1.13.2-eksbuild.24
1.33	v1.12.4-eksbuild.31
1.32	v1.11.4-eksbuild.53
1.31	v1.11.4-eksbuild.53

Important

如果您要自行管理此附加元件，資料表中的版本可能與可用的自我管理版本不同。如需更新此附加元件之自我管理類型的詳細資訊，請參閱 the section called “更新 (自我管理)”。

重要的 CoreDNS 升級考量

- CoreDNS 更新會利用 PodDisruptionBudget，以協助在更新程序期間維持 DNS 服務可用性。
- 為了提高 CoreDNS 部署的穩定性與可用性，版本 `v1.9.3-eksbuild.6` 及更新版本和 `v1.10.1-eksbuild.3` 會隨 `PodDisruptionBudget` 一起部署。如果您已部署現有 `PodDisruptionBudget`，則可能無法升級至上述版本。如果升級失敗，完成下列其中一項任務即可解決問題：
  - 升級 Amazon EKS 附加元件時，選擇將覆寫現有設定作為衝突解決方案選項。如果您已對部署制定自訂設定，則務必在升級前備份設定，以便在升級後重新套用其他自訂設定。
  - 移除現有的 `PodDisruptionBudget`，然後再次嘗試升級。
- 在 EKS 附加元件版本 `v1.9.3-eksbuild.3` 和更新版本以及 `v1.10.1-eksbuild.6` 和更新版本中，CoreDNS 部署將 `readinessProbe` 設定為使用 `/ready` 端點。此端點已在的 CoreDNS 的 `Corefile` 組態檔案中啟用。

  如果您使用自訂 `Corefile`，則必須將 `ready` 外掛程式新增至組態，以便在 CoreDNS 中使用 `/ready` 端點才能使用探查。

- 在 EKS 附加元件版本以v1.9.3-eksbuild.7及更新版本v1.10.1-eksbuild.4和更新版本中，您可以變更 PodDisruptionBudget。您可以使用下列範例的欄位，在選擇性組態設定中編輯附加元件並變更這些設定。此範例顯示預設值PodDisruptionBudget。

```
{
    "podDisruptionBudget": {
        "enabled": true,
        "maxUnavailable": 1
        }
}
```

  您可以設定 maxUnavailable 或 minAvailable，但不能同時在單一 PodDisruptionBudget 進行設定。如需有關 PodDisruptionBudgets 的詳細資訊，請參閱 Kubernetes 文件中的指定 PodDisruptionBudget。

  請注意，如果您將 enabled 設定為 false，則 PodDisruptionBudget 不會移除。將此欄位設定為false之後，您必須刪除PodDisruptionBudget物件。同樣地，如果您在升級至具有 PodDisruptionBudget 的版本之後編輯附加元件以使用較舊版本的附加元件 (降級附加元件)，則 PodDisruptionBudget 不會移除。執行下列命令以刪除 PodDisruptionBudget：

```
kubectl delete poddisruptionbudget coredns -n kube-system
```

- 在 EKS 附加元件版本 v1.10.1-eksbuild.5 和更新版本中，將預設容差從 node-role.kubernetes.io/master:NoSchedule 變更為 node-role.kubernetes.io/control-plane:NoSchedule，以符合 KEP 2067 的規定。如需有關 KEP 2067 的詳細資訊，請參閱 GitHub 上 Kubernetes Enhancement Proposals (KEP) 中的 KEP-2067：重新命名 kubeadm「主要」標籤和污點。

  在 EKS 附加元件版本 v1.8.7-eksbuild.8 和更新版本以及 v1.9.3-eksbuild.9 和更新版本中，兩個容差均設定為與每個 Kubernetes 版本相容。

- 在 EKS 附加元件版本 v1.9.3-eksbuild.11 和 v1.10.1-eksbuild.7 及更新版本中，CoreDNS 部署會設定 topologySpreadConstraints 的預設值。如果多個可用區域中有可用的節點，預設值可確保 CoreDNS Pod 分散到可用區域。您可以設定自訂值，並使用該值取代預設值。預設值如下：

```
topologySpreadConstraints:
  - maxSkew: 1
    topologyKey: topology.kubernetes.io/zone
    whenUnsatisfiable: ScheduleAnyway
```

```
    labelSelector:
      matchLabels:
        k8s-app: kube-dns
```

- 自訂 Corefile 組態和附加元件更新 — Amazon EKS 附加元件可完整管理 CoreDNS ConfigMap (`data.Corefile`)。如果您`Corefile`直接修改（例如，為了新增現場部署 DNS 解析的條件式轉送器），附加元件會在下次更新時覆寫您的變更。若要跨附加元件更新保留自訂`Corefile`組態，請在附加元件的 `configurationValues` 參數中使用 `corefile`金鑰。如需詳細資訊，請參閱the section called “您可自訂的欄位”。

CoreDNS **v1.11** 升級考量

- 在 EKS 附加元件版本 `v1.11.1-eksbuild.4` 和更新版本中，容器映像的基礎為 Amazon EKS Distro 維護的最低基礎映像，其中包含最少套件且沒有 Shell。如需詳細資訊，請參閱 Amazon EKS Distro。CoreDNS 映像的用量和故障診斷保持不變。

建立 CoreDNS Amazon EKS 附加元件

建立 CoreDNS Amazon EKS 附加元件。您必須先擁有叢集，方能建立附加元件。如需詳細資訊，請參閱 the section called “建立 叢集”。

1. 查看叢集上目前安裝了哪些附加元件版本。

   ```
   kubectl describe deployment coredns --namespace kube-system | grep coredns: | cut -
   d : -f 3
   ```

   範例輸出如下。

   ```
   v1.10.1-eksbuild.13
   ```

2. 查看叢集上安裝的附加元件類型。視您用來建立叢集的工具而定，您的叢集上目前可能沒有安裝 Amazon EKS 附加元件類型。使用您叢集的名稱取代 *my-cluster*。

   ```
   aws eks describe-addon --cluster-name my-cluster --addon-name coredns --query
    addon.addonVersion --output text
   ```

   如果傳回版本編號，則表明已在叢集上安裝 Amazon EKS 類型的附加元件，並且無需完成此程序中的剩餘步驟。如果傳回錯誤，則表明沒有在叢集上安裝 Amazon EKS 類型的附加元件。完成此程序的剩餘步驟以安裝該類型。

3. 儲存您目前安裝的附加元件。

```
kubectl get deployment coredns -n kube-system -o yaml > aws-k8s-coredns-old.yaml
```

4. 使用 AWS CLI 建立附加元件。如果您想要使用 AWS 管理主控台 或 eksctl 建立附加元件，請參閱 the section called “建立附加元件” 並指定 coredns 作為附加元件名稱。將隨後的命令複製到您的裝置。視需要對命令進行下列修改，然後執行修改後的命令。
   - 使用您叢集的名稱取代 *my-cluster*。
   - 將 *v1.11.3-eksbuild.1* 取代為叢集版本的最新版本資料表中列出的最新版本。

   ```
   aws eks create-addon --cluster-name my-cluster --addon-name coredns --addon-version
    v1.11.3-eksbuild.1
   ```

   如果您已將自訂設定套用至與 Amazon EKS 附加元件的預設設定衝突的目前附加元件，建立動作可能會失敗。若建立失敗，您會收到錯誤，其中的訊息有助於您解決問題。或者，您可以將 --resolve-conflicts OVERWRITE 新增至上一條命令。這可讓附加元件覆寫任何現有的自訂設定。建立附加元件後，可使用自訂設定來更新該附加元件。

5. 確認叢集 Kubernetes 版本的附加元件的最新版本已新增至叢集。使用您叢集的名稱取代 *my-cluster*。

   ```
   aws eks describe-addon --cluster-name my-cluster --addon-name coredns --query
    addon.addonVersion --output text
   ```

   建立附加元件的動作可能需要幾秒鐘的時間才能完成。

   範例輸出如下。

   ```
   v1.11.3-eksbuild.1
   ```

6. 如果您對原始附加元件制定自訂設定，請在建立 Amazon EKS 附加元件之前，使用在上一步中儲存的組態，以您的自訂設定更新 Amazon EKS 附加元件。如需更新附加元件的說明，請參閱 the section called “更新 (EKS 附加元件)”。

更新 CoreDNS Amazon EKS 附加元件

更新 Amazon EKS 類型的附加元件。如果您尚未將 Amazon EKS 附加元件新增至叢集，請進行新增或參閱 the section called “更新 (自我管理)”。

開始之前，請檢閱升級考量事項。如需詳細資訊，請參閱 the section called “重要的 CoreDNS 升級考量”。

1. 查看叢集上目前安裝了哪些附加元件版本。使用您的叢集名稱取代 *my-cluster*。

```
aws eks describe-addon --cluster-name my-cluster --addon-name coredns --query
 "addon.addonVersion" --output text
```

   範例輸出如下。

```
v1.10.1-eksbuild.13
```

   如果傳回的版本與最新版本資料表中的叢集 Kubernetes 版本相同，則表明您已經在叢集上安裝最新版本，不需要完成此程序的剩餘步驟。若您收到錯誤而非版本編號，則表明叢集上沒有安裝 Amazon EKS 類型的附加元件。您必須先建立附加元件，才能使用此程序進行更新。

2. 儲存您目前安裝的附加元件。

```
kubectl get deployment coredns -n kube-system -o yaml > aws-k8s-coredns-old.yaml
```

3. 使用 AWS CLI 更新您的附加元件。如果您想要使用 AWS 管理主控台 或 eksctl 更新附加元件，請參閱 the section called “更新附加元件”。將隨後的命令複製到您的裝置。視需要對命令進行下列修改，然後執行修改後的命令。
   - 使用您叢集的名稱取代 *my-cluster*。
   - 將 *v1.11.3-eksbuild.1* 取代為叢集版本的最新版本資料表中列出的最新版本。
   - `--resolve-conflicts`*PRESERVE* 選項會保留附加元件的現有組態值。如果對於附加元件設定，您已設定自訂值，但未使用此選項，Amazon EKS 會以其預設值覆寫您的值。如果您使用此選項，建議您在更新生產叢集上的附加元件之前，測試非生產叢集上的任何欄位和值變更。如果您將此值變更為 OVERWRITE，則所有設定都會變更為 Amazon EKS 預設值。如果您已設定任何設定的自訂值，則可能會使用 Amazon EKS 預設值覆寫這些值。如果您將此值變更為 none，Amazon EKS 不會變更任何設定的值，但更新可能會失敗。若更新失敗，您會收到錯誤訊息，以協助您解決衝突。
   - 如果您不更新組態設定，請從命令中移除 `--configuration-values '{`*"replicaCount":3*`}'`。如果您要更新組態設定，請以您要進行的設定取代 *"replicaCount":3*。在此範例中，CoreDNS 的複本數目設定為 3。您指定的值必須對組態結構描述有效。若您不知道組態結構描述，請執行 `aws eks describe-addon-configuration --addon-name coredns --addon-version` *v1.11.3-eksbuild.1*

，其中請用要查看組態的附加元件版本編號取代 *v1.11.3-eksbuild.1*。結構描述會在輸出中傳回。如果您有任何現有的自訂組態，並且想要全部移除，同時將所有設定的值設定回 Amazon EKS 預設值，請從命令中移除 *"replicaCount":3*，這樣就會得到空白的 {}。如需有關 CoreDNS 設定的詳細資訊，請參閱 Kubernetes 文件中的自訂 DNS 服務。

```
aws eks update-addon --cluster-name my-cluster --addon-name coredns --addon-version
 v1.11.3-eksbuild.1 \
    --resolve-conflicts PRESERVE --configuration-values '{"replicaCount":3}'
```

更新動作可能需要幾秒鐘的時間才能完成。

4. 確認附加元件版本已更新。使用您叢集的名稱取代 *my-cluster*。

```
aws eks describe-addon --cluster-name my-cluster --addon-name coredns
```

更新動作可能需要幾秒鐘的時間才能完成。

範例輸出如下。

```
{
    "addon": {
        "addonName": "coredns",
        "clusterName": "my-cluster",
        "status": "ACTIVE",
        "addonVersion": "v1.11.3-eksbuild.1",
        "health": {
            "issues": []
        },
        "addonArn": "arn:aws:eks:region:111122223333:addon/my-cluster/coredns/
d2c34f06-1111-2222-1eb0-24f64ce37fa4",
        "createdAt": "2023-03-01T16:41:32.442000+00:00",
        "modifiedAt": "2023-03-01T18:16:54.332000+00:00",
        "tags": {},
        "configurationValues": "{\"replicaCount\":3}"
    }
}
```

更新 CoreDNS Amazon EKS 自我管理的附加元件

> **Important**
>
> 建議將附加元件的 Amazon EKS 類型新增到叢集，而不是使用附加元件的自我管理類型。如果您不熟悉類型之間的差異，則請參閱 the section called “Amazon EKS 附加元件”。如需將 Amazon EKS 附加元件新增至叢集的詳細資訊，請參閱 the section called “建立附加元件”。如果您無法使用 Amazon EKS 附加元件，我們建議您提交有關為何無法存取容器藍圖 GitHub 儲存庫的問題。

開始之前，請檢閱升級考量事項。如需詳細資訊，請參閱 the section called “重要的 CoreDNS 升級考量”。

1. 確認您的叢集上已安裝附加元件的自我管理類型。使用您叢集的名稱取代 *my-cluster*。

   ```
   aws eks describe-addon --cluster-name my-cluster --addon-name coredns --query
    addon.addonVersion --output text
   ```

   如果傳回錯誤訊息，則表明您的叢集上安裝了附加元件的自我管理類型。完成此程序的剩餘步驟。如果傳回版本編號，則表明已在叢集上安裝附加元件的 Amazon EKS 類型。若要更新附加元件的 Amazon EKS 類型，請使用更新 CoreDNS Amazon EKS 附加元件中的程序，而不是使用此程序。如果您不熟悉附加元件類型之間的差異，請參閱 the section called “Amazon EKS 附加元件”。

2. 查看叢集上目前安裝了哪些容器映像版本。

   ```
   kubectl describe deployment coredns -n kube-system | grep Image | cut -d ":" -f 3
   ```

   範例輸出如下。

   ```
   v1.8.7-eksbuild.2
   ```

3. 如果您目前的 CoreDNS 版本為 `v1.5.0` 或更新版本，但早於 CoreDNS 版本資料表中列出的版本，則略過此步驟。如果您目前的版本比 `1.5.0` 還要舊，則需要修改 CoreDNS 的 `ConfigMap`，以使用正向附加元件，而不是代理附加元件。

   a. 使用下列命令開啟 `ConfigMap`。

      ```
      kubectl edit configmap coredns -n kube-system
      ```

b. 使用 forward 取代為下列行中的 proxy：儲存檔案，然後退出編輯器。

```
proxy . /etc/resolv.conf
```

4. 如果您原先在 Kubernetes 1.17 或舊版本上部署叢集，那麼您可能需要從您的 CoreDNS 資訊清單中移除已終止的行。

> **Important**
>
> 您必須在升級至 CoreDNS 版本 1.7.0 之前完成此步驟，即使您正在升級至較舊版本，也建議您先完成此步驟。

a. 檢查您的 CoreDNS 資訊清單是否有這一行。

```
kubectl get configmap coredns -n kube-system -o jsonpath='{$.data.Corefile}' |
 grep upstream
```

如果沒有傳回任何輸出，則資訊清單沒有該行，您可跳至下一個步驟來更新 CoreDNS。如果傳回輸出，那麼您需要移除該行。

b. 使用下列命令編輯 ConfigMap，移除檔案中有文字 upstream 的行。不要變更檔案中的任何其他內容。移除行之後，儲存變更。

```
kubectl edit configmap coredns -n kube-system -o yaml
```

5. 擷取您當前的 CoreDNS 映像版本：

```
kubectl describe deployment coredns -n kube-system | grep Image
```

範例輸出如下。

```
602401143452.dkr.ecr.region-code.amazonaws.com/eks/coredns:v1.8.7-eksbuild.2
```

6. 如果要更新到 CoreDNS 1.8.3 或更新版本，則需要將 endpointslices 許可新增至 system:coredns Kubernetes clusterrole。

```
kubectl edit clusterrole system:coredns -n kube-system
```

在檔案的 rules 區段的現有許可行下，新增以下行。

```
[...]
- apiGroups:
  - discovery.k8s.io
  resources:
  - endpointslices
  verbs:
  - list
  - watch
[...]
```

7. 使用上一個步驟中返回的值取代 *602401143452* 和 *region-code*，從而更新 CoreDNS 附加元件。將 *v1.11.3-eksbuild.1* 取代為適用於您 Kubernetes 版本的最新版本資料表中列出的 CoreDNS 版本。

```
kubectl set image deployment.apps/coredns -n kube-system
 coredns=602401143452.dkr.ecr.region-code.amazonaws.com/eks/coredns:v1.11.3-
eksbuild.1
```

範例輸出如下。

```
deployment.apps/coredns image updated
```

8. 再次檢查容器映像版本，以確認其已更新至您在上一步中指定的版本。

```
kubectl describe deployment coredns -n kube-system | grep Image | cut -d ":" -f 3
```

範例輸出如下。

```
v1.11.3-eksbuild.1
```

為高 DNS 流量擴展 CoreDNS Pod

當您啟動具有至少一個節點的 Amazon EKS 叢集時，預設會部署 CoreDNS 映像的兩個複本的部署，無論叢集中部署的節點數目為何。CoreDNS Pod 為叢集中的所有 Pod 提供名稱解析。應用程式使用名稱解析來連線至叢集中的 Pod 和服務，並連線至叢集外部的服務。隨著來自 Pod 的名稱解析請求 (查詢) 數量增加，CoreDNS Pod 可能會變得不堪重負和速度變慢，並拒絕 Pod 無法處理的請求。

為應對 CoreDNS Pod 增加的負載，請考慮為 CoreDNS 設定自動擴展系統。Amazon EKS 可在 CoreDNS 的 EKS 附加元件版本中管理 CoreDNS 部署的自動擴展。此 CoreDNS 自動擴展器持續監控叢集狀態，包括節點數和 CPU 核心數。根據該資訊，控制器會動態調整 EKS 叢集中 CoreDNS 部署的複本數量。此功能適用於 CoreDNS v1.9 及更新版本。有關哪些版本與 CoreDNS 自動擴展相容的更多資訊，請參閱下一節。

該系統會根據叢集中的節點和 CPU 核心數目，使用動態公式自動管理 CoreDNS 複本，計算方式為 (numberOfNodes 除以 16) 和 (numberOfCPUCores 除以 256) 的最大值。它會評估 10 分鐘尖峰時段的需求，在需要時立即向上擴展以處理增加的 DNS 查詢負載，同時每 3 分鐘縮減複本規模 33%，以維持系統穩定性並避免中斷。

我們建議您將此功能與其他 EKS Cluster Autoscaling 最佳實務結合使用，以改善整體應用程式可用性和叢集可擴展性。

先決條件

要讓 Amazon EKS 擴展您的 CoreDNS 部署，有三個先決條件：

- 必須使用 CoreDNS 的 EKS 附加元件版本。
- 叢集必須執行至少最低要求的叢集版本和平台版本。
- 叢集必須執行至少最低要求的 CoreDNS EKS 附加元件版本。

最低叢集版本

CoreDNS 的自動擴展由叢集控制平面中的新元件負責，該元件由 Amazon EKS 管理。因此，您必須將叢集升級至支援具有新元件的最低平台版本的 EKS 版本。

新的 Amazon EKS 叢集。若要部署叢集，請參閱 開始使用。叢集必須執行下表所列的其中一種 Kubernetes 版本和平台版本或更高版本。請注意，也支援比上列任何 Kubernetes 與平台版本更新的版本。您可使用叢集名稱取代下列命令的 *my-cluster*，然後執行修改的命令來檢查目前 Kubernetes 版本：

```
aws eks describe-cluster --name my-cluster --query cluster.version --output text
```

Kubernetes 版本	平台版本
未列示	全部平台版本

Kubernetes 版本	平台版本
1.29.3	eks.7
1.28.8	eks.13
1.27.12	eks.17
1.26.15	eks.18

> **Note**
>
> 也支援更高 Kubernetes 版本的每個平台版本，例如從 eks.1 開始的 Kubernetes 版本 1.30。

最低 EKS 附加元件版本

Kubernetes 版本	1.29	1.28
	v1.11.1-eksbuild.9	v1.10.1-eksbuild.11

在 中設定 CoreDNS 自動調整規模 AWS 管理主控台

1. 確保您的叢集達到或高於最低叢集版本。

   Amazon EKS 會自動在同一 Kubernetes 版本的不同平台版本之間升級叢集，您無法自行啟動此程序。相反，您可以將叢集升級到下一個 Kubernetes 版本，叢集將升級到該 K8s 版本和最新的平台版本。

   新的 Kubernetes 版本有時會加入重大變更。因此，我們推薦您在更新生產叢集之前，使用新 Kubernetes 版本的單獨叢集測試應用程式的行為。

   要將叢集升級至新的 Kubernetes 版本，請遵循將現有叢集更新至全新 Kubernetes 版本中的程序。
2. 確保您擁有 CoreDNS 的 EKS 附加元件，而不是自我管理的 CoreDNS 部署。

   視您用來建立叢集的工具而定，您的叢集上目前可能沒有安裝 Amazon EKS 附加元件類型。要查看叢集上安裝的附加元件類型，可執行以下命令。使用您叢集的名稱取代 my-cluster。

```
aws eks describe-addon --cluster-name my-cluster --addon-name coredns --query
 addon.addonVersion --output text
```

如果傳回版本編號，則表明已在叢集上安裝附加元件的 Amazon EKS 類型，並且您可以繼續後續步驟。如果傳回錯誤，則表明沒有在叢集上安裝 Amazon EKS 類型的附加元件。完成程序建立 CoreDNS Amazon EKS 附加元件的其餘步驟，以用 Amazon EKS 附加元件取代自我管理版本。

3. 確保 CoreDNS 的 EKS 附加元件版本等於或高於最低 EKS 附加元件版本。

   查看叢集上目前安裝了哪些附加元件版本。您可以在 中查看 AWS 管理主控台 或執行下列命令：

   ```
   kubectl describe deployment coredns --namespace kube-system | grep coredns: | cut -
   d : -f 3
   ```

   範例輸出如下。

   ```
   v1.10.1-eksbuild.13
   ```

   將此版本與上一節中的最低要求 EKS 附加元件版本進行比較。如有需要，請遵循程序更新 CoreDNS Amazon EKS 附加元件將 EKS 附加元件升級到更高版本。

4. 可將自動調整規模組態新增至 EKS 附加元件的選用組態設定。

   a. 開啟 Amazon EKS 主控台。

   b. 在左側導覽窗格中，選取 Clusters (叢集)，然後選取您要為其設定附加元件之叢集的名稱。

   c. 選擇附加元件索引標籤。

   d. 選取 CoreDNS 附加元件方塊右上方的方塊，然後選擇編輯。

   e. 在設定 CoreDNS 頁面上：

      i. 選取您要使用的版本。建議您保留與上一步驟相同的版本，並執行單獨的動作來更新版本與組態。

      ii. 展開選用組態設定。

      iii. 請輸入組態值中包含索引鍵 `"enabled":` 和值 `true` 的巢狀 JSON 物件的 JSON 索引鍵 `"autoscaling":` 和值。產生的文字必須是有效的 JSON 物件。如果此金鑰和值是文字方塊中唯一的資料，請以大括號 `{ }` 括住該金鑰和值。以下範例顯示已啟用自動擴展：

      ```
      {
        "autoScaling": {
          "enabled": true
      ```

```
  }
}
```

iv. (選用) 您可以提供自動擴展可以將 CoreDNS Pod 數量擴展到的最小值和最大值。

以下範例顯示已啟用自動擴展，並且所有選用鍵都有值。我們建議 CoreDNS Pod 的最小數量始終大於 2，以提供叢集中 DNS 服務的彈性。

```
{
  "autoScaling": {
    "enabled": true,
    "minReplicas": 2,
    "maxReplicas": 10
  }
}
```

f. 要透過取代 CoreDNS Pod 來套用新組態，請選擇儲存變更。

Amazon EKS 藉由推展適用於 CoreDNS 的 Kubernetes 部署，來套用變更至 EKS 附加元件。您可以在 和 的附加元件更新歷史記錄中追蹤推展的狀態 AWS 管理主控台 `kubectl rollout status deployment/coredns --namespace kube-system`。

`kubectl rollout` 具有下列命令：

```
kubectl rollout

history  -- View rollout history
pause    -- Mark the provided resource as paused
restart  -- Restart a resource
resume   -- Resume a paused resource
status   -- Show the status of the rollout
undo     -- Undo a previous rollout
```

若推展時間太長，Amazon EKS 會復原推展，並且類型為附加元件更新及狀態為失敗的訊息，則會新增至附加元件的更新歷程記錄。要調查任何問題，請從推展的歷史記錄開始，並在 CoreDNS Pod 上執行 `kubectl logs`，以查看 CoreDNS 的日誌。

5. 若更新歷程記錄中新項目的狀態為成功，則表示推展已完成，並且在所有 CoreDNS Pod 身分識別代理程式 Pod 中，附加元件均會使用新的組態。當您變更叢集中節點的節點和 CPU 核心數時，Amazon EKS 會擴展 CoreDNS 部署的複本數量。

在 AWS 命令列界面中設定 CoreDNS 自動調整規模

1. 確保您的叢集達到或高於最低叢集版本。

   Amazon EKS 會自動在同一 Kubernetes 版本的不同平台版本之間升級叢集，您無法自行啟動此程序。相反，您可以將叢集升級到下一個 Kubernetes 版本，叢集將升級到該 K8s 版本和最新的平台版本。

   新的 Kubernetes 版本有時會加入重大變更。因此，我們推薦您在更新生產叢集之前，使用新 Kubernetes 版本的單獨叢集測試應用程式的行為。

   要將叢集升級至新的 Kubernetes 版本，請遵循將現有叢集更新至全新 Kubernetes 版本中的程序。

2. 確保您擁有 CoreDNS 的 EKS 附加元件，而不是自我管理的 CoreDNS 部署。

   視您用來建立叢集的工具而定，您的叢集上目前可能沒有安裝 Amazon EKS 附加元件類型。要查看叢集上安裝的附加元件類型，可執行以下命令。使用您叢集的名稱取代 `my-cluster`。

   ```
   aws eks describe-addon --cluster-name my-cluster --addon-name coredns --query
    addon.addonVersion --output text
   ```

   如果傳回版本編號，則表明已在叢集上安裝附加元件的 Amazon EKS 類型。如果傳回錯誤，則表明沒有在叢集上安裝 Amazon EKS 類型的附加元件。完成程序建立 CoreDNS Amazon EKS 附加元件的其餘步驟，以用 Amazon EKS 附加元件取代自我管理版本。

3. 確保 CoreDNS 的 EKS 附加元件版本等於或高於最低 EKS 附加元件版本。

   查看叢集上目前安裝了哪些附加元件版本。您可以在 中查看 AWS 管理主控台 或執行下列命令：

   ```
   kubectl describe deployment coredns --namespace kube-system | grep coredns: | cut -
   d : -f 3
   ```

   範例輸出如下。

   ```
   v1.10.1-eksbuild.13
   ```

   將此版本與上一節中的最低要求 EKS 附加元件版本進行比較。如有需要，請遵循程序更新 CoreDNS Amazon EKS 附加元件將 EKS 附加元件升級到更高版本。

4. 可將自動調整規模組態新增至 EKS 附加元件的選用組態設定。

執行下列 AWS CLI 命令。將 my-cluster 取代為您的叢集名稱，並將 IAM 角色 ARN 取代為您正在使用的角色。

```
aws eks update-addon --cluster-name my-cluster --addon-name coredns \
    --resolve-conflicts PRESERVE --configuration-values '{"autoScaling":
{"enabled":true}}'
```

Amazon EKS 藉由推展適用於 CoreDNS 的 Kubernetes 部署，來套用變更至 EKS 附加元件。您可以在 和 中的附加元件更新歷史記錄中追蹤推展的狀態 AWS 管理主控台 `kubectl rollout status deployment/coredns --namespace kube-system`。

`kubectl rollout` 具有下列命令：

```
kubectl rollout

history  -- View rollout history
pause    -- Mark the provided resource as paused
restart  -- Restart a resource
resume   -- Resume a paused resource
status   -- Show the status of the rollout
undo     -- Undo a previous rollout
```

若推展時間太長，Amazon EKS 會復原推展，並且類型為附加元件更新及狀態為失敗的訊息，則會新增至附加元件的更新歷程記錄。要調查任何問題，請從推展的歷史記錄開始，並在 CoreDNS Pod 上執行 `kubectl logs`，以查看 CoreDNS 的日誌。

5. (選用) 您可以提供自動擴展可以將 CoreDNS Pod 數量擴展到的最小值和最大值。

   以下範例顯示已啟用自動擴展，並且所有選用鍵都有值。我們建議 CoreDNS Pod 的最小數量始終大於 2，以提供叢集中 DNS 服務的彈性。

   ```
   aws eks update-addon --cluster-name my-cluster --addon-name coredns \
       --resolve-conflicts PRESERVE --configuration-values '{"autoScaling":
   {"enabled":true,"minReplicas":2,"maxReplicas":10}}'
   ```

6. 透過執行以下命令檢查附加元件更新的狀態：

   ```
   aws eks describe-addon --cluster-name my-cluster --addon-name coredns
   ```

如果您看到此行：`"status": "ACTIVE"`，則表示推展已完成，且附加元件在所有 CoreDNS Pod 中使用新組態。當您變更叢集中節點的節點和 CPU 核心數時，Amazon EKS 會擴展 CoreDNS 部署的複本數量。

使用 CoreDNS 指標來監控 Kubernetes DNS 解析

CoreDNS 作為 EKS 附加元件，會在 CoreDNS 服務中以 Prometheus 格式在連接埠 9153 上公開來自 `kube-dns` 的指標。您可以使用 Prometheus、Amazon CloudWatch 代理程式或任何其他相容的系統來湊集 (收集) 這些指標。

如需與 Prometheus 和 CloudWatch 代理程式相容的範例湊集組態，請參閱《Amazon CloudWatch 使用者指南》中的適用於 Prometheus 的 CloudWatch 代理程式組態。

在 Amazon EKS 叢集中管理 **kube-proxy**

> **Tip**
>
> 使用 Amazon EKS 自動模式，您無需安裝或升級聯網附加元件。自動模式包括 Pod 聯網和負載平衡功能。
> 如需詳細資訊，請參閱EKS 自動模式。

建議將附加元件的 Amazon EKS 類型新增到叢集，而不是使用附加元件的自我管理類型。如果您不熟悉類型之間的差異，則請參閱 the section called “Amazon EKS 附加元件”。如需將 Amazon EKS 附加元件新增至叢集的詳細資訊，請參閱 the section called “建立附加元件”。如果您無法使用 Amazon EKS 附加元件，我們建議您提交有關為何無法存取容器藍圖 GitHub 儲存庫的問題。

`kube-proxy` 附加元件會部署在 Amazon EKS 叢集中的每個 Amazon EC2 節點上。其會維護節點上的網路規則，並啟用與 Pod 的網路通訊。附加元件未部署至叢集中的 Fargate 節點。如需詳細資訊，請參閱 Kubernetes 文件中的 kube-proxy。

安裝 Amazon EKS 附加元件

kube-proxy 版本

下表列出適用於每個 Kubernetes 版本的 Amazon EKS 附加元件類型的最新版本。

Kubernetes 版本	**kube-proxy** 版本
1.36	v1.36.0-eksbuild.17
1.35	v1.35.3-eksbuild.21
1.34	v1.34.6-eksbuild.21
1.33	v1.33.10-eksbuild.21
1.32	v1.32.13-eksbuild.24
1.31	v1.31.14-eksbuild.28

> **Note**
>
> 如果您要自行管理此附加元件，資料表中的版本可能與可用的自我管理版本不同。

kube-proxy 容器映像

此 kube-proxy 容器映像類型的基礎為 Amazon EKS Distro 維護的最低基礎映像，其中包含最少套件且沒有 Shell。如需詳細資訊，請參閱 Amazon EKS Distro。

下表列出每個 Amazon EKS 叢集版本最新可用的自我管理 kube-proxy 容器映像版本。

版本	kube-proxy
1.36	v1.36.0-eksbuild.17
1.35	v1.35.3-eksbuild.21
1.34	v1.34.6-eksbuild.21
1.33	v1.33.10-minimal-eksbuild.21
1.32	v1.32.13-minimal-eksbuild.24
1.31	v1.31.14-minimal-eksbuild.28

更新 Amazon EKS 附加元件類型時，您必須指定有效的 Amazon EKS 附加元件版本，該版本可能不是此資料表中列出的版本。這是因為 Amazon EKS 附加元件版本不一定與更新此附加元件的自我管理類型時指定的容器映像版本相符。當您更新此附加元件的自我管理類型時，請指定此資料表中列出的有效容器映像版本。

更新 Kubernetes **kube-proxy** 自我管理的附加元件

> **Important**
>
> 建議將附加元件的 Amazon EKS 類型新增到叢集，而不是使用附加元件的自我管理類型。如果您不熟悉類型之間的差異，則請參閱 the section called “Amazon EKS 附加元件”。如需將 Amazon EKS 附加元件新增至叢集的詳細資訊，請參閱 the section called “建立附加元件”。如果您無法使用 Amazon EKS 附加元件，我們建議您提交有關為何無法存取容器藍圖 GitHub 儲存庫的問題。

先決條件

- 現有 Amazon EKS 叢集。若要部署叢集，請參閱 開始使用。

考量事項

- Amazon EKS 叢集上的 Kube-proxy 有與 Kubernetes 相同的相容性和差異政策。了解如何驗證 Amazon EKS 附加元件版本與叢集的相容性。
  1. 確認您的叢集上已安裝附加元件的自我管理類型。使用您叢集的名稱取代 *my-cluster*。

     ```
     aws eks describe-addon --cluster-name my-cluster --addon-name kube-proxy --query
      addon.addonVersion --output text
     ```

     如果傳回錯誤訊息，則表明您的叢集上安裝了附加元件的自我管理類型。本主題中的其餘步驟用於更新附加元件的自我管理類型。如果傳回版本編號，則表明已在叢集上安裝附加元件的 Amazon EKS 類型。若要將其更新，請使用更新 Amazon EKS 附加元件中的程序，而不是使用本主題中的程序。如果您不熟悉附加元件類型之間的差異，請參閱 the section called “Amazon EKS 附加元件”。
  2. 查看叢集上目前安裝了哪些容器映像版本。

     ```
     kubectl describe daemonset kube-proxy -n kube-system | grep Image
     ```

範例輸出如下。

```
Image:    602401143452.dkr.ecr.region-code.amazonaws.com/eks/kube-proxy:v1.29.1-eksbuild.2
```

在範例輸出中，安裝在叢集上的版本為 *v1.29.1-eksbuild.2*。

3. 使用上一個步驟中輸出的值取代 *602401143452* 和 *region-code*，從而更新 kube-proxy 附加元件。使用每個 Amazon EKS 叢集版本的最新可用自我管理 kube-proxy 容器映像版本表列出的 kube-proxy 版本取代 *v1.30.6-eksbuild.3*。

> **Important**
>
> 每個映像類型的資訊清單皆不相同，並且與預設或最小映像類型之間不相容。您必須使用與上一個映像相同的映像類型，以便進入點和引數相符。

```
kubectl set image daemonset.apps/kube-proxy -n kube-system kube-proxy=602401143452.dkr.ecr.region-code.amazonaws.com/eks/kube-proxy:v1.30.6-eksbuild.3
```

範例輸出如下。

```
daemonset.apps/kube-proxy image updated
```

4. 確認您的叢集上現在已安裝新版本。

```
kubectl describe daemonset kube-proxy -n kube-system | grep Image | cut -d ":" -f 3
```

範例輸出如下。

```
v1.30.0-eksbuild.3
```

5. 如果您在同一個叢集中使用 x86 和 Arm 節點，而且您的叢集是在 2020 年 8 月 17 日之前部署的。然後，編輯您的 kube-proxy 清單檔案，以包含多個硬體架構的節點選取器，並使用以下命令。這是一次性操作。將選取器新增至資訊清單後，不需要在每次更新附加元件時進行新增。如果您的叢集是在 2020 年 8 月 17 日或之後部署的，則 kube-proxy 已經具備多架構能力。

```
kubectl edit -n kube-system daemonset/kube-proxy
```

將下列節點選取器新增至編輯器中的檔案，然後儲存檔案。如需在編輯器中包含此文字的範例，請參閱 GitHub 上的 CNI 清單檔案檔案。這樣一來，可讓 Kubernetes 根據節點的硬體架構提取正確的硬體映像。

```
- key: "kubernetes.io/arch"
  operator: In
  values:
  - amd64
  - arm64
```

6. 如果您的叢集最初是使用 Kubernetes 版本 1.14 或更新版本建立的，則可以略過此步驟，因為 kube-proxy 已包含此 Affinity Rule。如果您最初使用 Kubernetes 版本 1.13 或更舊版本建立 Amazon EKS 叢集，並打算使用叢集中的 Fargate 節點，則請編輯 kube-proxy 資訊清單以包含 NodeAffinity 規則，進而防止 kube-proxy Pod 在 Fargate 節點上排程。這是一次性編輯。將 Affinity Rule 新增至資訊清單後，不需要在每次更新附加元件時進行新增。編輯您的 kube-proxy DaemonSet。

```
kubectl edit -n kube-system daemonset/kube-proxy
```

新增下列 Affinity Rule 到編輯器的檔案中的 DaemonSet spec 部分，然後儲存檔案。如需在編輯器中包含此文字的範例，請參閱 GitHub 上的 CNI 清單檔案檔案。

```
- key: eks.amazonaws.com/compute-type
  operator: NotIn
  values:
  - fargate
```

了解如何將工作負載和附加元件部署至 Amazon EKS

您的工作負載會部署在容器中，而這些容器會部署在 Kubernetes 的 Pod 中。Pod 包含一或多個容器。一般而言，提供相同服務的一或多個 Pod 會部署在 Kubernetes Service 中。部署多個提供相同服務的 Pod 後，您可以：

- 使用 AWS 管理主控台 檢視在每個叢集上執行之工作負載的相關資訊。
- 使用 Kubernetes Vertical Pod Autoscaler 垂直擴展 Pod。
- 水平擴展所需的 Pod 數量，以滿足使用 Kubernetes Horizontal Pod Autoscaler 進行縮放的需求。
- 建立外部 (針對網際網路可存取的 Pod) 或內部 (針對私有 Pod) Network Load Balancer 來平衡 Pod 間的網路流量。負載平衡器會在 OSI 模型的第 4 層路由流量。
- 建立 Application Load Balancer 來平衡 Pod 間的應用程式流量。Application Load Balancer 會在 OSI 模型的第 7 層路由流量。
- 如果您是 Kubernetes 新手，則本主題可協助您部署範例應用程式。
- 您可以使用 `externalIPs` 限制可指派給服務的 IP 地址。

在 Linux 上部署範例應用程式

在本主題中，您將範例應用程式部署到 Linux 節點上的叢集。

先決條件

- 一個現有的 Kubernetes 叢集至少會有一個節點。如果尚無現有 Amazon EKS 叢集，可以使用 開始使用 中的其中一項指南來部署叢集。
- 安裝在電腦上的 `Kubectl`。如需詳細資訊，請參閱the section called “設定 kubectl 和 eksctl”。
- 設定好的 `Kubectl`，以便與叢集通訊。如需詳細資訊，請參閱the section called “透過 kubectl 存取叢集”。
- 如果打算將範例工作負載部署至 Fargate，則必須擁有現有的 Fargate 設定檔，其中包含在本教學課程中建立的相同命名空間，也就是 `eks-sample-app`，除非您變更名稱。如果您使用 中的其中一個指南建立叢集開始使用，則必須建立新的設定檔，或將命名空間新增至現有的設定檔，因為入門指南中建立的設定檔不會指定本教學課程中所使用的命名空間。您的 VPC 也必須至少有一個私有子網路。

雖然許多變數可在下列步驟中變更，但建議只在指定的位置變更變值。更加了解 Kubernetes Pod、部署和服務後，您可以嘗試變更其他值。

建立命名空間。

命名空間可讓您將 Kubernetes 中的資源分組。如需詳細資訊，請參閱 Kubernetes 文件中的命名空間。如果您打算部署範例應用程式以使用 AWS Fargate 簡化運算管理，請確定 定義啟動時使用 AWS Fargate 之 Pod namespace的 值為 eks-sample-app。

```
kubectl create namespace eks-sample-app
```

建立 Kubernetes 部署

建立 Kubernetes 部署。此範例部署會從公有儲存庫提取容器映像，並將其三個複本 (個別 Pod) 部署至您的叢集。若要進一步了解，請參閱 Kubernetes 文件中的部署。

1. 將下列內容儲存到名為 eks-sample-deployment.yaml 的檔案中。範例應用程式中的容器不會使用網路儲存，但您可能有需要使用網路儲存的應用程式。如需詳細資訊，請參閱應用程式資料儲存。
   - 在 kubernetes.io/arch 金鑰下方的 amd64 或 arm64 values 代表可以部署應用程式至其中一個硬體架構 (如果叢集中有兩個)。此操作可行，因為此映像是一個多架構映像，但並非所有映像都是。透過檢視要從中提取映像的儲存庫中的映像詳細資訊，可以確定支援該映像的硬體架構。部署不支援硬體架構類型的映像時，或者不希望部署映像時，請從清單檔案中移除該類型。如需詳細資訊，請參閱 Kubernetes 文件中的已知標籤、註釋和污點。
   - kubernetes.io/os: linux nodeSelector 表示，如果叢集中有 Linux 和 Windows 節點 (舉例來說)，則該映像只會部署至 Linux 節點。如需詳細資訊，請參閱 Kubernetes 文件中的已知標籤、註釋和污點。

   ```
   apiVersion: apps/v1
   kind: Deployment
   metadata:
     name: eks-sample-linux-deployment
     namespace: eks-sample-app
     labels:
       app: eks-sample-linux-app
   spec:
     replicas: 3
     selector:
       matchLabels:
   ```

```
      app: eks-sample-linux-app
  template:
    metadata:
      labels:
        app: eks-sample-linux-app
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: kubernetes.io/arch
                operator: In
                values:
                - amd64
                - arm64
      containers:
      - name: nginx
        image: public.ecr.aws/nginx/nginx:1.23
        ports:
        - name: http
          containerPort: 80
        imagePullPolicy: IfNotPresent
      nodeSelector:
        kubernetes.io/os: linux
```

2. 將部署清單檔案套用至叢集。

```
kubectl apply -f eks-sample-deployment.yaml
```

建立服務

服務可讓您透過單一 IP 地址或名稱存取所有複本。如需詳細資訊，請參閱 Kubernetes 文件中的服務。雖然未在範例應用程式中實作，但如果您的應用程式需要與其他 AWS 服務互動，我們建議您為 Pod 建立 Kubernetes 服務帳戶，並將其與 AWS IAM 帳戶建立關聯。指定服務帳戶後，您的 Pod 僅能擁有您為它們指定與其他服務互動所需的最低許可。如需詳細資訊，請參閱the section called “具有 IRSA 的憑證”。

1. 將下列內容儲存到名為 `eks-sample-service.yaml` 的檔案中。Kubernetes 會為服務指派自己的 IP 地址，這些地址只能從叢集內存取。若要從叢集外部存取服務，請部署 AWS Load Balancer 控制器，以將應用程式或網路流量負載平衡至服務。

```
apiVersion: v1
kind: Service
metadata:
  name: eks-sample-linux-service
  namespace: eks-sample-app
  labels:
    app: eks-sample-linux-app
spec:
  selector:
    app: eks-sample-linux-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

2. 將服務資訊清單套用至叢集。

```
kubectl apply -f eks-sample-service.yaml
```

檢閱建立的資源

1. 檢視 eks-sample-app 命名空間中存在的所有資源。

```
kubectl get all -n eks-sample-app
```

範例輸出如下。

```
NAME                                                READY   STATUS    RESTARTS   AGE
pod/eks-sample-linux-deployment-65b7669776-m6qxz    1/1     Running   0          27m
pod/eks-sample-linux-deployment-65b7669776-mmxvd    1/1     Running   0          27m
pod/eks-sample-linux-deployment-65b7669776-qzn22    1/1     Running   0          27m

NAME                                TYPE          CLUSTER-IP      EXTERNAL-IP   PORT(S)
   AGE
service/eks-sample-linux-service    ClusterIP     10.100.74.8     <none>        80/TCP
   32m

NAME                                           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/eks-sample-linux-deployment    3/3     3            3           27m
```

```
NAME                                                    DESIRED   CURRENT   READY
 AGE
replicaset.apps/eks-sample-linux-deployment-776d8f8fd8   3         3         3
 27m
```

在輸出中，您會看到在先前步驟中所部署範例清單檔案中指定的服務和部署。您也會看到三個 Pod。這是因為範例清單檔案中指定了 3 個 `replicas`。如需 Pod 的詳細資訊，請參閱 Kubernetes 文件中的 Pod。Kubernetes 會自動建立 `replicaset` 資源，即使其未在範例清單檔案中指定。如需 `ReplicaSets` 的詳細資訊，請參閱 Kubernetes 文件中的 ReplicaSet。

> **Note**
>
> Kubernetes 會維護清單檔案中指定的複本數量。如果這是生產部署，並且您希望 Kubernetes 橫向擴展複本數量或垂直擴展 Pod 的運算資源，請使用透過水平 Pod 自動擴展器來擴展 Pod 部署和透過垂直 Pod 自動擴展器來調整 Pod 資源來執行此操作。

2. 檢視已部署服務的詳細資訊。

```
kubectl -n eks-sample-app describe service eks-sample-linux-service
```

範例輸出如下。

```
Name:              eks-sample-linux-service
Namespace:         eks-sample-app
Labels:            app=eks-sample-linux-app
Annotations:       <none>
Selector:          app=eks-sample-linux-app
Type:              ClusterIP
IP Families:       <none>
IP:                10.100.74.8
IPs:               10.100.74.8
Port:              <unset>  80/TCP
TargetPort:        80/TCP
Endpoints:         192.168.24.212:80,192.168.50.185:80,192.168.63.93:80
Session Affinity:  None
Events:            <none>
```

在先前的輸出中，`IP:` 值是可從叢集內的任一個節點或 Pod 連線的唯一 IP 位址，但無法從叢集外部連線。`Endpoints` 值是從 VPC 內指派至服務一部分的 Pod 的 IP 位址。

3. 在先前步驟中檢視命名空間時，請檢視輸出中列出的其中一個 Pod 詳細資訊。將 *65b7669776-m6qxz* 取代為其中一個 Pod 傳回的值。

```
kubectl -n eks-sample-app describe pod eks-sample-linux-deployment-65b7669776-m6qxz
```

縮寫範例輸出

```
Name:         eks-sample-linux-deployment-65b7669776-m6qxz
Namespace:    eks-sample-app
Priority:     0
Node:         ip-192-168-45-132.us-west-2.compute.internal/192.168.45.132
[...]
IP:           192.168.63.93
IPs:
  IP:           192.168.63.93
Controlled By:  ReplicaSet/eks-sample-linux-deployment-65b7669776
[...]
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
[...]
Events:
  Type    Reason     Age    From
 Message
  ----    ------     ----   ----
 -------
  Normal  Scheduled  3m20s  default-scheduler
 Successfully assigned eks-sample-app/eks-sample-linux-deployment-65b7669776-m6qxz to
 ip-192-168-45-132.us-west-2.compute.internal
[...]
```

在先前的輸出中，IP: 的值是唯一的 IP，從指派給節點所在子網路的 CIDR 區塊指派給 Pod。如果希望從不同的 CIDR 區塊指派 Pod IP 地址，則可以變更預設行為。如需詳細資訊，請參閱the section called “自訂聯網”。您也可以看到 Kubernetes 排程器使用 IP 位址 *192.168.45.132* 排定了 Node 上的 Pod。

> **Tip**
>
> 若不使用命令列，您可以在 AWS 管理主控台中檢視有關 Pod、服務、部署和其他 Kubernetes 資源的許多詳細資訊。如需詳細資訊，請參閱the section called “存取叢集資源”。

在 Pod 上執行 Shell

1. 在先前步驟中描述的 Pod 上執行 Shell，將 *65b7669776-m6qxz* 取代為其中一個 Pod 的 ID。

```
kubectl exec -it eks-sample-linux-deployment-65b7669776-m6qxz -n eks-sample-app -- /
bin/bash
```

2. 在 Pod Shell 中，檢視在先前步驟中與部署一起安裝的 Web 伺服器輸出。您只需要指定服務名稱。CoreDNS 會將服務名稱解析為服務的 IP 位址；CoreDNS 預設會與 Amazon EKS 叢集一起部署。

```
curl eks-sample-linux-service
```

   範例輸出如下。

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
[...]
```

3. 從 Pod Shell 中檢視 Pod 的 DNS 伺服器。

```
cat /etc/resolv.conf
```

   範例輸出如下。

```
nameserver 10.100.0.10
search eks-sample-app.svc.cluster.local svc.cluster.local cluster.local us-
west-2.compute.internal
options ndots:5
```

在先前的輸出中，10.100.0.10 會自動指派為讓所有 Pod 部署至叢集的 nameserver。

4. 透過輸入 exit 中斷從 Pod 的連線。
5. 完成範例應用程式後，您可以使用以下命令來移除範例命名空間、服務和部署。

```
kubectl delete namespace eks-sample-app
```

後續步驟

在部署範例應用程式之後，您可能需要嘗試下面列出的部分練習：

- 透過 Application Load Balancer 路由應用程式與 HTTP 流量
- 透過 Network Load Balancer 路由 TCP 與 UDP 流量

部署 Windows 上的範例應用程式

在這個主題中，您要部署範例應用程式至 Windows 節點上的叢集。

先決條件

- 一個現有的 Kubernetes 叢集至少會有一個節點。如果尚無現有 Amazon EKS 叢集，可以使用 開始使用 中的其中一項指南來部署叢集。您必須為叢集啟用 Windows 支援，以及啟用至少一個 Amazon EC2 Windows 節點。
- 安裝在電腦上的 Kubectl。如需詳細資訊，請參閱the section called “設定 kubectl 和 eksctl”。
- 設定好的 Kubectl，以便與叢集通訊。如需詳細資訊，請參閱the section called “透過 kubectl 存取叢集”。
- 如果打算將範例工作負載部署至 Fargate，則必須擁有現有的 Fargate 設定檔，其中包含在本教學課程中建立的相同命名空間，也就是 eks-sample-app，除非您變更名稱。如果您使用 中的其中一個指南建立叢集開始使用，則必須建立新的設定檔，或將命名空間新增至現有的設定檔，因為入門指南中建立的設定檔不會指定本教學課程中所使用的命名空間。您的 VPC 也必須至少有一個私有子網路。

雖然許多變數可在下列步驟中變更，但建議只在指定的位置變更變值。更加了解 Kubernetes Pod、部署和服務後，您可以嘗試變更其他值。

建立命名空間。

命名空間可讓您將 Kubernetes 中的資源分組。如需詳細資訊，請參閱 Kubernetes 文件中的命名空間。如果您打算部署範例應用程式以使用 AWS Fargate 簡化運算管理，請確定 定義啟動時使用 AWS Fargate 之 Pod namespace的 值為 eks-sample-app。

```
kubectl create namespace eks-sample-app
```

建立 Kubernetes 部署

此範例部署會從公有儲存庫提取容器映像，並將其三個複本 (個別 Pod) 部署至您的叢集。若要進一步了解，請參閱 Kubernetes 文件中的部署。

1. 將下列內容儲存到名為 eks-sample-deployment.yaml 的檔案中。範例應用程式中的容器不會使用網路儲存，但您可能有需要使用網路儲存的應用程式。如需詳細資訊，請參閱應用程式資料儲存。
   - kubernetes.io/os: windows nodeSelector 表示，如果叢集中有 Windows 和 Linux 節點 (舉例來說)，則該映像只會部署至 Windows 節點。如需詳細資訊，請參閱 Kubernetes 文件中的已知標籤、註釋和污點。

     ```
     apiVersion: apps/v1
     kind: Deployment
     metadata:
       name: eks-sample-windows-deployment
       namespace: eks-sample-app
       labels:
         app: eks-sample-windows-app
     spec:
       replicas: 3
       selector:
         matchLabels:
           app: eks-sample-windows-app
       template:
         metadata:
           labels:
             app: eks-sample-windows-app
         spec:
           affinity:
             nodeAffinity:
               requiredDuringSchedulingIgnoredDuringExecution:
     ```

```
            nodeSelectorTerms:
            - matchExpressions:
              - key: kubernetes.io/arch
                operator: In
                values:
                - amd64
      containers:
      - name: windows-server-iis
        image: mcr.microsoft.com/windows/servercore:ltsc2019
        ports:
        - name: http
          containerPort: 80
        imagePullPolicy: IfNotPresent
        command:
        - powershell.exe
        - -command
        - "Add-WindowsFeature Web-Server; Invoke-WebRequest -UseBasicParsing
 -Uri 'https://dotnetbinaries.blob.core.windows.net/servicemonitor/2.0.1.6/
ServiceMonitor.exe' -OutFile 'C:\\ServiceMonitor.exe'; echo '<html><body><br/
><br/><marquee><H1>Hello EKS!!!<H1><marquee></body><html>' > C:\\inetpub\\wwwroot\
\default.html; C:\\ServiceMonitor.exe 'w3svc'; "
      nodeSelector:
        kubernetes.io/os: windows
```

2. 將部署清單檔案套用至叢集。

```
kubectl apply -f eks-sample-deployment.yaml
```

建立服務

服務可讓您透過單一 IP 地址或名稱存取所有複本。如需詳細資訊，請參閱 Kubernetes 文件中的服務。雖然未在範例應用程式中實作，但如果您的應用程式需要與其他 AWS 服務互動，我們建議您為 Pod 建立 Kubernetes 服務帳戶，並將其與 AWS IAM 帳戶建立關聯。指定服務帳戶後，您的 Pod 僅能擁有您為它們指定與其他服務互動所需的最低許可。如需詳細資訊，請參閱the section called “具有 IRSA 的憑證”。

1. 將下列內容儲存到名為 `eks-sample-service.yaml` 的檔案中。Kubernetes 會為服務指派自己的 IP 地址，這些地址只能從叢集內存取。若要從叢集外部存取服務，請部署 AWS Load Balancer 控制器，以將應用程式或網路流量負載平衡至服務。

```
apiVersion: v1
```

```
kind: Service
metadata:
  name: eks-sample-windows-service
  namespace: eks-sample-app
  labels:
    app: eks-sample-windows-app
spec:
  selector:
    app: eks-sample-windows-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

2. 將服務資訊清單套用至叢集。

```
kubectl apply -f eks-sample-service.yaml
```

檢閱建立的資源

1. 檢視 eks-sample-app 命名空間中存在的所有資源。

```
kubectl get all -n eks-sample-app
```

範例輸出如下。

```
NAME                                                 READY   STATUS    RESTARTS   AGE
pod/eks-sample-windows-deployment-65b7669776-m6qxz   1/1     Running   0          27m
pod/eks-sample-windows-deployment-65b7669776-mmxvd   1/1     Running   0          27m
pod/eks-sample-windows-deployment-65b7669776-qzn22   1/1     Running   0          27m

NAME                                 TYPE         CLUSTER-IP      EXTERNAL-IP   PORT(S)
   AGE
service/eks-sample-windows-service   ClusterIP    10.100.74.8     <none>        80/
TCP    32m

NAME                                            READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/eks-sample-windows-deployment 3/3     3            3           27m

NAME                                                       DESIRED   CURRENT   READY
 AGE
```

```
replicaset.apps/eks-sample-windows-deployment-776d8f8fd8    3         3         3
   27m
```

在輸出中，您會看到在先前步驟中所部署範例清單檔案中指定的服務和部署。您也會看到三個 Pod。這是因為範例清單檔案中指定了 3 個 `replicas`。如需 Pod 的詳細資訊，請參閱 Kubernetes 文件中的 Pod。Kubernetes 會自動建立 `replicaset` 資源，即使其未在範例清單檔案中指定。如需 `ReplicaSets` 的詳細資訊，請參閱 Kubernetes 文件中的 ReplicaSet。

> **Note**
>
> Kubernetes 會維護清單檔案中指定的複本數量。如果這是生產部署，並且您希望 Kubernetes 橫向擴展複本數量或垂直擴展 Pod 的運算資源，請使用透過水平 Pod 自動擴展器來擴展 Pod 部署和透過垂直 Pod 自動擴展器來調整 Pod 資源來執行此操作。

2. 檢視已部署服務的詳細資訊。

```
kubectl -n eks-sample-app describe service eks-sample-windows-service
```

範例輸出如下。

```
Name:              eks-sample-windows-service
Namespace:         eks-sample-app
Labels:            app=eks-sample-windows-app
Annotations:       <none>
Selector:          app=eks-sample-windows-app
Type:              ClusterIP
IP Families:       <none>
IP:                10.100.74.8
IPs:               10.100.74.8
Port:              <unset>  80/TCP
TargetPort:        80/TCP
Endpoints:         192.168.24.212:80,192.168.50.185:80,192.168.63.93:80
Session Affinity:  None
Events:            <none>
```

在先前的輸出中，`IP:` 值是可從叢集內的任一個節點或 Pod 連線的唯一 IP 位址，但無法從叢集外部連線。`Endpoints` 值是從 VPC 內指派至服務一部分的 Pod 的 IP 位址。

3. 在先前步驟中檢視命名空間時，請檢視輸出中列出的其中一個 Pod 詳細資訊。將 *65b7669776-m6qxz* 取代為其中一個 Pod 傳回的值。

```
kubectl -n eks-sample-app describe pod eks-sample-windows-deployment-65b7669776-m6qxz
```

縮寫範例輸出

```
Name:         eks-sample-windows-deployment-65b7669776-m6qxz
Namespace:    eks-sample-app
Priority:     0
Node:         ip-192-168-45-132.us-west-2.compute.internal/192.168.45.132
[...]
IP:           192.168.63.93
IPs:
  IP:           192.168.63.93
Controlled By:  ReplicaSet/eks-sample-windows-deployment-65b7669776
[...]
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
[...]
Events:
  Type    Reason     Age    From
 Message
  ----    ------     ----   ----
 -------
  Normal  Scheduled  3m20s  default-scheduler
 Successfully assigned eks-sample-app/eks-sample-windows-deployment-65b7669776-m6qxz
 to ip-192-168-45-132.us-west-2.compute.internal
[...]
```

在先前的輸出中，IP: 的值是唯一的 IP，從指派給節點所在子網路的 CIDR 區塊指派給 Pod。如果希望從不同的 CIDR 區塊指派 Pod IP 地址，則可以變更預設行為。如需詳細資訊，請參閱the section called “自訂聯網”。您也可以看到 Kubernetes 排程器使用 IP 位址 *192.168.45.132* 排定了 Node 上的 Pod。

> **Tip**
>
> 若不使用命令列，您可以在 AWS 管理主控台中檢視有關 Pod、服務、部署和其他 Kubernetes 資源的許多詳細資訊。如需詳細資訊，請參閱the section called “存取叢集資源”。

在 Pod 上執行 Shell

1. 在先前步驟中描述的 Pod 上執行 Shell，將 *65b7669776-m6qxz* 取代為其中一個 Pod 的 ID。

```
kubectl exec -it eks-sample-windows-deployment-65b7669776-m6qxz -n eks-sample-app --
 powershell.exe
```

2. 在 Pod Shell 中，檢視在先前步驟中與部署一起安裝的 Web 伺服器輸出。您只需要指定服務名稱。CoreDNS 會將服務名稱解析為服務的 IP 位址；CoreDNS 預設會與 Amazon EKS 叢集一起部署。

```
Invoke-WebRequest -uri eks-sample-windows-service/default.html -UseBasicParsing
```

   範例輸出如下。

```
StatusCode        : 200
StatusDescription : OK
Content           : < h t m l > < b o d y > < b r / > < b r / > < m a r q u e e > < H
 1 > H e l l o
                      E K S ! ! ! < H 1 > < m a r q u e e > < / b o d y > < h t m l >
```

3. 從 Pod Shell 中檢視 Pod 的 DNS 伺服器。

```
Get-NetIPConfiguration
```

   縮寫輸出

```
InterfaceAlias       : vEthernet
[...]
IPv4Address          : 192.168.63.14
[...]
```

```
DNSServer            : 10.100.0.10
```

在先前的輸出中，`10.100.0.10` 會自動指派為讓所有 Pod 部署至叢集的 DNS 伺服器。

4. 透過輸入 `exit` 中斷從 Pod 的連線。
5. 完成範例應用程式後，您可以使用以下命令來移除範例命名空間、服務和部署。

```
kubectl delete namespace eks-sample-app
```

後續步驟

在部署範例應用程式之後，您可能需要嘗試下面列出的部分練習：

- 透過 Application Load Balancer 路由應用程式與 HTTP 流量
- 透過 Network Load Balancer 路由 TCP 與 UDP 流量

透過垂直 Pod 自動擴展器來調整 Pod 資源

Kubernetes 垂直 Pod 自動擴展器可自動調整保留給裝置的 CPU 和記憶體，以便 Pod 協助保持應用程式的「適當大小」。這項調整可以改善叢集資源使用率，並釋放 CPU 和記憶體給其他 Pod。本主題協助您將 Vertical Pod Autoscaler 部署到您的叢集，並驗證是否正常運作。

- 您擁有現有的 Amazon EKS 叢集。如果您沒有，則請參閱 開始使用。
- 您已安裝 Kubernetes 指標伺服器。如需詳細資訊，請參閱 the section called “指標伺服器”。
- 您所使用的 `kubectl` 用戶端已設定將與您的 Amazon EKS 叢集通訊。
- 裝置上已安裝 OpenSSL `1.1.1` 或更新版本。

部署 Vertical Pod Autoscaler

在本節中，您會將 Vertical Pod Autoscaler 部署到叢集。

1. 開啟終端機視窗，導覽至您要下載 Vertical Pod Autoscaler 原始程式碼的目錄。
2. 複製 kubernetes/autoscaler GitHub 儲存庫。

```
git clone https://github.com/kubernetes/autoscaler.git
```

3. 切換至 `vertical-pod-autoscaler` 目錄。

```
cd autoscaler/vertical-pod-autoscaler/
```

4. (選用) 如果您已部署另一個版本的 Vertical Pod Autoscaler，請使用下列命令將其移除。

```
./hack/vpa-down.sh
```

5. 如果您的節點對 `registry.k8s.io` 容器登錄檔無網際網路存取權，則您需要提取以下映像，並將其推送到您的私有儲存庫。如需有關如何提取映像以及將映像推送到您的私有儲存庫的詳細資訊，請參閱 the section called “將映像複製到儲存庫”。

```
registry.k8s.io/autoscaling/vpa-admission-controller:0.10.0
registry.k8s.io/autoscaling/vpa-recommender:0.10.0
registry.k8s.io/autoscaling/vpa-updater:0.10.0
```

   如果要將映像推送到私有 Amazon ECR 儲存庫，請使用您的登錄檔取代清單檔案中的 `registry.k8s.io`。使用您的帳戶 ID 取代 *`111122223333`*。將 *`region-code`* 取代為您的叢集所在的 AWS 區域。以下命令假定您將自己的儲存庫命名為與清單檔案中儲存庫相同的名稱。如果您將自己的儲存庫命名為其他名稱，您也需要變更清單檔案中儲存庫的名稱。

```
sed -i.bak -e 's/registry.k8s.io/111122223333.dkr.ecr.region-code.amazonaws.com/' ./
deploy/admission-controller-deployment.yaml
sed -i.bak -e 's/registry.k8s.io/111122223333.dkr.ecr.region-code.amazonaws.com/' ./
deploy/recommender-deployment.yaml
sed -i.bak -e 's/registry.k8s.io/111122223333.dkr.ecr.region-code.amazonaws.com/' ./
deploy/updater-deployment.yaml
```

6. 使用下列命令，將 Vertical Pod Autoscaler 部署到您的叢集。

```
./hack/vpa-up.sh
```

7. 確認已成功建立垂直 Pod 自動擴展器 Pod。

```
kubectl get pods -n kube-system
```

   範例輸出如下。

```
NAME                                        READY   STATUS    RESTARTS   AGE
[...]
```

```
metrics-server-8459fc497-kfj8w              1/1     Running   0          83m
vpa-admission-controller-68c748777d-ppspd   1/1     Running   0          7s
vpa-recommender-6fc8c67d85-gljpl            1/1     Running   0          8s
vpa-updater-786b96955c-bgp9d                1/1     Running   0          8s
```

測試您的 Vertical Pod Autoscaler 安裝

在本節中，您將部署範例應用程式，以驗證 Vertical Pod Autoscaler 是否正常運作。

1. 使用以下命令部署 `hamster.yaml` Vertical Pod Autoscaler 範例。

```
kubectl apply -f examples/hamster.yaml
```

2. 從 `hamster` 範例應用程式取得 Pod。

```
kubectl get pods -l app=hamster
```

範例輸出如下。

```
hamster-c7d89d6db-rglf5   1/1     Running   0          48s
hamster-c7d89d6db-znvz5   1/1     Running   0          48s
```

3. 描述其中一個 Pod，以檢視其 `cpu` 和 `memory` 的保留情況。使用在上一個步驟傳回的輸出中的其中一個 ID 取代 *`c7d89d6db-rglf5`*。

```
kubectl describe pod hamster-c7d89d6db-rglf5
```

範例輸出如下。

```
[...]
Containers:
  hamster:
    Container ID:  docker://
e76c2413fc720ac395c33b64588c82094fc8e5d590e373d5f818f3978f577e24
    Image:         registry.k8s.io/ubuntu-slim:0.1
    Image ID:      docker-pullable://registry.k8s.io/ubuntu-
slim@sha256:b6f8c3885f5880a4f1a7cf717c07242eb4858fdd5a84b5ffe35b1cf680ea17b1
    Port:          <none>
    Host Port:     <none>
    Command:
```

```
      /bin/sh
    Args:
      -c
      while true; do timeout 0.5s yes >/dev/null; sleep 0.5s; done
    State:          Running
      Started:      Fri, 27 Sep 2019 10:35:16 -0700
    Ready:          True
    Restart Count:  0
    Requests:
      cpu:        100m
      memory:     50Mi
[...]
```

您可以看到原始 Pod 保留 100 millicpu 的 CPU 和 50 MiB 的記憶體。在這個範例應用程式中，100 millicpu 小於 Pod 執行所需，因此 CPU 受限制。保留的記憶體也遠少於所需。垂直 Pod 自動擴展器 `vpa-recommender` 部署會分析 hamster Pod，以檢查 CPU 和記憶體需求是否合適。如果需要調整，`vpa-updater` 會以更新的值重新啟動 Pod。

4. 等待 `vpa-updater`啟動新的倉鼠 Pod。這需要一兩分鐘。您可使用下列命令來監控 Pod。

> **Note**
>
> 如果您不確定新的 Pod 是否已啟動，請將 Pod 名稱與之前的清單進行比較。當新的 Pod 啟動時，您會看到新的 Pod 名稱。

```
kubectl get --watch Pods -l app=hamster
```

5. 當新的倉鼠 Pod 啟動時，請加以描述並檢視更新的 CPU 和記憶體保留。

```
kubectl describe pod hamster-c7d89d6db-jxgfv
```

範例輸出如下。

```
[...]
Containers:
  hamster:
    Container ID:
 docker://2c3e7b6fb7ce0d8c86444334df654af6fb3fc88aad4c5d710eac3b1e7c58f7db
    Image:         registry.k8s.io/ubuntu-slim:0.1
```

```
    Image ID:      docker-pullable://registry.k8s.io/ubuntu-
slim@sha256:b6f8c3885f5880a4f1a7cf717c07242eb4858fdd5a84b5ffe35b1cf680ea17b1
    Port:          <none>
    Host Port:     <none>
    Command:
      /bin/sh
    Args:
      -c
      while true; do timeout 0.5s yes >/dev/null; sleep 0.5s; done
    State:          Running
      Started:      Fri, 27 Sep 2019 10:37:08 -0700
    Ready:          True
    Restart Count:  0
    Requests:
      cpu:        587m
      memory:     262144k
[...]
```

在先前的輸出中，您可以看到 cpu 保留已增加到 587 millicpu，這已是原始值的 5 倍。memory 已增加到 262,144 KB，大約是 250 MiB 或原始值的 5 倍。此 Pod 的資源不足，垂直 Pod 自動擴展器已依照更適當的值來更正預估值。

6. 描述 hamster-vpa 資源以檢視新的建議。

```
kubectl describe vpa/hamster-vpa
```

範例輸出如下。

```
Name:         hamster-vpa
Namespace:    default
Labels:       <none>
Annotations:  kubectl.kubernetes.io/last-applied-configuration:
                {"apiVersion":"autoscaling.k8s.io/
v1beta2","kind":"VerticalPodAutoscaler","metadata":{"annotations":{},"name":"hamster-
vpa","namespace":"d...
API Version:  autoscaling.k8s.io/v1beta2
Kind:         VerticalPodAutoscaler
Metadata:
  Creation Timestamp:  2019-09-27T18:22:51Z
  Generation:          23
  Resource Version:    14411
```

```
  Self Link:           /apis/autoscaling.k8s.io/v1beta2/namespaces/default/
verticalpodautoscalers/hamster-vpa
  UID:                 d0d85fb9-e153-11e9-ae53-0205785d75b0
Spec:
  Target Ref:
    API Version:  apps/v1
    Kind:         Deployment
    Name:         hamster
Status:
  Conditions:
    Last Transition Time:  2019-09-27T18:23:28Z
    Status:                True
    Type:                  RecommendationProvided
  Recommendation:
    Container Recommendations:
      Container Name:  hamster
      Lower Bound:
        Cpu:     550m
        Memory:  262144k
      Target:
        Cpu:     587m
        Memory:  262144k
      Uncapped Target:
        Cpu:     587m
        Memory:  262144k
      Upper Bound:
        Cpu:     21147m
        Memory:  387863636
Events:          <none>
```

7. 當您完成範例應用程式的實驗後，您可以使用下列命令將其刪除。

```
kubectl delete -f examples/hamster.yaml
```

使用 Horizontal Pod Autoscaler 擴展 Pod 部署

Kubernetes Horizontal Pod Autoscaler 會根據資源的 CPU 使用率，自動擴展部署、複寫控制器或複本集中的 Pod 數量。這可協助您的應用程式水平擴展以滿足增加的需求，或在不需要資源時縮減，從而釋出節點供其他應用程式使用。當您設定目標 CPU 使用率百分比時，Horizontal Pod Autoscaler 會擴展或縮減您的應用程式，以嘗試滿足該目標。

Horizontal Pod Autoscaler 是 Kubernetes 中的標準 API 資源，只需要在 Amazon EKS 叢集上安裝指標來源 (例如 Kubernetes 指標伺服器) 即可運作。您不需要在叢集上部署或安裝 Horizontal Pod Autoscaler，即可開始調整您的應用程式。如需詳細資訊，請參閱 Kubernetes 文件中的 Horizontal Pod Autoscaler。

使用此主題為您的 Amazon EKS 叢集準備 Horizontal Pod Autoscaler，並以範例應用程式驗證是否正常運作。

> **Note**
>
> 此主題是根據 Kubernetes 文件中的 Horizontal Pod Autoscaler 演練。

- 您擁有現有的 Amazon EKS 叢集。如果您沒有，則請參閱 開始使用。
- 您已安裝 Kubernetes 指標伺服器。如需詳細資訊，請參閱 the section called “指標伺服器”。
- 您所使用的 `kubectl` 用戶端已設定將與您的 Amazon EKS 叢集通訊。

執行 Horizontal Pod Autoscaler 測試應用程式

在本節中，您將部署範例應用程式，以驗證 Horizontal Pod Autoscaler 是否正常運作。

> **Note**
>
> 此範例是根據 Kubernetes 文件中的 Horizontal Pod Autoscaler 演練。

1. 使用下列命令部署簡單的 Apache Web 伺服器應用程式。

   ```
   kubectl apply -f https://k8s.io/examples/application/php-apache.yaml
   ```

   此 Apache Web 伺服器 Pod 獲得 500 millicpu CPU 的限制，而在連接埠 80 上提供服務。
2. 建立 `php-apache` 部署的 Horizontal Pod Autoscaler 資源。

   ```
   kubectl autoscale deployment php-apache --cpu-percent=50 --min=1 --max=10
   ```

   此命令會建立自動擴展器，其目標為 50% CPU 使用率用於部署，以及至少一個 Pod 和最多十個 Pod。當平均 CPU 負載低於 50% 時，自動擴展器會嘗試降低部署中的 Pod 數量，最少降低至一

個。當負載大於 50% 時，自動擴展器會嘗試增加部署中的 Pod 數量，最多十個。如需詳細資訊，請參閱 Kubernetes 文件中的 HorizontalPodAutoscaler 如何運作？。

3. 使用以下命令描述自動擴展器，以檢視其詳細資訊。

```
kubectl get hpa
```

範例輸出如下。

```
NAME         REFERENCE                  TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
php-apache   Deployment/php-apache   0%/50%    1         10        1          51s
```

如您所見，目前的 CPU 負載是 0%，因為伺服器上還沒有負載。Pod 數量已在其最低底限 (一個)，因此無法縮減。

4. 透過執行容器來建立 Web 伺服器的負載。

```
kubectl run -i \
    --tty load-generator \
    --rm --image=busybox \
    --restart=Never \
    -- /bin/sh -c "while sleep 0.01; do wget -q -O- http://php-apache; done"
```

5. 若要觀察向外擴展部署，請定期在不同的終端機中，從您執行上一個步驟的終端機執行下列命令。

```
kubectl get hpa php-apache
```

範例輸出如下。

```
NAME         REFERENCE                  TARGETS    MINPODS   MAXPODS   REPLICAS   AGE
php-apache   Deployment/php-apache   250%/50%   1         10        5          4m44s
```

複本計數可能需要一分多鐘的時間才會增加。只要實際 CPU 百分比高於目標百分比，複本計數就會增加，最多可增加至 10。在這種情況下，它會是 250%，所以 REPLICAS 的數量會繼續增加。

> **Note**
>
> 您可能需要幾分鐘才會看到複本計數達到其最大值。例如，如果 CPU 負載只需要 6 個複本維持在 50% 或以下，則負載將不會擴展超出 6 個複本。

6. 停止該負載。在您正在產生負載的終端窗口中，按住 Ctrl+C 鍵來停用負載。透過在正在觀察縮減的終端中再次執行下列命令，您會看到複本縮減回 1。

```
kubectl get hpa
```

範例輸出如下。

```
NAME         REFERENCE                TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
php-apache   Deployment/php-apache    0%/50%    1         10        1          25m
```

> **Note**
>
> 縮減的預設時間範圍為 5 分鐘，因此需要一些時間才會看到複本計數再次達到 1，即使目前的 CPU 百分比為 0。時間範圍是可修改的。如需詳細資訊，請參閱 Kubernetes 文件中的 Horizontal Pod Autoscaler。

7. 當您完成範例應用程式的實驗後，請刪除 php-apache 資源。

```
kubectl delete deployment.apps/php-apache service/php-apache
 horizontalpodautoscaler.autoscaling/php-apache
```

透過 Network Load Balancer 路由 TCP 與 UDP 流量

> **Note**
>
> 新功能：Amazon EKS 自動模式現可自動執行負載平衡的常規任務。如需詳細資訊，請參閱：
>
> - the section called “部署負載平衡器”
> - the section called “建立服務”

網路流量在 OSI 模型的 L4 處於負載平衡狀態。若要在 負載平衡應用程式流量L7，您可以部署 Kubernetes ingress，其會佈建 AWS Application Load Balancer。如需詳細資訊，請參閱the section called “應用程式負載平衡”。若要進一步了解兩種負載平衡類型之間的差異，請參閱 AWS 網站上的 Elastic Load Balancing 功能。

當您建立類型 Service的 Kubernetes 時LoadBalancer， AWS 雲端提供者負載平衡器控制器預設會建立 AWS Classic Load Balancer，但也可以建立 AWS Network Load Balancer。此控制器將來僅接收關鍵錯誤修正。如需使用 AWS 雲端提供者負載平衡器控制器的詳細資訊，請參閱 Kubernetes 文件中的AWS 雲端提供者負載平衡器控制器。本主題中不涉及其使用方式。

我們建議您使用版本 2.7.2或更新版本的 AWS Load Balancer Controller，而不是 AWS 雲端提供者負載平衡器控制器。The AWS Load Balancer 控制器會建立 AWS Network Load Balancer，但不會建立 AWS Classic Load Balancer。本主題的其餘部分是關於使用 AWS Load Balancer控制器。

AWS Network Load Balancer 可以將網路流量負載平衡到部署到 Amazon EC2 IP 和執行個體目標的 Pod、 AWS Fargate IP 目標或部署到 Amazon EKS 混合節點做為 IP 目標。如需詳細資訊，請參閱 GitHub 上的 AWS Load Balancer 控制器。

先決條件

您必須先符合下列需求，才能使用 AWS Load Balancer網路流量。

- 擁有現有的叢集。如果沒有現有叢集，請參閱 開始使用。如果您需要更新現有叢集的版本，請參閱 the section called “更新 Kubernetes 版本”。
- 在您的叢集上部署 AWS Load Balancer控制器。如需詳細資訊，請參閱the section called “ AWS Load Balancer控制器”。我們建議使用 2.7.2 版或更新版本。
- 至少有一個子網路。若在可用區域中找到多個標記的子網路，控制器會根據其子網路 ID 依詞典編纂順序來選擇第一個子網路。子網路必須至少有 8 個可用的 IP 地址。
- 如果您使用的是 AWS Load Balancer控制器版本 2.1.1或更早版本，子網路必須標記如下。如果使用版本 2.1.2 或更新版本，則此標籤是選用的。如果您有多個叢集在相同 VPC 中執行，或多個 AWS 服務在 VPC 中共用子網路，並希望進一步控制為每個叢集佈建負載平衡器的位置，您可能會想要標記子網路。如果您明確指定子網路 IDs做為服務物件上的註釋，則 Kubernetes 和 AWS Load Balancer控制器會直接使用這些子網路來建立負載平衡器。如果選擇使用此方法來佈建負載平衡器，而且可以略過下列私有和公有子網路標記要求，則不需要標記子網路。使用您的叢集名稱取代 *my-cluster*。
  - 索引鍵：kubernetes.io/cluster/<my-cluster>
  - 值：shared 或 owned
- 您的公有和私有子網路必須符合下列要求，除非您明確指定子網路 ID 作為服務或傳入物件上的註釋。如果您透過明確指定子網路 IDs做為服務或輸入物件的註釋來佈建負載平衡器，則 Kubernetes 和 AWS Load Balancer控制器會直接使用這些子網路來建立負載平衡器，而且不需要下列標籤。
  - 私有子網路：必須使用下列格式進行標記。這是為了讓 Kubernetes 和 AWS Load Balancer控制器知道子網路可用於內部負載平衡器。如果您在 2020 年 3 月 26 日之後使用 eksctl或 Amazon

EKS AWS CloudFormation 範本建立 VPC，則子網路會在建立時適當地加上標籤。如需 Amazon EKS AWS CloudFormation VPC 範本的詳細資訊，請參閱 the section called “建立 VPC”。

- 索引鍵：`kubernetes.io/role/internal-elb`
- 值：1

- 公有子網路：必須使用下列格式進行標記。這是為了讓 Kubernetes 僅以這些子網路用於外部負載平衡器，而非在每個可用區域 (依據子網路 ID 的字典順序) 中選擇公有子網路。如果您在 2020 年 3 月 26 日之後使用 `eksctl`或 Amazon EKS AWS CloudFormation 範本建立 VPC，則子網路會在建立時適當地加上標籤。如需 Amazon EKS AWS CloudFormation VPC 範本的詳細資訊，請參閱 the section called “建立 VPC”。
  - 索引鍵：`kubernetes.io/role/elb`
  - 值：1

如果沒有明確新增子網路角色標籤，則 Kubernetes Service 控制器會檢查叢集 VPC 子網路的路由表，以判定子網路是私有還是公有。建議您不要依賴此行為，而是明確地新增私有或公有角色標籤。The AWS Load Balancer 控制器不會檢查路由表，需要私有和公有標籤才能成功自動探索。

考量事項

- 負載平衡器的組態是由加入至該服務清單檔案的註釋所控制。使用 AWS Load Balancer控制器的服務註釋與使用 AWS 雲端提供者負載平衡器控制器的服務註釋不同。部署服務之前，請務必檢閱 AWS Load Balancer控制器的註釋。
- 使用適用於 Kubernetes 的 Amazon VPC CNI 外掛程式時， AWS Load Balancer控制器可以將平衡負載至 Amazon EC2 IP 或執行個體目標和 Fargate IP 目標。當使用替代相容 CNI 外掛程式時，控制器僅能針對執行個體目標進行負載平衡，除非您要對 Amazon EKS 混合節點進行負載平衡。對於混合節點，控制器可以對 IP 目標進行負載平衡。如需 Network Load Balancer 目標類型的詳細資訊，請參閱 Network Load Balancer 使用者指南中的目標類型。
- 如果要在建立負載平衡器時或之後將標籤新增至負載平衡器，請在服務規格中新增下列註釋。如需詳細資訊，請參閱 AWS Load Balancer控制器文件中AWS 的資源標籤。

```
service.beta.kubernetes.io/aws-load-balancer-additional-resource-tags
```

- 您可以透過新增以下注釋將彈性 IP 地址指派至 Network Load Balancer。使用彈性 IP 地址`Allocation IDs`的 取代範例值。`Allocation IDs` 的數量必須與負載平衡器所使用的子網路數量一致。如需詳細資訊，請參閱 AWS Load Balancer 控制器說明文件。

```
service.beta.kubernetes.io/aws-load-balancer-eip-allocations: eipalloc-
xxxxxxxxxxxxxxxx,eipalloc-yyyyyyyyyyyyyyyyy
```

- Amazon EKS 會為用戶端流量新增一個傳入規則到節點的安全群組，並為 VPC 中的每個負載平衡器子網路新增一個規則，以對您所建立的每個 Network Load Balancer 進行運作狀態檢查。如果 Amazon EKS 嘗試建立超過安全群組允許之規則數量上限配額的規則，則 `LoadBalancer` 類型的服務部署可能會失敗。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中 Amazon VPC 配額的安全群組。請考慮下列選項，將超過安全群組規則數量上限的機會降至最低：
  - 請求提高每個安全群組規則的配額。如需詳細資訊，請參閱《Service Quotas 使用者指南》中的請求提高配額。
  - 使用 IP 目標，而不是執行個體目標。使用 IP 目標時，您可以共用相同目標連接埠的規則。您可以使用註釋手動指定負載平衡器子網路。如需詳細資訊，請參閱 GitHub 上的注釋。
  - 使用傳入，而不是類型服務 `LoadBalancer` 將流量傳送至您的服務。The AWS Application Load Balancer 所需的規則少於 Network Load Balancer。您可在多個輸入之間共用 ALB。如需詳細資訊，請參閱the section called “應用程式負載平衡”。您不能在多個服務之間共用 Network Load Balancer。
  - 將叢集部署到多個帳戶。
- 如果您的 Pod 在 Windows 上執行，則在 Amazon EKS 叢集中，具有負載平衡器的單一服務最多可支援 1024 個後端 Pod。每個 Pod 都有自己的唯一 IP 地址。
- 我們建議僅使用負載平衡器控制器建立新的 Network AWS Load Balancer。嘗試取代使用 AWS 雲端提供者負載平衡器控制器建立的現有 Network Load Balancer 可能會導致多個 Network Load Balancer，這可能會導致應用程式停機。

建立 Network Load Balancer

您可以建立具有 IP 或執行個體目標的 Network Load Balancer。

建立 Network Load Balancer — IP 目標

- 您可以將 IP 目標與部署到 Amazon EC2 節點、Fargate 或 Amazon EKS 混合節點的 Pod 搭配使用。您的 Kubernetes 服務必須以 `LoadBalancer` 類型建立。如需詳細資訊，請參閱 Kubernetes 文件中的 LoadBalancer 類型。

  若要建立使用 IP 目標的負載平衡器，請將下列註解新增至服務清單檔案，然後部署您的服務。`external` 的值`aws-load-balancer-type`是導致 AWS Load Balancer控制器而非 AWS 雲

端提供者負載平衡器控制器建立 Network Load Balancer 的原因。您可以檢視具有注釋的範例服務清單檔案。

```
service.beta.kubernetes.io/aws-load-balancer-type: "external"
service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: "ip"
```

> **Note**
>
> 如果您要對 IPv6 Pod 進行負載平衡，請新增以下註釋。您只能在 IPv6 上負載平衡至 IP 目標，而不能負載平衡執行個體目標。如果沒有此註釋，負載平衡將在 IPv4 上執行。

```
service.beta.kubernetes.io/aws-load-balancer-ip-address-type: dualstack
```

依預設，使用 `internal` `aws-load-balancer-scheme` 建立 Network Load Balancer。您可以在叢集 VPC 的任何子網路中啟動 Network Load Balancer，包括建立叢集時未指定的子網路。

Kubernetes 會檢查您子網路的路由表，以判別其為公有或私有。公有子網路具備使用網際網路閘道直接通向網際網路的路由，而私有子網路則不然。

若要在公有子網路中建立 Network Load Balancer 對 Amazon EC2 節點進行負載平衡 (Fargate 只能為私有)，請指定具有以下註釋的 `internet-facing`：

```
service.beta.kubernetes.io/aws-load-balancer-scheme: "internet-facing"
```

> **Note**
>
> 仍支援 `service.beta.kubernetes.io/aws-load-balancer-type: "nlb-ip"` 註釋以提供回溯相容性。不過，建議您針對新的負載平衡器使用先前的註釋，而不是 `service.beta.kubernetes.io/aws-load-balancer-type: "nlb-ip"`。

> **Important**
>
> 建立服務後，請勿編輯註釋。如果需要進行修改，請刪除服務物件，並使用此註釋所需的值重新建立服務物件。

建立 Network Load Balancer — 執行個體目標

- AWS 雲端提供者負載平衡器控制器只會使用執行個體目標建立 Network Load Balancer。版本 2.2.0 AWS Load Balancer控制器也會建立具有執行個體目標的 Network Load Balancer。我們建議您使用它，而不是 AWS 雲端提供者負載平衡器控制器，來建立新的 Network Load Balancer。您可以將 Network Load Balancer 執行個體目標與部署至 Amazon EC2 節點 (而非 Fargate) 的 Pod 搭配使用。若要在部署到 Fargate 的 Pod 之間負載平衡網路流量，則必須使用 IP 目標。

  若要將 Network Load Balancer 部署至私有子網路，則您的服務規格必須具有下列註釋。您可以檢視具有注釋的範例服務清單檔案。external 的值aws-load-balancer-type是導致 AWS Load Balancer控制器而非 AWS 雲端提供者負載平衡器控制器建立 Network Load Balancer 的原因。

  ```
  service.beta.kubernetes.io/aws-load-balancer-type: "external"
  service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: "instance"
  ```

  依預設，使用 internal aws-load-balancer-scheme 建立 Network Load Balancer。若為內部 Network Load Balancer，您的 Amazon EKS 叢集必須設定至少使用 VPC 中的一個私有子網路。Kubernetes 會檢查您子網路的路由表，以判別其為公有或私有。公有子網路具備使用網際網路閘道直接通向網際網路的路由，而私有子網路則不然。

  如果您想要在公有子網路中建立 Network Load Balancer 以將負載平衡至 Amazon EC2 節點，internet-facing請使用下列註釋來指定 ：

  ```
  service.beta.kubernetes.io/aws-load-balancer-scheme: "internet-facing"
  ```

  > **Important**
  >
  > 建立服務後，請勿編輯註釋。如果需要進行修改，請刪除服務物件，並使用此註釋所需的值重新建立服務物件。

(選用) 部署範例應用程式

- 叢集 VPC 中必須至少有一個公有或私有子網路。
- 在您的叢集上部署 AWS Load Balancer控制器。如需詳細資訊，請參閱the section called “ AWS Load Balancer控制器”。我們建議使用 2.7.2 版或更新版本。

1. 若要部署至 Fargate，請確保 VPC 中有一個可用的私有子網路，並建立一個 Fargate 設定檔。如果您沒有部署到 Fargate，請略過此步驟。您可以透過執行下列命令來建立描述檔，也可以使用命令中 name 和 namespace 相同的值藉由 AWS 管理主控台 進行。使用自己的取代範例值。

```
eksctl create fargateprofile \
    --cluster my-cluster \
    --region region-code \
    --name nlb-sample-app \
    --namespace nlb-sample-app
```

2. 部署範例應用程式。

   a. 建立應用程式的命名空間。

```
kubectl create namespace nlb-sample-app
```

   b. 將下列內容儲存到電腦上名為 sample-deployment.yaml 的檔案中。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nlb-sample-app
  namespace: nlb-sample-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: public.ecr.aws/nginx/nginx:1.23
          ports:
            - name: tcp
              containerPort: 80
```

   c. 將清單檔案套用至叢集。

```
kubectl apply -f sample-deployment.yaml
```

3. 使用負載平衡至 IP 目標的網際網路取向的 Network Load Balancer 建立服務。

   a. 將下列內容儲存到電腦上名為 `sample-service.yaml` 的檔案中。若要部署至 Fargate 節點，請移除 `service.beta.kubernetes.io/aws-load-balancer-scheme: internet-facing` 行。

```
apiVersion: v1
kind: Service
metadata:
  name: nlb-sample-service
  namespace: nlb-sample-app
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-type: external
    service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: ip
    service.beta.kubernetes.io/aws-load-balancer-scheme: internet-facing
spec:
  ports:
    - port: 80
      targetPort: 80
      protocol: TCP
  type: LoadBalancer
  selector:
    app: nginx
```

   b. 將清單檔案套用至叢集。

```
kubectl apply -f sample-service.yaml
```

4. 確認服務已部署。

```
kubectl get svc nlb-sample-service -n nlb-sample-app
```

   範例輸出如下。

```
NAME             TYPE           CLUSTER-IP         EXTERNAL-IP
                                                 PORT(S)        AGE
sample-service   LoadBalancer   10.100.240.137   k8s-nlbsampl-nlbsampl-xxxxxxxxxx-
xxxxxxxxxxxxxxxx.elb.region-code.amazonaws.com  80:32400/TCP   16h
```

> **Note**
>
> *10.100.240.137* 和 *xxxxxxxxxx-xxxxxxxxxxxxxxxx* 的值將與範例輸出不同（它們對於您的負載平衡器是唯一的），而且 *us-west-2* 可能有所不同，取決於叢集所在的 AWS 區域。

5. 開啟 Amazon EC2 AWS 管理主控台。選取左側導覽窗格中的 Target Groups (目標群組) (在 Load Balancing (負載平衡) 下)。在名稱資料欄中選取目標群組名稱，其負載平衡器資料欄中的值必須與上一個步驟中 EXTERNAL-IP 資料欄的部分輸出值名稱相符。例如，若輸出值名稱與先前的輸出值名稱相同，請選取名為 k8s-default-samplese-*xxxxxxxxxx* 的目標群組。Target type (目標類型) 為 IP，因為其已在範例服務清單檔案中指定。
6. 選取 Target group (目標群組)，然後選擇 Targets (目標) 索引標籤。在 Registered targets (已註冊目標) 下，您應該會看到在上一個步驟中部署之三個複本的三個 IP 地址。待所有目標狀態變為 healthy (狀態良好) 再繼續。可能需要幾分鐘的時間，所有目標才能變為 healthy。目標在變更為 healthy 狀態前，可能處於 unhealthy 狀態。
7. 將流量傳送至服務，使用 EXTERNAL-IP 在上一個步驟的輸出中傳回的值，來取代 *xxxxxxxxxx-xxxxxxxxxxxxxxxx* 和 *us-west-2*。如果您部署至私有子網路，則需要從 VPC 內的裝置 (例如堡壘主機) 檢視頁面。如需詳細資訊，請參閱 AWS上的 Linux 堡壘主機。

```
curl k8s-default-samplese-xxxxxxxxxx-xxxxxxxxxxxxxxxx.elb.region-code.amazonaws.com
```

範例輸出如下。

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
[...]
```

8. 在完成範例部署、服務和命名空間後，請將其移除。

```
kubectl delete namespace nlb-sample-app
```

透過 Application Load Balancer 路由應用程式與 HTTP 流量

> **Note**
>
> 新功能：Amazon EKS 自動模式現可自動執行負載平衡的常規任務。如需詳細資訊，請參閱：
>
> - the section called “部署負載平衡器”
> - the section called “建立傳入類別”

當您建立 Kubernetes 時`ingress`，會佈建 an AWS Application Load Balancer (ALB)，以平衡應用程式流量。如需進一步了解，請參閱《Application Load Balancer 使用者指南》中的什麼是 Application Load Balancer？和 Kubernetes 文件中的傳入。ALBs可與部署到節點或 AWS Fargate 的 Pod 搭配使用。您可以將 ALB 部署到公有或私有子網路。

應用程式流量會在 OSI 模型的 L7 平衡。若要負載平衡 L4 的網路流量，您可以部署 `LoadBalancer` 類型的 Kubernetes `service`。此類型佈建 AWS Network Load Balancer。如需詳細資訊，請參閱the section called “網路負載平衡”。若要進一步了解兩種負載平衡類型之間的差異，請參閱 AWS 網站上的 Elastic Load Balancing 功能。

先決條件

在將應用程式流量負載平衡到應用程式之前，您必須符合以下要求。

- 擁有現有的叢集。如果沒有現有叢集，請參閱 開始使用。如果您需要更新現有叢集的版本，請參閱 the section called “更新 Kubernetes 版本”。
- 在您的叢集上部署 AWS Load Balancer控制器。如需詳細資訊，請參閱the section called “ AWS Load Balancer控制器”。我們建議使用 `2.7.2` 版或更新版本。
- 至少兩個子網路位於不同的可用區域。 AWS Load Balancer控制器會從每個可用區域選擇一個子網路。當在可用區域中找到多個標記的子網路時，控制器會根據其子網路 ID 依詞典編纂順序來選擇子網路。每個子網路必須至少有 8 個可用的 IP 地址。

  如果您使用的是連接到工作者節點的多個安全群組，則必須僅標記一個安全群組，如下所示。使用您的叢集名稱取代 *`my-cluster`*。

  - 索引鍵：`kubernetes.io/cluster/<my-cluster>`
  - 值：`shared` 或 `owned`

- 如果您使用的是 AWS Load Balancer控制器版本 2.1.1或更早版本，子網路必須以下列格式進行標記。如果使用版本 2.1.2 或更新版本，則不一定要加上標籤。不過，如果出現下列任何情況，我們建議您標記子網路。您有多個叢集正在相同的 VPC 中執行，或有多個 AWS 服務在 VPC 中共用子網路。或者，您希望更好地控制為每個叢集佈建負載平衡器的位置。使用您的叢集名稱取代 *my-cluster*。
  - 索引鍵：kubernetes.io/cluster/<my-cluster>
  - 值：shared 或 owned
- 您的公有和私有子網路必須符合下列要求，除非明確指定子網路 ID 作為服務或傳入物件上的註釋。假設您透過明確指定子網路 ID 作為服務或傳入物件上的註釋來佈建負載平衡器。在此情況下，Kubernetes 和 AWS 負載平衡器控制器會直接使用這些子網路來建立負載平衡器，而不需要下列標籤。
  - 私有子網路：必須使用下列格式進行標記。這是為了讓 Kubernetes 和 AWS 負載平衡器控制器知道子網路可用於內部負載平衡器。如果您在 2020 年 3 月 26 日之後使用 eksctl或 Amazon EKS AWS CloudFormation 範本建立 VPC，子網路會在建立時適當標記。如需 Amazon EKS AWS CloudFormation VPC 範本的詳細資訊，請參閱 the section called “建立 VPC”。
    - 索引鍵：kubernetes.io/role/internal-elb
    - 值：1
  - 公有子網路：必須使用下列格式進行標記。這是為了讓 Kubernetes 知道只使用為外部負載平衡器指定的子網路。如此一來，Kubernetes 不會在每個可用區域中選擇公有子網路 (根據其子網路 ID 依詞典編纂順序)。如果您在 2020 年 3 月 26 日之後使用 eksctl或 Amazon EKS AWS CloudFormation 範本建立 VPC，子網路會在建立時適當標記。如需 Amazon EKS AWS CloudFormation VPC 範本的詳細資訊，請參閱 the section called “建立 VPC”。
    - 索引鍵：kubernetes.io/role/elb
    - 值：1

  如果沒有明確新增子網路角色標籤，則 Kubernetes 服務控制器會檢查叢集 VPC 子網路的路由表。這是為了判斷子網路是私有還是公有。建議您不要依賴此行為。而是明確地新增私有或公有角色標籤。The AWS Load Balancer 控制器不會檢查路由表。其還需要存在私有和公有標籤才能成功自動探索。
- 每當在具有 kubernetes.io/ingress.class: alb註釋的叢集上建立 Kubernetes 輸入 AWS 資源時，AWS Load Balancer控制器都會建立 ALBs 和必要的支援資源。傳入資源設定 ALB 在叢集內透過 HTTP 或 HTTPS 路由到不同的 Pod。若要確保您的輸入物件使用 AWS Load Balancer控制器，請將下列註釋新增至 Kubernetes 輸入規格。如需詳細資訊，請參閱 GitHub 上的傳入規格。

```
annotations:
```

```
    kubernetes.io/ingress.class: alb
```

> **Note**
>
> 如果您正在負載平衡至 IPv6 Pod，請新增以下註釋到您的傳入規格。您只能在 IPv6 上負載平衡至 IP 目標，而不能負載平衡執行個體目標。如果沒有此註釋，負載平衡將在 IPv4 上執行。

```
alb.ingress.kubernetes.io/ip-address-type: dualstack
```

- AWS Load Balancer控制器支援下列流量模式：
  - 執行個體 – 將叢集中的節點註冊作為 ALB 的目標。到達 ALB 的流量被路由到服務的 NodePort，然後代理到您的 Pod。這是預設的流量模式。您也可以使用 `alb.ingress.kubernetes.io/target-type: instance` 註釋明確指定它。

    > **Note**
    >
    > 您的 Kubernetes 服務必須指定要使用此流量模式的 NodePort 或 LoadBalancer 類型。

  - IP：註冊 Pod 作為 ALB 的目標。到達 ALB 的流量會直接路由到服務的 Pod。您必須指定 `alb.ingress.kubernetes.io/target-type: ip` 註釋才能使用此流量模式。當目標 Pod 在 Fargate 或 Amazon EKS 混合節點上執行時，必須使用 IP 目標類型。
- 要標記由控制器建立的 ALB，請新增以下註釋至控制器：`alb.ingress.kubernetes.io/tags`。如需 AWS Load Balancer控制器支援的所有可用註釋清單，請參閱 GitHub 上的傳入註釋。
- 升級或降級 ALB 控制器版本可能會導致依賴控制器的功能發生重大變化。如需有關每個版本中引入的重大變更的詳細資訊，請參閱 GitHub 上的 ALB 控制器版本備註。

透過輸入群組重複使用 ALB

您可使用 IngressGroups 跨多個服務資源共用 Application Load Balancer。

若要將傳入加入群組，請將下列註解新增至 Kubernetes 傳入資源規格。

```
alb.ingress.kubernetes.io/group.name: my-group
```

群組名稱必須：

- 是 63 個字元或以下的長度。
- 由小寫字母、數字、- 和 . 組成
- 開頭和結尾為字母或數字。

控制器會自動合併相同傳入群組中所有傳入的傳入規則。控制器透過單個 ALB 提供支援。在傳入中定義的大多數註解僅適用於該傳入定義的路徑。根據預設，傳入資源不屬於任何傳入群組。

> **Warning**
>
> 潛在的安全風險
> 僅當所有具有建立或修改傳入資源之 RBAC 許可的 Kubernetes 使用者都在相同的信任界限內時，才為傳入指定傳入群組。如果您使用群組名稱新增註釋，其他 Kubernetes 使用者可以建立或修改其傳入，使其屬於同一傳入群組。這樣做可能會產生不良行為，例如以較高優先順序規則覆寫現有規則。

您可以新增傳入資源的訂單編號。

```
alb.ingress.kubernetes.io/group.order: '10'
```

該編號可以是 1 到 1000 之間的數字。系統會先評估相同傳入群組中所有傳入數量的最低數量。系統會使用 0 值評估沒有此注釋的所有傳入。具有較高數量的重複規則可以覆寫具有較低數量的規則。根據預設，相同輸入群組內輸入之間的規則順序會根據命名空間和名稱以詞典方式決定。

> **Important**
>
> 確定相同傳入群組中的每個傳入都有唯一的優先順序編號。您不能在整個傳入中有重複的訂單編號。

(選用) 部署範例應用程式

- 叢集 VPC 中必須至少有一個公有或私有子網路。
- 在您的叢集上部署 AWS Load Balancer控制器。如需詳細資訊，請參閱the section called “ AWS Load Balancer控制器”。我們建議使用 2.7.2 版或更新版本。

您可以在具有 Amazon EC2 節點、Fargate Pod 或兩者兼有的叢集上執行範例應用程式。

1. 如果您沒有部署到 Fargate，請略過此步驟。如果您要部署到 Fargate，請建立 Fargate 描述檔。您可以透過執行下列命令來建立描述檔，也可以使用命令中 `name` 和 `namespace` 相同的值藉由 AWS 管理主控台 進行。使用自己的取代範例值。

```
eksctl create fargateprofile \
    --cluster my-cluster \
    --region region-code \
    --name alb-sample-app \
    --namespace game-2048
```

2. 將遊戲 2048 部署為範例應用程式，以確認 AWS Load Balancer控制器建立 ALB AWS 作為輸入物件的結果。針對您正在部署的子網路類型，完成相關的步驟。

   a. 如果要部署到使用 IPv6 系列建立的叢集中的 Pod，請跳到下一步。

   - 公有::

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.14.1/docs/examples/2048/2048_full.yaml
```

   - 私有::

     A. 下載清單檔案。

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.14.1/docs/examples/2048/2048_full.yaml
```

     B. 編輯檔案並尋找顯示 `alb.ingress.kubernetes.io/scheme: internet-facing` 的行。

     C. 將`######`變更為 `internal` 並儲存檔案。

     D. 將清單檔案套用至叢集。

```
kubectl apply -f 2048_full.yaml
```

   b. 如果要部署到使用 IPv6 系列建立的叢集中的 Pod，請完成下列步驟。

      i. 下載清單檔案。

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.14.1/docs/examples/2048/2048_full.yaml
```

ii. 在編輯器中開啟檔案，然後在傳入規格中新增以下一行註釋。

```
alb.ingress.kubernetes.io/ip-address-type: dualstack
```

iii. 如果您正在負載平衡至內部 Pod，而不是面向網際網路的 Pod，請變更該行，即將 `alb.ingress.kubernetes.io/scheme: internet-facing` 變更為 `alb.ingress.kubernetes.io/scheme: internal`

iv. 儲存檔案。

v. 將清單檔案套用至叢集。

```
kubectl apply -f 2048_full.yaml
```

3. 幾分鐘後，使用下列指令來驗證傳入資源已建立。

```
kubectl get ingress/ingress-2048 -n game-2048
```

範例輸出如下。

```
NAME           CLASS    HOSTS   ADDRESS
                      PORTS   AGE
ingress-2048   <none>   *       k8s-game2048-ingress2-xxxxxxxxxx-yyyyyyyyyy.region-
code.elb.amazonaws.com   80      2m32s
```

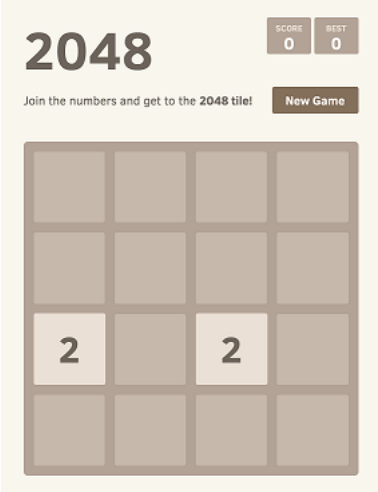
> **Note**
>
> 若在私有子網路中建立負載平衡器，則先前輸出中位於 ADDRESS 下的值前面會加上 `internal-`。

如果您的輸入在幾分鐘後未成功建立，請執行下列命令來檢視 AWS Load Balancer控制器日誌。這些日誌可能包含錯誤訊息，您可用於診斷部署的相關問題。

```
kubectl logs -f -n kube-system -l app.kubernetes.io/instance=aws-load-balancer-
controller
```

1. 如果您部署公有子網路，請開啟瀏覽器，再導覽至先前指令輸出中的 ADDRESS URL 查看範例應用程式。如果沒有看到任何內容，請重新整理瀏覽器，然後再試一次。如果您部署至私有子網路，則

需要從 VPC 內的裝置 (例如堡壘主機) 檢視頁面。如需詳細資訊，請參閱 AWS上的 Linux 堡壘主機。



2. 當您完成範例應用程式的實驗時，請執行下列其中一個命令將其刪除。

   - 如果您應用了清單檔案，而不是應用下載的複本，請使用以下命令。

     ```
     kubectl delete -f https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.14.1/docs/examples/2048/2048_full.yaml
     ```

   - 如果您下載並編輯清單檔案，請使用下列命令。

     ```
     kubectl delete -f 2048_full.yaml
     ```

限制可指派給服務的外部 IP 位址

Kubernetes 服務可透過以下方式從叢集內部連線：

- Kubernetes 自動指派的叢集 IP 地址
- 您為服務規則中的 `externalIPs` 屬性指定的 IP 地址。外部 IP 地址不是由 Kubernetes 管理，是叢集管理員的責任。使用 `externalIPs` 指定的外部 IP 地址不同於雲端提供商指派給類型服務 `LoadBalancer` 的外部 IP 地址。

如需進一步了解 Kubernetes 服務，請參閱 Kubernetes 文件中的服務。您可以限制可以在服務規格中為 `externalIPs` 指定的 IP 地址。

1. 部署 `cert-manager` 以管理網路掛鉤憑證。如需詳細資訊，請參閱 cert-manager 文件。

```
kubectl apply -f https://github.com/jetstack/cert-manager/releases/download/v1.5.4/
cert-manager.yaml
```

2. 確認是否正在執行 cert-manager Pod。

```
kubectl get pods -n cert-manager
```

範例輸出如下。

```
NAME                                       READY   STATUS    RESTARTS   AGE
cert-manager-58c8844bb8-nlx7q              1/1     Running   0          15s
cert-manager-cainjector-745768f6ff-696h5   1/1     Running   0          15s
cert-manager-webhook-67cc76975b-4v4nk      1/1     Running   0          14s
```

3. 檢閱您現有的服務，以確保其中沒有指派給這些服務的外部 IP 位址，而這些地址不包含在要限制地址的 CIDR 區塊中。

```
kubectl get services -A
```

範例輸出如下。

```
NAMESPACE                     NAME                                      TYPE
 CLUSTER-IP       EXTERNAL-IP     PORT(S)         AGE
cert-manager                  cert-manager                              ClusterIP
 10.100.102.137   <none>          9402/TCP        20m
cert-manager                  cert-manager-webhook                      ClusterIP
 10.100.6.136     <none>          443/TCP         20m
default                       kubernetes                                ClusterIP
 10.100.0.1       <none>          443/TCP         2d1h
externalip-validation-system  externalip-validation-webhook-service     ClusterIP
 10.100.234.179   <none>          443/TCP         16s
kube-system                   kube-dns                                  ClusterIP
 10.100.0.10      <none>          53/UDP,53/TCP   2d1h
my-namespace                  my-service                                ClusterIP
 10.100.128.10    192.168.1.1     80/TCP          149m
```

如果有任何值是不位於要限制存取之區塊內的 IP 位址，則您需要將地址變更為位於區塊內，然後重新部署服務。例如，上一個輸出中的 my-service 服務具有指派給其的外部 IP 位址，而該地址不位於步驟 5 中的 CIDR 區塊範例內。

4. 下載外部 IP Webhook 清單檔案。您也可以檢視 GitHub 上的 Webhook 來源程式碼

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/docs/externalip-webhook.yaml
```

5. 指定 CIDR 區塊。在編輯器中開啟下載的檔案，並在下列這幾行的開頭移除 \#。

```
#args:
#- --allowed-external-ip-cidrs=10.0.0.0/8
```

使用您自己的 CIDR 區塊取代 10.0.0.0/8。您可以指定需要的區塊，不限數量。如果指定多個區塊，請在區塊之間新增逗號。

6. 如果您的叢集不在 us-west-2 AWS 區域中，請使用下列命令取代 檔案amazonaws.com中的 us-west-2602401143452、 和 。執行命令之前，請將 *region-code* 和 *111122223333* 取代為 Amazon EKS 附加元件檢視 Amazon 容器映像登錄檔清單中 AWS 的區域值。

```
sed -i.bak -e 's|602401143452|111122223333|' externalip-webhook.yaml
sed -i.bak -e 's|us-west-2|region-code|' externalip-webhook.yaml
sed -i.bak -e 's|amazonaws.com||' externalip-webhook.yaml
```

7. 將清單檔案套用至叢集。

```
kubectl apply -f externalip-webhook.yaml
```

嘗試使用為 指定的 IP 地址將服務部署到您的叢集externalIPs，而該 IP 地址未包含在您在指定 CIDR 區塊步驟中指定的區塊中，將會失敗。

將容器映像從一個儲存庫複製到另一個儲存庫

本主題介紹如何從節點無法存取的儲存庫中提取容器映像，並將該映像推送到節點可存取的儲存庫。您可以將映像推送到 Amazon ECR 或節點可存取的替代儲存庫。

- 電腦上已安裝和設定 Docker 引擎。如需相關說明，請參閱 Docker 文件中的安裝 Docker 引擎。
- 在您的裝置或 AWS CloudShell 上安裝和設定的 AWS 命令列界面 (AWS CLI) 版本 1.27.160 2.12.3或更新版本。若要檢查您目前的版本，請使用 aws --version | cut -d / -f2 | cut -d ' ' -f1。適用於 macOS 的 yum、 apt-get或 Homebrew 等套件管理員通常是最新版本 CLI AWS 後面的幾個版本。若要安裝最新版本，請參閱《 AWS 命令列界面使用者指南》中的使用 aws 設定安裝 和快速組態。 https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-quickstart.html#cli-configure-quickstart-config在 AWS CloudShell 中安裝的 AWS CLI 版本也可能是

最新版本後面的幾個版本。若要更新它，請參閱《CloudShell AWS 使用者指南》中的將 CLI 安裝到您的主目錄。 AWS CloudShell

- 如果您希望節點透過 Amazon 網路推送容器映像到私有 Amazon ECR 儲存庫，或從中提取容器映像，則需使用適用於 Amazon ECR 的介面 VPC 端點。如需詳細資訊，請參閱《Amazon 彈性容器登錄檔使用者指南》中的建立適用於 Amazon ECR 的 VPC 端點。

請完成以下步驟，以從某個儲存庫中提取容器映像，並將其推送到您的儲存庫。在本主題所提供的以下範例中，會提取適用於 Kubernetes 專用 Amazon VPC CNI 外掛程式指標協助程式的映像。當您依照以下步驟操作時，請務必使用您的值取代 example values。

1. 如果您尚未擁有 Amazon ECR 儲存庫或其他儲存庫，請先建立您的節點可存取的儲存庫。以下命令會建立 Amazon ECR 私有儲存庫。Amazon ECR 私有儲存庫名稱必須以字母開頭。名稱僅能包含小寫字母、數字、連字號 (-)、底線 (_) 和正斜線 (/)。如需詳細資訊，請參閱《Amazon 彈性容器登錄檔使用者指南》中的建立私有儲存庫。

   您可以使用所選的任何項目取代 *cni-metrics-helper*。最佳實務是為每個映像建立個別儲存庫。建議您採取此做法，因為映像標籤在儲存庫中必須是唯一的。使用 Amazon ECR 支援的AWS 區域取代 *region-code*。

   ```
   aws ecr create-repository --region region-code --repository-name cni-metrics-helper
   ```

2. 判定節點需要提取的映像的登錄檔、儲存庫和標籤 (選用)。此資訊採用 `registry/repository[:tag]` 格式。

   許多有關於安裝映像的 Amazon EKS 主題都要求您套用清單檔案檔案或使用 Helm Chart 來安裝映像。然而在套用清單檔案檔案或安裝 Helm Chart 之前，請檢視資訊清單的內容或圖表的 `values.yaml` 檔案。如此一來，您可以判定要提取的登錄檔、儲存庫和標籤。

   例如，您可在適用於 Kubernetes 專用 Amazon VPC CNI 外掛程式指標協助程式的資訊清單檔案中找到以下這行。登錄檔為 `602401143452.dkr.ecr.us-west-2.amazonaws.com`，這是一個 Amazon ECR 私有登錄檔。儲存庫為 `cni-metrics-helper`。

   ```
   image: "602401143452.dkr.ecr.us-west-2.amazonaws.com/cni-metrics-helper:v1.12.6"
   ```

   您可能會看到如以下映像位置的變化：

   - 僅有 `repository-name:tag`。在此案例中，`docker.io` 通常是未經指定的登錄檔，因為在沒有指定登錄檔的情況下，依照預設 Kubernetes 會將其置於儲存庫名稱的前面。

- `repository-name/repository-namespace/repository:tag`。儲存庫命名空間為選用，但有時會由儲存庫擁有者指定，以將映像分類。例如，Amazon ECR Public Gallery 中的所有 Amazon EC2 映像都使用 `aws-ec2` 命名空間。

  在使用 Helm 安裝映像之前，請檢視 Helm `values.yaml` 檔案，以判定映像位置。例如，適用於 Kubernetes 專用 Amazon VPC CNI 外掛程式指標協助程式 values.yaml 檔案包含以下行。

```
image:
  region: us-west-2
  tag: v1.12.6
  account: "602401143452"
  domain: "amazonaws.com"
```

3. 提取清單檔案檔案中指定的容器映像。

   a. 如果您要從公有登錄檔 (例如 Amazon ECR 公共映像庫) 進行提取，則可跳到下一個子步驟，因為不需要身分驗證。在此範例中，您將向包含 CNI 指標協助程式映像的儲存庫的 Amazon ECR 私有登錄檔進行身分驗證。Amazon EKS 會在檢視 Amazon EKS 附加元件的 Amazon 容器映像登錄檔中列出的每個登錄檔中維護映像。您可使用不同登錄檔的資訊取代 *602401143452* 和 *region-code*，以對任何登錄檔進行身分驗證。每個支援 Amazon EKS 的AWS 區域中都有一個個別的登錄檔。

```
aws ecr get-login-password --region region-code | docker login --username AWS --
password-stdin 602401143452.dkr.ecr.region-code.amazonaws.com
```

   b. 提取映像。在此範例中，您將從上一個子步驟中所驗證的登錄表中提取。將 *602401143452* 和 *region-code* 取代為您在上一個子步驟中提供的資訊。

```
docker pull 602401143452.dkr.ecr.region-code.amazonaws.com/cni-metrics-
helper:v1.12.6
```

4. 標記與登錄檔、儲存庫和標籤一起提取的映像。以下範例假設您從清單檔案檔案中提取映像，並將其推送到您在第一個步驟中建立的 Amazon ECR 私有儲存庫。使用您的帳戶 ID 取代 *111122223333*。將 *region-code* 取代為您建立 Amazon ECR 私有儲存庫 AWS 的區域。

```
docker tag cni-metrics-helper:v1.12.6 111122223333.dkr.ecr.region-code.amazonaws.com/
cni-metrics-helper:v1.12.6
```

5. 對登錄檔進行身分驗證。在此範例中，您需對您在第一個步驟中建立的 Amazon ECR 私有登錄檔進行身分驗證。如需詳細資訊，請參閱《Amazon 彈性容器登錄檔使用者指南》中的登錄檔身分驗證。

```
aws ecr get-login-password --region region-code | docker login --username AWS --
password-stdin 111122223333.dkr.ecr.region-code.amazonaws.com
```

6. 將映像推送到您的儲存庫。在此範例中，您需將映像推送到您在第一個步驟中建立的 Amazon ECR 私有儲存庫。如需詳細資訊，請參閱《Amazon 彈性容器登錄檔使用者指南》中的推送 Docker 映像。

```
docker push 111122223333.dkr.ecr.region-code.amazonaws.com/cni-metrics-helper:v1.12.6
```

7. 針對您所推送的映像，使用 `registry/repository:tag` 更新您在先前步驟中用於確定映像的清單檔案檔案。如果您是使用 Helm Chart 進行安裝，則通常會出現一個可供指定 `registry/repository:tag` 的選項。安裝圖表時，請為推送到儲存庫的映像指定 `registry/repository:tag`。

檢視 Amazon EKS 附加元件的 Amazon 容器映像登錄檔

將 AWS Amazon EKS 附加元件部署到叢集時，節點會從附加元件安裝機制中指定的登錄檔中提取所需容器映像，例如，安裝清單檔案或 Helm `values.yaml` 檔案。從 Amazon EKS Amazon ECR 私有儲存庫中提取映像。Amazon EKS 會將映像複寫到每個 Amazon EKS 支援 AWS 區域中的儲存庫。您的節點可以透過網際網路，從以下任何登錄檔中提取容器映像。或者，如果您在 VPC 中為 Amazon ECR (AWS PrivateLink) 建立界面 VPC 端點，則節點可以透過 Amazon 網路提取映像。登錄檔需要 IAM AWS 帳戶的身分驗證。您的節點使用 Amazon EKS 節點 IAM 角色進行驗證，該角色具有與其相關聯 AmazonEC2ContainerRegistryReadOnly 受管 IAM 政策的權限。

AWS 區域	登錄檔
af-south-1	877085696533.dkr.ecr.af-south-1.amazonaws.com
ap-east-1	800184023465.dkr.ecr.ap-east-1.amazonaws.com
ap-east-2	533267051163.dkr.ecr.ap-east-2.amazonaws.com

AWS 區域	登錄檔
ap-southeast-3	296578399912.dkr.ecr.ap-southeast-3.amazonaws.com
ap-south-2	900889452093.dkr.ecr.ap-south-2.amazonaws.com
ap-southeast-4	491585149902.dkr.ecr.ap-southeast-4.amazonaws.com
ap-southeast-5	151610086707.dkr.ecr.ap-southeast-5.amazonaws.com
ap-southeast-6	333609536671.dkr.ecr.ap-southeast-6.amazonaws.com
ap-south-1	602401143452.dkr.ecr.ap-south-1.amazonaws.com
ap-northeast-3	602401143452.dkr.ecr.ap-northeast-3.amazonaws.com
ap-northeast-2	602401143452.dkr.ecr.ap-northeast-2.amazonaws.com
ap-southeast-1	602401143452.dkr.ecr.ap-southeast-1.amazonaws.com
ap-southeast-2	602401143452.dkr.ecr.ap-southeast-2.amazonaws.com
ap-southeast-7	121268973566.dkr.ecr.ap-southeast-7.amazonaws.com
ap-northeast-1	602401143452.dkr.ecr.ap-northeast-1.amazonaws.com
cn-north-1	918309763551.dkr.ecr.cn-north-1.amazonaws.com.cn

AWS 區域	登錄檔
cn-northwest-1	961992271922.dkr.ecr.cn-northwest-1.amazonaws.com.cn
eu-central-1	602401143452.dkr.ecr.eu-central-1.amazonaws.com
eu-west-1	602401143452.dkr.ecr.eu-west-1.amazonaws.com
eu-west-2	602401143452.dkr.ecr.eu-west-2.amazonaws.com
eu-south-1	590381155156.dkr.ecr.eu-south-1.amazonaws.com
eu-west-3	602401143452.dkr.ecr.eu-west-3.amazonaws.com
eu-south-2	455263428931.dkr.ecr.eu-south-2.amazonaws.com
eu-north-1	602401143452.dkr.ecr.eu-north-1.amazonaws.com
eu-central-2	900612956339.dkr.ecr.eu-central-2.amazonaws.com
il-central-1	066635153087.dkr.ecr.il-central-1.amazonaws.com
mx-central-1	730335286997.dkr.ecr.mx-central-1.amazonaws.com
me-south-1	558608220178.dkr.ecr.me-south-1.amazonaws.com
me-central-1	759879836304.dkr.ecr.me-central-1.amazonaws.com

AWS 區域	登錄檔
us-east-1	602401143452.dkr.ecr.us-east-1.amazonaws.com
us-east-2	602401143452.dkr.ecr.us-east-2.amazonaws.com
us-west-1	602401143452.dkr.ecr.us-west-1.amazonaws.com
us-west-2	602401143452.dkr.ecr.us-west-2.amazonaws.com
ca-central-1	602401143452.dkr.ecr.ca-central-1.amazonaws.com
ca-west-1	761377655185.dkr.ecr.ca-west-1.amazonaws.com
sa-east-1	602401143452.dkr.ecr.sa-east-1.amazonaws.com
us-gov-east-1	151742754352.dkr.ecr.us-gov-east-1.amazonaws.com
us-gov-west-1	013241004608.dkr.ecr.us-gov-west-1.amazonaws.com
eusc-de-east-1	877088126301.dkr.ecr.eusc-de-east-1.amazonaws.eu

Amazon EKS 附加元件

附加元件是提供 Kubernetes 應用程式支援操作功能的軟體，但並不專用於應用程式。這包括可觀測性代理程式或 Kubernetes 驅動程式等軟體，允許叢集與基礎 AWS 資源互動，以進行聯網、運算和儲存。附加元件軟體通常是由 Kubernetes 社群、雲端供應商等 AWS或第三方廠商所建置和維護。Amazon EKS 會自動安裝自我管理的附加元件，例如 Kubernetes 專用 Amazon VPC CNI 外掛程

式、kube-proxy，以及每個叢集的 CoreDNS。請注意，VPC CNI 附加元件與 Amazon EKS 混合節點不相容，並且不會部署到混合節點。您可以變更附加元件的預設設定，並在需要時進行更新。

Amazon EKS 附加元件提供 Amazon EKS 叢集精選附加元件集的安裝和管理。所有 Amazon EKS 附加元件都包含最新的安全修補程式、錯誤修正，並由 驗證 AWS 以使用 Amazon EKS。Amazon EKS 附加元件可讓您持續確保 Amazon EKS 叢集安全穩定，並降低安裝、設定和更新附加元件所需的工作量。如果是自我管理的附加元件 (例如 kube-proxy) 已在叢集上執行，且可作為 Amazon EKS 附加元件使用，則可以安裝 kube-proxy Amazon EKS 附加元件開始受益於 Amazon EKS 附加元件的功能。

您可以透過 Amazon EKS API 更新 Amazon EKS 附加元件的特定 Amazon EKS 受管組態欄位。您也可以在附加元件啟動後，直接在 Kubernetes 叢集中修改非由 Amazon EKS 管理的組態欄位。這包括定義附加元件適用的特定組態欄位。這些變更一旦進行，就不會被 Amazon EKS 覆寫。可以使用 Kubernetes 伺服器端套用功能來實現。如需詳細資訊，請參閱the section called “您可自訂的欄位”。

您可以使用 Amazon EKS 附加元件搭配任何 Amazon EKS 節點類型。如需詳細資訊，請參閱管理運算。

您可以使用 Amazon EKS API、、 AWS CLI 和 來新增 AWS 管理主控台、更新或刪除 Amazon EKS 附加元件eksctl。您也可以使用 AWS CloudFormation 建立 Amazon EKS 附加元件。

考量事項

當您使用 Amazon EKS 附加元件時，請考慮下列情況：

- 若要設定叢集的附加元件，則您的 IAM 主體必須具有 IAM 許可才能使用附加元件。如需詳細資訊，請參閱 Amazon Elastic Kubernetes Service 定義的動作中在其名稱中具有 Addon 的動作。
- Amazon EKS 附加元件會在您為叢集佈建或設定的節點上執行。節點類型包括 Amazon EC2 執行個體、Fargate 和混合節點。
- 您可以修改不由 Amazon EKS 管理的欄位，以自訂 Amazon EKS 附加元件的安裝。如需詳細資訊，請參閱the section called “您可自訂的欄位”。
- 如果您使用 建立叢集 AWS 管理主控台， AWS 管理主控台 會自動將 Amazon EKS kube-proxy、適用於 Kubernetes 的 Amazon VPC CNI 外掛程式，以及 CoreDNS 附加元件新增至您的叢集。如果您在eksctl不使用 config 檔案的情況下使用 建立叢集， 會將適用於 Kuberneteskube-proxy、CoreDNS 和 metrics-server 的 、Amazon VPC CNI 外掛程式eksctl安裝為從 0.184.0 版開始的 Amazon EKS 附加元件。對於舊版 eksctl，或者如果您使用任何其他工具，請改為安裝自我管理的 kube-proxy、適用於 Kubernetes 的 Amazon VPC CNI 外掛程式，以及 CoreDNS 附加元件。您可以自行管理這些附加元件，也可以在建立叢集之後手動新增 Amazon EKS 附加元件。無

論您使用何種方法來建立叢集，適用於 Kubernetes 附加元件的 Amazon VPC CNI 外掛程式不會安裝在混合節點上。

- 該 `eks:addon-cluster-admin` `ClusterRoleBinding` 將 `cluster-admin` `ClusterRole` 繫結至 `eks:addon-manager` Kubernetes 身分。此角色具有建立 Kubernetes 命名空間並將附加元件安裝到命名空間的 `eks:addon-manager` 身分所需的許可。若已移除 `eks:addon-cluster-admin` `ClusterRoleBinding`，Amazon EKS 叢集會繼續運作，但 Amazon EKS 將無法再管理任何附加元件。以下平台版本開始的所有叢集皆會使用新的 `ClusterRoleBinding`。
- 的 EKS 附加元件子集 AWS 已經過驗證，可與 Amazon EKS 混合節點相容。如需詳細資訊，請參閱 the section called “ AWS 附加元件” 上的相容性資料表。

附加元件的自訂命名空間

對於社群和 AWS 附加元件，您可以在建立附加元件期間選擇性地指定自訂命名空間。一旦您在特定命名空間中安裝了附加元件，必須移除並重新建立附加元件才能更改其命名空間。

如果您未指定命名空間，它將使用附加元件的預定義命名空間。

使用自訂命名空間可以更好地組織和隔離 EKS 叢集中的附加元件物件。這種彈性有助於您將附加元件與您的運作需求和現有命名空間策略保持一致。

您可在建立附加元件時設定自訂命名空間。如需詳細資訊，請參閱the section called “建立附加元件”。

取得附加元件的預定義命名空間

附加元件的預定義命名空間是若您未指定命名空間，將安裝到其中的命名空間。

要取得附加元件的預定義命名空間，請使用以下命令：

```
aws eks describe-addon-versions --addon-name <addon-name> --query
 "addons[].defaultNamespace"
```

輸出範例：

```
[
    "kube-system"
]
```

Amazon EKS 自動模式的考量事項

Amazon EKS 自動模式包含提供基本叢集功能的能力，包括：

- 裝置聯網
- 網路服務
- 叢集 DNS
- 自動擴展
- 區塊儲存
- Load Balancer 控制器
- Pod 身分識別代理程式
- 節點監控代理程式

使用自動模式運算，許多常用的 EKS 附加元件變得冗餘，例如：

- Amazon VPC CNI
- kube-proxy
- CoreDNS
- Amazon EBS CSI 驅動程式
- EKS Pod 身分識別代理程式

不過，如果您的叢集將自動模式與其他運算選項結合，例如自我管理的 EC2 執行個體、受管節點群組或 AWS Fargate，這些附加元件仍然是必要的。 AWS 具有增強型 EKS 附加元件與反親和性規則，可自動確保附加元件 Pod 僅排程在支援的運算類型上。此外，使用者現在可利用 EKS 附加元件 `DescribeAddonVersions` API 來確認每個附加元件及其特定版本所支援的 computeTypes。此外，使用 EKS Auto 模式時，上述控制器會在 AWS 擁有的基礎設施上執行。因此，您甚至可能不會在帳戶中看到它們，除非您使用 EKS 自動模式搭配其他類型的運算，在這種情況下，您會看到您在叢集上安裝的控制器。

如果您計劃在現有叢集上啟用 EKS 自動模式，則可能需要升級某些附加元件的版本。如需詳細資訊，請參閱 EKS 自動模式的 the section called “所需的附加元件版本”。

支援

AWS 會發佈具有不同支援層級的多種附加元件類型。

- AWS 附加元件：這些附加元件由 AWS建置並完全支援。
  - 使用 AWS 附加元件來處理其他 AWS 服務，例如 Amazon EFS。

  - 如需詳細資訊，請參閱the section called “ AWS 附加元件”。
- AWS Marketplace 附加元件：這些附加元件由 掃描 AWS 並由獨立 AWS 合作夥伴支援。
  - 使用 Marketplace 附加元件為您的叢集新增有價值且複雜的功能。
  - 如需詳細資訊，請參閱the section called “Marketplace 附加元件”。
- 社群附加元件：這些附加元件由 掃描， AWS 但由開放原始碼社群支援。
  - 使用社群附加元件來降低安裝常見開放原始碼軟體的複雜性，例如 Kubernetes 指標伺服器。
  - 社群附加元件是從 source by AWS. AWS only 驗證社群附加元件的版本相容性。
  - 如需詳細資訊，請參閱the section called “社群附加元件”。

下表詳細說明每個附加元件類型的支援範圍：

Category	功能	AWS 附加元件	AWS Marketplace 附加元件	社群附加元件
開發	由 建置 AWS	是	否	是
開發	驗證者 AWS	是	否	是*
開發	合作夥伴驗證 AWS	否	是	否
Maintenance (維護)	掃描者 AWS	是	是	是
Maintenance (維護)	修補者 AWS	是	否	是
Maintenance (維護)	由 AWS 合作夥伴修補	否	是	否
發佈	發佈者 AWS	是	否	是
發佈	由 AWS 合作夥伴發佈	否	是	否
支援	的基本安裝支援 AWS	是	是	是

Category	功能	AWS 附加元件	AWS Marketplace 附加元件	社群附加元件
支援	完整 AWS 支援	是	否	否
支援	完整 AWS 合作夥伴支援	否	是	否

*：社群附加元件的驗證僅包含了 Kubernetes 版本相容性。例如，如果您在叢集上安裝社群附加元件， 會 AWS 檢查它是否與叢集的 Kubernetes 版本相容。

AWS Marketplace 附加元件可以從外部來源下載其他軟體相依性 AWS。這些外部相依性不會由 掃描或驗證 AWS。部署擷取外部相依性的 AWS Marketplace 附加元件時，請考慮您的安全需求。

AWS 附加元件

您可在叢集上建立下列 Amazon EKS 附加元件。您可以使用 `eksctl`、 AWS 管理主控台或 AWS CLI 檢視最新的可用附加元件清單。若要查看所有可用的附加元件或安裝附加元件，請參閱 the section called “建立附加元件”。如果附加元件需要 IAM 許可，則您必須擁有叢集的 IAM OpenID Connect (OIDC) 提供者。若要判斷您是否已經擁有一個，或是需要建立一個，請參閱 the section called “IAM OIDC 提供商”。您可以在安裝附加元件之後建立或刪除附加元件。如需詳細資訊，請參閱 the section called “更新附加元件” 或 the section called “移除附加元件” 。若要了解藉助 Amazon EKS 混合節點來執行 EKS 附加元件特定考量的相關詳細資訊，請參閱 the section called “設定附加元件”。

您可使用以下任意 Amazon EKS 附加元件。

說明	進一步了解	相容的運算類型
針對您的叢集提供原生 VPC 聯網	the section called “Kubernetes 專用 Amazon VPC CNI 外掛程式”	EC2
可充當 Kubernetes 叢集 DNS 的靈活、可擴展的 DNS 伺服器	the section called “CoreDNS”	EC2、Fargate、EKS 自動模式、EKS 混合節點
維護每個 Amazon EC2 節點上的網路規則	the section called “Kube-proxy ”	EC2、EKS 混合式節點

說明	進一步了解	相容的運算類型
針對您的叢集提供 Amazon EBS 儲存體	the section called “Amazon EBS CSI 驅動程式”	EC2
為您的叢集提供 Amazon EC2 執行個體儲存體	the section called “Amazon EC2 執行個體存放區 CSI 驅動程式”	EC2
針對您的叢集提供 Amazon EFS 儲存體	the section called “Amazon EFS CSI 驅動程式”	EC2、EKS 自動模式
為您的叢集提供 Amazon S3 檔案儲存	the section called “Amazon EFS CSI 驅動程式”	EC2、EKS 自動模式
針對您的叢集提供 Amazon FSx for Lustre 儲存體	the section called “Amazon FSx CSI 驅動程式”	EC2、EKS 自動模式
針對您的叢集提供 Amazon S3 儲存體	the section called “適用於 Amazon S3 的掛載點 CSI 驅動程式”	EC2、EKS 自動模式
偵測額外的節點運作狀態問題	the section called “節點監控代理程式”	EC2、EKS 混合式節點
在相容的 CSI 驅動程式中啟用快照功能，例如 Amazon EBS CSI 驅動程式。	the section called “CSI 快照控制器”	EC2、Fargate、EKS 自動模式、EKS 混合節點
SageMaker HyperPod 任務治理能夠將 Amazon EKS 叢集各團隊間的運算資源分配與用量最佳化，從而解決任務優先順序確定及資源共用方面的效率低下問題。	the section called “Amazon SageMaker HyperPod 任務治理”	EC2、EKS 自動模式

說明	進一步了解	相容的運算類型
Amazon SageMaker HyperPod 可觀測性附加元件為 HyperPod 叢集帶來了全方位的監控與可觀測性功能。	the section called “Amazon SageMaker HyperPod 可觀測性附加元件”	EC2、EKS 自動模式
藉助 Amazon SageMaker HyperPod 訓練運算子，可透過進階排程與資源管理功能，在 Amazon EKS 叢集上實現高效的分散式訓練。	the section called “Amazon SageMaker HyperPod 訓練運算子”	EC2、EKS 自動模式
Amazon SageMaker HyperPod 推論運算子可透過最佳化的資源使用率和成本效益，來部署和管理高效能 AI 推論工作負載。	the section called “Amazon SageMaker HyperPod 推論運算子”	EC2、EKS 自動模式
Kubernetes 代理程式可收集網路流量資料，以及將其報告給 Amazon CloudWatch，從而實現對叢集節點間 TCP 連線的全方位監控。	the section called “AWS 網路流量監控代理程式”	EC2、EKS 自動模式
OpenTelemetry 專案的安全、生產就緒、 AWS 支援的分發	the section called “AWS Distro for OpenTelemetry”	EC2、Fargate、EKS 自動模式、EKS 混合節點
分析和處理基礎資料來源的安全監控服務，包括 AWS CloudTrail 管理事件和 Amazon VPC 流程日誌。Amazon GuardDuty 也會處理各種功能，例如 Kubernetes 稽核日誌和執行時期監控。	the section called “Amazon GuardDuty 代理程式”	EC2、EKS 自動模式

說明	進一步了解	相容的運算類型
AWS提供的監控和可觀測性服務。此附加元件會安裝 CloudWatch 代理程式，並搭配 Amazon EKS 的增強可觀測性，啟用 CloudWatch Application Signals 和 CloudWatch Container Insights。	the section called “Amazon CloudWatch 可觀測性代理程式”	EC2、EKS 自動模式、EKS 混合節點
能夠管理應用程式憑證，類似 EC2 執行個體設定檔將憑證提供給 EC2 執行個體的方式	the section called “EKS Pod 身分識別代理程式”	EC2、EKS 混合式節點
讓 cert-manager 從 AWS Private CA 發行 X.509 憑證。要求使用 cert-manager。	the section called “AWS Kubernetes 專用私有 CA 連接器”	EC2、Fargate、EKS 自動模式、EKS 混合節點
生成 SR-IOV 網路裝置效能的相關 Prometheus 指標	the section called “SR-IOV 網路指標匯出工具”	EC2
從 AWS Secrets Manager 擷取秘密，並從 AWS Systems Manager 參數存放區擷取參數，並將其掛載為 Kubernetes Pod 中的檔案。	the section called “AWS Secrets Store CSI 驅動程式提供者”	EC2、EKS 自動模式、EKS 混合節點
使用 Spaces，您可以建立和管理 JupyterLab 和程式碼編輯器應用程式，以執行互動式 ML 工作負載。	the section called “Amazon SageMaker 空間”	Hyperpod

Kubernetes 專用 Amazon VPC CNI 外掛程式

適用於 Kubernetes Amazon EKS 附加元件的 Amazon VPC CNI 外掛程式是一種 Kubernetes 容器網路介面 (CNI) 外掛程式，能夠針對您的叢集提供原生 VPC 聯網。此附加元件的自我管理類型或受管類

型預設會安裝在每個 Amazon EC2 節點上。如需詳細資訊，請參閱 Kubernetes 容器網路介面 (CNI) 外掛程式。

> **Note**
>
> 您無須在 Amazon EKS 自動模式叢集上安裝此附加元件。如需詳細資訊，請參閱the section called “Amazon EKS 自動模式的考量事項”。

Amazon EKS 附加元件名稱是 `vpc-cni`。

所需的 IAM 許可

此附加元件會利用 Amazon EKS 的服務帳戶的 IAM 角色功能。如需詳細資訊，請參閱the section called “具有 IRSA 的憑證”。

如果您的叢集使用 IPv4 系列，則需要 AmazonEKS_CNI_Policy 中的許可。如果您的叢集使用 IPv6 系列，則必須在 IPv6 模式下建立具有許可的 IAM 政策。您可以建立 IAM 角色，將其中一個政策連接至該角色，並使用下列命令為附加元件使用的 Kubernetes 服務帳戶加上注釋。

將 *my-cluster* 取代為您的叢集名稱，並將 *AmazonEKSVPCCNIRole* 取代為您的角色名稱。如果您的叢集使用 IPv6 系列，則請將 *AmazonEKS_CNI_Policy* 取代為您建立的政策名稱。此命令要求您在裝置上安裝 eksctl。如果您需要使用其他工具來建立角色、為其附加政策並註釋 Kubernetes 服務帳戶，請參閱 the section called “指派 IAM 角色”。

```
eksctl create iamserviceaccount --name aws-node --namespace kube-system --cluster my-
cluster --role-name AmazonEKSVPCCNIRole \
    --role-only --attach-policy-arn arn:aws: iam::aws:policy/AmazonEKS_CNI_Policy --
approve
```

更新資訊

您一次只能更新一個次要版本。例如，如果目前的版本是 `1.28.`*x*`-eksbuild.`*y* ，並且您想要更新至 `1.30.`*x*`-eksbuild.`*y* ，則您必須將目前的版本更新至 `1.29.`*x*`-eksbuild.`*y* ，然後再更新至 `1.30.`*x*`-eksbuild.`*y* 。如需更新附加元件的詳細資訊，請參閱 the section called “更新 (EKS 附加元件)”。

CoreDNS

CoreDNS Amazon EKS 附加元件是一個靈活、可擴展的 DNS 伺服器，可作為 Kubernetes 集群 DNS。根據預設，在您建立叢集時，會安裝此附加元件的自我管理類型或受管理類型。當您啟動具

有至少一個節點的 Amazon EKS 叢集時，依預設會部署兩個 CoreDNS 映像複本，而不論叢集中部署的節點數量為何。CoreDNS Pod 為叢集中的所有 Pod 提供名稱解析。如果您的叢集包含 Fargate Profile，且該設定檔的命名空間與 CoreDNS 的命名空間相符，您可將 CoreDNS Pod 部署至 Fargate 節點。如需詳細資訊，請參閱the section called “定義設定檔”。

> **Note**
>
> 您無須在 Amazon EKS 自動模式叢集上安裝此附加元件。如需詳細資訊，請參閱the section called “Amazon EKS 自動模式的考量事項”。

Amazon EKS 附加元件名稱是 `coredns`。

所需的 IAM 許可

此附加元件無須任何許可。

其他資訊

若要了解 CoreDNS 的相關詳細資訊，請參閱 Kubernetes 文件中的使用 CoreDNS 進行服務探索和自訂 DNS 服務。

Kube-proxy

`Kube-proxy` Amazon EKS 附加元件會維護每個 Amazon EC2 節點上的網路規則。其可以與您的 Pod 進行網路通訊。此附加元件的自我管理類型或受管類型預設會安裝在叢集的每個 Amazon EC2 節點上。

> **Note**
>
> 您無須在 Amazon EKS 自動模式叢集上安裝此附加元件。如需詳細資訊，請參閱the section called “Amazon EKS 自動模式的考量事項”。

Amazon EKS 附加元件名稱是 `kube-proxy`。

所需的 IAM 許可

此附加元件無須任何許可。

更新資訊

在更新目前版本之前，請考量下列需求：

- Amazon EKS 叢集上的 Kube-proxy 有與 Kubernetes 相同的相容性和差異政策。

其他資訊

若要了解 kube-proxy 的相關詳細資訊，請參閱 Kubernetes 文件中的 kube-proxy。

Amazon EBS CSI 驅動程式

Amazon EBS CSI 驅動程式 Amazon EKS 附加元件是一種 Kubernetes Container Storage Interface (CSI) 外掛程式，能夠針對您的叢集提供 Amazon EBS 儲存體。

> **Note**
>
> 您無須在 Amazon EKS 自動模式叢集上安裝此附加元件。自動模式包括區塊儲存功能。如需詳細資訊，請參閱the section called “部署具狀態工作負載”。

Amazon EKS 附加元件名稱是 aws-ebs-csi-driver。

所需的 IAM 許可

此附加元件會利用 Amazon EKS 的服務帳戶的 IAM 角色功能。如需詳細資訊，請參閱the section called “具有 IRSA 的憑證”。此附加元件需要下列其中一個 AWS 受管政策：適用於標籤型範圍的 AmazonEBSCSIDriverPolicyV2、適用於叢集範圍隔離的 AmazonEBSCSIDriverEKSClusterScopedPolicy，或如果您不希望任何標籤型限制，則為 AmazonEBSCSIDriverPolicy。建立 IAM 角色，並按照下列命令連接受管政策。使用您的叢集名稱取代 *my-cluster*，並使用您的角色名稱取代 *AmazonEKS_EBS_CSI_DriverRole*。此命令要求您在裝置上安裝 eksctl。如果您需要使用其他工具，或需要使用自訂 KMS 金鑰進行加密，請參閱 the section called “步驟 1：建立 IAM 角色”。

如果從 遷移AmazonEBSCSIDriverPolicy，請參閱 EBS CSI 驅動程式政策遷移。

```
eksctl create iamserviceaccount \
    --name ebs-csi-controller-sa \
    --namespace kube-system \
    --cluster my-cluster \
```

```
    --role-name AmazonEKS_EBS_CSI_DriverRole \
    --role-only \
    --attach-policy-arn arn:aws: iam::aws:policy/service-role/
AmazonEBSCSIDriverPolicyV2 \
    --approve
```

其他資訊

若要了解附加元件的相關詳細資訊，請參閱 the section called “Amazon EBS”。

Amazon EC2 執行個體存放區 CSI 驅動程式

Amazon EC2 執行個體存放區 CSI 驅動程式 Amazon EKS 附加元件是一種 Kubernetes Container Storage Interface (CSI) 外掛程式，可為叢集提供 Amazon EC2 執行個體存放區 NVMe 儲存體。

Amazon EKS 附加元件名稱是 `aws-ec2-local-instance-store-csi-driver`。

所需的 IAM 許可

此附加元件無須任何許可。驅動程式會透過本機 Linux 操作與後端 NVMe 裝置互動，不會進行 AWS API 呼叫。

其他資訊

若要了解附加元件的相關詳細資訊，請參閱 the section called “Amazon EC2 執行個體儲存體”。

Amazon EFS CSI 驅動程式

Amazon EFS CSI 驅動程式 Amazon EKS 附加元件是 Kubernetes 容器儲存介面 (CSI) 外掛程式，可為叢集提供 Amazon EFS 和 Amazon S3 檔案儲存。

Amazon EKS 附加元件名稱是 `aws-efs-csi-driver`。

所需的 IAM 許可

此附加元件會利用 Amazon EKS 的服務帳戶的 IAM 角色功能。如需詳細資訊，請參閱the section called “具有 IRSA 的憑證”。

您需要的特定 AWS 受管政策取決於您要使用的檔案系統類型：

- 僅限 Amazon EFS 檔案系統：連接 AmazonEFSCSIDriverPolicy 受管政策。
- 僅適用於 Amazon S3 檔案系統：連接 `AmazonS3FilesCSIDriverPolicy`受管政策。

- 對於 Amazon EFS 和 Amazon S3 檔案系統：連接 AmazonEFSCSIDriverPolicy和 AmazonS3FilesCSIDriverPolicy受管政策。

您可以建立 IAM 角色，並使用下列命令連接受管政策。使用您的叢集名稱取代 *my-cluster*，並使用您的角色名稱取代 *AmazonEKS_EFS_CSI_DriverRole*。下列範例會連接 AmazonEFSCSIDriverPolicy Amazon EFS 檔案系統的 。如果您使用的是 Amazon S3 檔案系統，請將政策 ARN 取代為 arn:aws: iam::aws:policy/service-role/AmazonS3FilesCSIDriverPolicy。如果您同時使用這兩種檔案系統類型，請使用第二個政策 ARN 新增額外的--attach-policy-arn旗標。這些命令要求您在裝置上安裝 eksctl。如果您需要使用不同的工具，請參閱the section called “步驟 1：建立 IAM 角色”適用於 Amazon EFS 的 或the section called “步驟 1：建立 IAM 角色”適用於 Amazon S3 檔案的 。

```
export cluster_name=my-cluster
export role_name=AmazonEKS_EFS_CSI_DriverRole
eksctl create iamserviceaccount \
    --name efs-csi-controller-sa \
    --namespace kube-system \
    --cluster $cluster_name \
    --role-name $role_name \
    --role-only \
    --attach-policy-arn arn:aws: iam::aws:policy/service-role/AmazonEFSCSIDriverPolicy
 \
    --approve
TRUST_POLICY=$(aws iam get-role --output json --role-name $role_name --query
 'Role.AssumeRolePolicyDocument' | \
    sed -e 's/efs-csi-controller-sa/efs-csi-*/' -e 's/StringEquals/StringLike/')
aws iam update-assume-role-policy --role-name $role_name --policy-document
 "$TRUST_POLICY"
```

> **Note**
>
> 上述範例只會設定 efs-csi-controller-sa。如果您使用的是 Amazon S3 檔案系統，您也需要設定 efs-csi-node-sa。如需完整的 S3 檔案 IAM 設定the section called “步驟 1：建立 IAM 角色”，請參閱 。

其他資訊

若要了解附加元件的相關詳細資訊，請參閱 the section called “Amazon EFS”。

Amazon FSx CSI 驅動程式

Amazon FSx CSI 驅動程式 Amazon EKS 附加元件是一種 Kubernetes Container Storage Interface (CSI) 外掛程式，能夠針對您的叢集提供 Amazon FSx for Lustre 儲存體。

Amazon EKS 附加元件名稱是 `aws-fsx-csi-driver`。

> **Note**
>
> - 叢集中預先存在的 Amazon FSx CSI 驅動程式安裝可能會導致附加元件安裝失敗。當您嘗試安裝 Amazon EKS 附加元件版本而又存在非 EKS FSx CSI 驅動程式時，安裝會因資源衝突而失敗。在安裝期間使用 `OVERWRITE` 標記以解決此問題：
>
>   ```
>   aws eks create-addon --addon-name aws-fsx-csi-driver --cluster-name my-cluster
>    --resolve-conflicts OVERWRITE
>   ```
>
> - Amazon FSx CSI 驅動程式 EKS 附加元件支援透過 EKS Pod Identity 或 IAM Roles for Service Accounts (IRSA) 進行身分驗證。若要使用 EKS Pod Identity，請在部署 FSx CSI 驅動程式附加元件之前或之後安裝 Pod Identity 代理程式。如需詳細資訊，請參閱the section called “設定代理程式”。若要改為使用 IRSA，請參閱 the section called “IAM OIDC 提供商”。

所需的 IAM 許可

此附加元件會利用 Amazon EKS 的服務帳戶的 IAM 角色功能。如需詳細資訊，請參閱the section called “具有 IRSA 的憑證”。AmazonFSxFullAccess AWS 受管政策中的許可為必要項目。建立 IAM 角色，並按照下列命令連接受管政策。使用您的叢集名稱取代 *`my-cluster`*，並使用您的角色名稱取代 *`AmazonEKS_FSx_CSI_DriverRole`*。此命令要求您在裝置上安裝 eksctl。

```
eksctl create iamserviceaccount \
    --name fsx-csi-controller-sa \
    --namespace kube-system \
    --cluster my-cluster \
    --role-name AmazonEKS_FSx_CSI_DriverRole \
    --role-only \
    --attach-policy-arn arn:aws: iam::aws:policy/AmazonFSxFullAccess \
    --approve
```

其他資訊

若要了解附加元件的相關詳細資訊，請參閱 the section called “Amazon FSx for Lustre”。

適用於 Amazon S3 的掛載點 CSI 驅動程式

適用於 Amazon S3 的掛載點 CSI 驅動程式 Amazon EKS 附加元件是一種 Kubernetes Container Storage Interface (CSI) 外掛程式，能夠針對您的叢集提供 Amazon S3 儲存。

Amazon EKS 附加元件名稱是 `aws-mountpoint-s3-csi-driver`。

所需的 IAM 許可

此附加元件會利用 Amazon EKS 的服務帳戶的 IAM 角色功能。如需詳細資訊，請參閱the section called “具有 IRSA 的憑證”。

建立的 IAM 角色將需要提供 S3 存取權的政策。建立政策時，請遵循掛載點 IAM 許可建議。或者，您可以使用 AWS 受管政策 AmazonS3FullAccess，但此受管政策授予的許可超過掛載點所需的許可。

您可以建立 IAM 角色，並使用下列命令將政策連接至該角色。將 *my-cluster* 取代為您的叢集名稱、將 *region-code* 取代為正確的 AWS 區域碼、將 *AmazonEKS_S3_CSI_DriverRole* 取代為您的角色名稱，並將 *AmazonEKS_S3_CSI_DriverRole_ARN* 取代為角色 ARN。這些命令要求您在裝置上安裝 eksctl。如需使用 IAM 主控台或 AWS CLI 的說明，請參閱 the section called “步驟 2：建立 IAM 角色”。

```
CLUSTER_NAME=my-cluster
REGION=region-code
ROLE_NAME=AmazonEKS_S3_CSI_DriverRole
POLICY_ARN=AmazonEKS_S3_CSI_DriverRole_ARN
eksctl create iamserviceaccount \
    --name s3-csi-driver-sa \
    --namespace kube-system \
    --cluster $CLUSTER_NAME \
    --attach-policy-arn $POLICY_ARN \
    --approve \
    --role-name $ROLE_NAME \
    --region $REGION \
    --role-only
```

其他資訊

若要了解附加元件的相關詳細資訊，請參閱 the section called “適用於 Amazon S3 的掛載點”。

CSI 快照控制器

容器儲存介面 (CSI) 快照控制器可讓您在相容的 CSI 驅動程式 (例如 Amazon EBS CSI 驅動程式) 中使用快照拍攝功能。

Amazon EKS 附加元件名稱是 `snapshot-controller`。

所需的 IAM 許可

此附加元件無須任何許可。

其他資訊

若要了解附加元件的相關詳細資訊，請參閱 the section called “CSI 快照器控制器”。

Amazon SageMaker HyperPod 任務治理

SageMaker HyperPod 任務治理是強大的管理系統，旨在簡化資源配置，並確保 Amazon EKS 叢集跨團隊和專案有效利用運算資源。這讓管理員能夠設定：

- 各種任務的優先順序層級
- 每個團隊的運算配置
- 每個團隊如何借出和借入閒置運算
- 如果團隊先佔自己的任務

HyperPod 任務治理也提供 Amazon EKS 叢集可觀測性，提供叢集容量的即時可見性。這包括運算可用性和使用情況、團隊配置和使用率，以及任務執行和等待時間資訊，讓您為明智的決策和主動資源管理做好準備。

Amazon EKS 附加元件名稱是 `amazon-sagemaker-hyperpod-taskgovernance`。

所需的 IAM 許可

此附加元件無須任何許可。

其他資訊

如需了解附加元件的相關詳細資訊，請參閱 SageMaker HyperPod 任務治理

Amazon SageMaker HyperPod 可觀測性附加元件

Amazon SageMaker HyperPod 可觀測性附加元件為 HyperPod 叢集帶來了全方位的監控與可觀測性功能。此附加元件可自動部署及管理必要的監控元件，包括節點匯出工具、DCGM 匯出工具、kube-state-metrics，以及 EFA 匯出工具。該附加元件還可收集指標，將其轉傳至客戶指定的 Amazon Managed Prometheus (AMP) 執行個體，以及公開自訂指標的 OTLP 端點及客戶訓練任務的事件擷取。

附加元件藉由從各種元件中湊集指標，包括 HyperPod 任務治理附加元件、HyperPod 訓練運算子、Kubeflow 及 KEDA 等，可與更廣泛的 HyperPod 生態系統整合。收集的所有指標均集中於 Amazon Managed Prometheus，從而支援客戶透過 Amazon Managed Grafana 儀表板來實現統一的可觀測性檢視。這提供了整個 HyperPod 環境中叢集運作狀態、資源使用率及訓練任務效能的端對端可視性。

Amazon EKS 附加元件名稱是 `amazon-sagemaker-hyperpod-observability`。

所需的 IAM 許可

此附加元件會利用 Amazon EKS 的服務帳戶的 IAM 角色功能。如需詳細資訊，請參閱the section called “具有 IRSA 的憑證”。需要下列受管政策：

- `AmazonPrometheusRemoteWriteAccess` - 用於在遠端將指標從叢集寫入 AMP
- `CloudWatchAgentServerPolicy` - 用於在遠端將日誌從叢集寫入 CloudWatch

其他資訊

如需了解附加元件及其功能的相關詳細資訊，請參閱 SageMaker HyperPod 可觀測性。

Amazon SageMaker HyperPod 訓練運算子

Amazon SageMaker HyperPod 訓練運算子透過跨大型 GPU 叢集有效管理分散式訓練，協助您加速生成式 AI 模型開發。它引入智慧型故障復原、當掉任務偵測和程序層級管理功能，將訓練中斷減至最低並降低成本。與發生故障時需要完整重新啟動任務的傳統訓練基礎結構不同，此運算子會實作手術程序復原，讓您的訓練任務順利執行。

運算子也會使用 HyperPod 的運作狀態監控和可觀測性函數，提供訓練執行的即時可見性，並自動監控損失尖峰和輸送量降低等關鍵指標。您可以透過簡單的 YAML 組態定義復原政策，而無需變更程式碼，可讓您快速回應無法復原的訓練狀態並從中復原。這些監控和復原功能會共同運作，以維持最佳的訓練效能，同時將操作負荷降至最低。

Amazon EKS 附加元件名稱是 `amazon-sagemaker-hyperpod-training-operator`。

如需詳細資訊，請參閱《Amazon SageMaker 開發人員指南》中的使用 HyperPod 訓練運算子。

所需的 IAM 許可

此附加元件需要 IAM 許可，以及使用 EKS Pod 身分識別。

AWS 建議 `AmazonSageMakerHyperPodTrainingOperatorAccess`受管政策。

如需詳細資訊，請參閱 Amazon SageMaker 開發人員指南中的安裝訓練運算子。

其他資訊

如需了解附加元件的相關詳細資訊，請參閱 SageMaker HyperPod 訓練運算子。

Amazon SageMaker HyperPod 推論運算子

Amazon SageMaker HyperPod end-to-end體驗，支援從互動式實驗和訓練到推論和訓練後工作流程的 AI 開發完整生命週期。它現在提供全面的推論平台，結合了 Kubernetes 的靈活性和受管體驗的卓越營運。在整個模型生命週期中使用相同的 HyperPod 運算，以企業級可靠性部署、擴展和最佳化 GenAI 模型。

Amazon SageMaker HyperPod 提供靈活的部署介面，可讓您透過多種方法部署模型，包括 kubectl、Python SDK、Amazon SageMaker Studio UI 或 HyperPod CLI。功能提供進階的自動擴展功能，具有動態資源配置，可根據需求自動調整。此外，它還包含全面的可觀測性和監控功能，這些功能會追蹤關鍵指標，例如 time-to-first-token、延遲和 GPU 使用率，以協助您最佳化效能。

Amazon EKS 附加元件名稱是 `amazon-sagemaker-hyperpod-inference`。

安裝方法

您可以使用下列其中一種方法來安裝此附加元件：

- SageMaker 主控台（建議）：透過引導式組態提供簡化的安裝體驗。
- EKS 附加元件主控台或 CLI：在安裝推論運算子之前，需要手動安裝相依性附加元件。如需必要的相依性，請參閱下列先決條件一節。

先決條件

透過 EKS 附加元件主控台或 CLI 安裝推論運算子附加元件之前，請確定已安裝下列相依性。

必要的 EKS 附加元件：

- Amazon S3 掛載點 CSI 驅動程式（最低版本：v1.14.1-eksbuild.1)
- Metrics Server（最低版本：v0.7.2-eksbuild.4)
- Amazon FSx CSI 驅動程式（最低版本：v1.6.0-eksbuild.1)
- Cert Manager（最低版本：v1.18.2-eksbuild.2)

如需每個相依性的詳細安裝說明，請參閱安裝推論運算子。

所需的 IAM 許可

此附加元件需要 IAM 許可，並使用 OIDC/IRSA。

建議使用以下 受管政策，因為它們提供最低範圍的許可：

- `AmazonSageMakerHyperPodInferenceAccess` - 提供設定推論運算子所需的管理員權限
- `AmazonSageMakerHyperPodGatedModelAccess` - 提供 SageMaker HyperPod 存取 SageMaker Jumpstart 中的門控模型（例如 Meta Llama、GPT-Neo)

如需詳細資訊，請參閱安裝推論運算子。

其他資訊

若要進一步了解 Amazon SageMaker HyperPod 推論運算子，請參閱 SageMaker HyperPod 推論運算子。

如需故障診斷資訊，請參閱對 SageMaker HyperPod 模型部署進行故障診斷。

AWS 網路流量監控代理程式

Amazon CloudWatch 網路流量監視器代理程式是一種 Kubernetes 應用程式，能夠從叢集的所有節點收集 TCP 連線統計資料，然後將網路流量報告發布至 Amazon CloudWatch Network Flow Monitor Ingestion API。

Amazon EKS 附加元件名稱是 `aws-network-flow-monitoring-agent`。

所需的 IAM 許可

此附加元件需要 IAM 許可。

您需要將 CloudWatchNetworkFlowMonitorAgentPublishPolicy 受管政策連接至該附加元件。

若要了解必要 IAM 設定的相關詳細資訊，請參閱 Amazon CloudWatch 網路流量監視器代理程式 GitHub 儲存庫上的 IAM 政策。

若要了解受管政策的相關詳細資訊，請參閱《Amazon CloudWatch 使用者指南》中的 CloudWatchNetworkFlowMonitorAgentPublishPolicy。

其他資訊

如需了解附加元件的相關詳細資訊，請參閱 Amazon CloudWatch 網路流量監視器代理程式 GitHub 儲存庫。

節點監控代理程式

節點監控代理程式 Amazon EKS 附加元件可偵測額外的節點運作狀態問題。選用節點自動修復功能還可善用這些額外的運作狀態訊號，依據需要來自動修復節點。

> **Note**
>
> 您無須在 Amazon EKS 自動模式叢集上安裝此附加元件。如需詳細資訊，請參閱the section called “Amazon EKS 自動模式的考量事項”。

Amazon EKS 附加元件名稱是 eks-node-monitoring-agent。

所需的 IAM 許可

此附加元件無須額外許可。

其他資訊

如需詳細資訊，請參閱the section called “節點運作狀態和修復”。

AWS Distro for OpenTelemetry

AWS Distro for OpenTelemetry Amazon EKS 附加元件是 OpenTelemetry 專案的安全、生產就緒、AWS 支援的分佈。如需詳細資訊，請參閱 GitHub 上的 AWS Distro for OpenTelemetry。

Amazon EKS 附加元件名稱是 adot。

所需的 IAM 許可

只有當您使用其中一個可透過進階配置選擇加入的預先設定自訂資源時，此附加元件才需要 IAM 許可。

其他資訊

如需詳細資訊，請參閱 AWS Distro for OpenTelemetry 文件中的使用 EKS 附加元件的 Distro for OpenTelemetry 入門。 AWS OpenTelemetry

ADOT 需要`cert-manager`將附加元件部署在叢集上做為先決條件，否則如果直接使用 https://registry.terraform.io/modules/terraform-aws-modules/eks/aws/latest`cluster_addons` 屬性部署，此附加元件將無法運作。如需更多需求，請參閱 AWS Distro for OpenTelemetry 文件中的使用 EKS 附加元件的 Distro for OpenTelemetry 入門需求。 AWS OpenTelemetry

Amazon GuardDuty 代理程式

Amazon GuardDuty 代理程式 Amazon EKS 附加元件可從您的 EKS 叢集節點收集執行時期事件 (檔案存取、程序執行、網路連線)，以便 GuardDuty 運行監控進行分析。GuardDuty 本身 （ 而非代理程式 ） 是安全性監控服務，可分析和處理基礎資料來源，包括 AWS CloudTrail 管理事件和 Amazon VPC 流程日誌，以及 Kubernetes 稽核日誌和執行期監控等功能。

Amazon EKS 附加元件名稱是 `aws-guardduty-agent`。

所需的 IAM 許可

此附加元件無須任何許可。

其他資訊

如需詳細資訊，請參閱 Amazon GuardDuty 中 Amazon EKS 叢集的運行監控。

- 若要偵測 Amazon EKS 叢集中潛在的安全威脅，請啟用 Amazon GuardDuty 執行時間監控，並將 GuardDuty 安全代理程式部署到您的 Amazon EKS 叢集。

Amazon CloudWatch 可觀測性代理程式

Amazon CloudWatch 可觀測性代理程式 Amazon EKS 附加元件是由 提供的監控和可觀測性服務 AWS。此附加元件會安裝 CloudWatch 代理程式，並搭配 Amazon EKS 的增強可觀測性，啟用 CloudWatch Application Signals 和 CloudWatch Container Insights。如需詳細資訊，請參閱 Amazon CloudWatch 代理程式。

Amazon EKS 附加元件名稱是 amazon-cloudwatch-observability。

所需的 IAM 許可

此附加元件會利用 Amazon EKS 的服務帳戶的 IAM 角色功能。如需詳細資訊，請參閱the section called “具有 IRSA 的憑證”。AWSXrayWriteOnlyAccess 和 CloudWatchAgentServerPolicy AWS 受管政策中的許可為必要條件。您可以建立 IAM 角色，為該角色附加受管政策，並使用下列命令為附加元件使用的 Kubernetes 服務帳戶加上注釋。使用您的叢集名稱取代 *my-cluster*，並使用您的角色名稱取代 *AmazonEKS_Observability_role*。此命令要求您在裝置上安裝 eksctl。如果您需要使用其他工具來建立角色、為其附加政策並註釋 Kubernetes 服務帳戶，請參閱 the section called “指派 IAM 角色”。

```
eksctl create iamserviceaccount \
    --name cloudwatch-agent \
    --namespace amazon-cloudwatch \
    --cluster my-cluster \
    --role-name AmazonEKS_Observability_Role \
    --role-only \
    --attach-policy-arn arn:aws: iam::aws:policy/AWSXrayWriteOnlyAccess \
    --attach-policy-arn arn:aws: iam::aws:policy/CloudWatchAgentServerPolicy \
    --approve
```

其他資訊

如需詳細資訊，請參閱安裝 CloudWatch 代理程式。

AWS Kubernetes 專用私有 CA 連接器

AWS Private CA Connector for Kubernetes 是 cert-manager 的附加元件，可讓使用者從 AWS Private Certificate Authority (AWS Private CA) 取得憑證。

- Amazon EKS 附加元件名稱是 aws-privateca-connector-for-kubernetes。
- 附加元件命名空間是 aws-privateca-issuer。

此附加元件要求使用 cert-manager。Amazon EKS 提供了 cert-manager 作為社群附加元件。若要了解此附加元件的相關詳細資訊，請參閱 the section called “Cert Manager”。若要了解安裝附加元件的相關詳細資訊，請參閱 the section called “建立附加元件”。

所需的 IAM 許可

此附加元件需要 IAM 許可。

使用 EKS Pod 身分識別，連接 AWSPrivateCAConnectorForKubernetesPolicy IAM 政策與 Kubernetes aws-privateca-issuer 服務帳戶。如需詳細資訊，請參閱the section called “使用 Pod 身分識別”。

如需必要許可的相關資訊，請參閱《 AWS 受管政策參考》中的 AWSPrivateCAConnectorForKubernetesPolicy。

其他資訊

如需詳細資訊，請參閱適用於 Kubernetes GitHub 儲存庫的AWS Private CA 發行者。

若要了解設定附加元件的相關詳細資訊，請參閱 aws-privateca-issuer GitHub 儲存庫中的 values.yaml。請確認 values.yaml 版本與叢集上安裝的附加元件版本是否相符。

此附加元件可容忍 EKS 自動模式的 system NodePool 所用的 CriticalAddonsOnly 污點。如需詳細資訊，請參閱the section called “執行關鍵附加元件”。

EKS Pod 身分識別代理程式

EKS Pod 身分識別代理程式 Amazon EKS 附加元件提供管理應用程式憑證的功能，類似 Amazon EC2 執行個體設定檔將憑證提供給 Amazon EC2 執行個體的方式。

> **Note**
>
> 您無須在 Amazon EKS 自動模式叢集上安裝此附加元件。Amazon EKS 自動模式與 EKS Pod 身分識別整合。如需詳細資訊，請參閱the section called “Amazon EKS 自動模式的考量事項”。

Amazon EKS 附加元件名稱是 eks-pod-identity-agent。

所需的 IAM 許可

Pod 身分識別代理程式附加元件本身無須 IAM 角色。該附加元件使用 Amazon EKS 節點 IAM 角色的許可來運作，但不需要用於附加元件的專用 IAM 角色。

更新資訊

您一次只能更新一個次要版本。例如，如果目前的版本是 1.28.x-eksbuild.y，並且您想要更新至 1.30.x-eksbuild.y，則您必須將目前的版本更新至 1.29.x-eksbuild.y，然後再更新至 1.30.x-eksbuild.y。如需更新附加元件的詳細資訊，請參閱 the section called “更新附加元件”。

SR-IOV 網路指標匯出工具

SR-IOV 網路指標匯出工具 Amazon EKS 附加元件使用 Prometheus 格式，來收集及公開 SR-IOV 網路裝置的相關指標。該附加元件支援在 EKS 裸機節點上監控 SR-IOV 網路效能。匯出工具在搭載 SR-IOV-capable 網路介面的節點上作為 DaemonSet 來執行，然後匯出 Prometheus 可湊集的指標。

> **Note**
>
> 此附加元件需要搭載 SR-IOV-capable 網路介面的節點。

屬性	Value
附加元件名稱	`sriov-network-metrics-exporter`
命名空間	`monitoring`
文件	SR-IOV 網路指標匯出工具 GitHub 儲存庫
服務帳戶名稱	無
受管 IAM 政策	無
自訂 IAM 許可	無

AWS Secrets Store CSI 驅動程式提供者

Secrets Store CSI 驅動程式的 AWS 提供者是一種附加元件，可讓您從 AWS Secrets Manager 擷取秘密，並從 AWS Systems Manager 參數存放區擷取參數，並將其掛載為 Kubernetes Pod 中的檔案。

所需的 IAM 許可

附加元件不需要 IAM 許可。不過，應用程式 Pod 需要 IAM 許可，才能從 AWS Secrets Manager 擷取秘密，並從 AWS Systems Manager 參數存放區擷取參數。安裝附加元件後，必須透過服務帳戶 (IRSA) 的 IAM 角色或 EKS Pod 身分來設定存取權。若要使用 IRSA，請參閱 Secrets Manager IRSA 設定文件。若要使用 EKS Pod 身分，請參閱 Secrets Manager Pod 身分設定文件。

AWS 建議 `AWSSecretsManagerClientReadOnlyAccess`受管政策。

如需所需許可的詳細資訊，請參閱《 AWS 受管政策參考`AWSSecretsManagerClientReadOnlyAccess`》中的 。

其他資訊

如需詳細資訊，請參閱 secrets-store-csi-driver-provider-aws GitHub 儲存庫。

若要進一步了解附加元件，請參閱AWS 附加元件的 Secrets Manager 文件。

Amazon SageMaker 空間

Amazon SageMaker Spaces 附加元件可讓您在 EKS 或 HyperPod-EKS 叢集上執行 IDEs 和筆記本。管理員可以使用 EKS 主控台在其叢集上安裝附加元件，並定義預設空間組態，例如影像、運算資源、筆記本設定的本機儲存體 （ 附加儲存體以連接至其空間）、檔案系統和初始化指令碼。

AI 開發人員可以使用 kubectl 來建立、更新和刪除空間。他們可以靈活地使用管理員提供的預設組態或自訂設定。AI 開發人員可以使用其本機 VS Code IDEs 和/或其 Web 瀏覽器，在管理員設定的自訂 DNS 網域上託管其 JupyterLab 或 CodeEditor IDE，在 EKS 或 HyperPod-EKS 上存取其空間。他們也可以使用 kubernetes 的連接埠轉送功能來存取其 Web 瀏覽器中的空間。

Amazon EKS 附加元件名稱是 `amazon-sagemaker-spaces`。

所需的 IAM 許可

此附加元件需要 IAM 許可。如需所需 IAM 設定的詳細資訊，請參閱《Amazon SageMaker 開發人員指南》中的 IAM 許可設定。

其他資訊

若要進一步了解附加元件及其功能，請參閱《Amazon SageMaker 開發人員指南》中的 HyperPod 上的 SageMaker AI 筆記本。 Amazon SageMaker

社群附加元件

您可以使用 AWS APIs社群附加元件，例如 Kubernetes Metrics Server。您可選擇將社群附加元件安裝為 Amazon EKS 附加元件，以降低在多個叢集上維護軟體的複雜性。

例如，您可以使用 AWS API、CLI 或 管理主控台來安裝社群附加元件。您可在叢集建立期間安裝社群附加元件。

您可像管理現有的 Amazon EKS 附加元件一樣管理社群附加元件。社群附加元件與現有的附加元件不同，它們具有獨特的支援範圍。

> **Note**
>
> 是否使用社群附加元件由您自行決定。作為您與 之間共同責任模型的一部分 AWS，您應該了解您在叢集中為這些第三方外掛程式安裝什麼。您也需要負責滿足叢集安全需求的社群附加元件。如需詳細資訊，請參閱the section called “支援部署至 EKS 的軟體”。

社群附加元件並非由 建置 AWS。 AWS 只會驗證社群附加元件的版本相容性。例如，如果您在叢集上安裝社群附加元件， 會 AWS 檢查它是否與叢集的 Kubernetes 版本相容。

重要的是， AWS 不提供社群附加元件的完整支援。 僅 AWS 支援使用 AWS APIs 完成的生命週期操作，例如安裝附加元件或刪除附加元件。

如果您需要社群附加元件的支援，請藉助現有的專案資源。例如，您可在專案的儲存庫上建立 GitHub 問題。

確定附加元件的類型

您可以使用 AWS CLI 來判斷 Amazon EKS 附加元件的類型。

使用下列 CLI 命令擷取附加元件的相關資訊。您可將 `metrics-server` 取代為任何附加元件的名稱。

```
aws eks describe-addon-versions --addon-name metrics-server
```

檢閱 `owner` 欄位的 CLI 輸出。

```
{
    "addons": [
        {
            "addonName": "metrics-server",
            "type": "observability",
            "owner": "community",
            "addonVersions": [
```

如果 的值`owner`是 `community`，則附加元件是社群附加元件。 AWS 僅支援安裝、更新和移除附加元件。如果您對附加元件本身的功能和操作有任何疑問，請利用社群資源，例如 GitHub 問題。

安裝或更新社群附加元件

您安裝或更新社群附加元件的方式與安裝或更新其他 Amazon EKS 附加元件相同。

- the section called “建立附加元件”
- the section called “更新附加元件”
- the section called “移除附加元件”

可用的社群附加元件

以下社群附加元件可從 Amazon EKS 取得。

- the section called “Kubernetes 指標伺服器”
- the section called “kube-state-metrics”
- the section called “Prometheus Node Exporter”
- the section called “Cert Manager”
- the section called “外部 DNS”
- the section called “Fluent Bit”

Kubernetes 指標伺服器

Kubernetes 指標伺服器是一個可擴展且高效的容器資源指標來源，用於 Kubernetes 內建的自動擴展管道。它從 Kubelets 收集資源指標，並透過 Metrics API 在 Kubernetes apiserver 中公開，以供 Horizontal Pod Autoscaler 和 Vertical Pod Autoscaler 使用。

屬性	Value
附加元件名稱	`metrics-server`
命名空間	`kube-system`
文件	GitHub 讀我檔案
服務帳戶名稱	無
受管 IAM 政策	無
自訂 IAM 許可	無

kube-state-metrics

用於產生和公開叢集層級指標的附加元件代理程式。

Kubernetes API 中的 Kubernetes 物件狀態可公開為指標。稱為 kube-state-metrics 的附加元件代理程式可以連線至 Kubernetes API 伺服器，並使用從叢集中個別物件狀態產生的指標公開 HTTP 端點。它會公開物件狀態的各種資訊，例如標籤和註釋、啟動和終止時間、狀態或物件目前所處的階段。

屬性	Value
附加元件名稱	`kube-state-metrics`
命名空間	`kube-state-metrics`
文件	Kubernetes 文件中 Kubernetes 物件狀態的指標
服務帳戶名稱	無
受管 IAM 政策	無
自訂 IAM 許可	無

Prometheus Node Exporter

用於 *NIX 核心公開的硬體和作業系統指標的 Prometheus 匯出器，以 Go 語言編寫，並具有可插拔的指標收集器。Prometheus Node Exporter 公開各種硬體及核心相關指標。

屬性	Value
附加元件名稱	`prometheus-node-exporter`
命名空間	`prometheus-node-exporter`
文件	使用 Prometheus 文件中的 Node Exporter 監控 Linux 主機指標
服務帳戶名稱	無
受管 IAM 政策	無

屬性	Value
自訂 IAM 許可	無

Cert Manager

Cert Manager 可用於管理憑證的建立和續約。

屬性	Value
附加元件名稱	`cert-manager`
命名空間	`cert-manager`
文件	Cert Manager 文件
服務帳戶名稱	無
受管 IAM 政策	無
自訂 IAM 許可	無

外部 DNS

外部 DNS EKS 附加元件可用於透過 Kubernetes 資源管理 Route53 DNS 記錄。

外部 DNS 許可能縮減為對您希望管理的託管區域的 `route53:ChangeResourceRecordSets`、`route53:ListHostedZones` 和 `route53:ListResourceRecordSets`。

屬性	Value
附加元件名稱	`external-dns`
命名空間	`external-dns`
文件	GitHub 讀我檔案
服務帳戶名稱	`external-dns`

屬性	Value
受管 IAM 政策	`arn:aws: iam::aws:policy/AmazonRoute53FullAccess`
自訂 IAM 許可	無

Fluent Bit

Fluent Bit 是一個輕量級且高效的日誌處理器和轉發器。它可讓您從不同的來源收集資料/日誌，統一它們，並將其傳送至多個目的地，包括 Amazon CloudWatch Logs、Amazon S3 和 Amazon Data Firehose。Fluent Bit 的設計考量了效能和資源效率，使其成為 Kubernetes 環境的理想選擇。

此附加元件在預設組態中不需要 IAM 許可。不過，如果您設定 AWS 輸出位置，您可能需要授予此附加元件 IAM 許可。如需詳細資訊，請參閱the section called “使用 Pod 身分識別”。

屬性	Value
附加元件名稱	`fluent-bit`
命名空間	`fluent-bit`
文件	Fluent 位元文件
服務帳戶名稱	`fluent-bit`
受管 IAM 政策	無
自訂 IAM 許可	無

檢視屬性

您可下載社群附加元件的開放原始碼屬性和授權資訊。

1. 確認您要下載歸屬資訊的附加元件的名稱與版本。
2. 使用名稱和版本更新以下命令：

```
curl -O https://amazon-eks-docs.s3.amazonaws.com/attributions/<add-on-name>/<add-on-version>/attributions.zip
```

例如：

```
curl -O https://amazon-eks-docs.s3.amazonaws.com/attributions/kube-state-metrics/v2.14.0-eksbuild.1/attributions.zip
```

3. 使用該命令下載檔案。

使用此 zip 檔案來檢視有關授權屬性的資訊。

AWS Marketplace 附加元件

除了先前的 Amazon EKS 附加元件清單之外，您還可以新增來自獨立軟體廠商的各種操作軟體 Amazon EKS 附加元件。選擇附加元件以進一步了解它及其安裝需求。

Accuknox

附加元件名稱是 accuknox_kubearmor 且命名空間是 kubearmor。Accuknox 發布了附加元件。

若要了解附加元件的相關資訊，請參閱 KubeArmor 文件中的 KubeArmor 入門。

服務帳戶名稱

服務帳戶不適用於此附加元件。

AWS 受管 IAM 政策

受管政策不適用於此附加元件。

自訂 IAM 許可

自訂許可不適用於此附加元件。

Akuity

附加元件名稱是 `akuity_agent` 且命名空間是 `akuity`。Akuity 發布了附加元件。

如需有關附加元件的資訊，請參閱 Akuity 平台文件中使用 Akuity EKS 附加元件在 Amazon EKS 上安裝 Akuity 代理程式。

服務帳戶名稱

服務帳戶不適用於此附加元件。

AWS 受管 IAM 政策

受管政策不適用於此附加元件。

自訂 IAM 許可

自訂許可不適用於此附加元件。

Calyptia

附加元件名稱是 `calyptia_fluent-bit` 且命名空間是 `calyptia-fluentbit`。Calyptia 發布了附加元件。

若要了解附加元件的相關資訊，請參閱 Calyptia 文件網站上的 Calyptia 核心代理程式入門。

服務帳戶名稱

服務帳戶名稱是 `calyptia-fluentbit`。

AWS 受管 IAM 政策

此附加元件會使用 AWSMarketplaceMeteringRegisterUsage 受管政策。如需詳細資訊，請參閱《AWS 受管政策參考指南》中的 AWSMarketplaceMeteringRegisterUsage。

建立必要 IAM 角色的命令

您需要具有 IAM OpenID Connect (OIDC) 提供者才能使用下列命令。若要判斷您是否已經擁有一個，或是需要建立一個，請參閱 the section called “IAM OIDC 提供商”。使用您的叢集名稱取代 *my-cluster*，並使用您的角色名稱取代 *my-calyptia-role*。此命令要求您在裝置上安裝 eksctl。如果您需要使用其他工具來建立角色並為 Kubernetes 服務帳戶加上註釋，請參閱 the section called “指派 IAM 角色”。

```
eksctl create iamserviceaccount --name service-account-name  --namespace calyptia-
fluentbit --cluster my-cluster --role-name my-calyptia-role \
    --role-only --attach-policy-arn arn:aws: iam::aws:policy/
AWSMarketplaceMeteringRegisterUsage --approve
```

Cisco Observability Collector

附加元件名稱是 cisco_cisco-cloud-observability-collectors 且命名空間是 appdynamics。Cisco 發布了附加元件。

如需附加元件的資訊，請參閱 Cisco AppDynamics 文件中的使用 Cisco Cloud Observability AWS Marketplace 附加元件。 AppDynamics

服務帳戶名稱

服務帳戶不適用於此附加元件。

AWS 受管 IAM 政策

受管政策不適用於此附加元件。

自訂 IAM 許可

自訂許可不適用於此附加元件。

Cisco Observability Operator

附加元件名稱是 cisco_cisco-cloud-observability-operators 且命名空間是 appdynamics。Cisco 發布了附加元件。

如需附加元件的資訊，請參閱 Cisco AppDynamics 文件中的使用 Cisco Cloud Observability AWS Marketplace 附加元件。 AppDynamics

服務帳戶名稱

服務帳戶不適用於此附加元件。

AWS 受管 IAM 政策

受管政策不適用於此附加元件。

自訂 IAM 許可

自訂許可不適用於此附加元件。

CLOUDSOFT

附加元件名稱是 `cloudsoft_cloudsoft-amp` 且命名空間是 `cloudsoft-amp`。CLOUDSOFT 發布了附加元件。

若要了解附加元件的相關資訊，請參閱 CLOUDSOFT 文件中的 Amazon EKS ADDON。

服務帳戶名稱

服務帳戶不適用於此附加元件。

AWS 受管 IAM 政策

受管政策不適用於此附加元件。

自訂 IAM 許可

自訂許可不適用於此附加元件。

Cribl

附加元件名稱是 `cribl_cribledge` 且命名空間是 `cribledge`。Cribl 發布了附加元件。

若要了解附加元件的相關資訊，請參閱 Cribl 文件中的安裝適用於 Edge 的 Cribl Amazon EKS 附加元件

服務帳戶名稱

服務帳戶不適用於此附加元件。

AWS 受管 IAM 政策

受管政策不適用於此附加元件。

自訂 IAM 許可

自訂許可不適用於此附加元件。

Dynatrace

附加元件名稱是 `dynatrace_dynatrace-operator` 且命名空間是 `dynatrace`。Dynatrace 發布了附加元件。

若要了解附加元件的相關資訊，請參閱 dynatrace 文件中的 Kubernetes 監控。

服務帳戶名稱

服務帳戶不適用於此附加元件。

AWS 受管 IAM 政策

受管政策不適用於此附加元件。

自訂 IAM 許可

自訂許可不適用於此附加元件。

Datree

附加元件名稱是 `datree_engine-pro` 且命名空間是 `datree`。Datree 發布了附加元件。

如需附加元件的資訊，請參閱 Datree 文件中的 Amazon EKS 整合。

服務帳戶名稱

服務帳戶名稱是 datree-webhook-server-awsmp。

AWS 受管 IAM 政策

受管政策是 AWSLicenseManagerConsumptionPolicy。如需詳細資訊，請參閱《 AWS 受管政策參考指南》中的 AWSLicenseManagerConsumptionPolicy。

建立必要 IAM 角色的命令

您需要具有 IAM OpenID Connect (OIDC) 提供者才能使用下列命令。若要判斷您是否已經擁有一個，或是需要建立一個，請參閱 the section called “IAM OIDC 提供商”。使用您的叢集名稱取代 *my-*

cluster，並使用您的角色名稱取代 *my-datree-role*。此命令要求您在裝置上安裝 eksctl。如果您需要使用其他工具來建立角色並為 Kubernetes 服務帳戶加上註釋，請參閱 the section called “指派 IAM 角色”。

```
eksctl create iamserviceaccount --name datree-webhook-server-awsmp --namespace datree
 --cluster my-cluster --role-name my-datree-role \
    --role-only --attach-policy-arn arn:aws: iam::aws:policy/service-role/
AWSLicenseManagerConsumptionPolicy --approve
```

自訂許可

自訂許可不適用於此附加元件。

Datadog

附加元件名稱是 `datadog_operator` 且命名空間是 `datadog-agent`。Datadog 發布了附加元件。

若要了解附加元件的相關資訊，請參閱 Datadog 文件中的藉助 Datadog Operator 附加元件在 Amazon EKS 上安裝 Datadog Agent。

服務帳戶名稱

服務帳戶不適用於此附加元件。

AWS 受管 IAM 政策

受管政策不適用於此附加元件。

自訂 IAM 許可

自訂許可不適用於此附加元件。

Groundcover

附加元件名稱是 `groundcover_agent` 且命名空間是 `groundcover`。Groundcover 發布了附加元件。

若要了解附加元件的相關資訊，請參閱 Groundcover 文件中的安裝 Groundcover Amazon EKS 附加元件。

服務帳戶名稱

服務帳戶不適用於此附加元件。

AWS 受管 IAM 政策

受管政策不適用於此附加元件。

自訂 IAM 許可

自訂許可不適用於此附加元件。

IBM Instana

附加元件名稱是 `instana-agent` 且命名空間是 `instana-agent`。IBM 發布了附加元件。

如需附加元件的資訊，請參閱 AWS 部落格中的使用 Instana Amazon EKS 附加元件實作 Amazon EKS 工作負載的可觀測性，以及使用 IBM Instana 和 Kubecost 監控和最佳化 Amazon EKS 成本。

Instana Observability (Instana) 提供了 Amazon EKS 附加元件，能夠部署 Instana 代理程式至 Amazon EKS 叢集。客戶可藉助此附加元件來收集及分析即時效能資料，以便深入洞察其容器化應用程式。Instana Amazon EKS 附加元件在 Kubernetes 環境中提供了可視性。部署後，Instana 代理程式可自動探索 Amazon EKS 叢集內的元件，包括節點、命名空間、部署、服務與 Pod。

服務帳戶名稱

服務帳戶不適用於此附加元件。

AWS 受管 IAM 政策

受管政策不適用於此附加元件。

自訂 IAM 許可

自訂許可不適用於此附加元件。

Grafana Labs

附加元件名稱是 `grafana-labs_kubernetes-monitoring` 且命名空間是 `monitoring`。Grafana Labs 發布了附加元件。

若要了解附加元件的相關資訊，請參閱 Grafana Labs 文件中的藉助 Amazon EKS 將 Kubernetes 監控設定為附加元件。

服務帳戶名稱

服務帳戶不適用於此附加元件。

AWS 受管 IAM 政策

受管政策不適用於此附加元件。

自訂 IAM 許可

自訂許可不適用於此附加元件。

Guance

- 發布者 – GUANCE
- 名稱 – `guance_datakit`
- 命名空間：`datakit`
- 服務帳戶名稱 – 服務帳戶不適用於此附加元件。
- AWS 受管 IAM 政策 – 受管政策不適用於此附加元件。
- 自訂 IAM 許可 – 自訂許可不適用於此附加元件。
- 設定與使用說明 – 請參閱 Guance 文件中的使用 Amazon EKS 附加元件。

HA 代理

名稱是 `haproxy-technologies_kubernetes-ingress-ee` 且命名空間是 `haproxy-controller`。HA Proxy 發布了附加元件。

如需附加元件的詳細資訊，請參閱 HA Proxy 文件中的在 上安裝 AWS 。

服務帳戶名稱

服務帳戶名稱是 `customer defined`。

AWS 受管 IAM 政策

受管政策是 AWSLicenseManagerConsumptionPolicy。如需詳細資訊，請參閱《 AWS 受管政策參考指南》中的 AWSLicenseManagerConsumptionPolicy。

建立必要 IAM 角色的命令

您需要具有 IAM OpenID Connect (OIDC) 提供者才能使用下列命令。若要判斷您是否已經擁有一個，或是需要建立一個，請參閱 the section called “IAM OIDC 提供商”。使用您的叢集名稱取代 *`my-cluster`*，並使用您的角色名稱取代 *`my-haproxy-role`*。此命令要求您在裝置上安裝 eksctl。如果

您需要使用其他工具來建立角色並為 Kubernetes 服務帳戶加上註釋，請參閱 the section called “指派 IAM 角色”。

```
eksctl create iamserviceaccount --name service-account-name  --namespace haproxy-
controller --cluster my-cluster --role-name my-haproxy-role \
    --role-only --attach-policy-arn arn:aws: iam::aws:policy/service-role/
AWSLicenseManagerConsumptionPolicy --approve
```

自訂許可

自訂許可不適用於此附加元件。

Kpow

附加元件名稱是 `factorhouse_kpow` 且命名空間是 `factorhouse`。Factorhouse 發布了附加元件。

若要了解附加元件的相關資訊，請參閱 Kpow 文件中的 AWS Marketplace LM。

服務帳戶名稱

服務帳戶名稱是 kpow。

AWS 受管 IAM 政策

受管政策是 AWSLicenseManagerConsumptionPolicy。如需詳細資訊，請參閱《 AWS 受管政策參考指南》中的 AWSLicenseManagerConsumptionPolicy。

建立必要 IAM 角色的命令

您需要具有 IAM OpenID Connect (OIDC) 提供者才能使用下列命令。若要判斷您是否已經擁有一個，或是需要建立一個，請參閱 the section called “IAM OIDC 提供商”。將 *my-cluster* 取代為您的叢集名稱，並將 *my-role-name* 取代為您的角色名稱。此命令要求您在裝置上安裝 eksctl。如果您需要使用其他工具來建立角色並為 Kubernetes 服務帳戶加上註釋，請參閱 the section called “指派 IAM 角色”。

```
eksctl create iamserviceaccount --name kpow --namespace factorhouse --cluster my-
cluster --role-name my-kpow-role \
    --role-only --attach-policy-arn arn:aws: iam::aws:policy/service-role/
AWSLicenseManagerConsumptionPolicy --approve
```

自訂許可

自訂許可不適用於此附加元件。

Kubecost

附加元件名稱是 `kubecost_kubecost` 且命名空間是 `kubecost`。Kubecost 發布了附加元件。

若要了解附加元件的相關資訊，請參閱 Kubecost 文件中的AWS 雲端帳單整合。

叢集上必須安裝具有 Amazon EBS 的 Store Kubernetes 磁碟區。否則，您會收到錯誤。

服務帳戶名稱

服務帳戶不適用於此附加元件。

AWS 受管 IAM 政策

受管政策不適用於此附加元件。

自訂 IAM 許可

自訂許可不適用於此附加元件。

Kasten

附加元件名稱是 `kasten_k10` 且命名空間是 `kasten-io`。Kasten by Veeam 發布了附加元件。

如需有關附加元件的資訊，請參閱 Kasten 文件中的AWS 使用 Amazon EKS 附加元件在 上安裝 K10。

您必須透過預設的 `StorageClass`，在叢集上安裝 Amazon EBS CSI 驅動程式。

服務帳戶名稱

服務帳戶名稱是 `k10-k10`。

AWS 受管 IAM 政策

受管政策是 AWSLicenseManagerConsumptionPolicy。如需詳細資訊，請參閱《 AWS 受管政策參考指南》中的 AWSLicenseManagerConsumptionPolicy。

建立必要 IAM 角色的命令

您需要具有 IAM OpenID Connect (OIDC) 提供者才能使用下列命令。若要判斷您是否已經擁有一個，或是需要建立一個，請參閱 the section called “IAM OIDC 提供商”。使用您的叢集名稱取代 *`my-cluster`*，並使用您的角色名稱取代 *`my-kasten-role`*。此命令要求您在裝置上安裝 eksctl。如果您需要使用其他工具來建立角色並為 Kubernetes 服務帳戶加上註釋，請參閱 the section called “指派 IAM 角色”。

```
eksctl create iamserviceaccount --name k10-k10 --namespace kasten-io --cluster my-
cluster --role-name my-kasten-role \
    --role-only --attach-policy-arn arn:aws: iam::aws:policy/service-role/
AWSLicenseManagerConsumptionPolicy --approve
```

自訂許可

自訂許可不適用於此附加元件。

Kong

附加元件名稱是 kong_konnect-ri 且命名空間是 kong。Kong 發布了附加元件。

若要了解附加元件的相關資訊，請參閱 Kong 文件中的安裝 Kong Gateway EKS 附加元件。

叢集上必須安裝具有 Amazon EBS 的 Store Kubernetes 磁碟區。否則，您會收到錯誤。

服務帳戶名稱

服務帳戶不適用於此附加元件。

AWS 受管 IAM 政策

受管政策不適用於此附加元件。

自訂 IAM 許可

自訂許可不適用於此附加元件。

LeakSignal

附加元件名稱是 leaksignal_leakagent 且命名空間是 leakagent。LeakSignal 發布了附加元件。

如需附加元件的詳細資訊，請參閱 LeakSignal 文件中的 https://www.leaksignal.com/docs/LeakAgent/Deployment/AWS%20EKS%20Addon/【Install the LeakAgent add-on】

叢集上必須安裝具有 Amazon EBS 的 Store Kubernetes 磁碟區。否則，您會收到錯誤。

服務帳戶名稱

服務帳戶不適用於此附加元件。

AWS 受管 IAM 政策

受管政策不適用於此附加元件。

自訂 IAM 許可

自訂許可不適用於此附加元件。

NetApp

附加元件名稱是 `netapp_trident-operator` 且命名空間是 `trident`。NetApp 發布了附加元件。

若要了解附加元件的相關資訊，請參閱 NetApp 文件中的設定 Trident EKS 附加元件。

服務帳戶名稱

服務帳戶不適用於此附加元件。

AWS 受管 IAM 政策

受管政策不適用於此附加元件。

自訂 IAM 許可

自訂許可不適用於此附加元件。

New Relic

附加元件名稱是 `new-relic_kubernetes-operator` 且命名空間是 `newrelic`。New Relic 發布了附加元件。

若要了解附加元件的相關資訊，請參閱 New Relic 文件中的安裝適用於 EKS 的 New Relic 附加元件。

服務帳戶名稱

服務帳戶不適用於此附加元件。

AWS 受管 IAM 政策

受管政策不適用於此附加元件。

自訂 IAM 許可

自訂許可不適用於此附加元件。

Rafay

附加元件名稱是 `rafay-systems_rafay-operator` 且命名空間是 `rafay-system`。Rafay 發布了附加元件。

若要了解附加元件的相關資訊，請參閱 Rafay 文件中的安裝 Rafay Amazon EKS 附加元件。

服務帳戶名稱

服務帳戶不適用於此附加元件。

AWS 受管 IAM 政策

受管政策不適用於此附加元件。

自訂 IAM 許可

自訂許可不適用於此附加元件。

Rad Security

- 發布者 – RAD SECURITY
- 名稱 – `rad-security_rad-security`
- 命名空間：`ksoc`
- 服務帳戶名稱 – 服務帳戶不適用於此附加元件。
- AWS 受管 IAM 政策 – 受管政策不適用於此附加元件。
- 自訂 IAM 許可 – 自訂許可不適用於此附加元件。
- 設定和使用說明 – 請參閱 Rad Security 文件中的透過 AWS Marketplace 安裝 Rad。

SolarWinds

- 發布者 – SOLARWINDS
- 名稱 – `solarwinds_swo-k8s-collector-addon`
- 命名空間：`solarwinds`
- 服務帳戶名稱 – 服務帳戶不適用於此附加元件。
- AWS 受管 IAM 政策 – 受管政策不適用於此附加元件。
- 自訂 IAM 許可 – 自訂許可不適用於此附加元件。
- 設定與使用說明 – 請參閱 SolarWinds 文件中的監控 Amazon EKS 叢集。

Solo

附加元件名稱是 `solo-io_istio-distro` 且命名空間是 `istio-system`。Solo 發布了附加元件。

如需附加元件的詳細資訊，請參閱 https：//Solo.io 文件中的安裝 Istio。

服務帳戶名稱

服務帳戶不適用於此附加元件。

AWS 受管 IAM 政策

受管政策不適用於此附加元件。

自訂 IAM 許可

自訂許可不適用於此附加元件。

Snyk

- 發布者 – SNYK
- 名稱 – `snyk_runtime-sensor`
- 命名空間：`snyk_runtime-sensor`
- 服務帳戶名稱 – 服務帳戶不適用於此附加元件。
- AWS 受管 IAM 政策 – 受管政策不適用於此附加元件。
- 自訂 IAM 許可 – 自訂許可不適用於此附加元件。
- 設定與使用說明 – 請參閱 Snyk 使用者文件中的 Snyk 執行時期感應器。

Stormforge

附加元件名稱是 `stormforge_optimize-Live` 且命名空間是 `stormforge-system`。Stormforge 發布了附加元件。

若要了解附加元件的相關資訊，請參閱 StormForge 文件中的安裝 StormForge 代理程式。

服務帳戶名稱

服務帳戶不適用於此附加元件。

AWS 受管 IAM 政策

受管政策不適用於此附加元件。

自訂 IAM 許可

自訂許可不適用於此附加元件。

SUSE

- 發布者 – SUSE
- 名稱 – `suse_observability-agent`
- 命名空間：`suse-observability`
- 服務帳戶名稱 – 服務帳戶不適用於此附加元件。
- AWS 受管 IAM 政策 – 受管政策不適用於此附加元件。
- 自訂 IAM 許可 – 自訂許可不適用於此附加元件。
- 設定與使用說明 – 請參閱 SUSE 文件中 Quick Start。

Splunk

附加元件名稱是 `splunk_splunk-otel-collector-chart` 且命名空間是 `splunk-monitoring`。Splunk 發布了附加元件。

若要了解附加元件的相關資訊，請參閱 Splunk 文件中的安裝適用於 Amazon EKS 的 Splunk 附加元件。

服務帳戶名稱

服務帳戶不適用於此附加元件。

AWS 受管 IAM 政策

受管政策不適用於此附加元件。

自訂 IAM 許可

自訂許可不適用於此附加元件。

Teleport

附加元件名稱是 `teleport_teleport` 且命名空間是 `teleport`。Teleport 發布了附加元件。

若要了解附加元件的相關資訊，請參閱 Teleport 文件中的 Teleport 運作方式。

服務帳戶名稱

服務帳戶不適用於此附加元件。

AWS 受管 IAM 政策

受管政策不適用於此附加元件。

自訂 IAM 許可

自訂許可不適用於此附加元件。

Tetrate

附加元件名稱是 `tetrate-io_istio-distro` 且命名空間是 `istio-system`。Tetrate Io 發布了附加元件。

若要了解附加元件的相關資訊，請參閱 Tetrate Istio Distro 網站。

服務帳戶名稱

服務帳戶不適用於此附加元件。

AWS 受管 IAM 政策

受管政策不適用於此附加元件。

自訂 IAM 許可

自訂許可不適用於此附加元件。

Upbound Universal Crossplane

附加元件名稱是 `upbound_universal-crossplane` 且命名空間是 `upbound-system`。Upbound 發布了附加元件。

若要了解附加元件的相關資訊，請參閱 Upbound 文件中的 Upbound Universal Crossplane (UXP)。

服務帳戶名稱

服務帳戶不適用於此附加元件。

AWS 受管 IAM 政策

受管政策不適用於此附加元件。

自訂 IAM 許可

自訂許可不適用於此附加元件。

Upwind

附加元件名稱是 upwind 且命名空間是 upwind。Upwind 發布了附加元件。

若要了解附加元件的相關資訊，請參閱 Upwind 文件。

服務帳戶名稱

服務帳戶不適用於此附加元件。

AWS 受管 IAM 政策

受管政策不適用於此附加元件。

自訂 IAM 許可

自訂許可不適用於此附加元件。

建立 Amazon EKS 附加元件

Amazon EKS 附加元件是 Amazon EKS 叢集彙整的附加元件軟體。所有 Amazon EKS 附加元件：

- 包括最新的安全性修補程式和錯誤修正。
- 經過 驗證 AWS ，可使用 Amazon EKS。
- 減少管理附加元件軟體所需的工作量。

您可以使用 `eksctl`、 AWS 管理主控台或 CLI 建立 Amazon EKS AWS 附加元件。如果附加元件需要 IAM 角色，則請參閱 Amazon EKS 附加元件中特定附加元件的詳細資訊，以了解有關建立角色的詳細資訊。

先決條件

請完成以下操作再建立附加元件：

- 您必須先存在叢集，才能為其建立附加元件。如需詳細資訊，請參閱the section called “建立 叢集”。
- 檢查附加元件是否需要 IAM 角色。如需詳細資訊，請參閱the section called “確認相容性”。

- 確認 Amazon EKS 附加元件版本與叢集相容。如需詳細資訊，請參閱the section called “確認相容性”。
- 確認您的電腦或 AWS CloudShell 上安裝了 0.190.0 版或更新版本的eksctl命令列工具。如需詳細資訊，請參閱 eksctl 網站文章安裝。

程序

您可以使用 eksctl、 AWS 管理主控台或 CLI 建立 Amazon EKS AWS 附加元件。如果附加元件需要 IAM 角色，則請參閱 AWS中可用的 Amazon EKS 附加元件中的特定附加元件相關詳細資訊，了解有關建立角色的詳細資訊。

建立附加元件 (eksctl)

1. 檢視叢集版本可用附加元件的名稱。將 *1.36* 取代為您的叢集版本。

```
eksctl utils describe-addon-versions --kubernetes-version 1.36 | grep AddonName
```

範例輸出如下。

```
"AddonName": "aws-ebs-csi-driver",
                        "AddonName": "coredns",
                        "AddonName": "kube-proxy",
                        "AddonName": "vpc-cni",
                        "AddonName": "adot",
                        "AddonName": "dynatrace_dynatrace-operator",
                        "AddonName": "upbound_universal-crossplane",
                        "AddonName": "teleport_teleport",
                        "AddonName": "factorhouse_kpow",
                        [...]
```

2. 檢視您想要建立之附加元件的可用版本。將 *1.36* 取代為您的叢集版本。請使用您想要檢視其版本之附加元件的名稱取代 *name-of-addon*。名稱必須是之前步驟中傳回的其中一個名稱。

```
eksctl utils describe-addon-versions --kubernetes-version 1.36 --name name-of-addon |
 grep AddonVersion
```

下列輸出是針對名為 vpc-cni 的附加元件傳回的範例。您可以看到附加元件有數個可用版本。

```
"AddonVersions": [
```

```
    "AddonVersion": "v1.12.0-eksbuild.1",
    "AddonVersion": "v1.11.4-eksbuild.1",
    "AddonVersion": "v1.10.4-eksbuild.1",
    "AddonVersion": "v1.9.3-eksbuild.1",
```

a. 決定您要建立的附加元件是 Amazon EKS 還是 AWS Marketplace 附加元件。 AWS Marketplace 具有第三方附加元件，要求您完成建立附加元件的額外步驟。

```
eksctl utils describe-addon-versions --kubernetes-version 1.36 --name name-of-addon | grep ProductUrl
```

如果未傳回任何輸出，則附加元件是 Amazon EKS 附加元件。如果傳回輸出，則附加元件是 AWS Marketplace 附加元件。下列輸出適用於名為 `teleport_teleport` 的附加元件。

```
"ProductUrl": "https://aws.amazon.com/marketplace/pp?sku=3bda70bb-566f-4976-806c-f96faef18b26"
```

您可以使用傳回的 URL，進一步了解 AWS Marketplace 中的附加元件。如果附加元件需要訂閱，您可以透過 AWS Marketplace 訂閱附加元件。如果您要從 AWS Marketplace 建立附加元件，則您用來建立附加元件的 IAM 主體必須具有建立 AWSServiceRoleForAWSLicenseManagerRole 服務連結角色的許可。如需有關將許可指派給 IAM 實體的詳細資訊，請參閱《IAM 使用者指南》中的新增和移除 IAM 身分許可。

3. 建立 Amazon EKS 附加元件。複製命令並取代 *user-data*，如下所示：
   - 使用您叢集的名稱取代 *my-cluster*。
   - 使用您要建立之附加元件的名稱取代 *name-of-addon*。
   - 如果您想要比最新版本更早的附加元件版本，請使用您要使用的上一步輸出中傳回的版本編號取代*####*。
   - 如果附加元件使用服務帳戶角色，請使用您的帳戶 ID 取代 *111122223333*，並使用角色名稱取代 *role-name*。如需為您的服務帳戶建立角色的說明，請參閱您所建立之附加元件的文件。若要獲取附加元件清單，請參閱 the section called “ AWS 附加元件”。若要指定服務帳戶角色，您的叢集需要具有 IAM OpenID Connect (OIDC) 提供者。若要判定您的叢集是否已經擁有一個提供者，或是要建立一個提供者，請參閱 the section called “IAM OIDC 提供商”。

     如果附加元件未使用服務帳戶角色，請刪除 `--service-account-role-arn arn:aws:iam::111122223333:role/role-name`。
   - 此範例命令會覆寫任何現有自我管理附加元件版本的組態 (如果有的話)。如果您不想覆寫現有自我管理附加元件的組態，請移除 *--force* 選項。如果您移除該選項，並且 Amazon EKS 附加元

件需要覆蓋現有自我管理附加元件的組態，則建立 Amazon EKS 附加元件的操作會失敗並會出現一條錯誤訊息，以幫助您解決衝突。指定此選項之前，請確定 Amazon EKS 附加元件未管理您需要管理的設定，因為這些設定會被此選項覆寫。

```
eksctl create addon --cluster my-cluster --name name-of-addon --version latest \
    --service-account-role-arn arn:aws: iam::111122223333:role/role-name --force
```

您可以查看該命令所有可用選項的清單。

```
eksctl create addon --help
```

如需有關可用選項的詳細資訊，請參閱 `eksctl` 文件中的 Addons (附加元件)。

建立附加元件AWS （主控台）

1. 開啟 Amazon EKS 主控台。
2. 在左側導覽窗格中選擇叢集。
3. 選擇您想要建立附加元件的叢集名稱。
4. 選擇附加元件索引標籤。
5. 選擇取得更多附加元件。
6. 在 Select add-ons (選取附加元件) 頁面上，選擇您要新增至叢集的附加元件。您可以視需要新增任意數量的 Amazon EKS 附加元件和 AWS Marketplace 附加元件。

   對於 AWS Marketplace 附加元件，您用來建立附加元件的 IAM 主體必須具有從 AWS LicenseManager 讀取附加元件權利的許可。 AWS LicenseManager 需要 AWSServiceRoleForAWSLicenseManagerRole 服務連結角色 (SLR)，以允許 AWS 資源代表您管理授權。SLR 是每個帳戶的一次性要求，您不必為每個附加元件或每個叢集建立單獨的 SLR。如需有關將許可指派給 IAM 主體的詳細資訊，請參閱《IAM 使用者指南》中的新增和移除 IAM 身分許可。

   如果您想要安裝的 AWS Marketplace 附加元件未列出，您可以選擇頁面編號以檢視其他頁面結果或在搜尋方塊中搜尋。您也可以在篩選選項中，依類別、廠商或定價模式搜尋，然後從搜尋結果中選擇附加元件。選取您要安裝的附加元件後，請選擇 Next (下一步)。
7. 在設定選取的附加元件設定頁面上，執行以下操作：
   a. 選擇檢視訂閱選項即可開啟訂閱選項表單。檢閱定價詳細資料和法律區段，然後選擇訂閱按鈕以繼續。

b. 針對版本，選取您想要安裝的版本。除非您建立的個別附加元件建議使用不同的版本，否則我們建議您使用標記為最新的版本。若要判斷附加元件是否有建議的版本，請參閱您所建立之附加元件的文件。若要獲取附加元件清單，請參閱 the section called " AWS 附加元件"。

c. 您可使用兩個用於設定附加元件角色的選項：EKS Pod 身分識別 IAM 角色及服務帳戶 (IRSA) 的 IAM 角色。根據您喜好的選項，遵循下列適當步驟。如果您選取的所有附加元件在狀態下都有需要訂閱，請選擇下一步。在您建立叢集之後訂閱這些附加元件之前，您無法進一步設定這些附加元件。針對狀態下沒有需要訂閱的附加元件，請執行以下操作：

   i. 若是服務帳戶的 Pod 身分識別 IAM 角色，您可使用現有的 EKS Pod 身分識別 IAM 角色，或是使用建立建議的角色按鈕來建立一個。此欄位僅提供包含適當信任政策的選項。若沒有可選取的角色，則表示您沒有包含相符信任政策的現有角色。如需針對所選附加元件的服務帳戶設定 EKS Pod 身分識別 IAM 角色，則選擇建立建議的角色。角色建立精靈會在單獨的視窗中開啟。精靈會自動填入如下所示角色資訊。針對您想要建立 EKS Pod 身分識別 IAM 角色的每個附加元件，完成如下所示 IAM 精靈中的步驟。

      - 在選取信任實體步驟中，EKS AWS 的服務選項和 EKS - Pod Identity 的使用案例會預先選取，且會自動填入附加元件的適當信任政策。例如，將會如 the section called "EKS Pod 身分識別的優勢" 中詳述，透過包含 pods.eks.amazonaws.com IAM Principal 的適當信任政策來建立角色。選擇下一步。
      - 在新增許可步驟中，將會針對附加元件預先選取適用於該角色政策的適當受管政策。舉例來說，針對 Amazon VPC CNI 附加元件，將會透過如 the section called "Kubernetes 專用 Amazon VPC CNI 外掛程式" 中詳述的受管政策 `AmazonEKS_CNI_Policy` 來建立角色。選擇下一步。
      - 在命名、檢閱及建立步驟中，角色名稱會自動填入附加元件的預設角色名稱。舉例來說，針對 Amazon VPC CNI 附加元件，將會使用名稱 AmazonEKSPodIdentityAmazonVPCCNIRole 來建立角色。在描述中，預設描述會自動填入附加元件的適當描述。例如，對於 Amazon VPC CNI 附加元件，將使用描述建立角色 允許在 Amazon EKS 叢集中執行的 Pod 存取 AWS 資源。在信任政策中，檢視針對附加元件填入的信任政策。選擇建立角色。

        注意：保留預設角色名稱，能夠讓 EKS 在新叢集中或新增附加元件至現有叢集時，預先選取附加元件的角色。您仍可覆寫此名稱，並且可在叢集的附加元件使用該角色，但需要從下拉式選單中手動選取該角色。

   ii. 若要了解狀態項下沒有需要訂閱選項的附加元件，以及您想要使用 IRSA 設定角色的位置，請參閱您要建立的附加元件相關文件，以便建立 IAM 政策並與角色連接。若要獲取附加元件清單，請參閱 the section called " AWS 附加元件"。若要選取 IAM 角色，您的叢集需要具有 IAM

OpenID Connect (OIDC) 提供者。若要判定您的叢集是否已經擁有一個提供者，或是要建立一個提供者，請參閱 the section called “IAM OIDC 提供商”。

iii. 選擇 Optional configuration settings (選用組態設定)。

iv. 如果附加元件需要組態，請在 Configuration values (組態值) 方塊中輸入。若要判斷附加元件是否需要組態資訊，請參閱您所建立之附加元件的文件。若要獲取附加元件清單，請參閱 the section called “ AWS 附加元件”。

v. 對於衝突解決方法，請選取其中一個可用選項。對於衝突解決方法，如果您選擇覆寫，就可以使用 Amazon EKS 附加元件的設定來覆寫現有附加元件的一種或多種設定。若未啟用此選項，而且有設定與現有設定發生衝突，則操作會失敗。您可以使用產生的錯誤訊息對此衝突進行疑難排解。在選擇此選項之前，請確認 Amazon EKS 附加元件未管理您需要自我管理的設定。

vi. 如果您想要將附加元件安裝到特定命名空間，在命名空間欄位中輸入附加元件。對於 AWS 和社群附加元件，您可以定義要安裝附加元件的自訂 Kubernetes 命名空間。如需詳細資訊，請參閱the section called “附加元件的自訂命名空間”。

vii.選擇下一步。

8. 在檢閱並新增頁面上，選擇建立。附加元件安裝完成後，您會看到已安裝的附加元件。
9. 如果您安裝的任何附加元件需要訂閱，請完成下列步驟：
   a. 選擇附加元件右下角的 Subscribe (訂閱) 按鈕。系統會將您導向 AWS Marketplace 中附加元件的頁面。請閱讀附加元件的相關資訊，例如其產品概觀和定價資訊。
   b. 選取附加元件頁面右上方的 Continue to Subscribe (繼續訂閱) 按鈕。
   c. 請完整閱讀條款與條件。如果您同意，請選擇 Accept Terms (接受條款)。處理訂閱可能需要幾分鐘的時間。訂閱正在處理時，Return to Amazon EKS Console (返回至 Amazon EKS 主控台) 按鈕會呈現灰色。
   d. 訂閱完成處理後，Return to Amazon EKS Console (返回至 Amazon EKS 主控台) 按鈕將不再呈現灰色。選擇按鈕以返回叢集的 Amazon EKS 主控台 Add-ons (附加元件) 索引標籤。
   e. 針對您訂閱的附加元件，請選擇 Remove and reinstall (移除後重新安裝)，然後選擇 Reinstall add-on (重新安裝附加元件)。安裝附加元件可能需要幾分鐘時間。安裝完成後，您可以設定附加元件。

建立附加元件 (AWS CLI)

1. 您需要在裝置或 AWS CloudShell 上安裝和設定版本 `2.12.3`或更新版本或版本 `1.27.160`或更新版本的 AWS 命令列界面 (AWS CLI)。若要檢查您目前的版本，請使用 `aws --version | cut -d / -f2 | cut -d ' ' -f1`。適用於 macOS 的 `yum`、 `apt-get`或 Homebrew 等套件管理員

通常是最新版本 CLI AWS 後面的幾個版本。若要安裝最新版本，請參閱《 AWS 命令列界面使用者指南》中的使用 aws 設定安裝 和快速組態。 https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-quickstart.html#cli-configure-quickstart-config在 AWS CloudShell 中安裝的 AWS CLI 版本也可能是最新版本後面的幾個版本。若要更新它，請參閱《CloudShell AWS 使用者指南》中的將 CLI 安裝到您的主目錄。 AWS CloudShell

2. 判斷有哪些附加元件可用。您可以查看所有可用的附加元件、其類型及其發佈者。您也可以查看可透過 AWS Marketplace 取得的附加元件 URL。將 *1.36* 取代為您的叢集版本。

```
aws eks describe-addon-versions --kubernetes-version 1.36 \
    --query 'addons[].{MarketplaceProductUrl: marketplaceInformation.productUrl,
 Name: addonName, Owner: owner, Publisher: publisher, Type: type}' --output table
```

範例輸出如下。

```
----------------------------------------------------------------------------------------------------
|
 DescribeAddonVersions
  |
+-------------------------------------------------------------
+------------------------------+------------------+-------------
+---------------------+
|                     MarketplaceProductUrl                     |             Name
          |      Owner       |  Publisher   |        Type         |
+-------------------------------------------------------------
+------------------------------+------------------+-------------
+---------------------+
|  None                                                         |  aws-ebs-csi-driver
          |  aws             |  eks         |  storage            |
|  None                                                         |  coredns
          |  aws             |  eks         |  networking         |
|  None                                                         |  kube-proxy
          |  aws             |  eks         |  networking         |
|  None                                                         |  vpc-cni
          |  aws             |  eks         |  networking         |
|  None                                                         |  adot
          |  aws             |  eks         |  observability      |
| https://aws.amazon.com/marketplace/pp/prodview-brb73nceicv7u |
 dynatrace_dynatrace-operator |  aws-marketplace |  dynatrace   |  monitoring
 |
| https://aws.amazon.com/marketplace/pp/prodview-uhc2iwi5xysoc |  upbound_universal-
crossplane |  aws-marketplace |  upbound     |  infra-management   |
```

```
| https://aws.amazon.com/marketplace/pp/prodview-hd2ydsrgqy4li |  teleport_teleport
          |  aws-marketplace |  teleport    |  policy-management  |
| https://aws.amazon.com/marketplace/pp/prodview-vgghgqdsplhvc |  factorhouse_kpow
          |  aws-marketplace |  factorhouse |  monitoring         |
|  [...]                                                        |  [...]
           |  [...]           |  [...]       |  [...]              |
+-------------------------------------------------------------
+------------------------------+------------------+-------------
+---------------------+
```

您的輸出可能不同。在此範例輸出中，有三種不同 networking 類型的附加元件可用，以及五個具有 eks 類型發佈者的附加元件。Owner 欄位中具有 aws-marketplace 的附加元件可能需要訂閱才能安裝它們。您可以造訪 URL 以深入了解附加元件並訂閱該附加元件。

3. 您可以查看每個附加元件可用的版本。將 *1.36* 取代為您的叢集版本，並將 *vpc-cni* 取代為上一個步驟中傳回的附加元件名稱。

```
aws eks describe-addon-versions --kubernetes-version 1.36 --addon-name vpc-cni \
    --query 'addons[].addonVersions[].{Version: addonVersion, Defaultversion:
 compatibilities[0].defaultVersion}' --output table
```

範例輸出如下。

```
-----------------------------------------
|          DescribeAddonVersions         |
+-----------------+----------------------+
| Defaultversion  |       Version        |
+-----------------+----------------------+
|  False          |  v1.12.0-eksbuild.1  |
|  True           |  v1.11.4-eksbuild.1  |
|  False          |  v1.10.4-eksbuild.1  |
|  False          |  v1.9.3-eksbuild.1   |
+-----------------+----------------------+
```

Defaultversion 欄位中具有 True 的版本在預設情況下是用來建立附加元件的版本。

4. (選用) 執行下列命令，尋找您所選附加元件的組態選項：

```
aws eks describe-addon-configuration --addon-name vpc-cni --addon-version v1.12.0-
eksbuild.1
```

```
{
    "addonName": "vpc-cni",
    "addonVersion": "v1.12.0-eksbuild.1",
    "configurationSchema": "{\"$ref\":\"#/definitions/VpcCni\",\"$schema\":\"http://
json-schema.org/draft-06/schema#\",\"definitions\":{\"Cri\":{\"additionalProperties
\":false,\"properties\":{\"hostPath\":{\"$ref\":\"#/definitions/HostPath\"}},\"title
\":\"Cri\",\"type\":\"object\"},\"Env\":{\"additionalProperties\":false,\"properties
\":{\"ADDITIONAL_ENI_TAGS\":{\"type\":\"string\"},\"AWS_VPC_CNI_NODE_PORT_SUPPORT
\":{\"format\":\"boolean\",\"type\":\"string\"},\"AWS_VPC_ENI_MTU\":{\"format\":
\"integer\",\"type\":\"string\"},\"AWS_VPC_K8S_CNI_CONFIGURE_RPFILTER\":{\"format
\":\"boolean\",\"type\":\"string\"},\"AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG\":{\"format
\":\"boolean\",\"type\":\"string\"},\"AWS_VPC_K8S_CNI_EXTERNALSNAT\":{\"format\":
\"boolean\",\"type\":\"string\"},\"AWS_VPC_K8S_CNI_LOGLEVEL\":{\"type\":\"string\"},
\"AWS_VPC_K8S_CNI_LOG_FILE\":{\"type\":\"string\"},\"AWS_VPC_K8S_CNI_RANDOMIZESNAT
\":{\"type\":\"string\"},\"AWS_VPC_K8S_CNI_VETHPREFIX\":{\"type\":\"string\"},
\"AWS_VPC_K8S_PLUGIN_LOG_FILE\":{\"type\":\"string\"},\"AWS_VPC_K8S_PLUGIN_LOG_LEVEL
\":{\"type\":\"string\"},\"DISABLE_INTROSPECTION\":{\"format\":\"boolean\",\"type
\":\"string\"},\"DISABLE_METRICS\":{\"format\":\"boolean\",\"type\":\"string
\"},\"DISABLE_NETWORK_RESOURCE_PROVISIONING\":{\"format\":\"boolean\",\"type
\":\"string\"},\"ENABLE_POD_ENI\":{\"format\":\"boolean\",\"type\":\"string
\"},\"ENABLE_PREFIX_DELEGATION\":{\"format\":\"boolean\",\"type\":\"string\"},
\"WARM_ENI_TARGET\":{\"format\":\"integer\",\"type\":\"string\"},\"WARM_PREFIX_TARGET
\":{\"format\":\"integer\",\"type\":\"string\"}},\"title\":\"Env\",\"type\":
\"object\"},\"HostPath\":{\"additionalProperties\":false,\"properties\":{\"path
\":{\"type\":\"string\"}},\"title\":\"HostPath\",\"type\":\"object\"},\"Limits
\":{\"additionalProperties\":false,\"properties\":{\"cpu\":{\"type\":\"string
\"},\"memory\":{\"type\":\"string\"}},\"title\":\"Limits\",\"type\":\"object\"},
\"Resources\":{\"additionalProperties\":false,\"properties\":{\"limits\":{\"$ref\":
\"#/definitions/Limits\"},\"requests\":{\"$ref\":\"#/definitions/Limits\"}},\"title
\":\"Resources\",\"type\":\"object\"},\"VpcCni\":{\"additionalProperties\":false,
\"properties\":{\"cri\":{\"$ref\":\"#/definitions/Cri\"},\"env\":{\"$ref\":\"#/
definitions/Env\"},\"resources\":{\"$ref\":\"#/definitions/Resources\"}},\"title\":
\"VpcCni\",\"type\":\"object\"}}}"
}
```

輸出是標準的 JSON 結構描述。

以下是適用於上述結構描述的有效組態值 (以 JSON 格式) 範例。

```
{
  "resources": {
    "limits": {
```

```
      "cpu": "100m"
    }
  }
}
```

以下是適用於上述結構描述的有效組態值 (以 YAML 格式) 範例。

```
resources:
  limits:
    cpu: 100m
```

5. 確定附加元件是否需要 IAM 許可。若是如此，您需要 (1) 確定要使用 EKS Pod 身分識別還是服務帳戶的 IAM 角色 (IRSA)，(2) 確定用於附加元件的 IAM 角色 ARN，以及 (3) 確定附加元件使用的 Kubernetes 服務帳戶名稱。如需詳細資訊，請參閱the section called “擷取 IAM 資訊”。
   - 如果附加元件支援，Amazon EKS 建議使用 EKS Pod 身分識別。這需要您的叢集上已安裝 Pod 身分識別代理程式。如需搭配附加元件使用 Pod 身分識別的詳細資訊，請參閱 the section called “IAM 角色”。
   - 如果未設定 EKS Pod 身分的附加元件或叢集，請使用 IRSA。確認您的叢集上已設定 IRSA。
   - 檢閱 Amazon EKS 附加元件文件，以判斷附加元件是否需要 IAM 許可和相關聯 Kubernetes 服務帳戶的名稱。
     a. 建立 Amazon EKS 附加元件。將隨後的命令複製到您的裝置。視需要對命令進行下列修改，然後執行修改後的命令：
   - 使用您叢集的名稱取代 *my-cluster*。
   - 使用您想要建立之上一步輸出中傳回的附加元件名稱取代 *vpc-cni*。
   - 使用您想要使用之上一步輸出中傳回的版本取代 *version-number*。
   - 如果您想要將附加元件安裝到自訂 Kubernetes 命名空間，請新增 `--namespace-config 'namespace=<my-namespace>'` 選項。此選項僅適用於 AWS 和 社群附加元件。如需詳細資訊，請參閱the section called “附加元件的自訂命名空間”
   - 如果附加元件不需要 IAM 許可，則刪除 *<service-account-configuration>*。
   - 執行以下任意一項：
     - 如果附加元件 (1) 需要 IAM 許可，且 (2) 您的叢集使用 EKS Pod 身分識別，請將 *<service-account-configuration>* 取代為以下 Pod 身分識別關聯。將 *<service-account-name>* 取代為附加元件使用的服務帳戶名稱。將 *<role-arn>* 取代為 IAM 角色的 ARN。該角色必須具有 EKS Pod 身分識別所需的信任政策。

```
--pod-identity-associations 'serviceAccount=<service-account-name>,roleArn=<role-arn>'
```

- 如果附加元件 (1) 需要 IAM 許可，且 (2) 您的叢集使用 IRSA，請將 *<service-account-configuration>* 取代為以下 IRSA 組態。使用帳戶 ID 取代 *111122223333*，並且使用所建立現有 IAM 角色的名稱取代 *role-name*。如需建立角色的說明，請參閱您所建立之附加元件的文件。若要獲取附加元件清單，請參閱 the section called " AWS 附加元件"。若要指定服務帳戶角色，您的叢集需要具有 IAM OpenID Connect (OIDC) 提供者。若要判定您的叢集是否已經擁有一個提供者，或是要建立一個提供者，請參閱 the section called "IAM OIDC 提供商"。

```
--service-account-role-arn arn:aws: iam::111122223333:role/role-name
```

- 如果有的話，這些範例命令會覆寫任何現有自我管理版本的附加元件`--configuration-values`選項。將其取代為所需的組態值，例如字串或檔案輸入。如果您不想提供組態值，請刪除 `--configuration-values` 選項。如果您不希望 AWS CLI 覆寫現有自我管理附加元件的組態，請移除 *`--resolve-conflicts OVERWRITE`* 選項。如果您移除該選項，並且 Amazon EKS 附加元件需要覆蓋現有自我管理附加元件的組態，則建立 Amazon EKS 附加元件的操作會失敗並會出現一條錯誤訊息，以幫助您解決衝突。指定此選項之前，請確定 Amazon EKS 附加元件未管理您需要管理的設定，因為這些設定會被此選項覆寫。

```
aws eks create-addon --cluster-name my-cluster --addon-name vpc-cni --addon-version
 version-number \
    <service-account-configuration> --configuration-values '{"resources":
{"limits":{"cpu":"100m"}}}' --resolve-conflicts OVERWRITE
```

```
aws eks create-addon --cluster-name my-cluster --addon-name vpc-cni --addon-version
 version-number \
   <service-account-configuration> --configuration-values 'file://example.yaml' --
resolve-conflicts OVERWRITE
```

如需可用選項的完整清單，請參閱 Amazon EKS Command Line Reference (《Amazon EKS 命令列參考》) 中的 `create-addon` 。如果您建立的附加元件有 `aws-marketplace` 列在上一步的 `Owner` 欄位中，則建立操作可能會失敗，而且您可能會收到類似下列錯誤的錯誤訊息。

```
{
    "addon": {
        "addonName": "addon-name",
        "clusterName": "my-cluster",
```

```
        "status": "CREATE_FAILED",
        "addonVersion": "version",
        "health": {
            "issues": [
                {
                    "code": "AddonSubscriptionNeeded",
                    "message": "You are currently not subscribed to this add-on. To
 subscribe, visit the AWS Marketplace console, agree to the seller EULA, select the
 pricing type if required, then re-install the add-on"
                }
            ]
        }
    }
}
```

如果您收到類似上一個輸出中之錯誤的錯誤，請造訪上一步輸出中的 URL，以訂閱附加元件。訂閱後，再次執行 `create-addon` 命令。

更新 Amazon EKS 附加元件

當新版本發布或您將叢集更新到新的 Kubernetes 次要版本之後，Amazon EKS 不會自動更新附加元件。若要更新現有叢集的附加元件，您必須啟動更新。在您啟動更新後，Amazon EKS 會為您更新附加元件。在更新附加元件之前，請檢閱附加元件目前的文件。如需可用附加元件的清單，請參閱 the section called “ AWS 附加元件”。如果附加元件需要 IAM 角色，則請參閱 AWS中可用的 Amazon EKS 附加元件中的特定附加元件相關詳細資訊，了解有關建立角色的詳細資訊。

先決條件

在更新附加元件之前，請先完成下列動作：

- 檢查附加元件是否需要 IAM 角色。如需詳細資訊，請參閱the section called “Amazon EKS 附加元件”。
- 確認 Amazon EKS 附加元件版本與叢集相容。如需詳細資訊，請參閱the section called “確認相容性”。

程序

您可以使用 `eksctl`、 AWS 管理主控台或 CLI 更新 Amazon EKS AWS 附加元件。

更新附加元件 (eksctl)

1. 判斷叢集上安裝的目前附加元件和附加元件版本。使用您叢集的名稱取代 *my-cluster*。

```
eksctl get addon --cluster my-cluster
```

   範例輸出如下。

```
NAME        VERSION              STATUS  ISSUES  IAMROLE  UPDATE AVAILABLE
coredns     v1.8.7-eksbuild.2    ACTIVE  0
kube-proxy  v1.23.7-eksbuild.1   ACTIVE  0                v1.23.8-eksbuild.2
vpc-cni     v1.10.4-eksbuild.1   ACTIVE  0                v1.12.0-eksbuild.1,v1.11.4-
eksbuild.1,v1.11.3-eksbuild.1,v1.11.2-eksbuild.1,v1.11.0-eksbuild.1
```

   您的輸出可能看起來有所不同，具體取決於您的叢集上有哪些附加元件和版本。您可以看到，在前面的範例輸出中，叢集上的兩個現有附加元件在 UPDATE AVAILABLE 欄位中具有較新的版本。

2. 更新附加元件。

   a. 將隨後的命令複製到您的裝置。視需要對命令進行下列修改：

   - 使用您叢集的名稱取代 *my-cluster*。
   - 將 *region-code* 取代為您的叢集所在的 AWS 區域。
   - 使用您想要更新的上一步輸出中傳回的附加元件名稱取代 *vpc-cni*。
   - 如果您想要更新為比最新可用版本更早的版本，請使用您要使用的上一步輸出中傳回的版本編號取代 *latest*。某些附加元件有建議的版本。如需詳細資訊，請參閱您所更新之附加元件的文件。若要獲取附加元件清單，請參閱 the section called “ AWS 附加元件”。* 若附加元件使用 Kubernetes 服務帳戶和 IAM 角色，請使用您的帳戶 ID 取代 *111122223333*，並使用您已建立的現有 IAM 角色的名稱取代 *role-name*。如需建立角色的說明，請參閱您所建立之附加元件的文件。若要獲取附加元件清單，請參閱 the section called “ AWS 附加元件”。若要指定服務帳戶角色，您的叢集需要具有 IAM OpenID Connect (OIDC) 提供者。若要判定您的叢集是否已經擁有一個提供者，或是要建立一個提供者，請參閱 the section called “IAM OIDC 提供商”。

     如果附加元件未使用 Kubernetes 服務帳戶和 IAM 角色，請刪除 `serviceAccountRoleARN: arn:aws: iam::`*111122223333*`:role/`*role-name* 列。
   - *preserve* 選項會保留附加元件的現有值。如果您已為附加元件設定設定了自訂值，但未使用此選項，Amazon EKS 會以其預設值覆寫您的值。如果您使用此選項，建議您在更新生產叢集上的附加元件之前，測試非生產叢集上的任何欄位和值變更。如果您將此值變更為

`overwrite`，則所有設定都會變更為 Amazon EKS 預設值。如果您已設定任何設定的自訂值，則可能會使用 Amazon EKS 預設值覆寫這些值。如果您將此值變更為 `none`，Amazon EKS 不會變更任何設定的值，但更新可能會失敗。若更新失敗，您會收到錯誤訊息，以協助您解決衝突。

```
cat >update-addon.yaml <<EOF
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: my-cluster
  region: region-code

addons:
- name: vpc-cni
  version: latest
  serviceAccountRoleARN: arn:aws: iam::111122223333:role/role-name
  resolveConflicts: preserve
EOF
```

b. 執行修改後的命令來建立 `update-addon.yaml` 檔案。

c. 將組態檔案套用至叢集。

```
eksctl update addon -f update-addon.yaml
```

如需有關更新附加元件的詳細資訊，請參閱 `eksctl` 文件中的 更新附加元件。

更新附加元件AWS （主控台）

1. 開啟 Amazon EKS 主控台。
2. 在左側導覽窗格中選擇叢集。
3. 選擇您想要更新附加元件的叢集名稱。
4. 選擇附加元件索引標籤。
5. 選擇您想要更新的附加元件。
6. 選擇編輯。
7. 在設定######頁面上，執行以下操作：

a. 選擇您要使用的版本。附加元件可能有建議的版本。如需詳細資訊，請參閱您所更新之附加元件的文件。若要獲取附加元件清單，請參閱 the section called “ AWS 附加元件”。

b. 您可使用兩個用於設定附加元件角色的選項：EKS Pod 身分識別 IAM 角色及服務帳戶 (IRSA) 的 IAM 角色。根據您喜好的選項，遵循下列適當步驟。如果您選取的所有附加元件在狀態下都有需要訂閱，請選擇下一步。針對狀態下沒有需要訂閱的附加元件，請執行以下操作：

i. 若是服務帳戶的 Pod 身分識別 IAM 角色，您可使用現有的 EKS Pod 身分識別 IAM 角色，或是使用建立建議的角色按鈕來建立一個。此欄位僅提供包含適當信任政策的選項。若沒有可選取的角色，則表示您沒有包含相符信任政策的現有角色。如需針對所選附加元件的服務帳戶設定 EKS Pod 身分識別 IAM 角色，則選擇建立建議的角色。角色建立精靈會在單獨的視窗中開啟。精靈會自動填入如下所示角色資訊。針對您想要建立 EKS Pod 身分識別 IAM 角色的每個附加元件，完成如下所示 IAM 精靈中的步驟。

- 在選取信任實體步驟中，EKS AWS 的服務選項和 EKS - Pod Identity 的使用案例會預先選取，且會自動填入附加元件的適當信任政策。例如，將會如 the section called “EKS Pod 身分識別的優勢” 中詳述，透過包含 pods.eks.amazonaws.com IAM Principal 的適當信任政策來建立角色。選擇下一步。
- 在新增許可步驟中，將會針對附加元件預先選取適用於該角色政策的適當受管政策。舉例來說，針對 Amazon VPC CNI 附加元件，將會透過如 the section called “Kubernetes 專用 Amazon VPC CNI 外掛程式” 中詳述的受管政策 `AmazonEKS_CNI_Policy` 來建立角色。選擇下一步。
- 在命名、檢閱及建立步驟中，角色名稱會自動填入附加元件的預設角色名稱。舉例來說，針對 Amazon VPC CNI 附加元件，將會使用名稱 AmazonEKSPodIdentityAmazonVPCCNIRole 來建立角色。在描述中，預設描述會自動填入附加元件的適當描述。例如，對於 Amazon VPC CNI 附加元件，將使用描述建立角色 允許在 Amazon EKS 叢集中執行的 Pod 存取 AWS 資源。在信任政策中，檢視針對附加元件填入的信任政策。選擇建立角色。

> **Note**
>
> 保留預設角色名稱，能夠讓 EKS 在新叢集中或新增附加元件至現有叢集時，預先選取附加元件的角色。您仍可覆寫此名稱，並且可在所有叢集的附加元件使用該角色，但需要從下拉式選單中手動選取該角色。

ii. 若要了解狀態項下沒有需要訂閱選項的附加元件，以及您想要使用 IRSA 設定角色的位置，請參閱您要建立的附加元件相關文件，以便建立 IAM 政策並與角色連接。若要獲取附加元件清單，請參閱 the section called “ AWS 附加元件”。若要選取 IAM 角色，您的叢集需要具有 IAM OpenID Connect (OIDC) 提供者。若要判定您的叢集是否已經擁有一個提供者，或是要建立一個提供者，請參閱 the section called “IAM OIDC 提供商”。

c. 展開選用組態設定。

d. 在組態值中，輸入任意附加元件特定組態資訊。如需詳細資訊，請參閱您所更新之附加元件的文件。若要獲取附加程式清單，請參閱 the section called “ AWS 附加元件”...針對衝突解決方法，請選取其中一個選項。如果您已設定附加元件設定的自訂值，我們建議您使用 Preserve (保留) 選項。如果您不選擇此選項，Amazon EKS 會以其預設值覆寫您的值。如果您使用此選項，建議您在更新生產叢集上的附加元件之前，測試非生產叢集上的任何欄位和值變更。如果您將此值變更為「覆寫」，則所有設定都會變更為 Amazon EKS 預設值。如果您已設定任何設定的自訂值，則可能會使用 Amazon EKS 預設值覆寫這些值。如果您將此值變更為「無」，Amazon EKS 不會變更任何設定的值，但更新可能會失敗。若更新失敗，您會收到錯誤訊息，以協助您解決衝突。

8. 選擇儲存變更。

更新附加元件 (AWS CLI)

1. 您需要在裝置或 AWS CloudShell 上安裝和設定版本 `2.12.3`或更新版本或版本 `1.27.160`或更新版本的 AWS 命令列界面 (AWS CLI)。若要檢查您目前的版本，請使用 `aws --version | cut -d / -f2 | cut -d ' ' -f1`。適用於 macOS 的 `yum`、 `apt-get`或 Homebrew 等套件管理員通常是最新版本 CLI AWS 後面的幾個版本。若要安裝最新版本，請參閱《 AWS 命令列界面使用者指南》中的使用 aws 設定安裝 和快速組態。 https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-quickstart.html#cli-configure-quickstart-config在 AWS CloudShell 中安裝的 AWS CLI 版本也可能是最新版本後面的幾個版本。若要更新它，請參閱《CloudShell AWS 使用者指南》中的將 CLI 安裝到您的主目錄。 AWS CloudShell
2. 請參閱已安裝附加元件的清單。使用您叢集的名稱取代 *my-cluster*。

```
aws eks list-addons --cluster-name my-cluster
```

範例輸出如下。

```
{
    "addons": [
        "coredns",
        "kube-proxy",
        "vpc-cni"
    ]
}
```

3. 檢視您想要更新之附加元件的目前版本。請使用叢集名稱取代 *my-cluster* 並使用您想要更新之附加元件的名稱取代 *vpc-cni*。

```
aws eks describe-addon --cluster-name my-cluster --addon-name vpc-cni --query
 "addon.addonVersion" --output text
```

範例輸出如下。

```
v1.10.4-eksbuild.1
```

4. 判斷哪些版本的附加元件可用於您的叢集版本。將 *1.36* 取代為您的叢集版本，並將 *vpc-cni* 取代為您要更新的附加元件名稱。

```
aws eks describe-addon-versions --kubernetes-version 1.36 --addon-name vpc-cni \
    --query 'addons[].addonVersions[].{Version: addonVersion, Defaultversion:
 compatibilities[0].defaultVersion}' --output table
```

範例輸出如下。

```
-----------------------------------------
|          DescribeAddonVersions         |
+-----------------+----------------------+
| Defaultversion  |       Version        |
+-----------------+----------------------+
|  False          |  v1.12.0-eksbuild.1  |
|  True           |  v1.11.4-eksbuild.1  |
|  False          |  v1.10.4-eksbuild.1  |
|  False          |  v1.9.3-eksbuild.1   |
+-----------------+----------------------+
```

`Defaultversion` 欄位中具有 `True` 的版本在預設情況下是用來建立附加元件的版本。

5. 更新您的附加元件。將隨後的命令複製到您的裝置。視需要對命令進行下列修改，然後執行修改後的命令。若要了解此命令的相關詳細資訊，請參閱《Amazon EKS 命令列參考》中的 update-addon。
   - 使用您叢集的名稱取代 *my-cluster*。
   - 使用您想要更新之附加元件的名稱取代 *vpc-cni*，該名稱在上一步輸出中傳回。
   - 使用您想要更新之上一步輸出中傳回的版本取代 *version-number*。某些附加元件有建議的版本。如需詳細資訊，請參閱您所更新之附加元件的文件。若要獲取附加元件清單，請參閱 the section called “ AWS 附加元件”。* 若附加元件使用 Kubernetes 服務帳戶和 IAM 角色，請使用您的帳戶 ID 取代 *111122223333*，並使用您已建立的現有 IAM 角色的名稱取代 *role-name*。

如需建立角色的說明，請參閱您所建立之附加元件的文件。若要獲取附加元件清單，請參閱 the section called “ AWS 附加元件”。若要指定服務帳戶角色，您的叢集需要具有 IAM OpenID Connect (OIDC) 提供者。若要判定您的叢集是否已經擁有一個提供者，或是要建立一個提供者，請參閱 the section called “IAM OIDC 提供商”。

如果附加元件未使用 Kubernetes 服務帳戶和 IAM 角色，請刪除 `serviceAccountRoleARN: arn:aws: iam::111122223333:role/role-name` 列。

- `--resolve-conflicts PRESERVE` 選項會保留附加元件的現有值。如果您已為附加元件設定設定了自訂值，但未使用此選項，Amazon EKS 會以其預設值覆寫您的值。如果您使用此選項，建議您在更新生產叢集上的附加元件之前，測試非生產叢集上的任何欄位和值變更。如果您將此值變更為 OVERWRITE，則所有設定都會變更為 Amazon EKS 預設值。如果您已設定任何設定的自訂值，則可能會使用 Amazon EKS 預設值覆寫這些值。如果您將此值變更為 NONE，Amazon EKS 不會變更任何設定的值，但更新可能會失敗。若更新失敗，您會收到錯誤訊息，以協助您解決衝突。
- 如果您要移除所有自訂組態，請使用 *`--configuration-values '{}'`* 選項執行更新。這會將所有自訂組態設定回預設值。如果您不想變更自訂組態，請不要提供 *`--configuration-values`* 標記。如果您要調整自訂組態，請使用新參數取代 *`{}`*。

```
aws eks update-addon --cluster-name my-cluster --addon-name vpc-cni --addon-version
 version-number \
    --service-account-role-arn arn:aws: iam::111122223333:role/role-name --
configuration-values '{}' --resolve-conflicts PRESERVE
```

6. 查看更新狀態。請使用叢集名稱取代 *`my-cluster`* 並使用您所更新之附加元件的名稱取代 *`vpc-cni`*。

```
aws eks describe-addon --cluster-name my-cluster --addon-name vpc-cni
```

範例輸出如下。

```
{
    "addon": {
        "addonName": "vpc-cni",
        "clusterName": "my-cluster",
        "status": "UPDATING",
    }
}
```

當狀態為 ACTIVE 時，即表示更新已完成。

確認 Amazon EKS 附加元件版本與叢集的相容性

在建立 Amazon EKS 附加元件之前，需要確認 Amazon EKS 附加元件版本是否與您的叢集相容。

使用 describe-addon-versions API 來列出可用的 EKS 附加元件版本，以及每個附加元件版本所支援的 Kubernetes 版本。

1. 確認已安裝 AWS CLI 並使用 `aws sts get-caller-identity`。如果此命令無法運作，請了解如何開始使用 AWS CLI。
2. 確定您要擷取版本相容性資訊的附加元件名稱，例如 `amazon-cloudwatch-observability`。
3. 確定叢集的 Kubernetes 版本，例如 `1.36`。
4. 使用 AWS CLI 擷取與叢集的 Kubernetes 版本相容的附加元件版本。

```
aws eks describe-addon-versions --addon-name amazon-cloudwatch-observability --
kubernetes-version 1.36
```

範例輸出如下。

```
{
    "addons": [
        {
            "addonName": "amazon-cloudwatch-observability",
            "type": "observability",
            "addonVersions": [
                {
                    "addonVersion": "vX.X.X-eksbuild.X",
                    "architecture": [
                        "amd64",
                        "arm64"
                    ],
                    "computeTypes": [
                        "ec2",
                        "auto",
                        "hybrid"
                    ],
                    "compatibilities": [
                        {
```

```
                            "clusterVersion": "1.36",
                            "platformVersions": [
                                "*"
                            ],
                            "defaultVersion": true
                        }
                    ],
                }
            ]
        }
    ]
}
```

此輸出顯示附加元件版本 `vX.X.X-eksbuild.X` 與 Kubernetes 叢集版本 `1.36` 相容。

附加元件與運算類型的相容性

`describe-addon-versions` 輸出中 `computeTypes` 欄位表示附加元件與 EKS 自動模式受管節點或混合節點的相容性。標記的附加元件`auto`適用於 EKS Auto Mode 的雲端型 AWS受管基礎設施，而標記的附加元件`hybrid`可在連接到 EKS 雲端控制平面的內部部署節點上執行。

如需詳細資訊，請參閱the section called “Amazon EKS 自動模式的考量事項”。

從叢集移除 Amazon EKS 附加元件

您可以使用 `eksctl`、 AWS 管理主控台或 CLI 從叢集中移除 Amazon EKS AWS 附加元件。

若要從叢集移除 Amazon EKS 附加元件：

- 此附加元件提供的功能沒有任何停機時間。
- 使用服務帳戶 (IRSA) 的 IAM 角色，並且附加元件具有與其關聯的 IAM 角色時，IAM 角色不會移除。
- 若是使用 Pod 身分識別，附加元件擁有的任何 Pod 身分識別關聯皆會移除。如果您指定 CLI AWS `--preserve`的選項，則會保留關聯。
- Amazon EKS 停止管理附加元件的設定。
- 當有新版本可用時，主控台會停止通知您。
- 您無法使用任何 AWS 工具或 APIs更新附加元件。

- 您可以選擇將附加元件軟體保留在叢集上，以便您可以自我管理附加元件軟體，或從叢集中移除附加元件軟體。只有在叢集上沒有資源取決於附加元件提供的功能時，才應該從叢集中移除附加元件軟體。

先決條件

移除附加元件之前，請先完成下列動作：

- 現有 Amazon EKS 叢集。若要部署叢集，請參閱 開始使用。
- 檢查附加元件是否需要 IAM 角色。如需詳細資訊，請參閱the section called “擷取 IAM 資訊”。
- 裝置或 AWS CloudShell 上安裝的`eksctl`命令列工具版本 `0.215.0`或更新版本。如需有關安裝或更新 `eksctl` 的指示，請參閱 `eksctl` 文件中的安裝一節。

程序

移除 Amazon EKS 附加元件時，您有兩個選項：

- Preserve add-on software on your cluster (在叢集上保留附加元件軟體)：此選項會移除任何設定的 Amazon EKS 管理。其也會移除 Amazon EKS 通知您更新的功能，並在您啟動更新後自動更新 Amazon EKS 附加元件。不過，該選項會保留您叢集上的附加元件軟體。此選項會使附加元件成為自我管理安裝，而不是 Amazon EKS 附加元件。如果啟用此選項，附加元件就無須停機。
- Remove add-on software entirely from your cluster (從叢集中完全移除附加元件軟體)：只有叢集上沒有資源依賴於附加元件提供的功能時，我們才會建議您將 Amazon EKS 附加元件從叢集中移除。

您可以使用 `eksctl`、 AWS 管理主控台或 CLI 移除 Amazon EKS AWS 附加元件。

移除附加元件 (eksctl)

1. 判斷叢集上目前安裝的附加元件。使用您叢集的名稱取代 *`my-cluster`*。

```
eksctl get addon --cluster my-cluster
```

範例輸出如下。

```
NAME        VERSION              STATUS  ISSUES  IAMROLE  UPDATE AVAILABLE
coredns     v1.8.7-eksbuild.2    ACTIVE  0
kube-proxy  v1.23.7-eksbuild.1   ACTIVE  0
vpc-cni     v1.10.4-eksbuild.1   ACTIVE  0
```

```
[...]
```

您的輸出可能看起來有所不同，具體取決於您的叢集上有哪些附加元件和版本。

2. 移除附加元件。使用叢集名稱取代 *my-cluster*，並使用您想要移除之上一步輸出中傳回的附加元件名稱取代 *name-of-add-on*。若移除 *--preserve* 選項，除了 Amazon EKS 不再管理附加元件之外，附加元件軟體也會從叢集中刪除。

```
eksctl delete addon --cluster my-cluster --name name-of-addon --preserve
```

若要了解移除附加元件的相關詳細資訊，請參閱 eksctl 文件中的刪除附加元件。

移除附加元件AWS （ 主控台 ）

1. 開啟 Amazon EKS 主控台。
2. 在左側導覽窗格中選擇叢集。
3. 選擇要移除 Amazon EKS 附加元件的叢集名稱。
4. 選擇附加元件索引標籤。
5. 選擇要移除的附加元件。
6. 選擇移除。
7. 在移除：******確認對話方塊中，執行以下操作：
   a. 若希望 Amazon EKS 停止管理附加元件的設定，請選取在叢集上保留。若要在叢集上保留附加元件軟體，請執行此動作。如此一來，您就可以自行管理附加元件的所有設定。
   b. 輸入附加元件的名稱。
   c. 選擇移除。

移除附加元件 (AWS CLI)

1. 您需要在裝置0.215.0或 AWS CloudShell 上安裝版本 或更新版本的eksctl命令列工具。如需有關安裝或更新 eksctl 的指示，請參閱 eksctl 文件中的安裝一節。
2. 請參閱已安裝附加元件的清單。使用您叢集的名稱取代 *my-cluster*。

```
aws eks list-addons --cluster-name my-cluster
```

範例輸出如下。

```
{
    "addons": [
        "coredns",
        "kube-proxy",
        "vpc-cni",
        "name-of-addon"
    ]
}
```

3. 移除已安裝的附加元件。使用叢集名稱取代 *my-cluster*，並使用您想要移除的附加元件名稱取代 *name-of-add-on*。移除 *--preserve* 會將附加元件軟體從叢集中刪除。

```
aws eks delete-addon --cluster-name my-cluster --addon-name name-of-addon --preserve
```

縮寫的範例輸出如下所示。

```
{
    "addon": {
        "addonName": "name-of-add-on",
        "clusterName": "my-cluster",
        "status": "DELETING",
    }
}
```

4. 檢查移除狀態。使用叢集名稱取代 *my-cluster*，並使用您將移除的附加元件名稱取代 *name-of-addon*。

```
aws eks describe-addon --cluster-name my-cluster --addon-name name-of-addon
```

移除附加元件之後，範例輸出如下所示。

```
An error occurred (ResourceNotFoundException) when calling the DescribeAddon
 operation: No addon: name-of-addon found in cluster: my-cluster
```

Amazon EKS 附加元件的 IAM 角色

某些 Amazon EKS 附加元件需要 IAM 角色和許可才能呼叫 AWS APIs。例如，Amazon VPC CNI 附加元件會呼叫特定 AWS APIs來設定帳戶中的網路資源。這些附加元件需要使用 IAM 來授予許可。更具體地說，需要將執行附加元件的 Pod 的服務帳戶，與具有特定 IAM 政策的 IAM 角色建立關聯。

授予叢集工作負載 AWS 許可的建議方法是使用 Amazon EKS 功能 Pod 身分。您可使用 Pod 身分識別關聯，將附加元件的服務帳戶映射到一個 IAM 角色。如果 Pod 使用具有關聯的服務帳戶，則 Amazon EKS 會在 Pod 的容器中設定環境變數。環境變數設定 AWS SDKs，包括 AWS CLI，以使用 EKS Pod Identity 憑證。如需詳細資訊，請參閱the section called “Pod 身分識別”

Amazon EKS 附加元件有助於管理與該附加元件相對應的 Pod 身分識別關聯的生命週期。例如，您可在單個 API 呼叫中建立或更新一個 Amazon EKS 附加元件及其必要的 Pod 身分識別關聯。Amazon EKS 還提供了用於擷取建議的 IAM 政策的 API。

1. 確認您的叢集上已設定 Amazon EKS Pod 身分代理程式。
2. 判斷您要安裝的附加元件是否需要使用 AWS CLI 操作的 IAM `describe-addon-versions` 許可。如果 `requiresIamPermissions` 標誌為 `true`，則您應使用 `describe-addon-configurations` 操作來確定附加元件所需的許可。回應中會包含建議的受管 IAM 政策清單。
3. 使用 `describe-addon-configuration` CLI 操作擷取 Kubernetes 服務帳戶的名稱和 IAM 政策。根據安全需求評估建議政策的範圍。
4. 使用建議的許可政策及 Pod 身分識別所需的信任政策，建立 IAM 角色。如需詳細資訊，請參閱the section called “建立 Pod Identity 關聯AWS （主控台）”。
5. 使用 CLI 建立或更新 Amazon EKS 附加元件。至少指定一個 Pod 身分識別關聯。Pod 身分識別關聯包含 Kubernetes 服務帳戶的名稱和 IAM 角色的 ARN。
   - 使用附加元件 API 建立的 Pod 身分識別關聯由相應的附加元件所擁有。如果您刪除該附加元件，Pod 身分識別關聯也會一併被刪除。使用 CLI 或 API 刪除附加元件時，您可以使用 AWS `preserve`選項來防止此串聯刪除。必要時，您也可直接更新或刪除 Pod 身分識別關聯。附加元件無法接管現有 Pod 身分識別關聯的所有權。您必須刪除現有的關聯，並使用附加元件的建立或更新操作重新建立。
   - Amazon EKS 推薦使用 Pod 身分識別關聯來管理附加元件的 IAM 許可。先前的方法，即服務帳戶的 IAM 角色 (IRSA)，仍然受支援。您可以同時為一個附加元件指定 IRSA `serviceAccountRoleArn` 和 Pod 身分識別關聯。如果叢集上安裝了 EKS Pod 身分識別代理程式，則 `serviceAccountRoleArn` 將被忽略，EKS 將使用提供的 Pod 身分識別關聯。如果未啟用 Pod 身分識別，則將使用 `serviceAccountRoleArn`。

- 如果您更新現有附加元件的 Pod 身分識別關聯，Amazon EKS 會啟動附加元件 Pod 的滾動重啟。

擷取 Amazon EKS 附加元件的相關 IAM 資訊

建立附加元件之前，請使用 AWS CLI 判斷：

- 附加元件是否需要 IAM 許可
- 待使用的建議 IAM 政策

程序

1. 確定您想要安裝的附加元件名稱，以及您的叢集的 Kubernetes 版本。如需附加元件的詳細資訊，請參閱 the section called “Amazon EKS 附加元件”。
2. 使用 AWS CLI 判斷附加元件是否需要 IAM 許可。

```
aws eks describe-addon-versions \
--addon-name <addon-name> \
--kubernetes-version <kubernetes-version>
```

例如：

```
aws eks describe-addon-versions \
--addon-name aws-ebs-csi-driver \
--kubernetes-version 1.30
```

檢閱以下輸出範例。請注意，`requiresIamPermissions` 是 `true`，以及預設的附加元件版本。若擷取建議的 IAM 政策，您需指定附加元件版本。

```
{
    "addons": [
        {
            "addonName": "aws-ebs-csi-driver",
            "type": "storage",
            "addonVersions": [
                {
                    "addonVersion": "v1.31.0-eksbuild.1",
                    "architecture": [
                        "amd64",
```

```
                        "arm64"
                    ],
                    "compatibilities": [
                        {
                            "clusterVersion": "1.30",
                            "platformVersions": [
                                "*"
                            ],
                            "defaultVersion": true
                        }
                    ],
                    "requiresConfiguration": false,
                    "requiresIamPermissions": true
                },
[...]
```

3. 如果附加元件需要 IAM 許可，請使用 AWS CLI 擷取建議的 IAM 政策。

```
aws eks describe-addon-configuration \
--query podIdentityConfiguration \
--addon-name <addon-name> \
--addon-version <addon-version>
```

例如：

```
aws eks describe-addon-configuration \
--query podIdentityConfiguration \
--addon-name aws-ebs-csi-driver \
--addon-version v1.31.0-eksbuild.1
```

檢閱以下輸出。請記下 recommendedManagedPolicies。

```
[
    {
        "serviceAccount": "ebs-csi-controller-sa",
        "recommendedManagedPolicies": [
            "arn:aws: iam::aws:policy/service-role/AmazonEBSCSIDriverPolicyV2"
        ]
    }
]
```

4. 建立 IAM 角色，以及連接建議的受管政策。或者，檢閱受管政策，以及在適用時縮減許可範圍。如需更多資訊，請參閱the section called “建立 Pod Identity 關聯AWS （主控台）”。

Pod 身分識別支援參考

下面的資料表列出了某些 Amazon EKS 附加元件是否支援 EKS Pod 身分識別。

附加元件名稱	Pod 身分識別支援	要求的最低版本
Amazon EBS CSI 驅動程式	是	v1.26.0-eksbuild.1
Amazon VPC CNI	是	v1.15.5-eksbuild.1
Amazon EFS CSI 驅動程式	是	v2.0.5-eksbuild.1
AWS Distro for OpenTelemetry	是	v0.94.1-eksbuild.1
適用於 Amazon S3 的掛載點 CSI 驅動程式	否	N/A
Amazon CloudWatch 可觀測性代理程式	是	v3.1.0-eksbuild.1

這份資料表的最近更新日期為 2024 年 10 月 28 日。

使用 Pod 身分識別指派 IAM 角色到 Amazon EKS 附加元件

某些 Amazon EKS 附加元件需要 IAM 角色及許可。在您更新 Amazon EKS 附加元件以使用 Pod Identity 關聯之前，請確認要使用的角色和政策。如需詳細資訊，請參閱the section called “擷取 IAM 資訊”。

1. 確定：
   - `cluster-name` – 待安裝附加元件的叢集名稱。
   - `addon-name` – 待安裝附加元件名稱。
   - `service-account-name` – 附加元件使用的 Kubernetes 服務帳戶名稱。

- `iam-role-arn` – 具有充足附加元件許可的 IAM 角色的 ARN。角色必須具有 EKS Pod 身分識別所需的信任政策。如需更多資訊，請參閱the section called “建立 Pod Identity 關聯AWS （主控台）”。

2. 使用 CLI AWS 更新附加元件。您亦可在建立附加元件時，使用相同的 `--pod-identity-associations` 語法來指定 Pod 身分識別關聯。請注意，若在更新附加元件時指定 Pod 身分識別關聯，所有之前的 Pod 身分識別關聯均會被覆寫。

```
aws eks update-addon --cluster-name <cluster-name> \
--addon-name <addon-name> \
--pod-identity-associations '[{"serviceAccount": "<service-account-name>", "roleArn":
 "<role-arn>"}]'
```

例如：

```
aws eks update-addon --cluster-name mycluster \
--addon-name aws-ebs-csi-driver \
--pod-identity-associations '[{"serviceAccount": "ebs-csi-controller-sa", "roleArn":
 "arn:aws: iam::123456789012:role/StorageDriver"}]'
```

3. 請驗證是否已建立 Pod 身分識別關聯：

```
aws eks list-pod-identity-associations --cluster-name <cluster-name>
```

如果執行成功，您應該會看到類似下列的輸出。注意 EKS 附加元件的 OwnerARN。

```
{
    "associations": [
        {
            "clusterName": "mycluster",
            "namespace": "kube-system",
            "serviceAccount": "ebs-csi-controller-sa",
            "associationArn": "arn:aws: eks:us-
west-2:123456789012:podidentityassociation/mycluster/a-4wvljrezsukshq1bv",
            "associationId": "a-4wvljrezsukshq1bv",
            "ownerArn": "arn:aws: eks:us-west-2:123456789012:addon/mycluster/aws-ebs-
csi-driver/9cc7ce8c-2e15-b0a7-f311-426691cd8546"
        }
    ]
}
```

從 Amazon EKS 附加元件移除 Pod 身分識別關聯

從 Amazon EKS 附加元件移除 Pod 身分識別關聯。

1. 確定：
   - `cluster-name` - 待安裝附加元件的 EKS 叢集名稱。
   - `addon-name` - 待安裝 Amazon EKS 附加元件名稱。
2. 更新附加元件，以指定空的 Pod 身分識別關聯陣列。

```
aws eks update-addon --cluster-name <cluster-name> \
--addon-name <addon-name> \
--pod-identity-associations "[]"
```

針對 EKS 附加元件的 Pod 身分識別進行疑難排解

如果您的附加元件在嘗試 AWS API、 SDK 或 CLI 操作時遇到錯誤，請確認下列事項：

- Pod 身分識別代理程式已安裝到您的叢集中。
  - 有關如何安裝 Pod 身分識別代理程式的資訊，請參閱 the section called “設定代理程式”。
- 該附加元件具有有效的 Pod 身分識別關聯。
  - 使用 AWS CLI 擷取附加元件使用之服務帳戶名稱的關聯。

```
aws eks list-pod-identity-associations --cluster-name <cluster-name>
```

- IAM 角色具有 Pod 身分識別所需的信任政策。
  - 使用 AWS CLI 擷取 附加元件的信任政策。

```
aws iam get-role --role-name <role-name> --query Role.AssumeRolePolicyDocument
```

- IAM 角色具有該附加元件所需的必要許可。
  - 使用 AWS CloudTrail 來檢閱 `AccessDenied`或 `UnauthorizedOperation`事件。
- Pod 身分識別關聯中的服務帳戶名稱與附加元件所使用的服務帳戶名稱相符。
  - 有關可用附加元件的資訊，請參閱 the section called “ AWS 附加元件”。
- 檢查名為 `pod-identity-webhook` 的 MutatingWebhookConfiguration 組態
  - Webhook 的 `admissionReviewVersions` 需要是 `v1beta1`，並且與 `v1` 不相容。

確定您可為 Amazon EKS 附加元件自訂的欄位

使用標準的最佳實務組態將 Amazon EKS 附加元件安裝到您的叢集。如需將 Amazon EKS 附加元件新增至叢集的詳細資訊，請參閱 the section called “Amazon EKS 附加元件”。

您可能想要自訂 Amazon EKS 附加元件的組態，以啟用進階功能。Amazon EKS 使用 Kubernetes 伺服器端套用功能來啟用 Amazon EKS 管理附加元件，而不會覆寫非由 Amazon EKS 管理的設定組態。如需詳細資訊，請參閱 Kubernetes 文件中的伺服器端套用。為了達到此目的，Amazon EKS 會為其安裝的每個附加元件管理一組數量最少的欄位。您可以修改所有非由 Amazon EKS 管理的欄位，或另一個 Kubernetes 控制平面程序 (例如 `kube-controller-manager`)，沒有任何問題。

> **Important**
>
> 修改 Amazon EKS 管理的欄位會阻止 Amazon EKS 管理附加元件，並且可能會導致在更新附加元件時覆寫您的變更。

欄位管理語法

檢視 Kubernetes 物件的詳細資訊時，輸出中會傳回受管和未受管的欄位。受管欄位可以是以下類型之一：

- 全受管：欄位的所有索引鍵均由 Amazon EKS 管理。修改任何值皆會導致衝突。
- 部分受管：欄位的某些索引鍵由 Amazon EKS 管理。只有對明確由 Amazon EKS 管理的索引鍵進行修改才會造成衝突。

這兩種類型的欄位均使用 `manager: eks` 標記。

每個索引鍵可以是 `.` 表示欄位本身，其一律會映射到一個空集，或映射至表示子欄位或項目的字串。欄位管理的輸出包含下列類型的宣告：

- `f:`*`name`*，其中 *`name`* (名稱) 是清單中欄位的名稱。
- `k:`*`keys`*，其中 *`keys`* (金鑰) 是清單項目欄位的映射。
- `v:`*`value`*，其中 *`value`* (值) 是清單項目的精準 JSON 格式化值。
- `i:`*`index`*，其中*##*是清單中項目的位置。

CoreDNS 附加元件的下列輸出部分會說明先前的宣告：

- 全受管欄位：如果受管欄位具有指定的 f: (欄位)，但沒有 k: (索引鍵)，則會管理整個欄位。修改此欄位中的任何值皆會導致衝突。

  在下列輸出中，您可以看到名為 coredns 的容器由 eks 管理。args、image 及 imagePullPolicy 子欄位也由 eks 管理。修改這些欄位中的任何值皆會導致衝突。

```
[...]
f:containers:
  k:{"name":"coredns"}:
  .: {}
  f:args: {}
  f:image: {}
  f:imagePullPolicy: {}
[...]
manager: eks
[...]
```

- 部分受管欄位：如果受管索引鍵具有指定的值，則會針對該欄位管理宣告的索引鍵。修改指定的索引鍵會導致衝突。

  在下列輸出中，您可以看到 eks 管理 config-volume 和使用 name 索引鍵設定的 tmp 磁碟區。

```
[...]
f:volumes:
  k:{"name":"config-volume"}:
    .: {}
    f:configMap:
      f:items: {}
      f:name: {}
    f:name: {}
  k:{"name":"tmp"}:
    .: {}
    f:name: {}
[...]
manager: eks
[...]
```

- 將索引鍵新增至部分受管欄位：如果只管理特定鍵值，您可以安全地將其他索引鍵 (例如引數) 新增至欄位，而不會造成衝突。如果新增其他索引鍵，請確定未先對欄位進行管理。新增或修改受管的任何值會導致衝突。

在下列輸出中，您可以看到 name 索引鍵和 name 欄位皆為受管項目。新增或修改任何容器名稱會導致與此受管索引鍵發生衝突。

```
[...]
f:containers:
  k:{"name":"coredns"}:
[...]
    f:name: {}
[...]
manager: eks
[...]
```

程序

您可以使用 kubectl，查看哪些欄位由 Amazon EKS 管理，適用於任何 Amazon EKS 附加元件。

您可以修改所有非由 Amazon EKS 管理的欄位，或另一個 Kubernetes 控制平面程序 (例如 kube-controller-manager)，沒有任何問題。

1. 判定要檢查的附加元件。若要查看部署到叢集的所有 deployments 和 DaemonSets，請參閱 the section called “存取叢集資源”。
2. 執行下列命令以檢視附加元件的受管欄位：

```
kubectl get type/add-on-name -n add-on-namespace -o yaml
```

例如，您可以使用下列命令查看 CoreDNS 附加元件的受管欄位。

```
kubectl get deployment/coredns -n kube-system -o yaml
```

欄位管理會列在傳回輸出的下一區段中。

```
[...]
managedFields:
  - apiVersion: apps/v1
    fieldsType: FieldsV1
    fieldsV1:
[...]
```

> **Note**
>
> 如果您在輸出中看不到 `managedFields`，請新增 `--show-managed-fields` 至該命令並再次執行。您正在使用的 `kubectl` 版本會決定是否預設傳回受管欄位。

後續步驟

AWS 為您的附加元件自訂 未擁有的欄位。

在部署期間驗證容器映像簽章

如果您使用 AWS Signer 並想在部署時驗證已簽署的容器映像，您可以使用下列解決方案其中一種：

- Gatekeeper 與 Ratify – 使用 Gatekeeper 作為許可控制器，並使用 AWS Signer 外掛程式設定 Ratify 作為驗證簽章的 Webhook。
- Kyverno – 使用 AWS Signer 外掛程式設定以供驗證簽章之用的 Kubernetes 政策引擎。

> **Note**
>
> 在驗證容器映像簽章之前，請依照您選取的許可控制器的要求，設定 Notation 信任存放區和信任政策。

EKS 功能

> **Tip**
>
> 入門：建立 ACK 功能 | 建立 Argo CD 功能 | 建立 kro 功能

Amazon EKS 功能是一組分層的全受管叢集功能，可協助加速開發人員速度，並卸載使用 Kubernetes 建置和擴展的複雜性。EKS 功能是 Kubernetes 原生功能，用於宣告式持續部署、 AWS 資源管理和 Kubernetes 資源撰寫和協同運作，全部由 完全管理 AWS。透過 EKS 功能，您可以更專注於建置和擴展工作負載，並將這些基礎平台服務的操作負擔卸載至 AWS。這些功能會在 EKS 而非叢集中執行，無需在工作者節點上安裝、維護和擴展關鍵平台元件。

若要開始使用，您可以在新的或現有的 EKS 叢集上建立一或多個 EKS 功能。若要這樣做，您可以使用 AWS CLI、 AWS 管理主控台、EKS APIs、eksctl 或您偏好的infrastructure-as-code工具。雖然 EKS 功能旨在共同運作，但它們是獨立的雲端資源，您可以根據您的使用案例和需求來挑選和選擇。

EKS 功能支援 EKS 支援的所有 Kubernetes 版本。

> **Note**
>
> EKS 功能可在提供 Amazon EKS 的所有 AWS 商業區域中使用。如需支援的區域清單，請參閱《 AWS 一般參考》中的 Amazon EKS 端點和配額。

可用的功能

AWS Kubernetes (ACK) 的控制器

ACK 可讓您使用 Kubernetes APIs 管理 AWS 資源，讓您使用 Kubernetes 自訂 AWS 資源建立和管理 S3 儲存貯體、RDS 資料庫、IAM 角色和其他資源。ACK 會持續協調您所需的狀態與 中的實際狀態 AWS，並更正一段時間內的任何偏離，以保持您的系統運作狀態，並依指定設定您的資源。您可以使用相同的工具和工作流程來管理 AWS 資源與 Kubernetes 工作負載，並支援 50 多種 AWS 服務，包括 S3、RDS、DynamoDB 和 Lambda。ACK 支援跨帳戶和跨區域資源管理，啟用複雜的多帳戶、多叢集系統管理架構。ACK 支援唯讀資源和唯讀採用，有助於將其他基礎設施作為程式碼工具遷移到以 Kubernetes 為基礎的系統。

進一步了解 ACK →

Argo CD

Argo CD 使用 Git 儲存庫做為工作負載和系統狀態的事實來源，為您的應用程式實作 GitOps 型持續部署。Argo CD 會自動從 Git 儲存庫同步應用程式資源到您的叢集，偵測並修復偏離，以確保您部署的應用程式符合您所需的狀態。您可以從單一 Argo CD 執行個體跨多個叢集部署和管理應用程式，並在每次遞交變更時從 Git 儲存庫自動部署。同時使用 Argo CD 和 ACK 可提供基礎 GitOps 系統，簡化工作負載相依性管理，並支援整個系統設計，包括大規模的叢集和基礎設施管理。Argo CD 與 AWS Identity Center 整合以進行身分驗證和授權，並提供託管 Argo UI 以視覺化應用程式運作狀態和部署狀態。

進一步了解 Argo CD →

kro (Kube Resource Orchestrator)

kro 可讓您建立自訂 Kubernetes APIs，將多個資源組成更高階的抽象，讓平台團隊為常見的資源組合定義可重複使用的模式：雲端建置區塊。使用 kro，您可以將 Kubernetes AWS 和資源組合為統一抽象，使用簡單的語法來啟用動態組態和條件式邏輯。 kro 可讓平台團隊提供具有適當防護機制的自助式功能，讓開發人員使用簡單的專用 APIs 佈建複雜的基礎設施，同時維護組織標準和最佳實務。 kro 資源只是 Kubernetes 資源，並指定在 Kubernetes 資訊清單中，這些資訊清單可以存放在 Git 中，或推送至 Amazon ECR 等 OCI 相容登錄檔以進行廣泛的組織分佈。

進一步了解 kro →

EKS 功能的優點

EKS 功能完全由 管理 AWS，無需安裝、維護和擴展基礎叢集服務。 AWS 處理安全修補、更新和操作管理，讓您的團隊專注於使用 進行建置 AWS ，而不是叢集操作。與使用叢集資源的傳統 Kubernetes 附加元件不同，功能在 EKS 中執行，而不是在工作者節點上執行。這可為您的工作負載釋放叢集容量和資源，同時將管理叢集內控制器和其他平台元件的操作負擔降至最低。

透過 EKS 功能，您可以使用原生 Kubernetes APIs 和 等工具來管理部署、 AWS 資源、自訂 Kubernetes 資源和組合`kubectl`。所有功能都會在您的叢集內容中運作，自動偵測和修正應用程式和雲端基礎設施資源中的組態偏離。您可以從單一控制點跨多個叢集、 AWS 帳戶和區域部署和管理資源，簡化複雜、分散式環境中的操作。

EKS 功能專為 GitOps 工作流程而設計，提供宣告性、版本控制的基礎設施和應用程式管理。透過系統從 Git 變更流程，提供與現有開發實務整合的稽核追蹤、復原功能和協同合作工作流程。這種

Kubernetes 原生方法表示您不需要使用多個工具或管理叢集外部的infrastructure-as-code系統，而且有單一事實來源可供參考。您在版本控制的 Kubernetes 宣告式組態中定義的所需狀態會在整個環境中持續強制執行。

定價

使用 EKS 功能，無需預付承諾或最低費用。您需要為 Amazon EKS 叢集上每個作用中小時的每個功能資源付費。由 EKS 功能管理的特定 Kubernetes 資源也會以每小時費率計費。

如需目前定價資訊，請參閱 Amazon EKS 定價頁面。

> **Tip**
>
> 您可以使用 AWS Cost Explorer 和成本和用量報告，與其他 EKS 費用分開追蹤功能成本。您可以使用叢集名稱、功能類型和其他詳細資訊來標記您的功能，以用於成本分配目的。

EKS 功能的運作方式

每個功能都是您在 EKS 叢集上建立 AWS 的資源。建立後，此功能會在 EKS 中執行，並由 完全管理 AWS。

> **Note**
>
> 您可以為指定叢集建立每種類型的一個功能資源 (Argo CD、ACK 和 kro)。您無法在相同叢集上建立相同類型的多個功能資源。

您可以使用標準 Kubernetes APIs 和工具與叢集中的功能互動：

- 使用 `kubectl`套用 Kubernetes 自訂資源
- 使用 Git 儲存庫做為 GitOps 工作流程的事實來源

有些功能支援其他工具。例如：

- 使用 Argo CD CLI 來設定和管理 Argo CD 功能中的儲存庫和叢集
- 使用 Argo CD UI 視覺化和管理由 Argo CD 功能管理的應用程式

功能旨在共同運作，但獨立且完全選擇加入。您可以根據您的需求啟用一個、兩個或全部三個功能，並隨著需求發展更新您的組態。

所有 EKS 運算類型都支援與 EKS 功能搭配使用。如需詳細資訊，請參閱管理運算。

如需 IAM 角色的安全組態和詳細資訊，請參閱the section called “EKS 功能的考量事項”。如需多叢集架構模式，請參閱 the section called “考量事項”。

常用案例

適用於應用程式和基礎設施的 GitOps

使用 Argo CD 部署應用程式和操作元件，以及從 Git 儲存庫管理叢集組態和佈建基礎設施的 ACK。您的整個堆疊 - 應用程式、資料庫、儲存和聯網 - 定義為程式碼並自動部署。

範例：開發團隊將變更推送至 Git。Argo CD 部署更新的應用程式，而 ACK 會使用正確的組態佈建新的 RDS 資料庫。所有變更皆可稽核、可逆且跨環境保持一致。

具有自助式服務的平台工程

使用 kro 建立構成 ACK 和 Kubernetes 資源APIs。平台團隊使用護欄定義核准的模式。應用程式團隊使用簡單的高階 APIs 來佈建完整的堆疊。

範例：平台團隊會建立佈建部署、服務、輸入和 S3 儲存貯體的「WebApplication」 API。開發人員使用此 API，而不需要了解基礎複雜性或 AWS 許可。

多叢集應用程式管理

使用 Argo CD 在不同區域或帳戶中跨多個 EKS 叢集部署應用程式。使用一致的政策和工作流程，從單一 Argo CD 執行個體管理所有部署。

範例：將相同的應用程式部署到跨多個區域的開發、預備和生產叢集。Argo CD 可確保每個環境與其對應的 Git 分支保持同步。

多叢集管理

使用 ACK 定義和佈建 EKS 叢集、使用組織標準自訂叢集組態的 kro，以及管理叢集生命週期和組態的 Argo CD。這提供從建立到持續操作end-to-end叢集管理。

範例：使用 ACK 和 kro 定義 EKS 叢集來佈建和管理叢集基礎設施，定義聯網、安全政策、附加元件和其他組態的組織標準。使用 Argo CD，利用一致的標準和自動化生命週期管理，在整個機群中建立並持續管理叢集、組態和 Kubernetes 版本更新。

遷移和現代化

使用原生雲端資源佈建和 GitOps 工作流程簡化遷移至 EKS。使用 ACK 來採用現有的 AWS 資源，而不重新建立這些資源，並使用 Argo CD 從 Git 操作工作負載部署。

範例：從 EC2 遷移至 EKS 的團隊會使用 ACK 採用其現有的 RDS 資料庫和 S3 儲存貯體，然後使用 Argo CD 從 Git 部署容器化應用程式。遷移路徑很清楚，操作會從第一天開始標準化。

帳戶和區域引導

使用 Argo CD 和 ACK 自動化跨帳戶和區域的基礎設施推展。在 Git 中將基礎設施定義為程式碼，並讓功能處理部署和管理。

範例：平台團隊會維護 Git 儲存庫，定義標準帳戶組態：VPCs、IAM 角色、RDS 執行個體和監控堆疊。Argo CD 會自動將這些組態部署到新帳戶和區域，確保一致性並將手動設定時間從幾天縮短到幾分鐘。

使用 功能資源

列出、描述、更新和刪除 Amazon EKS 叢集上的功能資源。

EKS 功能資源

EKS 功能是可在 Amazon EKS 叢集上啟用受管功能 AWS 的資源。功能在 EKS 中執行，無需在工作者節點上安裝和維護控制器和其他操作元件。系統會針對特定 EKS 叢集建立功能，並在叢集的整個生命週期內與該叢集保持關聯。

每個功能資源都有：

- 叢集內的唯一名稱
- 功能類型 (ACK、ARGOCD 或 KRO)
- Amazon Resource Name (ARN)，同時指定名稱和類型
- 功能 IAM 角色
- 指出其目前狀態的狀態
- 組態，包括通用和特定於 功能類型

了解功能狀態

功能資源的狀態會指出其目前狀態。您可以在 EKS 主控台或使用 AWS CLI 檢視功能狀態和運作狀態。

主控台：

1. 在以下網址開啟 Amazon EKS 主控台：https://console.aws.amazon.com/eks/home#/clusters。
2. 選取您的叢集名稱。
3. 選擇功能索引標籤以檢視所有功能的狀態。
4. 如需詳細運作狀態資訊，請選擇可觀測性索引標籤，然後選擇監控叢集，然後選擇功能索引標籤。

AWS CLI：

```
aws eks describe-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-capability-name
```

功能狀態

CREATING：正在設定功能。您可以離開 主控台，此功能將繼續在背景中建立。

ACTIVE：功能正在執行並準備好使用。如果資源未如預期般運作，請檢查資源狀態和 IAM 許可。如需指引，請參閱the section called “故障診斷功能”。

正在更新：正在套用組態變更。等待狀態傳回 ACTIVE。

刪除：正在從叢集中移除功能。

CREATE_FAILED：設定發生錯誤。常見原因包括：

- IAM 角色信任政策不正確或遺失
- IAM 角色不存在或無法存取
- 叢集存取問題
- 無效的組態參數

如需特定錯誤詳細資訊，請參閱功能運作狀態區段。

UPDATE_FAILED：組態更新失敗。檢查功能運作狀態區段以取得詳細資訊，並確認 IAM 許可。

> **Tip**
>
> 如需詳細的故障診斷指引，請參閱：
>
> - the section called “故障診斷功能” - 一般功能疑難排解
> - the section called “疑難排解” - ACK 特定問題
> - the section called “疑難排解” - Argo CD 特定問題
> - the section called “疑難排解” - kro 特定問題

建立 功能

若要在您的叢集上建立功能，請參閱下列主題：

- the section called “建立 ACK 功能” – 建立 ACK 功能，以使用 Kubernetes APIs 管理 AWS 資源
- the section called “建立 Argo CD 功能” – 為 GitOps 持續交付建立 Argo CD 功能
- the section called “建立 kro 功能” – 建立資源合成和協同運作的 kro 功能

列出功能

您可以列出叢集上的所有功能資源。

主控台

1. 在以下網址開啟 Amazon EKS 主控台：https://console.aws.amazon.com/eks/home#/clusters。
2. 選取您的叢集名稱以開啟叢集詳細資訊頁面。
3. 選擇功能索引標籤。
4. 在受管功能下檢視功能資源。

AWS CLI

使用 `list-capabilities`命令來檢視叢集上的所有功能。將 *`region-code`* 取代為您的叢集所在的 AWS 區域，並將 *`my-cluster`* 取代為您的叢集名稱。

```
aws eks list-capabilities \
  --region region-code \
  --cluster-name my-cluster
```

```
{
    "capabilities": [
        {
            "capabilityName": "my-ack",
            "arn": "arn:aws:eks:us-west-2:111122223333:capability/my-cluster/ack/my-
ack/abc123",
            "type": "ACK",
            "status": "ACTIVE",
            "createdAt": "2025-11-02T10:30:00.000000-07:00",
            "modifiedAt": "2025-11-02T10:32:15.000000-07:00"
        },
        {
            "capabilityName": "my-kro",
            "arn": "arn:aws:eks:us-west-2:111122223333:capability/my-cluster/kro/my-
kro/abc123",
            "type": "KRO",
            "status": "ACTIVE",
            "version": "v0.6.3",
            "createdAt": "2025-11-02T10:30:00.000000-07:00",
            "modifiedAt": "2025-11-02T10:32:15.000000-07:00"
        },
        {
            "capabilityName": "my-argocd",
            "arn": "arn:aws:eks:us-west-2:111122223333:capability/my-cluster/argocd/my-
argocd/abc123",
            "type": "ARGOCD",
            "status": "ACTIVE",
            "version": "3.1.8-eks-1",
            "createdAt": "2025-11-21T08:22:28.486000-05:00",
            "modifiedAt": "2025-11-21T08:22:28.486000-05:00"
        }
    ]
}
```

描述功能

取得特定功能的詳細資訊，包括其組態和狀態。

主控台

1. 在以下網址開啟 Amazon EKS 主控台：https://console.aws.amazon.com/eks/home#/clusters。
2. 選取您的叢集名稱以開啟叢集詳細資訊頁面。
3. 選擇功能索引標籤。
4. 選擇您要從受管功能檢視的功能。
5. 檢視功能詳細資訊，包括狀態、組態和建立時間。

AWS CLI

使用 describe-capability命令來檢視詳細資訊。將 *region-code* 取代為您叢集所在的 AWS 區域，將 *my-cluster* 取代為您叢集的名稱，並將 *capability-name* 取代為您建立功能時提供的名稱 （ 例如 my-ack、 my-argocd或 my-kro)。

```
aws eks describe-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name capability-name
```

輸出範例：

```
{
  "capability": {
    "capabilityName": "my-ack",
    "capabilityArn": "arn:aws:eks:us-west-2:111122223333:capability/my-cluster/ack/my-
ack/abc123",
    "clusterName": "my-cluster",
    "type": "ACK",
    "roleArn": "arn:aws:iam::111122223333:role/AmazonEKSCapabilityACKRole",
    "status": "ACTIVE",
    "configuration": {},
    "tags": {},
    "health": {
      "issues": []
    },
    "createdAt": "2025-11-19T17:11:30.242000-05:00",
    "modifiedAt": "2025-11-19T17:11:30.242000-05:00",
    "deletePropagationPolicy": "RETAIN"
```

```
  }
}
```

更新 功能的組態

您可以在建立後更新功能組態的某些層面。特定組態選項會因功能類型而有所不同。

> **Note**
>
> EKS 功能資源受到完整管理，包括修補和版本更新。更新功能會更新資源組態，而且不會導致受管功能元件的版本更新。

AWS CLI

使用 `update-capability`命令來修改功能：

```
aws eks update-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name capability-name \
  --role-arn arn:aws:iam::111122223333:role/NewCapabilityRole
```

> **Note**
>
> 並非所有功能屬性都可以在建立後更新。如需可修改內容的詳細資訊，請參閱功能特定的文件。

刪除功能

當您不再需要叢集上的功能時，您可以刪除功能資源。

> **Important**
>
> 在刪除功能之前刪除叢集資源。
> 刪除功能資源不會自動刪除透過該功能建立的資源：

- 所有 Kubernetes 自訂資源定義 (CRDs仍安裝在您的叢集中。
- ACK 資源會保留在您的叢集中，而對應的 AWS 資源會保留在您的帳戶中
- Argo CD 應用程式及其 Kubernetes 資源會保留在您的叢集中
- kro ResourceGraphDefinitions 和執行個體會保留在您的叢集中

您應該在刪除功能之前刪除這些資源，以避免孤立的資源。
您可以選擇保留與 ACK Kubernetes AWS 資源相關聯的資源。請參閱 ACK 考量事項

主控台

1. 在以下網址開啟 Amazon EKS 主控台：https://console.aws.amazon.com/eks/home#/clusters。
2. 選取您的叢集名稱以開啟叢集詳細資訊頁面。
3. 選擇功能索引標籤。
4. 從受管功能清單中選擇要刪除的功能。
5. 選擇刪除功能。
6. 在確認對話方塊中，輸入 功能的名稱以確認刪除。
7. 選擇 刪除。

AWS CLI

使用 `delete-capability`命令來刪除功能資源：

將 *region-code* 取代為您的叢集所在的 AWS 區域，將 *my-cluster* 取代為您的叢集名稱，並將 *capability-name* 取代為要刪除的功能名稱。

```
aws eks delete-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name capability-name
```

後續步驟

- the section called “Kubernetes 資源” – 了解每個功能類型提供的 Kubernetes 資源

- the section called “ACK 概念” – 了解 ACK 概念和資源生命週期
- the section called “使用 Argo CD” – 使用適用於 GitOps 工作流程的 Argo CD 功能
- the section called “kro 概念” – 了解 kro 概念和資源合成

存取 EKS 功能控制器日誌

ACK、kro 和 Argo CD 的 EKS 功能控制器會在叢集外部執行受 AWS管基礎設施。您可以使用 Amazon CloudWatch Vended Logs 為這些控制器設定日誌交付，讓您了解控制器行為以進行監控和故障診斷。

日誌會以結構化 JSON 格式交付，並包含操作欄位，例如日誌層級、訊息、控制器名稱和對帳識別符。交付前會先篩選內部 AWS 中繼資料，您只會收到與功能控制器相關的操作日誌內容。

支援的日誌類型

每個 功能都有一或多個日誌類型，您可以獨立設定為 CloudWatch Vended Logs 交付來源。

ACK

- EKS_CAPABILITY_ACK_LOGS

kro

- EKS_CAPABILITY_KRO_LOGS

Argo CD

- EKS_CAPABILITY_ARGOCD_APPLICATION_LOGS
- EKS_CAPABILITY_ARGOCD_APPLICATIONSET_LOGS
- EKS_CAPABILITY_ARGOCD_COMMITSERVER_LOGS
- EKS_CAPABILITY_ARGOCD_REPOSERVER_LOGS
- EKS_CAPABILITY_ARGOCD_SERVER_LOGS

ACK 使用涵蓋所有 ACK 服務控制器的單一日誌類型。日誌記錄包含 `controllerGroup` 欄位（例如 `s3.services.k8s.aws`、`rds.services.k8s.aws`），可識別哪些 ACK 服務控制器產生日誌。您可以使用此欄位來篩選 CloudWatch Logs Insights 或其他查詢工具中特定服務控制器的日誌。

Argo CD 有五種不同的日誌類型，每個控制器元件一個。這可讓您僅啟用所需控制器的記錄，並將其路由到不同的目的地。

設定日誌交付

若要設定功能控制器日誌交付，請使用 AWS 管理主控台 或 Amazon CloudWatch Logs API。如需詳細設定指示，請參閱《Amazon CloudWatch Logs 使用者指南》中的從 AWS 服務啟用記錄。每個功能控制器都可以設定為個別的 CloudWatch Vended Logs 交付來源，因此您可以選擇要接收的日誌。

主控台

1. 在以下網址開啟 Amazon EKS 主控台：https://console.aws.amazon.com/eks/home#/clusters。
2. 選取您的叢集名稱。
3. 選擇功能索引標籤，然後選取您的功能。
4. 在日誌交付區段中，選擇新增。
5. 選取您 功能的日誌類型，然後選擇目的地。
6. 選擇新增以建立交付。

使用 Amazon CloudWatch APIs

設定記錄需要三個步驟：

1. 使用 CloudWatch PutDeliverySource API 建立 功能的交付來源。使用 功能 ARN 做為 ，resourceArn並指定所需的日誌類型。
2. 使用 建立交付目的地PutDeliveryDestination。指定 CloudWatch 日誌群組、S3 儲存貯體或 Amazon Data Firehose 交付串流的 ARN。
3. 建立交付以使用 連接來源和目的地CreateDelivery。

您可以使用 describe-capability命令擷取功能 ARN：

```
aws eks describe-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-capability \
  --query 'capability.capabilityArn' --output text
```

您可以設定單一功能（交付來源），透過建立多個交付將日誌傳送至多個目的地。您也可以將日誌從多個功能傳送至相同的目的地。

IAM 許可

根據選取的目的地，您可能需要為 CloudWatch 日誌群組、S3 儲存貯體或 Data Firehose 交付串流設定 IAM 政策或角色，以確保日誌交付成功。如果您要跨 AWS 帳戶傳送日誌，請使用 `PutDeliveryDestinationPolicy` API 來設定允許交付至目的地的 IAM 政策。如需其他資訊，請參閱 CloudWatch Vended Logs 許可文件。

範例：使用 CloudWatch Logs Insights 查詢日誌

若要檢視所有控制器的錯誤日誌：

```
fields @timestamp, controller, message, error
| filter level = "error"
| sort @timestamp desc
| limit 50
```

如需更多查詢範例，包括依服務控制器篩選、追蹤對帳週期和 Argo CD 應用程式篩選，請參閱使用控制器日誌進行故障診斷。

檢視您的日誌

設定日誌交付後，日誌會交付到您指定的目的地。存取日誌的方法取決於您選擇的目的地類型：

- CloudWatch Logs — 在 CloudWatch Logs 主控台中檢視日誌、使用 AWS CLI 命令，或搭配 CloudWatch Logs Insights 進行查詢。
- Amazon S3 — 透過 S3 主控台、 AWS CLI 或 Amazon Athena 等分析工具，將日誌作為 S3 儲存貯體中的物件存取。
- Amazon Data Firehose — 日誌會串流到您設定的 Firehose 目標（例如 S3、OpenSearch Service、Redshift 等）。

定價

根據您選擇的交付目的地，CloudWatch Vended Logs 會收取日誌交付和儲存的費用。與標準 CloudWatch Logs 相比，CloudWatch Vended Logs 以更低的價格使用內建 AWS 身分驗證和授權來實現可靠、安全的日誌交付。如需詳細資訊，請參閱 CloudWatch 定價頁面的已修訂日誌一節。

功能 Kubernetes 資源

在叢集上啟用功能後，您通常會透過在叢集中建立和管理 Kubernetes 自訂資源來與其互動。每個功能都提供自己的一組自訂資源定義 (CRDs)，以功能特定的功能擴展 Kubernetes API。

Argo CD 資源

啟用 Argo CD 功能時，您可以建立和管理下列 Kubernetes 資源：

Application (應用程式)

定義從 Git 儲存庫到目標叢集的部署。 `Application` 資源會指定來源儲存庫、目標命名空間和同步政策。每個 Argo CD 功能執行個體最多可以建立 3，000 個`Application`資源。

ApplicationSet

從 範本產生多個`Application`資源，啟用多叢集和多環境部署。 `ApplicationSet` 資源會使用產生器，根據叢集清單、Git 目錄或其他來源動態建立`Application`資源。

AppProject

提供 `Application` 資源的邏輯分組和存取控制。 `AppProject` 資源會定義`Application`資源可以使用哪些儲存庫、叢集和命名空間，進而啟用多租戶和安全界限。

資源範例`Application`：

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: my-app
  namespace: argocd
spec:
  project: default
  source:
    repoURL: https://github.com/org/repo
    targetRevision: main
    path: manifests
  destination:
    server: https://kubernetes.default.svc
    namespace: production
```

如需 Argo CD 資源和概念的詳細資訊，請參閱 the section called “Argo CD 概念”。

kro 資源

啟用 kro 功能時，您可以建立和管理下列 Kubernetes 資源：

ResourceGraphDefinition (RGD)

定義自訂 API，將多個 Kubernetes AWS 和資源組成為更高層級的抽象。平台團隊會建立 ResourceGraphDefinition 資源，以提供具有護欄的可重複使用模式。

自訂資源執行個體

建立ResourceGraphDefinition資源後，您可以建立由 定義的自訂 API 執行個體ResourceGraphDefinition。 kro 會自動建立和管理 中指定的資源ResourceGraphDefinition。

資源範例ResourceGraphDefinition：

```
apiVersion: kro.run/v1alpha1
kind: ResourceGraphDefinition
metadata:
  name: web-application
spec:
  schema:
    apiVersion: v1alpha1
    kind: WebApplication
    spec:
      name: string
      replicas: integer
  resources:
    - id: deployment
      template:
        apiVersion: apps/v1
        kind: Deployment
        # ... deployment spec
    - id: service
      template:
        apiVersion: v1
        kind: Service
        # ... service spec
```

範例WebApplication執行個體：

```
apiVersion: v1alpha1
kind: WebApplication
metadata:
  name: my-web-app
  namespace: default
spec:
  name: my-web-app
  replicas: 3
```

當您套用此執行個體時，kro 會自動建立 中Service定義的 Deployment和資源ResourceGraphDefinition。

如需 kro 資源和概念的詳細資訊，請參閱 the section called “kro 概念”。

ACK 資源

啟用 ACK 功能時，您可以使用 Kubernetes 自訂 AWS 資源來建立和管理資源。ACK 為超過 50 種 AWS 服務提供超過 200 CRDs，可讓您定義 AWS Kubernetes 工作負載的資源，以及使用 Kubernetes 管理專用 AWS 基礎設施資源。

ACK 資源的範例：

S3 儲存貯體

Bucket 資源會使用版本控制、加密和生命週期政策來建立和管理 Amazon S3 儲存貯體。

RDS DBInstance

DBInstance 資源使用自動備份和維護時段佈建和管理 Amazon RDS 資料庫執行個體。

DynamoDB 資料表

Table 資源會使用佈建或隨需容量建立和管理 DynamoDB 資料表。

IAM 角色

Role 資源定義具有信任政策和 AWS 服務存取許可政策的 IAM 角色。

Lambda 函數

Function 資源會使用程式碼、執行時間和執行角色組態來建立和管理 Lambda 函數。

Bucket 資源的範例規格：

```
apiVersion: s3.services.k8s.aws/v1alpha1
kind: Bucket
metadata:
  name: my-app-bucket
spec:
  name: my-unique-bucket-name-12345
  versioning:
    status: Enabled
  encryption:
    rules:
      - applyServerSideEncryptionByDefault:
          sseAlgorithm: AES256
```

如需 ACK 資源和概念的詳細資訊，請參閱 the section called “ACK 概念”。

資源限制

EKS 功能具有下列資源限制：

Argo CD 用量限制：

- 每個 Argo CD 功能執行個體最多 3，000 個`Application`資源
- 每個 Argo CD 功能執行個體最多設定 100 個遠端叢集

資源組態限制：

- Argo CD 中每個資源最多 150 個 Kubernetes `Application` 資源
- kro `ResourceGraphDefinition`中每個資源最多 64 個 Kubernetes 資源

> **Note**
>
> 這些限制適用於每個功能執行個體管理的資源數量。如果您需要更高的限制，您可以在多個叢集之間部署功能。

後續步驟

如需功能特定的任務和進階組態，請參閱下列主題：

- the section called “ACK 概念” – 了解 ACK 概念和資源生命週期
- the section called “使用 Argo CD” – 使用適用於 GitOps 工作流程的 Argo CD 功能
- the section called “kro 概念” – 了解 kro 概念和資源合成

EKS 功能考量

在部署 EKS 功能之前，請檢閱重要的考量事項，包括存取控制設計、在受管和自我管理解決方案之間進行選擇、多叢集部署的架構模式，以及操作最佳實務。

功能 IAM 角色和 Kubernetes RBAC

每個 EKS 功能資源都有一個已設定的功能 IAM 角色。功能角色用於授予 EKS 功能代您採取行動 AWS 的服務許可。例如，若要使用 ACK 的 EKS 功能來管理 Amazon S3 儲存貯體，您將授予該功能的 S3 儲存貯體管理許可，使其能夠建立和管理儲存貯體。

設定 功能後， AWS 即可使用叢集中的 Kubernetes 自訂資源建立和管理 中的 S3 資源。Kubernetes RBAC 是叢集內存取控制機制，用於判斷哪些使用者和群組可以建立和管理這些自訂資源。例如，授予特定 Kubernetes RBAC 使用者和群組在您選擇的命名空間中建立和管理`Bucket`資源的許可。

如此一來，IAM 和 Kubernetes RBAC 是end-to-end存取控制系統的一半，可管理與 EKS 功能和資源相關的許可。請務必為您的使用案例設計正確的 IAM 許可和 RBAC 存取政策組合。

如需功能 IAM 角色和 Kubernetes 許可的詳細資訊，請參閱 the section called “EKS 功能的考量事項”。

多叢集架構模式

在多個叢集之間部署功能時，請考慮這些常見的架構模式：

使用集中式管理進行對話

在集中受管叢集中執行這三種功能，以協調工作負載和管理跨多個工作負載叢集的雲端基礎設施。

- 管理叢集上的 Argo CD 會將應用程式部署到不同區域或帳戶中的工作負載叢集
- 管理叢集上的 ACK 會為所有叢集佈建 AWS 資源 (RDS、S3、IAM)
- 管理叢集上的 kro 會建立適用於所有叢集的可攜式平台抽象

此模式會集中工作負載和雲端基礎設施管理，並可簡化管理許多叢集的組織操作。

分散式 GitOps

工作負載和雲端基礎設施是由工作負載執行所在之相同叢集上的功能所管理。

- Argo CD 會管理本機叢集上的應用程式資源。
- ACK 資源用於叢集和工作負載需求。
- 安裝 kro 平台抽象並協調本機資源。

此模式會分散操作，讓團隊在一或多個叢集中管理自己的專用平台服務。

使用混合 ACK 部署的中樞和發言

結合集中式和分散式模型，並根據範圍和擁有權進行集中式應用程式部署和資源管理。

- Hub 叢集：
  - Argo CD 會管理本機叢集和所有遠端工作負載叢集的 GitOps 部署
  - ACK 用於管理範圍資源的管理叢集 （生產資料庫、IAM 角色、VPCs)
  - kro 用於管理叢集上，用於可重複使用的平台抽象化
- 呼叫叢集：
  - 工作負載是透過集中式中樞叢集上的 Argo CD 進行管理
  - ACK 在本機用於工作負載範圍的資源 (S3 儲存貯體、ElastiCache 執行個體、SQS 佇列）
  - kro 用於本機的資源組成和建置區塊模式

此模式會區隔考量，平台團隊會集中管理管理叢集上的關鍵基礎設施，選擇性包括工作負載叢集，而應用程式團隊則會指定和管理雲端資源與工作負載。

選擇模式

選取 架構時，請考慮下列因素：

- 組織結構：集中化平台團隊偏好中樞模式；分散式團隊可能偏好每個叢集的功能
- 資源範圍：管理員範圍的資源 （資料庫、IAM) 通常受益於集中管理；工作負載資源 （儲存貯體、佇列） 可在本機管理
- 自助式服務：集中式平台團隊可以撰寫和分發規範性自訂資源，以針對常見的工作負載需求啟用雲端資源的安全自助式服務
- 叢集機群管理：集中式管理叢集為 EKS 叢集機群管理提供客戶擁有的控制平面，以及其他管理員範圍的資源

- 合規要求：某些組織需要集中控制以進行稽核和控管
- 操作複雜性：功能執行個體較少可簡化操作，但可能會產生瓶頸

> **Note**
>
> 您可以從一個模式開始，並在您的平台成熟時發展為另一個模式。功能是獨立的，您可以根據您的需求，在叢集之間以不同的方式部署這些功能。

比較 EKS 功能與自我管理解決方案

EKS 功能為在 EKS 中執行的熱門 Kubernetes 工具和控制器提供全受管體驗。這與您在叢集中安裝和操作的自我管理解決方案不同。

主要差異

部署和管理

AWS 完全管理 EKS 功能，無需安裝、設定或維護元件軟體。 會自動在叢集中 AWS 安裝和管理所有必要的 Kubernetes 自訂資源定義 (CRDs)。

透過自我管理解決方案，您可以使用 Helm Chart、kubectl 或其他運算子來安裝和設定叢集軟體。您可以完全控制自我管理解決方案的軟體生命週期和執行期組態，在解決方案的任何層級提供自訂。

操作和維護

AWS 透過自動更新和安全性修補程式，管理 EKS 功能的修補和其他軟體生命週期操作。EKS 功能與簡化組態 AWS 的功能整合，提供內建的高可用性和容錯能力，並消除控制器工作負載的叢集內故障診斷。

自我管理解決方案需要您監控元件運作狀態和日誌、套用安全修補程式和版本更新、使用多個複本和 Pod 中斷預算設定高可用性、疑難排解和修復控制器工作負載問題，以及管理版本和版本。您可以完全控制部署，但這通常需要針對私有叢集存取和其他整合的自訂解決方案，這些解決方案必須符合組織標準和安全合規要求。

資源耗用量

EKS 功能會在 EKS 和叢集之外執行，釋放節點資源和叢集資源。功能不會使用叢集工作負載資源、不會在工作者節點上使用 CPU 或記憶體、自動擴展，以及對叢集容量規劃的影響降到最低。

自我管理的解決方案會在工作者節點上執行控制器和其他元件，並直接使用工作者節點資源、叢集 IPs 和其他叢集資源。管理叢集服務需要工作負載的容量規劃，而且需要資源請求和限制的規劃和組態，才能管理擴展和高可用性需求。

功能支援

EKS 功能是全受管服務功能，與自我管理解決方案相比，依其本質進行評價。雖然 功能將支援大多數功能和使用案例，但與自我管理解決方案相比，涵蓋範圍會有所不同。

透過自我管理解決方案，您可以完全控制軟體的組態、選用功能和其他功能層面。您可以選擇執行自己的自訂映像、自訂組態的所有層面，以及完全控制自我管理的解決方案功能。

成本考量事項

每個 EKS 功能資源都有相關的每小時成本，因功能類型而異。由 功能管理的叢集資源也具有相關聯的每小時成本和自己的定價。如需詳細資訊，請參閱 Amazon EKS 定價。

自我管理解決方案沒有與 AWS 費用相關的直接成本，但您需要支付控制器和相關工作負載所使用的叢集運算資源。除了節點和叢集資源消耗之外，使用自我管理解決方案的總體擁有成本包括營運開銷，以及維護、疑難排解和支援的費用。

在 EKS 功能和自我管理解決方案之間進行選擇

EKS 功能 當您想要降低營運開銷並專注於軟體和系統中的差異化價值，而不是叢集平台操作的基本需求時，請考慮此選項。當您想要將安全修補程式和軟體生命週期管理的操作負擔降至最低、釋放應用程式工作負載的節點和叢集資源、簡化組態和安全管理，以及從 AWS 支援涵蓋範圍中獲益時，請使用 EKS 功能。EKS 功能適用於大多數生產使用案例，是新部署的建議方法。

自我管理解決方案 當您需要特定的 Kubernetes 資源 API 版本、自訂控制器建置、根據自我管理部署建置現有的自動化和工具，或需要控制器執行期組態的深度自訂時，請考慮此選項。自我管理解決方案可為特殊使用案例提供彈性，而且您可以完全控制部署和執行期組態。

> **Note**
>
> EKS 功能可以與自我管理解決方案共存於叢集中，並且可以實現逐步遷移。

功能特定的比較

如需功能特定功能、上游差異和遷移路徑等詳細比較，請參閱：

- the section called “與自我管理的比較”
- the section called “與自我管理的比較”
- the section called “與自我管理的比較”

使用 Kubernetes 專用 AWS 控制器 (ACK) 從 Kubernetes 部署 AWS 資源

AWS Kubernetes (ACK) 的控制器可讓您直接從 Kubernetes 定義和管理 AWS 服務資源。使用 AWS Controllers for Kubernetes (ACK)，您可以使用 Kubernetes 自訂資源來管理工作負載資源和雲端基礎設施，以及使用熟悉的 Kubernetes APIs 和工具的應用程式工作負載。

透過 EKS 功能，ACK 完全由 管理 AWS，無需在叢集上安裝、維護和擴展 ACK 控制器。

ACK 的運作方式

ACK 會將 Kubernetes 自訂資源規格轉換為 AWS API 呼叫。當您建立、更新或刪除代表 AWS 服務資源的 Kubernetes 自訂資源時，ACK 會進行必要的 AWS API 呼叫來建立、更新或刪除 AWS 資源。

ACK 支援的每個 AWS 資源都有自己的自訂資源定義 (CRD)，可定義用於指定其組態的 Kubernetes API 結構描述。例如，ACK 為 S3 提供 CRDs，包括儲存貯體、儲存貯體政策和其他 S3 資源。

ACK 會持續將 AWS 資源的狀態與 Kubernetes 自訂資源中定義的所需狀態進行協調。如果資源偏離其所需狀態，ACK 會偵測到此情況，並採取修正動作，使其重新對齊。Kubernetes 資源的變更會立即反映在 AWS 資源狀態中，而上游 AWS 資源變更的被動偏離偵測和修復可能需要長達 10 小時 （ 重新同步期間 ），但通常會更快發生。

範例 S3 儲存貯體資源資訊清單

```
apiVersion: s3.services.k8s.aws/v1alpha1
kind: Bucket
metadata:
  name: my-ack-bucket
spec:
  name: my-unique-bucket-name
```

當您將此自訂資源套用至叢集時，如果帳戶尚未存在，ACK 會在帳戶中建立 Amazon S3 儲存貯體。此資源的後續變更，例如指定非預設儲存層或新增政策，將會套用至 中的 S3 資源 AWS。從叢集刪除此資源時，預設 AWS 會刪除 中的 S3 儲存貯體。

ACK 的優點

ACK 提供 Kubernetes 原生 AWS 資源管理，可讓您使用與應用程式相同的 Kubernetes APIs 和工具來管理 AWS 資源。這種統一的方法可簡化您的基礎設施管理工作流程，無需在不同工具之間切換或學習單獨的infrastructure-as-code系統。您可以在 Kubernetes 資訊清單中宣告地定義 AWS 資源，讓 GitOps 工作流程和基礎設施成為與現有開發程序無縫整合的程式碼實務。

ACK 會持續協調 AWS 資源的所需狀態及其實際狀態，修正偏離並確保基礎設施的一致性。此持續對帳表示對 AWS 資源進行必要的out-of-band變更會自動還原，以符合您宣告的組態，以維護基礎設施的完整性做為程式碼。您可以設定 ACK 以跨多個 AWS 帳戶和區域管理資源，啟用複雜的多帳戶架構，無需額外的工具。

對於從其他基礎設施管理工具遷移的組織，ACK 支援資源採用，允許您在 ACK 管理下使用現有 AWS 資源，而無需重新建立它們。ACK 也提供唯讀資源，用於不修改存取 AWS 的資源觀察，以及即使從叢集刪除 Kubernetes 資源，可選擇保留 AWS 資源的註釋。

若要進一步了解並開始使用適用於 ACK 的 EKS 功能，請參閱 the section called “ACK 概念”和 the section called “考量事項”。

支援 AWS 的服務

ACK 支援各種 AWS 服務，包括但不限於：

- Amazon EC2
- Amazon S3
- Amazon RDS
- Amazon DynamoDB
- Amazon ElastiCache
- Amazon EKS
- Amazon SQS
- Amazon SNS
- AWS Lambda
- AWS IAM

ACK 的 EKS 功能支援所有列為一般可用的上游 AWS 服務。如需支援 AWS 的服務完整清單，請參閱 ACK 網站上的 ACK 服務頁面。

與其他 EKS 受管功能整合

ACK 與其他 EKS 受管功能整合。

- Argo CD：使用 Argo CD 管理跨多個叢集的 ACK 資源部署，為您的 AWS 基礎設施啟用 GitOps 工作流程。
  - ACK 與 Argo CD 搭配使用時可延伸 GitOps 的優點，但 ACK 不需要與 git 整合。
- kro (Kube Resource Orchestrator)：使用 kro 從 ACK 資源編寫複雜的資源，建立更高階的抽象，簡化資源管理。
  - 您可以使用定義 Kubernetes 資源和資源的 kro 建立複合自訂 AWS 資源。團隊成員可以使用這些自訂資源快速部署複雜的應用程式。

ACK 入門

若要開始使用 ACK 的 EKS 功能：

1. 建立並設定具有必要許可的 IAM 功能角色，讓 ACK 代表您管理 AWS 資源。
2. 透過 AWS 主控台、 AWS CLI 或您偏好的基礎設施做為程式碼工具，在 EKS 叢集上建立 ACK 功能資源。
3. 將 Kubernetes 自訂資源套用至您的叢集，以開始在 Kubernetes 中管理您的 AWS 資源。

建立 ACK 功能

本章說明如何在 Amazon EKS 叢集上建立 ACK 功能。

先決條件

建立 ACK 功能之前，請確定您有：

- Amazon EKS 叢集
- IAM 功能角色，具有 ACK 管理 AWS 資源的許可
- 足夠的 IAM 許可，可在 EKS 叢集上建立功能資源
- 安裝和設定適當的 CLI 工具，或存取 EKS 主控台

如需建立 IAM 功能角色的說明，請參閱 the section called “功能 IAM 角色”。

> **Important**
>
> ACK 是一種基礎設施管理功能，可授予建立、修改和刪除 AWS 資源的能力。這是應謹慎控制的管理員範圍功能。有權在叢集中建立 Kubernetes 資源的任何人都可以有效透過 ACK 建立 AWS 資源，但需受 IAM 功能角色許可的約束。您提供的 IAM 功能角色會決定 ACK 可以建立和管理哪些 AWS 資源。如需建立具有最低權限許可之適當角色的指引，請參閱 the section called “功能 IAM 角色”和 the section called “EKS 功能的考量事項”。

選擇您的工具

您可以使用、 AWS 管理主控台 AWS CLI 或 eksctl 建立 ACK 功能：

- the section called “主控台” - 使用 主控台進行引導式體驗
- the section called “ AWS CLI” - 使用 AWS CLI 進行指令碼編寫和自動化
- the section called “eksctl” - 使用 eksctl 獲得 Kubernetes 原生體驗

建立 ACK 功能時會發生什麼情況

當您建立 ACK 功能時：

1. EKS 會建立 ACK 功能服務，並將其設定為監控和管理叢集中的資源
2. 您的叢集中已安裝自訂資源定義 (CRDs)
3. 會自動為您的 IAM 功能角色建立存取項目，其中包含授予基準 Kubernetes 許可的特定功能存取項目政策（請參閱 the section called “EKS 功能的考量事項”）
4. 該功能會擔任您提供的 IAM 功能角色
5. ACK 開始在您的叢集中監看其自訂資源
6. 功能狀態從 變更為 `CREATING` `ACTIVE`

啟用後，您可以在叢集中建立 ACK 自訂資源來管理 AWS 資源。

> **Note**
>
> 自動建立的存取項目包含授予 ACK 管理 AWS 資源許可`AmazonEKSACKPolicy`的 。有些參考 Kubernetes 秘密的 ACK 資源（例如具有密碼的 RDS 資料庫）需要額外的存取項目政策。

若要進一步了解存取項目以及如何設定其他許可，請參閱 the section called “EKS 功能的考量事項”。

後續步驟

建立 ACK 功能之後：

- the section called “ACK 概念” - 了解 ACK 概念並開始使用 AWS 資源
- the section called “設定 許可” - 設定 IAM 許可和多帳戶模式

使用主控台建立 ACK 功能

使用 AWS 管理主控台在 Amazon EKS 叢集上建立 ACK 功能。

建立 ACK 功能

1. 在以下網址開啟 Amazon EKS 主控台：https://console.aws.amazon.com/eks/home#/clusters。
2. 選取您的叢集名稱以開啟叢集詳細資訊頁面。
3. 選擇功能索引標籤。
4. 在左側導覽中，選擇 Kubernetes (ACK) 的 AWS 控制器。
5. 選擇建立 Kubernetes 功能的 AWS 控制器。
6. 對於 IAM 功能角色：
   - 如果您已經有 IAM 功能角色，請從下拉式清單中選取它
   - 如果您需要建立角色，請選擇建立管理員角色

   這會在新標籤中開啟 IAM 主控台，其中包含預先填入的信任政策和 `AdministratorAccess`受管政策。您可以取消選取此政策，並視需要新增其他許可。

   建立角色後，返回 EKS 主控台並自動選取角色。

   > **Important**
   >
   > 建議`AdministratorAccess`的政策授予廣泛的許可，旨在簡化入門。對於生產用途，請以自訂政策取代此政策，該政策僅授予您計劃使用 ACK 管理之特定 AWS 服務所需的許可。如需建立最低權限政策的指引，請參閱 the section called “設定 許可”和 the section called “EKS 功能的考量事項”。

7. 選擇建立。

功能建立程序開始。

驗證功能是否作用中

1. 在功能索引標籤上，檢視 ACK 功能狀態。
2. 等待狀態從 變更為 CREATING ACTIVE。
3. 啟用後，此功能即可使用。

如需功能狀態和故障診斷的資訊，請參閱 the section called “使用 功能”。

確認可用的自訂資源

功能處於作用中狀態後，請確認叢集中是否有可用的 ACK 自訂資源。

使用主控台

1. 在 Amazon EKS 主控台中導覽至您的叢集
2. 選擇資源索引標籤
3. 選擇延伸模組
4. 選擇 CustomResourceDefinitions

您應該會看到一些針對 AWS 資源列出的 CRDs。

使用 kubectl

```
kubectl api-resources | grep services.k8s.aws
```

您應該會看到一些針對 AWS 資源列出的 APIs。

> **Note**
>
> 適用於 Kubernetes 的 AWS 控制器功能將為各種 AWS 資源安裝多個 CRDs。

後續步驟

- the section called “ACK 概念” - 了解 ACK 概念並開始使用

- the section called “設定 許可” - 設定其他服務的 IAM 許可 AWS
- the section called “使用 功能” - 管理您的 ACK 功能資源

使用 CLI 建立 ACK AWS 功能

使用 CLI 在 Amazon EKS 叢集上建立 ACK AWS 功能。此程序會逐步引導您建立 IAM 角色，並驗證功能是否處於作用中狀態。

先決條件

- AWS CLI – 版本 `2.12.3` 或更新版本。若要檢查您的版本，請執行 `aws --version`。如需詳細資訊，請參閱《 AWS 命令列界面使用者指南》中的安裝。
- **`kubectl`** ：命令列工具，適用於使用 Kubernetes 叢集。如需詳細資訊，請參閱the section called “設定 kubectl 和 eksctl”。

步驟 1：建立 IAM 功能角色

建立信任政策檔案：

```
cat > ack-trust-policy.json << 'EOF'
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "capabilities.eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ]
    }
  ]
}
EOF
```

建立 IAM 角色：

```
aws iam create-role \
```

```
  --role-name ACKCapabilityRole \
  --assume-role-policy-document file://ack-trust-policy.json
```

將 AdministratorAccess受管政策連接至角色：

```
aws iam attach-role-policy \
  --role-name ACKCapabilityRole \
  --policy-arn arn:aws:iam::aws:policy/AdministratorAccess
```

Important

建議AdministratorAccess的政策授予廣泛的許可，旨在簡化入門。對於生產用途，請以自訂政策取代此政策，該政策僅授予您計劃使用 ACK 管理之特定 AWS 服務所需的許可。如需建立最低權限政策的指引，請參閱 the section called “設定 許可”和 the section called “EKS 功能的考量事項”。

步驟 2：建立 ACK 功能

在叢集上建立 ACK 功能資源。將 *region-code* 取代為您的叢集所在的 AWS 區域，並將 *my-cluster* 取代為您的叢集名稱。

```
aws eks create-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-ack \
  --type ACK \
  --role-arn arn:aws:iam::$(aws sts get-caller-identity --query Account --output
 text):role/ACKCapabilityRole \
  --delete-propagation-policy RETAIN
```

命令會立即傳回，但當 EKS 建立所需的功能基礎設施和元件時，功能需要一些時間才會變成作用中。EKS 會在建立叢集時，在叢集中安裝與此功能相關的 Kubernetes 自訂資源定義。

Note

如果您收到叢集不存在或您沒有許可的錯誤，請驗證：

- 叢集名稱正確
- 您的 AWS CLI 已設定為正確的區域

- 您擁有必要的 IAM 許可

步驟 3：確認功能處於作用中狀態

等待 功能變成作用中。將 *region-code* 取代為您的叢集所在的 AWS 區域，並將 *my-cluster* 取代為您的叢集名稱。

```
aws eks describe-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-ack \
  --query 'capability.status' \
  --output text
```

當狀態顯示 時，此功能已就緒ACTIVE。在狀態為 之前，請勿繼續進行下一個步驟ACTIVE。

您也可以檢視完整的功能詳細資訊：

```
aws eks describe-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-ack
```

步驟 4：確認可用的自訂資源

功能處於作用中狀態後，請確認叢集中是否有可用的 ACK 自訂資源：

```
kubectl api-resources | grep services.k8s.aws
```

您應該會看到一些針對 AWS 資源列出的 APIs。

> **Note**
>
> 適用於 Kubernetes 的 AWS 控制器功能將為各種 AWS 資源安裝多個 CRDs。

後續步驟

- the section called “ACK 概念” - 了解 ACK 概念並開始使用

- the section called “設定 許可” - 設定其他服務的 IAM 許可 AWS
- the section called “使用 功能” - 管理您的 ACK 功能資源

使用 eksctl 建立 ACK 功能

使用 eksctl 在 Amazon EKS 叢集上建立 ACK 功能。

> **Note**
>
> 下列步驟需要 eksctl 版本 `0.215.0` 或更新版本。若要檢查您的版本，請執行 `eksctl version`。

步驟 1：建立 IAM 功能角色

建立信任政策檔案：

```
cat > ack-trust-policy.json << 'EOF'
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "capabilities.eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ]
    }
  ]
}
EOF
```

建立 IAM 角色：

```
aws iam create-role \
  --role-name ACKCapabilityRole \
  --assume-role-policy-document file://ack-trust-policy.json
```

將 AdministratorAccess受管政策連接至角色：

```
aws iam attach-role-policy \
  --role-name ACKCapabilityRole \
  --policy-arn arn:aws:iam::aws:policy/AdministratorAccess
```

> **Important**
>
> 建議AdministratorAccess的政策授予廣泛的許可，旨在簡化入門。對於生產用途，請以自訂政策取代此政策，該政策僅授予您計劃使用 ACK 管理之特定 AWS 服務所需的許可。如需建立最低權限政策的指引，請參閱 the section called “設定 許可”和 the section called “EKS 功能的考量事項”。

步驟 2：建立 ACK 功能

使用 eksctl 建立 ACK 功能。將 *region-code* 取代為您的叢集所在的 AWS 區域，並將 *my-cluster* 取代為您的叢集名稱。

```
eksctl create capability \
  --cluster my-cluster \
  --region region-code \
  --name ack \
  --type ACK \
  --role-arn arn:aws:iam::$(aws sts get-caller-identity --query Account --output
 text):role/ACKCapabilityRole
```

命令會立即傳回，但 功能需要一些時間才會變成作用中。

步驟 3：確認功能處於作用中狀態

檢查功能狀態：

```
eksctl get capability \
  --cluster my-cluster \
  --region region-code \
  --name ack
```

當狀態顯示 時，此功能已就緒ACTIVE。

步驟 4：確認可用的自訂資源

功能處於作用中狀態後，請確認叢集中是否有可用的 ACK 自訂資源：

```
kubectl api-resources | grep services.k8s.aws
```

您應該會看到一些針對 AWS 資源列出的 APIs。

> **Note**
>
> 適用於 Kubernetes 的 AWS 控制器功能將為各種 AWS 資源安裝多個 CRDs。

後續步驟

- the section called “ACK 概念” - 了解 ACK 概念並開始使用
- the section called “設定 許可” - 設定其他服務的 IAM 許可 AWS
- the section called “使用 功能” - 管理您的 ACK 功能資源

ACK 概念

ACK 透過 Kubernetes APIs 持續協調資訊清單中的所需狀態與實際狀態，來管理 AWS 資源 AWS。當您建立或更新 Kubernetes 自訂資源時，ACK 會進行必要的 AWS API 呼叫，以建立或修改對應的 AWS 資源，然後監控它是否有偏離，並更新 Kubernetes 狀態以反映目前的狀態。此方法可讓您使用熟悉的 Kubernetes 工具和工作流程來管理基礎設施，同時保持叢集與 之間的一致性 AWS。

本主題說明 ACK 如何透過 Kubernetes APIs 管理 AWS 資源的基本概念。

ACK 入門

建立 ACK 功能後 （ 請參閱 the section called “建立 ACK 功能”)，您可以開始使用叢集中的 Kubernetes 資訊清單來管理 AWS 資源。

例如，在 中建立此 S3 儲存貯體資訊清單`bucket.yaml`，選擇您自己的唯一儲存貯體名稱。

```
apiVersion: s3.services.k8s.aws/v1alpha1
kind: Bucket
metadata:
  name: my-test-bucket
```

```
  namespace: default
spec:
  name: my-unique-bucket-name-12345
```

套用資訊清單：

```
kubectl apply -f bucket.yaml
```

檢查狀態：

```
kubectl get bucket my-test-bucket
kubectl describe bucket my-test-bucket
```

驗證儲存貯體是否已在 中建立 AWS：

```
aws s3 ls | grep my-unique-bucket-name-12345
```

刪除 Kubernetes 資源：

```
kubectl delete bucket my-test-bucket
```

確認儲存貯體已從下列位置刪除 AWS：

```
aws s3 ls | grep my-unique-bucket-name-12345
```

儲存貯體不應再出現在清單中，示範 ACK 會管理 AWS 資源的完整生命週期。

如需 ACK 入門的詳細資訊，請參閱 ACK 網站上的 ACK 入門。

資源生命週期和對帳

ACK 使用持續對帳迴圈，確保您的 AWS 資源符合 Kubernetes 資訊清單中定義的所需狀態。

對帳的運作方式：

1. 您可以建立或更新 Kubernetes 自訂資源 （ 例如 S3 儲存貯體 ）

2. ACK 會偵測變更，並將所需的狀態與 中的實際狀態進行比較 AWS
3. 如果不同，ACK 會呼叫 AWS API 以協調差異
4. ACK 會更新 Kubernetes 中的資源狀態，以反映目前狀態
5. 迴圈會持續重複，通常每隔幾個小時重複一次

當您建立新的 Kubernetes 資源、更新現有資源的 `spec`，或當 ACK AWS 從 ACK 外部的手動變更偵測到偏離時，就會觸發調校。此外，ACK 會以 10 小時的重新同步期間執行定期調校。Kubernetes 資源的變更會觸發立即對帳，而上游 AWS 資源變更的被動偏離偵測則會在定期重新同步期間發生。

完成上述入門範例時，ACK 會執行下列步驟：

1. 檢查儲存貯體是否存在於 AWS
2. 如果沒有， 會呼叫 `s3:CreateBucket`
3. 使用儲存貯體 ARN 和狀態更新 Kubernetes 狀態
4. 繼續監控偏離

如需 ACK 運作方式的詳細資訊，請參閱 ACK 網站上的 ACK 調校。

狀態條件

ACK 資源使用狀態條件來傳達其狀態。了解這些條件可協助您疑難排解問題並了解資源運作狀態。

- 就緒：表示資源已就緒可供使用 （ 標準化 Kubernetes 條件 ）。
- ACK.ResourceSynced：表示資源規格符合 AWS 資源狀態。
- ACK.Terminal：表示發生無法復原的錯誤。
- ACK.Adopted：表示資源是從現有資源採用，而不是建立新的 AWS 資源。
- ACK.Recoverable：表示可在不更新規格的情況下解決的可復原錯誤。
- ACK.Advisory：提供有關資源的諮詢資訊。
- ACK.LateInitialized：指出欄位的延遲初始化是否已完成。
- ACK.ReferencesResolved：指出是否已解析所有`AWSResourceReference`欄位。
- ACK.IAMRoleSelected：指出是否已選取 IAMRoleSelector 來管理此資源。

檢查資源狀態：

```
# Check if resource is ready
kubectl get bucket my-bucket -o jsonpath='{.status.conditions[?
(@.type=="Ready")].status}'

# Check for terminal errors
kubectl get bucket my-bucket -o jsonpath='{.status.conditions[?
(@.type=="ACK.Terminal")]}'
```

範例狀態：

```
status:
  conditions:
  - type: Ready
    status: "True"
    lastTransitionTime: "2024-01-15T10:30:00Z"
  - type: ACK.ResourceSynced
    status: "True"
    lastTransitionTime: "2024-01-15T10:30:00Z"
  - type: ACK.Terminal
    status: "False"
  ackResourceMetadata:
    arn: arn:aws:s3:::my-unique-bucket-name
    ownerAccountID: "111122223333"
    region: us-west-2
```

如需 ACK 狀態和條件的詳細資訊，請參閱 ACK 網站上的 ACK 條件。

刪除政策

ACK 的刪除政策會控制當您刪除 Kubernetes 資源時 AWS ，資源會發生什麼情況。

刪除 （ 預設 ）

刪除 Kubernetes 資源時會刪除 AWS 資源：這是預設行為。

```
# No annotation needed - this is the default
apiVersion: s3.services.k8s.aws/v1alpha1
kind: Bucket
metadata:
  name: temp-bucket
spec:
```

```
  name: temporary-bucket
```

刪除此資源會刪除其中的 S3 儲存貯體 AWS。

保留

當您刪除 Kubernetes 資源時， AWS 資源會保留：

```
apiVersion: s3.services.k8s.aws/v1alpha1
kind: Bucket
metadata:
  name: important-bucket
  annotations:
    services.k8s.aws/deletion-policy: "retain"
spec:
  name: production-data-bucket
```

刪除此資源會將其從 Kubernetes 中移除，但會保留 S3 儲存貯體 AWS。

此`retain`政策適用於生產資料庫，這些資料庫應超過 Kubernetes 資源、多個應用程式使用的共用資源、具有不應意外刪除之重要資料的資源，或您採用資源的暫時 ACK 管理、進行設定，然後將其釋出給手動管理。

如需 ACK 刪除政策的詳細資訊，請參閱 ACK 網站上的 ACK 刪除政策。

資源採用

採用可讓您在 ACK 管理下使用現有 AWS 資源，而無需重新建立這些資源。

何時使用採用：

- 將現有基礎設施遷移至 ACK 管理
- 在 Kubernetes 中意外刪除 AWS 資源時復原孤立的資源
- 匯入其他工具建立的資源 (CloudFormation、Terraform)

採用方式的運作方式：

```
apiVersion: s3.services.k8s.aws/v1alpha1
kind: Bucket
metadata:
```

```
  name: existing-bucket
  annotations:
    services.k8s.aws/adoption-policy: "adopt-or-create"
spec:
  name: my-existing-bucket-name
```

當您建立此資源時：

1. ACK 會檢查具有該名稱的儲存貯體是否存在於 AWS
2. 如果找到，ACK 會採用它（沒有要建立的 API 呼叫）
3. ACK 從 讀取目前的組態 AWS
4. ACK 會更新 Kubernetes 狀態以反映實際狀態
5. 未來的更新會正常協調資源

採用後，資源會像任何其他 ACK 資源一樣受到管理，除非您使用`retain`刪除政策，否則刪除 Kubernetes 資源將會刪除 AWS 資源。

採用資源時， AWS 資源必須已存在，且 ACK 需要讀取許可才能探索資源。如果資源存在，`adopt-or-create`政策會採用該資源，否則會建立該資源。當您想要可運作資源是否存在的宣告式工作流程時，此功能非常有用。

如需 ACK 資源採用的詳細資訊，請參閱 ACK 網站上的 ACK 資源採用。

跨帳戶和跨區域資源

ACK 可以從單一叢集管理不同 AWS 帳戶和區域中的資源。

跨區域資源註釋

您可以使用 註釋指定 AWS 資源的區域：

```
apiVersion: s3.services.k8s.aws/v1alpha1
kind: Bucket
metadata:
  name: eu-bucket
  annotations:
    services.k8s.aws/region: eu-west-1
spec:
  name: my-eu-bucket
```

您也可以指定在指定命名空間中建立的所有 AWS 資源的區域：

命名空間註釋

為命名空間中的所有資源設定預設區域：

```
apiVersion: v1
kind: Namespace
metadata:
  name: production
  annotations:
    services.k8s.aws/default-region: us-west-2
```

在此命名空間中建立的資源會使用此區域，除非以資源層級註釋覆寫。

跨帳戶

使用 IAM 角色選取器將特定 IAM 角色映射至命名空間：

```
apiVersion: services.k8s.aws/v1alpha1
kind: IAMRoleSelector
metadata:
  name: target-account-config
spec:
  arn: arn:aws:iam::444455556666:role/ACKTargetAccountRole
  namespaceSelector:
    names:
      - production
```

在映射命名空間中建立的資源會自動使用指定的角色。

若要進一步了解 IAM 角色選取器，請參閱 ACK 跨帳戶資源管理。如需跨帳戶組態詳細資訊，請參閱 the section called “設定 許可”。

錯誤處理和重試行為

ACK 會自動處理暫時性錯誤並重試失敗的操作。

重試策略：

- 暫時性錯誤（速率限制、暫時性服務問題、許可不足）觸發自動重試

- 指數退避可防止壓倒 AWS APIs
- 重試嘗試次數上限因錯誤類型而異
- 永久錯誤 （無效的參數、資源名稱衝突） 不會重試

使用 檢查資源狀態的錯誤詳細資訊`kubectl describe`：

```
kubectl describe bucket my-bucket
```

尋找包含錯誤訊息的狀態條件、顯示最近調校嘗試的事件，以及狀態條件中解釋失敗的 `message` 欄位。常見的錯誤包括 IAM 許可不足、資源名稱衝突 AWS、 中的組態值無效`spec`，以及超出 AWS 服務配額。

如需常見錯誤的疑難排解，請參閱 the section called “疑難排解”。

kro 的資源合成

若要編寫和連接多個 ACK 資源，請使用適用於 kro 的 EKS 功能 (Kube Resource Orchestrator)。kro 提供宣告式方法來定義資源群組，直接在資源之間傳遞組態以管理複雜的基礎設施模式。

如需使用 ACK 資源建立自訂資源組合的詳細範例，請參閱 the section called “kro 概念”。

後續步驟

- the section called “考量事項” - EKS 特定的模式和整合策略

設定 ACK 許可

ACK 需要 IAM 許可，才能代表您建立和管理 AWS 資源。本主題說明 IAM 如何與 ACK 搭配使用，並提供設定不同使用案例許可的指引。

IAM 如何與 ACK 搭配使用

ACK 使用 IAM 角色對 資源進行身分驗證 AWS 和執行動作。提供許可給 ACK 的方式有兩種：

功能角色：您在建立 ACK 功能時提供的 IAM 角色。根據預設，所有 ACK 操作都會使用此角色。

IAM 角色選取器：可映射至特定命名空間或資源的其他 IAM 角色。這些角色會覆寫其範圍內資源的功能角色。

當 ACK 需要建立或管理資源時，它會決定要使用的 IAM 角色：

1. 檢查 IAMRoleSelector 是否符合資源的命名空間
2. 如果找到相符項目，請擔任該 IAM 角色
3. 否則，請使用功能角色

此方法可實現從簡單的單一角色設定到複雜多帳戶、多團隊組態的彈性許可管理。

入門：簡易許可設定

對於開發、測試或簡單使用案例，您可以將所有必要的服務許可直接新增至功能角色。

此方法在下列情況下運作良好：

- 您即將開始使用 ACK
- 所有資源都在同一個 AWS 帳戶中
- 單一團隊管理所有 ACK 資源
- 您信任所有 ACK 使用者都具有相同的許可

生產最佳實務：IAM 角色選取器

對於生產環境，請使用 IAM 角色選取器實作最低權限存取和命名空間層級隔離。

使用 IAM 角色選取器時，能力角色只需要 `sts:AssumeRole`和 `sts:TagSession`許可，即可擔任服務特定的角色。您不需要將任何 AWS 服務許可（例如 S3 或 RDS) 新增至功能角色本身，這些許可會授予功能角色擔任的個別 IAM 角色。

在許可模型之間進行選擇：

在下列情況下使用直接許可（將服務許可新增至功能角色）：

- 您正在開始使用，並想要最簡單的設定
- 所有資源都與您的叢集位於相同的帳戶
- 您有全叢集的管理許可要求
- 所有團隊都可以共用相同的許可

在下列情況下使用 IAM 角色選取器：

- 跨多個 AWS 帳戶管理資源
- 不同的團隊或命名空間需要不同的許可
- 您需要每個命名空間的精細存取控制
- 您想要遵循最低權限的安全實務

您可以從直接許可開始，並在需求增加之後遷移至 IAM 角色選取器。

為什麼要在生產環境中使用 IAM 角色選取器：

- 最低權限：每個命名空間只會取得所需的許可
- 團隊隔離：團隊 A 不會意外使用團隊 B 的許可
- 更容易稽核：明確映射哪個命名空間使用哪個角色
- 跨帳戶支援：管理多個帳戶中的資源時需要
- 問題分離：不同的服務或環境使用不同的角色

基本 IAM 角色選取器設定

步驟 1：建立服務特定的 IAM 角色

建立具有特定 AWS 服務許可的 IAM 角色：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:*"
      ],
      "Resource": "*"
    }
  ]
}
```

設定信任政策以允許能力角色擔任它：

```
{
```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/ACKCapabilityRole"
      },
      "Action": ["sts:AssumeRole", "sts:TagSession"]
    }
  ]
}
```

步驟 2：將 AssumeRole 許可授予能力角色

將許可新增至功能角色以擔任服務特定角色：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["sts:AssumeRole", "sts:TagSession"],
      "Resource": "arn:aws:iam::111122223333:role/ACK-S3-Role"
    }
  ]
}
```

步驟 3：建立 IAMRoleSelector

將 IAM 角色映射至命名空間：

```
apiVersion: services.k8s.aws/v1alpha1
kind: IAMRoleSelector
metadata:
  name: s3-namespace-config
spec:
  arn: arn:aws:iam::111122223333:role/ACK-S3-Role
  namespaceSelector:
    names:
      - s3-resources
```

步驟 4：在映射的命名空間中建立資源

s3-resources 命名空間中的資源會自動使用指定的角色：

```
apiVersion: s3.services.k8s.aws/v1alpha1
kind: Bucket
metadata:
  name: my-bucket
  namespace: s3-resources
spec:
  name: my-production-bucket
```

多帳戶管理

使用 IAM 角色選取器來管理多個 AWS 帳戶的資源。

步驟 1：建立跨帳戶 IAM 角色

在目標帳戶 (444455556666 中，建立信任來源帳戶能力角色的角色：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/ACKCapabilityRole"
      },
      "Action": ["sts:AssumeRole", "sts:TagSession"]
    }
  ]
}
```

將服務特定的許可連接到此角色。

步驟 2：授予 AssumeRole 許可

在來源帳戶 (111122223333 中，允許能力角色擔任目標帳戶角色：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```
      "Effect": "Allow",
      "Action": ["sts:AssumeRole", "sts:TagSession"],
      "Resource": "arn:aws:iam::444455556666:role/ACKTargetAccountRole"
    }
  ]
}
```

步驟 3：建立 IAMRoleSelector

將跨帳戶角色映射至命名空間：

```
apiVersion: services.k8s.aws/v1alpha1
kind: IAMRoleSelector
metadata:
  name: production-account-config
spec:
  arn: arn:aws:iam::444455556666:role/ACKTargetAccountRole
  namespaceSelector:
    names:
      - production
```

步驟 4：建立資源

production 命名空間中的資源會在目標帳戶中建立：

```
apiVersion: s3.services.k8s.aws/v1alpha1
kind: Bucket
metadata:
  name: my-bucket
  namespace: production
spec:
  name: my-cross-account-bucket
```

工作階段標籤

EKS ACK 功能會自動在所有 AWS API 請求上設定工作階段標籤。這些標籤透過識別每個請求的來源來啟用精細存取控制和稽核。

可用的工作階段標籤

ACK 所做的每個 AWS API 呼叫都會包含下列工作階段標籤：

標籤索引鍵	說明
eks:eks-capability-arn	提出請求之 EKS 功能的 ARN
eks:kubernetes-namespace	受管資源的 Kubernetes 命名空間
eks:kubernetes-api-group	資源的 Kubernetes API 群組（例如 s3.services.k8s.aws）

使用工作階段標籤進行存取控制

您可以在 IAM 政策條件下使用這些工作階段標籤，以限制 ACK 可以管理哪些資源。這除了以命名空間為基礎的 IAM 角色選取器之外，還提供額外的安全層。

範例：依命名空間限制

只有在請求來自production命名空間時，才允許 ACK 建立 S3 儲存貯體：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:CreateBucket",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalTag/eks:kubernetes-namespace": "production"
        }
      }
    }
  ]
}
```

範例：依功能限制

僅允許來自特定 ACK 功能的動作：

```
{
```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:*",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalTag/eks:eks-capability-arn": "arn:aws:eks:us-
west-2:111122223333:capability/my-cluster/ack/my-ack"
        }
      }
    }
  ]
}
```

> **Note**
>
> 工作階段標籤與自我管理 ACK 不同，其預設不會設定這些標籤。這可透過 受管功能啟用更精細的存取控制。

進階 IAM 角色選取器模式

如需進階組態，包括跨帳戶角色映射、命名空間範圍角色選擇和其他範例，請參閱 ACK 網站上的精細 IAM 角色。

後續步驟

- the section called “ACK 概念” - 了解 ACK 概念和資源生命週期
- the section called “EKS 功能的考量事項” - 了解 功能的安全最佳實務

EKS 的 ACK 考量事項

本主題涵蓋使用 ACK 的 EKS 功能的重要考量，包括 IAM 組態、多帳戶模式，以及與其他 EKS 功能的整合。

IAM 組態模式

ACK 功能使用 IAM 功能角色進行身分驗證 AWS。根據您的需求選擇正確的 IAM 模式。

簡單：單一功能角色

對於開發、測試或簡單使用案例，請將所有必要的許可直接授予能力角色。

使用時機：

- ACK 入門
- 單一帳戶部署
- 由一個團隊管理的所有資源
- 開發與測試環境

範例：使用資源標記條件將 S3 和 RDS 許可新增至您的能力角色：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["s3:*"],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestedRegion": ["us-west-2", "us-east-1"]
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": ["rds:*"],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestedRegion": ["us-west-2", "us-east-1"],
          "aws:ResourceTag/ManagedBy": "ACK"
        }
      }
    }
  ]
}
```

此範例會將 S3 和 RDS 操作限制在特定區域，並要求 RDS 資源具有ManagedBy： ACK標籤。

生產：IAM 角色選取器

對於生產環境，請使用 IAM 角色選取器實作最低權限存取和命名空間層級隔離。

使用時機：

- 生產環境
- 多團隊叢集
- 多帳戶資源管理
- 最低權限的安全需求
- 不同的服務需要不同的許可

優點：

- 每個命名空間只會取得所需的許可
- 團隊隔離 - 團隊 A 無法使用團隊 B 的許可
- 更輕鬆地稽核和合規
- 跨帳戶資源管理的必要項目

如需詳細的 IAM 角色選取器組態，請參閱 the section called “設定 許可”。

與其他 EKS 功能整合

使用 Argo CD 的 GitOps

使用適用於 Argo CD 的 EKS 功能從 Git 儲存庫部署 ACK 資源，啟用基礎設施管理的 GitOps 工作流程。

考量：

- 將 ACK 資源與end-to-end GitOps 的應用程式資訊清單一起存放
- 根據您的團隊結構，依環境、服務或資源類型進行組織
- 使用 Argo CD 的自動同步進行持續調校
- 啟用剔除以自動移除已刪除的資源
- 考慮多叢集基礎設施管理的hub-and-spoke模式

GitOps 提供稽核追蹤、復原功能和宣告式基礎設施管理。如需 Argo CD 的詳細資訊，請參閱 the section called “使用 Argo CD”。

kro 的資源合成

使用 kro 的 EKS 功能 (Kube Resource Orchestrator)，將多個 ACK 資源組成更高階的抽象和自訂 APIs。

何時搭配 ACK 使用 kro：

- 建立常用基礎設施堆疊的可重複使用模式（資料庫 + 備份 + 監控）
- 為應用程式團隊建置具有簡化 APIs自助式平台
- 管理資源相依性，並在資源之間傳遞值 (S3 儲存貯體 ARN 至 Lambda 函數）
- 標準化跨團隊的基礎設施組態
- 透過隱藏自訂資源後方的實作詳細資訊來降低複雜性

範例模式：

- 應用程式堆疊：S3 儲存貯體 + SQS 佇列 + 通知組態
- 資料庫設定：RDS 執行個體 + 參數群組 + 安全群組 + 秘密
- 網路：VPC + 子網路 + 路由表 + 安全群組

kro 會處理編寫資源的相依性排序、狀態傳播和生命週期管理。如需 kro 的詳細資訊，請參閱 the section called “kro 概念”。

組織您的 資源

使用 Kubernetes 命名空間和資源標籤組織 ACK AWS 資源，以獲得更好的管理、存取控制和成本追蹤。

命名空間組織

使用 Kubernetes 命名空間，以邏輯方式依環境（生產、預備、開發）、團隊（平台、資料、ml) 或應用程式分隔 ACK 資源。

優點：

- 用於存取控制的命名空間範圍 RBAC

- 使用註釋設定每個命名空間的預設區域
- 更輕鬆的資源管理和清理
- 符合組織結構的邏輯分離

資源標記

EKS ACK 功能會自動將預設標籤套用至其建立的所有 AWS 資源。這些標籤與自我管理的 ACK 不同，並提供增強的可追蹤性。

功能套用的預設標籤：

標籤索引鍵	說明
`eks:controller-version`	ACK 控制器的版本
`eks:kubernetes-namespace`	ACK 資源的 Kubernetes 命名空間
`eks:kubernetes-resource-name`	Kubernetes 資源的名稱
`eks:kubernetes-api-group`	Kubernetes API 群組（例如 `s3.services.k8s.aws`）
`eks:eks-capability-arn`	EKS ACK 功能的 ARN

> **Note**
>
> 自我管理 ACK 使用不同的預設標籤：`services.k8s.aws/controller-version`和 `services.k8s.aws/namespace`。功能的標籤使用 `eks:`字首，以與其他 EKS 功能保持一致。

其他建議標籤：

為成本分配、所有權追蹤和組織目的新增自訂標籤：

- 環境（生產、預備、開發）
- 團隊或部門擁有權
- 帳單分配的成本中心
- 應用程式或服務名稱

從其他Infrastructure-as-code工具遷移

許多組織在工作負載協同運作之外，都在 Kubernetes 上尋找標準化的價值。將基礎設施 AWS 和資源管理遷移至 ACK 可讓您使用 Kubernetes APIs 和應用程式工作負載來標準化基礎設施管理。

在 Kubernetes 上標準化基礎設施的優點：

- 單一事實來源：管理 Kubernetes 中的應用程式和基礎設施，實現end-to-end GitOps 實務
- 統一工具：團隊使用 Kubernetes 資源和工具，而不是學習多個工具和架構
- 一致的對帳：ACK 會持續對 Kubernetes 對工作負載所做的 AWS 資源進行對帳，並與必要工具相比，偵測和修正偏離
- 原生組合：使用 kro 和 ACK，直接在應用程式和資源資訊清單中參考 AWS 資源，在資源之間傳遞連線字串和 ARNs
- 簡化的操作：在整個系統中部署、復原和可觀測性的單一控制平面

ACK 支援採用現有 AWS 資源，而無需重新建立這些資源，可從 CloudFormation、Terraform 或叢集外部的資源進行零停機時間遷移。

採用現有資源：

```
apiVersion: s3.services.k8s.aws/v1alpha1
kind: Bucket
metadata:
  name: existing-bucket
  annotations:
    services.k8s.aws/adoption-policy: "adopt-or-create"
spec:
  name: my-existing-bucket-name
```

一旦採用，資源將由 ACK 管理，並且可以透過 Kubernetes 資訊清單進行更新。您可以逐步遷移，視需要採用資源，同時維護其他資源的現有 IaC 工具。

ACK 也支援唯讀資源。對於您希望參考但未修改的其他團隊或工具管理的資源，請將採用與`retain`刪除政策結合，並僅授予讀取 IAM 許可。這可讓應用程式透過 Kubernetes API 探索共用基礎設施 (VPCs、IAM 角色、KMS 金鑰），而不會有修改的風險。 APIs

如需資源採用的詳細資訊，請參閱the section called “ACK 概念”。

刪除政策

當您刪除對應的 AWS Kubernetes 資源時，刪除政策會控制資源會發生的情況。根據資源生命週期和您的操作需求，選擇正確的政策。

刪除 （預設）

AWS 刪除 Kubernetes 資源時會刪除資源。這可維持叢集與 之間的一致性 AWS，確保資源不會累積。

何時使用刪除：

- 清理很重要的開發和測試環境
- 與應用程式生命週期繫結的暫時性資源 （測試資料庫、臨時儲存貯體）
- 不應讓應用程式過期的資源 (SQS 佇列、ElastiCache 叢集）
- 成本最佳化 - 自動清除未使用的資源
- 使用 GitOps 管理的環境，其中從 Git 移除資源應刪除基礎設施

預設刪除政策符合 Kubernetes 的宣告式模型：叢集中的內容符合 中存在的內容 AWS。

保留

當您刪除 Kubernetes 資源時， AWS 資源會保留。這可保護關鍵資料，並允許資源超過其 Kubernetes 表示。

何時使用 保留：

- 生產資料庫具有必須承受叢集變更的關鍵資料
- 具有合規或稽核要求的長期儲存貯體
- 多個應用程式或團隊使用的共用資源
- 要遷移至不同管理工具的資源

- 您希望保留基礎設施的災難復原案例
- 具有複雜相依性的資源需要謹慎解除委任

```
apiVersion: rds.services.k8s.aws/v1alpha1
kind: DBInstance
metadata:
  name: production-db
  annotations:
    services.k8s.aws/deletion-policy: "retain"
spec:
  dbInstanceIdentifier: prod-db
  # ... configuration
```

> **Important**
>
> 保留的資源會持續產生 AWS 成本，且必須在不再需要 AWS 時從 手動刪除。使用資源標記來追蹤保留的資源以進行清除。

如需刪除政策的詳細資訊，請參閱the section called “ACK 概念”。

上游文件

如需使用 ACK 的詳細資訊，請參閱 ACK 網站上的下列頁面：

- 如需建立和管理資源，請參閱 ACK 使用指南。
- 如需所有 服務的完整 API 文件，請參閱 ACK API 參考。
- 如需完整的使用者文件，請參閱 ACK 文件。

後續步驟

- the section called “設定 許可” - 設定 IAM 許可和多帳戶模式
- the section called “ACK 概念” - 了解 ACK 概念和資源生命週期
- the section called “疑難排解” - 故障診斷 ACK 問題
- the section called “使用 Argo CD” - 使用 GitOps 部署 ACK 資源
- the section called “kro 概念” - 將 ACK 資源編寫為更高層級的抽象概念

對 ACK 功能的問題進行故障診斷

> **Note**
>
> EKS 功能是完全受管的，並在叢集外部執行。您無法直接存取控制器命名空間。您可以設定控制器日誌交付，以掌握控制器行為。請參閱 the section called “功能控制器日誌”。故障診斷著重於功能運作狀態、資源狀態和 IAM 組態。

功能為 ACTIVE，但資源尚未建立

如果您的 ACK 功能顯示ACTIVE狀態，但未在其中建立資源 AWS，請檢查功能運作狀態、資源狀態和 IAM 許可。

檢查功能運作狀態：

您可以在 EKS 主控台或使用 AWS CLI 檢視功能運作狀態和狀態問題。

主控台：

1. 在以下網址開啟 Amazon EKS 主控台：https://console.aws.amazon.com/eks/home#/clusters。
2. 選取您的叢集名稱。
3. 選擇可觀測性索引標籤。
4. 選擇監控叢集。
5. 選擇功能索引標籤以檢視所有功能的運作狀態和狀態。

AWS CLI：

```
# View capability status and health
aws eks describe-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-ack

# Look for issues in the health section
```

常見原因：

- IAM 許可遺失：功能角色缺少 AWS 服務的許可
- 命名空間錯誤：在命名空間中建立的資源沒有適當的 IAMRoleSelector
- 無效的資源規格：檢查資源狀態條件是否有驗證錯誤
- API 限流：達到 AWS API 速率限制
- 許可 Webhook：許可 Webhook 封鎖控制器修補資源狀態

檢查資源狀態：

```
# Describe the resource to see conditions and events
kubectl describe bucket my-bucket -n default

# Look for status conditions
kubectl get bucket my-bucket -n default -o jsonpath='{.status.conditions}'

# View resource events
kubectl get events --field-selector involvedObject.name=my-bucket -n default
```

驗證 IAM 許可：

```
# View the Capability Role's policies
aws iam list-attached-role-policies --role-name my-ack-capability-role
aws iam list-role-policies --role-name my-ack-capability-role

# Get specific policy details
aws iam get-role-policy --role-name my-ack-capability-role --policy-name policy-name
```

在 中建立 AWS 但未在 Kubernetes 中顯示的資源

ACK 只會追蹤透過 Kubernetes 資訊清單建立的資源。若要使用 ACK 管理現有 AWS 資源，請使用採用功能。

```
apiVersion: s3.services.k8s.aws/v1alpha1
kind: Bucket
metadata:
  name: existing-bucket
  annotations:
    services.k8s.aws/adoption-policy: "adopt-or-create"
spec:
```

```
  name: my-existing-bucket-name
```

如需資源採用的詳細資訊，請參閱the section called “ACK 概念”。

未建立跨帳戶資源

如果使用 IAM 角色選取器時未在目標 AWS 帳戶中建立資源，請驗證信任關係和 IAMRoleSelector 組態。

驗證信任關係：

```
# Check the trust policy in the target account role
aws iam get-role --role-name cross-account-ack-role --query
 'Role.AssumeRolePolicyDocument'
```

信任政策必須允許來源帳戶的 功能角色擔任該角色。

確認 IAMRoleSelector 組態：

```
# List IAMRoleSelectors (cluster-scoped)
kubectl get iamroleselector

# Describe specific selector
kubectl describe iamroleselector my-selector
```

驗證命名空間對齊：

IAMRoleSelectors 是叢集範圍的資源，但以特定命名空間為目標。確保您的 ACK 資源位於符合 IAMRoleSelector 命名空間選擇器的命名空間中：

```
# Check resource namespace
kubectl get bucket my-cross-account-bucket -n production

# List all IAMRoleSelectors (cluster-scoped)
kubectl get iamroleselector

# Check which namespace the selector targets
kubectl get iamroleselector my-selector -o jsonpath='{.spec.namespaceSelector}'
```

檢查 IAMRoleSelected 條件：

檢查ACK.IAMRoleSelected條件，確認 IAMRoleSelector 已成功符合您的資源：

```
# Check if IAMRoleSelector was matched
kubectl get bucket my-cross-account-bucket -n production -o
 jsonpath='{.status.conditions[?(@.type=="ACK.IAMRoleSelected")]}'
```

如果條件為 False或遺失，IAMRoleSelector 的命名空間選擇器不符合資源的命名空間。確認選取器的 namespaceSelector符合您資源的命名空間標籤。

檢查功能角色許可：

功能角色對目標帳戶角色的需求sts:AssumeRole和sts:TagSession許可：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["sts:AssumeRole", "sts:TagSession"],
      "Resource": "arn:aws:iam::444455556666:role/cross-account-ack-role"
    }
  ]
}
```

如需詳細跨帳戶組態，請參閱 the section called “設定 許可”。

後續步驟

- the section called “考量事項” - ACK 考量事項和最佳實務
- the section called “設定 許可” - 設定 IAM 許可和多帳戶模式
- the section called “ACK 概念” - 了解 ACK 概念和資源生命週期
- the section called “故障診斷功能” - 一般功能故障診斷指引

比較 ACK 與自我管理 ACK 的 EKS 功能

ACK 的 EKS 功能提供與自我管理 ACK 控制器相同的功能，但具有顯著的操作優勢。如需 EKS 功能與自我管理解決方案的一般比較，請參閱 the section called “考量事項”。本主題著重於 ACK 特定的差異。

與上游 ACK 的差異

ACK 的 EKS 功能是以上游 ACK 控制器為基礎，但在 IAM 整合中有所不同。

IAM 功能角色：此功能使用專用 IAM 角色搭配信任政策，允許`capabilities.eks.amazonaws.com`服務主體，而不是 IRSA（服務帳戶的 IAM 角色）。您可以直接將 IAM 政策連接至功能角色，而不需要建立或註釋 Kubernetes 服務帳戶或設定 OIDC 供應商。生產使用案例的最佳實務是使用 設定服務許可`IAMRoleSelector`。如需詳細資訊，請參閱the section called “設定 許可”。

工作階段標籤： 受管功能會自動在所有 AWS API 請求上設定工作階段標籤，啟用精細存取控制和稽核。標籤包括 `eks:eks-capability-arn`、 `eks:kubernetes-namespace`和 `eks:kubernetes-api-group`。這與自我管理 ACK 不同，其預設不會設定這些標籤。如需在 the section called “設定 許可” IAM 政策中使用工作階段標籤的詳細資訊，請參閱 。

資源標籤： 功能會將不同的預設標籤套用至 AWS 資源，而非自我管理的 ACK。此功能使用字`eks:`首標籤（例如 `eks:kubernetes-namespace`、`eks:eks-capability-arn`），而不是自我管理 ACK 使用的`services.k8s.aws/`標籤。如需預設資源標籤的完整清單，the section called “考量事項”請參閱 。

資源相容性：ACK 自訂資源的運作方式與上游 ACK 相同，不會變更您的 ACK 資源 YAML 檔案。該功能使用相同的 Kubernetes APIs 和 CRDs，因此 等工具`kubectl`的運作方式相同。此功能僅支援上游 ACK 中一般可用的控制器和資源 (GA)。此功能不包含位於預覽上游的控制器。控制器的狀態可能會隨著時間從預覽變更為上游 GA，然後 功能可以開始自動管理它。如果您搭配 功能執行自我管理預覽控制器，請在遷移the section called “預覽控制器和自動提升”之前檢閱 。

如需完整的 ACK 文件和服務特定指南，請參閱 ACK 網站上的 ACK 文件。

遷移路徑

您可以將自我管理的 ACK 遷移至受管功能，並將 AWS 資源的中斷降至最低。遷移依賴於 Kubernetes 領導者選擇：自我管理控制器和功能爭奪相同的租用，因此其中只有一個會隨時協調指定的資源。為此，兩者都必須在相同的命名空間中共用租用。此功能不會強制從執行中的自我管理控制器取得租用，因此您可以透過向下擴展自我管理的控制器來控制切換的發生時間。

> **Important**
>
> 開始之前，請授予 IAM 功能角色許可，該許可等同於您的自我管理控制器目前使用的許可。此功能會透過`capabilities.eks.amazonaws.com`服務主體，而不是透過您自我管理控制

器目前使用的機制，例如 IRSA 或 EKS Pod Identity，以專用功能角色進行驗證（請參閱 the section called “設定 許可”）。如果功能角色缺少許可，該功能會採用您的 資源。然後，它無法協調它們並記錄AccessDenied錯誤。

完成下列步驟以進行遷移。這些步驟使用 S3 控制器 (ack-s3-controller) 做為範例。針對您要遷移至 功能的每個自我管理 ACK 控制器重複這些項目，並取代對應的控制器名稱和 Helm Chart。

> **Note**
>
> 搭配 功能執行自我管理的控制器旨在作為遷移期間的暫時狀態，而不是作為長期組態。當兩者都執行時，任一端的中斷（例如功能部署或自我管理控制器的升級）可以釋出租用，並讓另一端取得租用，導致對帳在它們之間意外切換。為每個控制器完成遷移，而不是無限期地隨 功能一起自行管理。

1. 在自我管理 ACK 控制器上啟用領導者選擇，並將其租用移至 kube-system：

```
helm upgrade --install ack-s3-controller \
  oci://public.ecr.aws/aws-controllers-k8s/s3-chart \
  --namespace ack-system \
  --set leaderElection.enabled=true \
  --set leaderElection.namespace=kube-system
```

   您必須設定這兩個值。在 ACK Helm Chart 中，只有在 leaderElection.enabled為時才會套用--leader-election-namespace旗標true，且領導者選擇預設為停用。leaderElection.namespace 單獨設定沒有效果。控制器會在沒有租用的情況下繼續執行，而且兩個控制器都會在建立功能後同時協調相同的資源。這適用於每個 ACK 服務控制器圖表，而不只是 S3。

   這會將控制器的租用移至 kube-system，允許受管功能與其協調。

2. 在叢集上建立 ACK 功能（請參閱 the section called “建立 ACK 功能”）。此功能會啟動並爭奪租用，但自我管理的控制器仍會保留它。您的自我管理控制器會持續協調您的資源，而 功能會等待領導，而不是強制接管。
3. 當您準備好開始遷移時，請將自我管理控制器縮減至零個複本。這會釋出租用，讓 功能可以取得領導權並接管對帳：

```
kubectl scale deployment ack-s3-controller \
```

```
  --namespace ack-system --replicas=0
```

在您縮減自我管理控制器的規模後，此功能通常會在短時間內取得租用並開始對帳。擴展自我管理控制器備份不會傳回租用，因為 功能會繼續保留和續約租用。若要將對帳傳回自我管理控制器，請將其擴展到至少一個複本，然後刪除 ACK 功能。刪除功能後，自我管理控制器會重新取得租用並繼續對帳。

採用時，此功能會套用自己的預設資源標籤 (eks: 字首） 取代自我管理 ACK 使用的services.k8s.aws/標籤（請參閱 the section called “考量事項”)。預期針對您採用的資源進行一次性標記 API 呼叫，並更新services.k8s.aws/標籤字首上索引鍵的任何成本分配或政策工具。

4. 確認功能運作狀態良好，並已接管資源的調校。確認 資源報告 Synced的條件，True且 功能未記錄AccessDenied錯誤。
5. 在您確認功能正確管理資源後，請移除自我管理控制器：

```
helm uninstall ack-s3-controller --namespace ack-system
```

此方法可讓兩個控制器在遷移期間安全地共存。在您發佈租用後， 受管功能會採用先前由自我管理控制器管理的資源，以確保持續對帳而不會發生衝突。

預覽控制器和自動提升

此功能僅支援上游 ACK 中為 GA 的控制器。目前處於預覽狀態的控制器稍後可以提升為上游 GA。發生這種情況時， 功能會開始自動管理該控制器，而不需要採取任何動作。

如果您在叢集上執行也為其他控制器使用 功能的自我管理預覽控制器，這會產生風險。您可以將預覽控制器作為單一複本執行，並停用領導選擇，因為沒有其他控制器可與之協調。當該控制器提升為 GA 時， 功能會開始管理它。此時，兩個對帳程式會在沒有共用租用的相同資源上運作，以協調這些資源。結果是與遷移步驟旨在防止的相同雙重調校衝突。兩個對帳程式都會發出競爭的 AWS API 呼叫，並將衝突的更新寫入自訂資源狀態。

為了避免這種情況，在您執行自我管理預覽控制器以及 功能之前：

- 在自我管理預覽控制器上啟用領導者選擇kube-system，並使用遷移步驟中顯示的相同 leaderElection.enabled=true和 leaderElection.namespace=kube-system設定，將其租用指向 。這可確保如果提升控制器且 功能接管，則兩個座標會透過共用租用而非平行調校。
- 追蹤您依賴的任何預覽控制器的上游 GA 狀態，並計劃在升級時遵循遷移路徑進行遷移。您可以在 ACK 網站上的 ACK 服務頁面上檢查每個控制器的目前狀態。

後續步驟

- the section called “建立 ACK 功能” - 建立 ACK 功能資源
- the section called “ACK 概念” - 了解 ACK 概念和資源生命週期
- the section called “設定 許可” - 設定 IAM 和許可

使用 Argo CD 持續部署

Argo CD 是一種適用於 Kubernetes 的宣告式 GitOps 持續交付工具。使用 Argo CD，您可以自動化跨多個叢集和環境的應用程式部署和生命週期管理。Argo CD 支援多種來源類型，包括 Git 儲存庫、Helm 登錄檔 (HTTP 和 OCI) 和 OCI 映像，為具有不同安全和合規要求的組織提供靈活性。

透過 EKS 功能，Argo CD 完全由 管理 AWS，無需在叢集上安裝、維護和擴展 Argo CD 控制器及其相依性。

Argo CD 的運作方式

Argo CD 遵循 GitOps 模式，其中您的應用程式來源 (Git 儲存庫、Helm 登錄檔或 OCI 映像） 是定義所需應用程式狀態的事實來源。當您建立 Argo CD `Application` 資源時，您可以指定包含應用程式資訊清單的來源，以及目標 Kubernetes 叢集和命名空間。Argo CD 會持續監控叢集中的來源和即時狀態，自動同步任何變更，以確保叢集狀態符合所需的狀態。

> **Note**
>
> 使用適用於 Argo CD 的 EKS 功能，Argo CD 軟體會在 AWS 控制平面中執行，而不是在您的工作者節點上執行。這表示您的工作者節點不需要直接存取 Git 儲存庫或 Helm 登錄檔，此功能會處理來自 AWS 帳戶的來源存取。

Argo CD 提供三種主要資源類型：

- 應用程式：定義從 Git 儲存庫到目標叢集的部署
- ApplicationSet：從多叢集部署的範本產生多個應用程式
- AppProject：提供應用程式的邏輯分組和存取控制

範例：建立 Argo CD 應用程式

下列範例示範如何建立 Argo CD `Application` 資源：

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: guestbook
  namespace: argocd
spec:
  project: default
  source:
    repoURL: https://github.com/argoproj/argocd-example-apps.git
    targetRevision: HEAD
    path: guestbook
  destination:
    name: in-cluster
    namespace: guestbook
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
```

> **Note**
>
> `destination.name` 使用 與您在註冊叢集時使用的叢集名稱 （ 例如`in-cluster`本機叢集的)。`destination.server` 欄位也適用於 EKS 叢集 ARNs，但建議使用叢集名稱以提高可讀性。

Argo CD 的優點

Argo CD 實作 GitOps 工作流程，您可以在 Git 儲存庫中定義應用程式組態，Argo CD 會自動同步您的應用程式以符合所需的狀態。這種以 Git 為中心的方法提供所有變更的完整稽核線索、輕鬆復原，並自然地與您現有的程式碼檢閱和核准程序整合。Argo CD 會自動偵測並協調 Git 中所需狀態與叢集中實際狀態之間的偏離，確保您的部署與您的宣告組態保持一致。

使用 Argo CD，您可以從單一 Argo CD 執行個體跨多個叢集部署和管理應用程式，簡化多叢集和多區域環境中的操作。Argo CD UI 提供視覺化和監控功能，可讓您檢視應用程式的部署狀態、運作狀態和歷史記錄。UI 與 AWS Identity Center （ 先前稱為 AWS SSO) 整合，以實現無縫身分驗證和授權，讓您能夠使用現有的身分管理基礎設施來控制存取。

作為 EKS 受管功能的一部分，Argo CD 完全由 管理 AWS，無需安裝、設定和維護 Argo CD 基礎設施。 會 AWS 處理擴展、修補和操作管理，讓您的團隊專注於應用程式交付，而不是工具維護。

與 AWS Identity Center 整合

EKS 受管功能可在 Argo CD 和 AWS Identity Center 之間直接整合，為您的使用者啟用無縫身分驗證和授權。當您啟用 Argo CD 功能時，您可以設定 AWS Identity Center 整合，將 Identity Center 群組和使用者映射至 Argo CD RBAC 角色，讓您控制誰可以存取和管理 Argo CD 中的應用程式。

與其他 EKS 受管功能整合

Argo CD 與其他 EKS 受管功能整合。

- AWS Kubernetes (ACK) 控制器：使用 Argo CD 管理跨多個叢集的 ACK 資源部署，為您的 AWS 基礎設施啟用 GitOps 工作流程。
- kro (Kube Resource Orchestrator)：使用 Argo CD 跨多個叢集部署 kro 組合，在您的 Kubernetes 資產中實現一致的資源組合。

Argo CD 入門

若要開始使用適用於 Argo CD 的 EKS 功能：

1. 建立並設定具有 Argo CD 必要許可的 IAM 功能角色，以存取您的來源和管理應用程式。
2. 透過 AWS 主控台、 AWS CLI 或您偏好的基礎設施做為程式碼工具，在 EKS 叢集上建立 Argo CD 功能資源。
3. 設定儲存庫存取並註冊叢集以進行應用程式部署。
4. 建立應用程式資源，從您的宣告來源部署應用程式。

建立 Argo CD 功能

本主題說明如何在 Amazon EKS 叢集上建立 Argo CD 功能。

先決條件

建立 Argo CD 功能之前，請確定您有：

- 執行支援 Kubernetes 版本的現有 Amazon EKS 叢集 （支援標準和延伸支援中的所有版本）
- AWS 已設定身分中心 - Argo CD 身分驗證需要 （不支援本機使用者）
- 具有 Argo CD 許可的 IAM 功能角色
- 足夠的 IAM 許可，可在 EKS 叢集上建立功能資源

- `kubectl` 設定為與您的叢集通訊
- （選用）為簡化叢集和儲存庫管理而安裝的 Argo CD CLI
- （適用於 CLI/eksctl) 安裝和設定適當的 CLI 工具

如需建立 IAM 功能角色的說明，請參閱 the section called “功能 IAM 角色”。如需 Identity Center 設定，請參閱 AWS Identity Center 入門。

> **Important**
>
> 您提供的 IAM 功能角色會決定 Argo CD 可存取 AWS 的資源。這包括透過 CodeConnections 和 Secrets Manager 中的秘密存取 Git 儲存庫。如需建立具有最低權限許可之適當角色的指引，請參閱 the section called “功能 IAM 角色”和 the section called “EKS 功能的考量事項”。

選擇您的工具

您可以使用、 AWS 管理主控台 AWS CLI 或 eksctl 建立 Argo CD 功能：

- the section called “主控台” - 使用 主控台進行引導式體驗
- the section called “ AWS CLI” - 使用 AWS CLI 進行指令碼編寫和自動化
- the section called “eksctl” - 使用 eksctl 進行 Kubernetes 原生體驗

當您建立 Argo CD 功能時會發生什麼情況

當您建立 Argo CD 功能時：

1. EKS 在 AWS 控制平面中建立 Argo CD 功能服務
2. 您的叢集中已安裝自訂資源定義 (CRDs)
3. 會自動為您的 IAM 功能角色建立存取項目，其中包含授予基準 Kubernetes 許可的特定功能存取項目政策（請參閱 the section called “EKS 功能的考量事項”)
4. Argo CD 開始觀看其自訂資源（應用程式、ApplicationSets、AppProjects)
5. 功能狀態從 變更為 `CREATING` `ACTIVE`
6. Argo CD UI 可透過其 URL 存取

啟用後，您可以在叢集中建立 Argo CD 應用程式，以從宣告性來源部署。

> **Note**
>
> 自動建立的存取項目不會授予將應用程式部署到叢集的許可。若要部署應用程式，您必須為每個目標叢集設定額外的 Kubernetes RBAC 許可。the section called “註冊叢集” 如需註冊叢集和設定存取的詳細資訊，請參閱 。

後續步驟

建立 Argo CD 功能之後：

- the section called “Argo CD 概念” - 了解 GitOps 原則、同步政策和多叢集模式
- the section called “使用 Argo CD” - 設定儲存庫存取、註冊目標叢集和建立應用程式
- the section called “Argo CD 考量事項” - 探索多叢集架構模式和進階組態

使用主控台建立 Argo CD 功能

使用 AWS 管理主控台在 Amazon EKS 叢集上建立 Argo CD 功能。

先決條件

- AWS 已設定 Identity Center – Argo CD 需要 AWS Identity Center 進行身分驗證。不支援本機使用者。如果您沒有設定 AWS Identity Center，請參閱 AWS Identity Center 入門以建立 Identity Center 執行個體，以及新增使用者和新增群組以建立使用者和群組以進行 Argo CD 存取。
- Identity AWS Center 中至少有一個使用者或群組 – 您必須在 Identity Center 執行個體中設定至少一個使用者或群組，以指派 Argo CD RBAC 角色映射並提供 Argo CD UI 的存取權。

建立 Argo CD 功能

1. 在以下網址開啟 Amazon EKS 主控台：https://console.aws.amazon.com/eks/home#/clusters。
2. 選取您的叢集名稱以開啟叢集詳細資訊頁面。
3. 選擇功能索引標籤。
4. 在左側導覽中，選擇 Argo CD。
5. 選擇建立 Argo CD 功能。
6. 對於 IAM 功能角色：
   - 如果您已經有 IAM 功能角色，請從下拉式清單中選取它

- 如果您需要建立角色，請選擇建立 Argo CD 角色

  這會在新標籤中開啟 IAM 主控台，其中包含預先填入的信任政策和 Secrets Manager 的完整讀取存取權。預設不會新增其他許可，但您可以視需要新增許可。如果您計劃使用 CodeCommit 儲存庫或其他 AWS 服務，請在建立角色之前新增適當的許可。

  建立角色後，返回 EKS 主控台並自動選取角色。

  > **Note**
  >
  > 如果您計劃使用與 AWS Secrets Manager 或 AWS CodeConnections 的選用整合，您將需要將許可新增至角色。如需 IAM 政策範例和組態指引，請參閱 the section called “ AWS Secrets Manager”和 the section called “ AWS CodeConnections”。

7. 設定 AWS Identity Center 整合：
   a. 選取啟用 AWS Identity Center 整合。
   b. 從下拉式清單中選擇您的 Identity Center 執行個體。
   c. 透過將使用者或群組指派給 Argo CD 角色 (ADMIN、EDITOR 或 VIEWER) 來設定 RBAC 的角色映射
8. （選用）設定私有端點：

   根據預設，Argo CD UI 和 API 端點可透過網際網路公開存取。如果您需要限制存取，您可以設定 VPC 端點。對於網路安全需求嚴格的環境，建議使用此選項。

   a. 在建立 功能之前，請在 VPC 中建立 `com.amazonaws.<region>.eks-capabilities`服務的介面 VPC 端點。VPC 端點應該：
      - 與不同可用區域中的子網路建立關聯，以獲得高可用性
      - 擁有安全群組，允許來自需要存取 Argo CD UI 和 API 之網路的傳入 HTTPS （連接埠 443) 流量
      - 如需建立和自訂 VPC 端點的詳細資訊，請參閱《 AWS PrivateLink 指南》中的建立 VPC 端點。
   b. 在 Argo CD 端點存取 - Argo CD 功能建立頁面的選用區段中，選取私有。
   c. 從下拉式清單中選擇您建立的 VPC 端點。

> **Note**
>
> 啟用私有端點時，只能透過 VPC 端點存取 Argo CD UI 和 API。使用者必須連線至 VPC（或對等網路）才能存取 Argo CD 界面。

9. 選擇建立。

功能建立程序開始。

驗證功能是否作用中

1. 在功能索引標籤上，檢視 Argo CD 功能狀態。
2. 等待狀態從 變更為 `CREATING` `ACTIVE`。
3. 啟用後，此功能即可使用。

如需功能狀態和故障診斷的資訊，請參閱 the section called “使用 功能”。

存取 Argo CD UI

啟用此功能後，您可以存取 Argo CD UI：

1. 在 Argo CD 功能頁面上，選擇開啟 Argo CD UI。
2. Argo CD UI 會在新的瀏覽器索引標籤中開啟。
3. 您現在可以透過 UI 建立應用程式和管理部署。

後續步驟

- the section called “使用 Argo CD” - 設定儲存庫、註冊叢集和建立應用程式
- the section called “Argo CD 考量事項” - 多叢集架構和進階組態
- the section called “使用 功能” - 管理您的 Argo CD 功能資源

使用 CLI 建立 Argo CD AWS 功能

使用 CLI 在 Amazon EKS 叢集上建立 Argo CD AWS 功能。此程序會逐步引導您建立 IAM 角色、設定 AWS Identity Center 整合，以及驗證功能是否作用中。

先決條件

- AWS CLI – 版本 2.12.3 或更新版本。若要檢查您的版本，請執行 aws --version。如需詳細資訊，請參閱《 AWS 命令列界面使用者指南》中的安裝。
- **kubectl** ：命令列工具，適用於使用 Kubernetes 叢集。如需詳細資訊，請參閱the section called “設定 kubectl 和 eksctl”。
- AWS 已設定 Identity Center – Argo CD 需要 AWS Identity Center 進行身分驗證。不支援本機使用者。如果您沒有設定 AWS Identity Center，請參閱 AWS Identity Center 入門以建立 Identity Center 執行個體，以及新增使用者和新增群組以建立使用者和群組以進行 Argo CD 存取。
- Identity AWS Center 中至少有一個使用者或群組 – 您必須在 Identity Center 執行個體中設定至少一個使用者或群組，以指派 Argo CD RBAC 角色映射並提供 Argo CD UI 的存取權。

步驟 1：建立 IAM 功能角色

建立信任政策檔案：

```
cat > argocd-trust-policy.json << 'EOF'
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "capabilities.eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ]
    }
  ]
}
EOF
```

建立 IAM 角色：

```
aws iam create-role \
  --role-name ArgoCDCapabilityRole \
  --assume-role-policy-document file://argocd-trust-policy.json
```

> **Note**
>
> 如果您計劃使用與 AWS Secrets Manager 或 AWS CodeConnections 的選用整合，請將許可新增至角色。如需 IAM 政策範例和組態指引，請參閱 the section called “ AWS Secrets Manager”和 the section called “ AWS CodeConnections”。

（選用）設定私有端點

根據預設，Argo CD UI 和 API 端點可透過網際網路公開存取。如果您需要限制存取，您可以設定 VPC 端點。對於網路安全需求嚴格的環境，建議使用此選項。

為 EKS 功能建立 VPC 端點

在 VPC 中為 EKS 功能服務建立介面 VPC 端點。將 *vpc-id*、*netnet-id-1 subnet-id-2*、*sg-id* 和 *region-code* 取代為您自己的值：

```
aws ec2 create-vpc-endpoint \
  --vpc-endpoint-type Interface \
  --service-name com.amazonaws.region-code.eks-capabilities \
  --vpc-id vpc-id \
  --subnet-ids subnet-id-1 subnet-id-2 \
  --security-group-ids sg-id \
  --region region-code
```

> **Note**
>
> - 子網路應位於不同的可用區域中，以實現高可用性。
> - 安全群組必須允許來自需要存取 Argo CD UI 和 API 之網路的傳入 HTTPS （連接埠 443) 流量。
> - 請注意此命令傳回的 VPC 端點 ID，建立 功能時需要它。

驗證 VPC 端點是否可用

```
aws ec2 describe-vpc-endpoints \
  --vpc-endpoint-ids vpce-xxxxxxxx \
  --query 'VpcEndpoints[0].State' \
```

```
  --output text \
  --region region-code
```

請等到狀態顯示available後再繼續。

步驟 2：建立 Argo CD 功能

在叢集上建立 Argo CD 功能資源。

首先，為您的 Identity Center 組態設定環境變數：

```
# Get your Identity Center instance ARN (replace region if your IDC instance is in a
 different region)
export IDC_INSTANCE_ARN=$(aws sso-admin list-instances --region region-code --query
 'Instances[0].InstanceArn' --output text)

# Get a user ID for RBAC mapping (replace with your username and region if needed)
export IDC_USER_ID=$(aws identitystore list-users \
  --region region-code \
  --identity-store-id $(aws sso-admin list-instances --region region-code --query
 'Instances[0].IdentityStoreId' --output text) \
  --query 'Users[?UserName==your-username].UserId' --output text)

echo "IDC_INSTANCE_ARN=$IDC_INSTANCE_ARN"
echo "IDC_USER_ID=$IDC_USER_ID"
```

使用 Identity Center 整合建立 功能。將 *region-code* 取代為您的叢集所在的 AWS 區域、將 *my-cluster* 取代為您的叢集名稱，並將 *idc-region-code* 取代為您的 Identity Center 執行個體所在的區域代碼：

```
aws eks create-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-argocd \
  --type ARGOCD \
  --role-arn arn:aws:iam::$(aws sts get-caller-identity --query Account --output
 text):role/ArgoCDCapabilityRole \
  --delete-propagation-policy RETAIN \
  --configuration '{
    "argoCd": {
      "awsIdc": {
        "idcInstanceArn": "'$IDC_INSTANCE_ARN'",
```

```
        "idcRegion": "'idc-region-code'"
      },
      "rbacRoleMappings": [{
        "role": "ADMIN",
        "identities": [{
          "id": "'$IDC_USER_ID'",
          "type": "SSO_USER"
        }]
      }]
    }
  }'
```

如果您已設定 VPC 端點進行私有存取，請包含 network-configuration 參數以使用私有端點建立功能。將 *vpce-xxxxxxxx* 取代為您的 VPC 端點 ID：

```
aws eks create-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-argocd \
  --type ARGOCD \
  --role-arn arn:aws:iam::$(aws sts get-caller-identity --query Account --output
 text):role/ArgoCDCapabilityRole \
  --delete-propagation-policy RETAIN \
  --network-configuration '{
    "elasticNetworkInterfaces": {
      "vpcEndpointId": "'vpce-xxxxxxxx'"
    }
  }' \
  --configuration '{
    "argoCd": {
      "awsIdc": {
        "idcInstanceArn": "'$IDC_INSTANCE_ARN'",
        "idcRegion": "'idc-region-code'"
      },
      "rbacRoleMappings": [{
        "role": "ADMIN",
        "identities": [{
          "id": "'$IDC_USER_ID'",
          "type": "SSO_USER"
        }]
      }]
    }
  }'
```

命令會立即傳回，但當 EKS 建立所需的功能基礎設施和元件時，功能需要一些時間才會變成作用中。EKS 會在功能建立期間，在您的叢集中安裝 Kubernetes 自訂資源定義 (CRDs)。

> **Note**
>
> 如果您收到叢集不存在或您沒有許可的錯誤，請驗證：
>
> - 叢集名稱正確
> - 您的 AWS CLI 已設定為正確的區域
> - 您擁有必要的 IAM 許可

步驟 3：確認功能處於作用中狀態

等待 功能變成作用中。將 *region-code* 取代為您的叢集所在的 AWS 區域，並將 *my-cluster* 取代為您的叢集名稱。

```
aws eks describe-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-argocd \
  --query 'capability.status' \
  --output text
```

當狀態顯示 時，此功能已就緒ACTIVE。請等到狀態為 ACTIVE，再繼續下一個步驟。

您也可以檢視完整的功能詳細資訊：

```
aws eks describe-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-argocd
```

步驟 4：確認可用的自訂資源

功能處於作用中狀態後，請確認叢集中是否有可用的 Argo CD 自訂資源：

```
kubectl api-resources | grep argoproj.io
```

您應該會看到列出的 ApplicationSet Application和資源類型。

後續步驟

- the section called “使用 Argo CD” - 設定儲存庫、註冊叢集和建立應用程式
- the section called “Argo CD 考量事項” - 多叢集架構和進階組態
- the section called “使用 功能” - 管理您的 Argo CD 功能資源

使用 eksctl 建立 Argo CD 功能

使用 eksctl 在 Amazon EKS 叢集上建立 Argo CD 功能。

> **Note**
>
> 下列步驟需要 eksctl 版本 `0.215.0` 或更新版本。若要檢查您的版本，請執行 `eksctl version`。

先決條件

- AWS 已設定 Identity Center – Argo CD 需要 AWS Identity Center 進行身分驗證。不支援本機使用者。如果您沒有設定 AWS Identity Center，請參閱 AWS Identity Center 入門以建立 Identity Center 執行個體，以及新增使用者和新增群組以建立使用者和群組以進行 Argo CD 存取。
- Identity AWS Center 中至少有一個使用者或群組 – 您必須在 Identity Center 執行個體中設定至少一個使用者或群組，以指派 Argo CD RBAC 角色映射並提供 Argo CD UI 的存取權。

步驟 1：建立 IAM 功能角色

建立信任政策檔案：

```
cat > argocd-trust-policy.json << 'EOF'
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "capabilities.eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
```

```
        "sts:TagSession"
      ]
    }
  ]
}
EOF
```

建立 IAM 角色：

```
aws iam create-role \
  --role-name ArgoCDCapabilityRole \
  --assume-role-policy-document file://argocd-trust-policy.json
```

> **Note**
>
> 對於此基本設定，不需要額外的 IAM 政策。如果您打算將 Secrets Manager 用於儲存庫登入資料或 CodeConnections，則需要將許可新增至角色。如需 IAM 政策範例和組態指引，請參閱 the section called “ AWS Secrets Manager”和 the section called “ AWS CodeConnections”。

步驟 2：取得 AWS Identity Center 組態

取得 RBAC 組態的 Identity Center 執行個體 ARN 和使用者 ID：

```
# Get your Identity Center instance ARN
aws sso-admin list-instances --query 'Instances[0].InstanceArn' --output text

# Get a user ID for admin access (replace 'your-username' with your Identity Center
 username)
aws identitystore list-users \
  --identity-store-id $(aws sso-admin list-instances --query
 'Instances[0].IdentityStoreId' --output text) \
  --query 'Users[?UserName==`your-username`].UserId' --output text
```

請注意這些值 - 您將在下一個步驟中使用這些值。

（選用）設定私有端點

根據預設，Argo CD UI 和 API 端點可透過網際網路公開存取。如果您需要限制存取，您可以設定 VPC 端點。對於網路安全需求嚴格的環境，建議使用此選項。

為 EKS 功能建立 VPC 端點

在 VPC 中為 EKS 功能服務建立介面 VPC 端點。將 *vpc-id*、*netnet-id-1 subnet-id-2*、*sg-id* 和 *region-code* 取代為您自己的值：

```
aws ec2 create-vpc-endpoint \
  --vpc-endpoint-type Interface \
  --service-name com.amazonaws.region-code.eks-capabilities \
  --vpc-id vpc-id \
  --subnet-ids subnet-id-1 subnet-id-2 \
  --security-group-ids sg-id \
  --region region-code
```

Note

- 子網路應位於不同的可用區域中，以實現高可用性。
- 安全群組必須允許來自需要存取 Argo CD UI 和 API 之網路的傳入 HTTPS （連接埠 443) 流量。
- 請注意此命令傳回的 VPC 端點 ID，建立 功能時需要它。

驗證 VPC 端點是否可用

```
aws ec2 describe-vpc-endpoints \
  --vpc-endpoint-ids vpce-xxxxxxxx \
  --query 'VpcEndpoints[0].State' \
  --output text \
  --region region-code
```

請等到狀態顯示`available`後再繼續。

步驟 3：建立 eksctl 組態檔案

建立名為 `argocd-capability.yaml` 且具有下列內容的檔案。將預留位置值取代為您叢集的名稱、叢集的區域、IAM 角色 ARN、身分中心執行個體 ARN、身分中心區域和使用者 ID：

```
apiVersion: eksctl.io/v1alpha5
```

```
kind: ClusterConfig

metadata:
  name: my-cluster
  region: cluster-region-code

capabilities:
  - name: my-argocd
    type: ARGOCD
    roleArn: arn:aws:iam::111122223333:role/ArgoCDCapabilityRole
    deletePropagationPolicy: RETAIN
    configuration:
      argocd:
        awsIdc:
          idcInstanceArn: arn:aws:sso:::instance/ssoins-123abc
          idcRegion: idc-region-code
        rbacRoleMappings:
          - role: ADMIN
            identities:
              - id: 38414300-1041-708a-01af-5422d6091e34
                type: SSO_USER
```

> **Note**
>
> 您可以將多個使用者或群組新增至 RBAC 映射。對於群組，請使用 type: SSO_GROUP並提供群組 ID。可用的角色為 ADMIN、 EDITOR和 VIEWER。

如果您已設定私有存取的 VPC 端點，請將 networkConfiguration區段新增至功能定義。將 *vpce-xxxxxxxx* 取代為您的 VPC 端點 ID：

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: my-cluster
  region: cluster-region-code

capabilities:
  - name: my-argocd
    type: ARGOCD
    roleArn: arn:aws:iam::111122223333:role/ArgoCDCapabilityRole
```

```
    deletePropagationPolicy: RETAIN
    networkConfiguration:
      elasticNetworkInterfaces:
        vpcEndpointId: vpce-xxxxxxxx
    configuration:
      argocd:
        awsIdc:
          idcInstanceArn: arn:aws:sso:::instance/ssoins-123abc
          idcRegion: idc-region-code
        rbacRoleMappings:
          - role: ADMIN
            identities:
              - id: 38414300-1041-708a-01af-5422d6091e34
                type: SSO_USER
```

> **Note**
>
> 啟用私有端點時，只能透過 VPC 端點存取 Argo CD UI 和 API。使用者必須連線至 VPC （或對等網路）才能存取 Argo CD 界面。

步驟 4：建立 Argo CD 功能

套用組態檔案：

```
eksctl create capability -f argocd-capability.yaml
```

命令會立即傳回，但 功能需要一些時間才會變成作用中。

步驟 5：確認功能為作用中

檢查功能狀態。將 *region-code* 取代為您的叢集所在的 AWS 區域，並將 *my-cluster* 取代為您的叢集名稱。

```
eksctl get capability \
  --region region-code \
  --cluster my-cluster \
  --name my-argocd
```

當狀態顯示 時，此功能已就緒ACTIVE。

步驟 6：確認可用的自訂資源

功能處於作用中狀態後，請確認叢集中是否有可用的 Argo CD 自訂資源：

```
kubectl api-resources | grep argoproj.io
```

您應該會看到列出的 `ApplicationSet` `Application`和資源類型。

後續步驟

- the section called “使用 Argo CD” - 了解如何建立和管理 Argo CD 應用程式
- the section called “Argo CD 考量事項” - 設定 SSO 和多叢集存取
- the section called “使用 功能” - 管理您的 Argo CD 功能資源

Argo CD 概念

Argo CD 透過將 Git 視為應用程式部署的單一事實來源來實作 GitOps。本主題會逐步解說實際範例，然後說明使用 Argo CD 的 EKS 功能時，您需要了解的核心概念。

Argo CD 入門

建立 Argo CD 功能後（請參閱 the section called “建立 Argo CD 功能”），您可以開始部署應用程式。此範例會逐步解說註冊叢集和建立應用程式。

步驟 1：設定

註冊您的叢集（必要）

註冊您要部署應用程式的叢集。在此範例中，我們將註冊執行 Argo CD 的相同叢集（您可以使用 名稱`in-cluster`來與大多數 Argo CD 範例相容）：

```
# Get your cluster ARN
CLUSTER_ARN=$(aws eks describe-cluster \
  --name my-cluster \
  --query 'cluster.arn' \
  --output text)

# Register the cluster using Argo CD CLI
argocd cluster add $CLUSTER_ARN \
  --aws-cluster-name $CLUSTER_ARN \
  --name in-cluster \
```

```
  --project default
```

> **Note**
>
> 如需設定 Argo CD CLI 以在 EKS 中使用 Argo CD 功能的詳細資訊，請參閱 the section called “搭配 受管功能使用 Argo CD CLI”。

或者，使用 Kubernetes Secret 註冊叢集（如需詳細資訊the section called “註冊叢集”，請參閱)。

設定儲存庫存取（選用）

此範例使用公有 GitHub 儲存庫，因此不需要儲存庫組態。對於私有儲存庫，請使用 AWS Secrets Manager、CodeConnections 或 Kubernetes Secrets 設定存取權（如需詳細資訊the section called “設定儲存庫”，請參閱)。

對於 AWS 服務 (ECR for Helm Charts、CodeConnections 和 CodeCommit)，您可以直接在應用程式資源中參考它們，而無需建立儲存庫。功能角色必須具有必要的 IAM 許可。如需詳細資訊，請參閱 the section called “設定儲存庫”。

步驟 2：建立應用程式

在 中建立此應用程式資訊清單`my-app.yaml`：

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: guestbook
  namespace: argocd
spec:
  project: default
  source:
    repoURL: https://github.com/argoproj/argocd-example-apps.git
    targetRevision: HEAD
    path: guestbook
  destination:
    name: in-cluster
    namespace: guestbook
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
```

```
    syncOptions:
    - CreateNamespace=true
```

套用應用程式：

```
kubectl apply -f my-app.yaml
```

套用此應用程式後，Argo CD：1。將應用程式從 Git 同步到您的叢集（初始部署）2. 監控 Git 儲存庫的變更 3。自動同步叢集 4 的後續變更。偵測並修正來自所需狀態 5 的任何偏離。在 UI 中提供運作狀態和同步歷史記錄

檢視應用程式狀態：

```
kubectl get application guestbook -n argocd
```

您也可以使用 Argo CD CLI 或 Argo CD UI（可從叢集功能索引標籤下的 EKS 主控台存取）檢視應用程式。

> **Note**
>
> 搭配 受管功能使用 Argo CD CLI 時，請指定命名空間字首為 的應用程式`argocd app get argocd/guestbook`。

> **Note**
>
> 在 中使用叢集名稱 `destination.name`（您在註冊叢集時使用的名稱）。受管功能不支援本機叢集內預設值 (`kubernetes.default.svc`)。

核心概念

GitOps 原則和來源類型

Argo CD 實作 GitOps，其中您的應用程式來源是部署的單一事實來源：

- 宣告 - 使用 YAML 資訊清單、Helm Chart 或 Kustomize 浮水印宣告所需狀態
- 已版本化 - 使用完整的稽核線索追蹤每個變更
- 自動化 - Argo CD 會持續監控來源並自動同步變更

- 自我修復 - 偵測並修正所需叢集狀態與實際叢集狀態之間的偏離

支援的來源類型：

- Git 儲存庫 - GitHub、GitLab、Bitbucket、CodeCommit (HTTPS、SSH 或 CodeConnections)
- Helm 登錄檔 - HTTP 登錄檔（如 `https://aws.github.io/eks-charts`) 和 OCI 登錄檔（如 `public.ecr.aws`)
- OCI 影像 - 包含資訊清單或 Helm Chart 的容器影像（例如 `oci://registry-1.docker.io/user/my-app`)

這種靈活性可讓組織選擇符合其安全和合規要求的來源。例如，限制從叢集存取 Git 的組織可以使用 Helm Chart 或 OCI 映像的 ECR。

如需詳細資訊，請參閱 Argo CD 文件中的應用程式來源。

同步和調校

Argo CD 會持續監控您的來源和叢集，以偵測和修正差異：

1. 輪詢來源以進行變更（預設：每 6 分鐘）
2. 比較所需狀態與叢集狀態
3. 將應用程式標記為 `Synced`或 `OutOfSync`
4. 自動同步變更（如果已設定）或等待手動核准
5. 同步後監控資源運作狀態

使用註釋同步波浪控制資源建立順序：

```
metadata:
  annotations:
    argocd.argoproj.io/sync-wave: "0"  # Default if not specified
```

資源會依波動順序套用（數字越小，包括負數，例如 `-1`)。如果未指定 Wave`0`，則預設為 Wave。這可讓您在部署 (wave `-1`) 之前建立相依性，例如 命名空間 (wave)，再部署 (wave `0`) 服務 (wave 1)。

自我修復會自動還原手動變更：

```
spec:
  syncPolicy:
```

```
    automated:
      selfHeal: true
```

> **Note**
>
> 受管功能使用註釋型資源追蹤（非標籤型），以便與 Kubernetes 慣例和其他工具更相容。

如需同步階段、勾點和進階模式的詳細資訊，請參閱 Argo CD 同步文件。

應用程式運作狀態

Argo CD 會監控應用程式中所有資源的運作狀態：

運作狀態：* 正常運作 - 所有資源如預期執行 * 進行中 - 正在建立或更新的資源 * 降級 - 部分資源運作狀態不佳 (Pod 當機、任務失敗） * 已暫停 - 應用程式刻意暫停 * 遺失 - Git 中定義的資源不存在於叢集中

Argo CD 具有常見 Kubernetes 資源（部署、StatefulSets、任務等）的內建運作狀態檢查，並支援 CRDs的自訂運作狀態檢查。

應用程式運作狀態由其所有資源決定 - 如果任何資源為 `Degraded`，則應用程式為 `Degraded`。

如需詳細資訊，請參閱 Argo CD 文件中的資源運作狀態。

多叢集模式

Argo CD 支援兩種主要部署模式：

Hub-and-spoke - 在部署到多個工作負載叢集的專用管理叢集上執行 Argo CD： * 集中控制和可見性 * 跨所有叢集的一致政策 * 一個要管理的 Argo CD 執行個體 * 控制平面和工作負載之間的明確分離

每個叢集 - 在每個叢集上執行 Argo CD，僅管理該叢集的應用程式：* 叢集分離（一個故障不會影響其他故障） * 簡化聯網（無跨叢集通訊） * 更輕鬆地初始設定（無叢集註冊）

為管理許多叢集的平台團隊選擇hub-and-spoke，或為獨立團隊選擇每個叢集，或當叢集必須完全隔離時。

如需多叢集組態的詳細資訊，請參閱 the section called “Argo CD 考量事項”。

專案

專案提供應用程式的邏輯分組和存取控制：

- 來源限制 - 限制哪些 Git 儲存庫可以使用
- 目的地限制 - 限制哪些叢集和命名空間可以成為目標
- 資源限制 - 限制可以部署哪些 Kubernetes 資源類型
- RBAC 整合 - 將專案映射至 AWS Identity Center 使用者和群組 IDs

應用程式屬於單一專案。如果未指定，他們會使用 `default`專案，該專案預設沒有限制。針對生產用途，請編輯`default`專案以限制存取，並建立具有適當限制的新專案。

如需專案組態和 RBAC 模式，請參閱 the section called “設定 許可”。

同步選項

使用常用選項微調同步行為：

- `CreateNamespace=true` - 自動建立目的地命名空間
- `ServerSideApply=true` - 使用伺服器端套用以獲得更好的衝突解決
- `SkipDryRunOnMissingResource=true` - 當 CRDs尚不存在時略過試轉 （適用於 kro 執行個體）

```
spec:
  syncPolicy:
    syncOptions:
    - CreateNamespace=true
    - ServerSideApply=true
    - SkipDryRunOnMissingResource=true
```

如需同步選項的完整清單，請參閱 Argo CD 同步選項文件。

後續步驟

- the section called “設定儲存庫” - 設定 Git 儲存庫存取
- the section called “註冊叢集” - 註冊目標叢集以進行部署
- the section called “建立應用程式” - 建立您的第一個應用程式
- the section called “Argo CD 考量事項” - EKS 特定模式、Identity Center 整合和多叢集組態
- Argo CD 文件 - 完整的 Argo CD 文件，包括同步勾點、運作狀態檢查和進階模式

設定 Argo CD 許可

Argo CD 受管功能與 AWS Identity Center 整合以進行身分驗證，並使用內建的 RBAC 角色進行授權。本主題說明如何設定使用者和團隊的許可。

許可如何使用 Argo CD

Argo CD 功能使用 AWS Identity Center 進行身分驗證，並提供三個內建的 RBAC 角色進行授權。

當使用者存取 Argo CD 時：

1. 他們使用 AWS Identity Center 進行身分驗證（可聯合到您的公司身分提供者）
2. AWS Identity Center 提供使用者和群組資訊給 Argo CD
3. Argo CD 會根據您的組態，將使用者和群組映射至 RBAC 角色
4. 使用者只會看到他們有權存取的應用程式和資源

內建 RBAC 角色

Argo CD 功能提供三個您映射到 AWS Identity Center 使用者和群組的內建角色。這些是全域範圍的角色，可控制對 Argo CD 資源的存取，例如專案、叢集和儲存庫。

> **Important**
>
> 全域角色控制對 Argo CD 本身的存取，而不是對應用程式等專案範圍的資源的存取。EDITOR 和 VIEWER 使用者預設無法查看或管理應用程式，他們需要專案角色才能存取專案範圍的資源。the section called “專案角色和專案範圍存取” 如需授予應用程式和其他專案範圍資源存取權的詳細資訊，請參閱 。

管理員

完整存取所有 Argo CD 資源和設定：

- 在任何專案中建立、更新和刪除應用程式和 ApplicationSets
- 管理 Argo CD 組態
- 註冊和管理部署目標叢集
- 設定儲存庫存取
- 建立和管理專案

- 檢視所有應用程式狀態和歷史記錄
- 列出和存取所有叢集和儲存庫

編輯器

可以更新專案並設定專案角色，但無法變更全域 Argo CD 設定：

- 更新現有專案（無法建立或刪除專案）
- 設定專案角色和許可
- 檢視 GPG 金鑰和憑證
- 無法變更全域 Argo CD 組態
- 無法直接管理叢集或儲存庫
- 無法查看或管理沒有專案角色的應用程式

檢視器

Argo CD 資源的唯讀存取權：

- 檢視專案組態
- 列出所有專案（包括未指派給使用者的專案）
- 檢視 GPG 金鑰和憑證
- 無法列出叢集或儲存庫
- 無法進行任何變更
- 無法查看或管理沒有專案角色的應用程式

> **Note**
>
> 若要授予 EDITOR 或 VIEWER 使用者對應用程式的存取權，ADMIN 或 EDITOR 必須建立專案角色，將 Identity Center 群組映射至專案中的特定許可。

專案角色和專案範圍存取

全域角色 (ADMIN、EDITOR、VIEWER) 控制對 Argo CD 本身的存取。專案角色控制對特定專案中資源和功能的存取，包括：

- 資源：Application、ApplicationSets、儲存庫登入資料、叢集登入資料
- 功能：日誌存取、執行應用程式 Pod 的存取

了解兩層許可模型：

- 全域範圍：內建角色決定使用者可以對專案、叢集、儲存庫和 Argo CD 設定執行的操作
- 專案範圍：專案角色決定使用者可以對特定專案中的資源和功能執行的動作

這表示：

- ADMIN 使用者可以存取所有專案資源和功能，無需額外的組態
- 必須授予 EDITOR 和 VIEWER 使用者專案角色，才能存取專案資源和功能
- EDITOR 使用者可以建立專案角色，以授予自己和其他人可以更新的專案存取權

工作流程範例：

1. ADMIN 會將 Identity Center 群組映射至全域的 EDITOR 角色
2. ADMIN 會為團隊建立專案
3. EDITOR 會設定該專案中的專案角色，以授予團隊成員對專案範圍資源的存取權
4. 團隊成員（可能有 VIEWER 全域角色）現在可以根據其專案角色許可來查看和管理該專案中的應用程式

如需設定專案角色的詳細資訊，請參閱 the section called “專案型存取控制”。

設定角色映射

在建立或更新功能時，將 AWS Identity Center 使用者和群組映射至 Argo CD 角色。

範例角色映射：

```
{
  "rbacRoleMappings": {
    "ADMIN": ["AdminGroup", "alice@example.com"],
    "EDITOR": ["DeveloperGroup", "DevOpsTeam"],
    "VIEWER": ["ReadOnlyGroup", "bob@example.com"]
  }
}
```

> **Note**
>
> 角色名稱區分大小寫，且必須為大寫 (ADMIN、EDITOR、VIEWER)。

> **Important**
>
> EKS 功能與 AWS Identity Center 整合支援每個 Argo CD 功能最多 1，000 個身分。身分可以是使用者或群組。

更新角色映射：

```
aws eks update-capability \
  --region us-east-1 \
  --cluster-name cluster \
  --capability-name capname \
  --role-arn "arn:aws:iam::111122223333:role/EKSCapabilityRole" \
  --configuration '{
    "argoCd": {
      "rbacRoleMappings": {
        "addOrUpdateRoleMappings": [
          {
            "role": "ADMIN",
            "identities": [
              { "id": "686103e0-f051-7068-b225-e6392b959d9e", "type": "SSO_USER" }
            ]
          }
        ]
      }
    }
  }'
```

管理員帳戶用量

管理員帳戶專為初始設定和管理任務而設計，例如註冊叢集和設定儲存庫。

當管理員帳戶適當時：

- 初始功能設定和組態
- Solo 開發或快速示範

- 管理任務（叢集註冊、儲存庫組態、專案建立）

管理員帳戶的最佳實務：

- 請勿將帳戶字符遞交至版本控制
- 如果權杖公開，請立即輪換權杖
- 將帳戶字符用量限制為設定和管理任務
- 設定短過期時間（最長 12 小時）
- 任何指定時間只能建立 5 個帳戶字符

改用專案型存取的時機：

- 與多個使用者共用開發環境
- 任何類似生產的環境
- 當您需要執行動作之人員的稽核線索時
- 當您需要強制執行資源限制或存取界限時

對於生產環境和多使用者案例，請使用專案型存取控制搭配映射至 AWS Identity Center 群組的專用 RBAC 角色。

專案型存取控制

使用 Argo CD Projects (AppProject) 為團隊提供精細的存取控制和資源隔離。

> **Important**
>
> 將使用者或群組指派給專案特定角色之前，您必須先在功能組態中將使用者或群組映射至全域 Argo CD 角色 (ADMIN、EDITOR 或 VIEWER)。使用者在沒有全域角色映射的情況下無法存取 Argo CD，即使他們已指派給專案角色。
>
> 考慮將使用者映射至全域 VIEWER 角色，然後透過專案特定角色授予其他許可。這可提供基準存取，同時允許在專案層級進行精細控制。

專案提供：

- 來源限制：限制哪些 Git 儲存庫可以使用

- 目的地限制：限制哪些叢集和命名空間可以成為目標
- 資源限制：限制可以部署哪些 Kubernetes 資源類型
- RBAC 整合：將專案映射至 AWS Identity Center 群組或 Argo CD 角色

團隊隔離的範例專案：

```
apiVersion: argoproj.io/v1alpha1
kind: AppProject
metadata:
  name: team-a
  namespace: argocd
spec:
  description: Team A applications

  # Required: Specify which namespaces this project watches for Applications
  sourceNamespaces:
  - argocd

  # Source restrictions
  sourceRepos:
  - https://github.com/myorg/team-a-apps

  # Destination restrictions
  destinations:
  - namespace: team-a-*
    server: arn:aws:eks:us-west-2:111122223333:cluster/production

  # Resource restrictions
  clusterResourceWhitelist:
  - group: ''
    kind: Namespace
  namespaceResourceWhitelist:
  - group: 'apps'
    kind: Deployment
  - group: ''
    kind: Service
  - group: ''
    kind: ConfigMap
```

來源命名空間

使用 EKS Argo CD 功能時，AppProject 定義中需要 `spec.sourceNamespaces` 欄位。此欄位指定哪些命名空間可以包含參考此專案的應用程式或 ApplicationSets。

> **Important**
>
> EKS Argo CD 功能僅支援應用程式和 ApplicationSets 的單一命名空間，這是您在建立功能時指定的命名空間（通常是 `argocd`）。這與支援多個命名空間的開放原始碼 Argo CD 不同。

AppProject 組態

所有 AppProjects都必須在 中包含 功能的已設定命名空間`sourceNamespaces`：

```
apiVersion: argoproj.io/v1alpha1
kind: AppProject
metadata:
  name: team-a-project
  namespace: argocd
spec:
  description: Applications for Team A

  # Required: Specify the capability's configured namespace
 (configuration.argoCd.namespace)
  sourceNamespaces:
    - argocd  # Must match your capability's namespace configuration

  # Source repositories this project can deploy from
  sourceRepos:
    - 'https://github.com/my-org/team-a-*'

  # Destination restrictions
  destinations:
    - namespace: 'team-a-*'
      server: arn:aws:eks:us-west-2:111122223333:cluster/my-cluster
```

> **Note**
>
> 如果您從 省略功能的命名空間`sourceNamespaces`，則該命名空間中的應用程式或 ApplicationSets 無法參考此專案，導致部署失敗。

將使用者指派給專案：

專案角色授予 EDITOR 和 VIEWER 使用者對專案資源（應用程式、ApplicationSets、儲存庫和叢集登入資料）和 功能（日誌、Exec) 的存取權。如果沒有專案角色，這些使用者就無法存取這些資源，即使他們具有全域角色存取權。

ADMIN 使用者可以存取所有應用程式，而不需要專案角色。

範例：將應用程式存取權授予團隊成員

```
apiVersion: argoproj.io/v1alpha1
kind: AppProject
metadata:
  name: team-a
  namespace: argocd
spec:
  # ... project configuration ...

  sourceNamespaces:
  - argocd

  # Project roles grant Application-level access
  roles:
  - name: developer
    description: Team A developers - can manage Applications
    policies:
    - p, proj:team-a:developer, applications, *, team-a/*, allow
    - p, proj:team-a:developer, clusters, get, *, allow  # See cluster names in UI
    groups:
    - 686103e0-f051-7068-b225-e6392b959d9e  # Identity Center group ID

  - name: viewer
    description: Team A viewers - read-only Application access
    policies:
    - p, proj:team-a:viewer, applications, get, team-a/*, allow
    - p, proj:team-a:viewer, clusters, get, *, allow  # See cluster names in UI
    groups:
    - 786203e0-f051-7068-b225-e6392b959d9f  # Identity Center group ID
```

> **Note**
>
> 包含在專案角色`clusters, get, *, allow`中，以允許使用者在 UI 中查看叢集名稱。如果沒有此許可，目的地叢集會顯示為「未知」。

了解專案角色政策：

政策格式為：`p, proj:<project>:<role>, <resource>, <action>, <object>, <allow/deny>`

資源政策：

- `applications, , team-a/, allow` - 團隊專案中所有應用程式的完整存取權
- `applications, get, team-a/*, allow` - 應用程式的唯讀存取
- `applications, sync, team-a/*, allow` - 可以同步應用程式，但無法建立/刪除
- `applications, delete, team-a/*, allow` - 可以刪除應用程式（請謹慎使用）
- `applicationsets, , team-a/, allow` - 完整存取 ApplicationSets
- `repositories, *, *, allow` - 存取儲存庫登入資料
- `clusters, *, *, allow` - 存取叢集登入資料

功能政策：

- `logs, , team-a/, allow` - 存取應用程式日誌
- `exec, , team-a/, allow` - 執行應用程式 Pod 的存取

> **Note**
>
> EDITOR 使用者可以建立專案角色，以授予自己和可以更新之專案中的其他許可。這可讓團隊主管控制團隊對專案範圍資源的存取，而不需要 ADMIN 介入。

> **Note**
>
> 在 `groups` 欄位中使用 Identity Center IDs （而非群組名稱）。您也可以使用 Identity Center 使用者 IDs進行個別使用者存取。在 AWS Identity Center 主控台或使用 CLI AWS 尋找這些 IDs。

常見許可模式

模式 1：具有完整存取權的管理團隊

```
{
  "rbacRoleMappings": {
    "ADMIN": ["PlatformTeam", "SRETeam"]
  }
}
```

ADMIN 使用者可以查看和管理所有專案範圍的資源，而無需額外的組態。

模式 2：團隊領導管理專案、開發人員透過專案角色存取

```
{
  "rbacRoleMappings": {
    "ADMIN": ["PlatformTeam"],
    "EDITOR": ["TeamLeads"],
    "VIEWER": ["AllDevelopers"]
  }
}
```

1. ADMIN 會為每個團隊建立專案
2. 團隊領導 (EDITOR) 設定專案角色，授予開發人員對專案資源 （應用程式、ApplicationSets、登入資料） 和功能 （日誌、執行） 的存取權
3. 開發人員 (VIEWER) 只能存取其專案角色允許的資源和功能

模式 3：具有專案角色的團隊型存取

1. ADMIN 建立專案並將團隊領導映射至全球 EDITOR 角色
2. 團隊負責人 (EDITOR) 將團隊成員指派給其專案中的專案角色
3. 團隊成員只需要 VIEWER 全域角色 - 專案角色提供專案資源和功能的存取權

```
{
  "rbacRoleMappings": {
    "ADMIN": ["PlatformTeam"],
    "EDITOR": ["TeamLeads"],
    "VIEWER": ["AllDevelopers"]
  }
}
```

最佳實務

使用群組而非個別使用者：將 AWS Identity Center 群組映射至 Argo CD 角色，而非個別使用者，以便於管理。

從最低權限開始：從 VIEWER 存取開始，並視需要授予 EDITOR 或 ADMIN。

使用專案進行團隊隔離：為不同的團隊或環境建立單獨的 AppProjects以強制執行界限。

利用 Identity Center 聯合：設定 AWS Identity Center 與您的公司身分提供者聯合，以進行集中式使用者管理。

定期存取檢閱：定期檢閱角色映射和專案指派，以確保適當的存取層級。

限制叢集存取：請記住，Argo CD RBAC 會控制對 Argo CD 資源和操作的存取，但不對應至 Kubernetes RBAC。具有 Argo CD 存取權的使用者可以將應用程式部署到 Argo CD 可存取的叢集。限制 Argo CD 可存取的叢集，並使用專案目的地限制來控制應用程式可部署的位置。

AWS 服務許可

若要直接在應用程式資源中使用 AWS 服務 （無需建立儲存庫資源），請將所需的 IAM 許可連接到功能角色。

Helm Chart 的 ECR：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecr:GetAuthorizationToken",
        "ecr:BatchCheckLayerAvailability",
        "ecr:GetDownloadUrlForLayer",
```

```
        "ecr:BatchGetImage"
      ],
      "Resource": "*"
    }
  ]
}
```

CodeCommit 儲存庫：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codecommit:GitPull"
      ],
      "Resource": "arn:aws:codecommit:region:account-id:repository-name"
    }
  ]
}
```

CodeConnections (GitHub、GitLab、Bitbucket)：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codeconnections:UseConnection"
      ],
      "Resource": "arn:aws:codeconnections:region:account-id:connection/connection-id"
    }
  ]
}
```

如需使用這些整合的詳細資訊the section called “設定儲存庫”，請參閱 。

後續步驟

- the section called “使用 Argo CD” - 了解如何建立應用程式和管理部署

- the section called “Argo CD 概念” - 了解 Argo CD 概念，包括專案
- the section called “EKS 功能的考量事項” - 檢閱 功能的安全最佳實務

使用 Argo CD

使用 Argo CD，您可以在 Git 儲存庫中定義應用程式，Argo CD 會自動將其同步到 Kubernetes 叢集。這可透過自動偏離偵測來啟用宣告式、版本控制的應用程式部署。

先決條件

在使用 Argo CD 之前，您需要：

- 建立 Argo CD 功能的 EKS 叢集 （ 請參閱 the section called “建立 Argo CD 功能”)
- 包含 Kubernetes 資訊清單的 Git 儲存庫
- `kubectl` 設定為與您的叢集通訊

一般任務

下列主題會引導您完成常見的 Argo CD 任務：

the section called “設定儲存庫” - 設定 Argo CD 以使用 AWS Secrets Manager、 AWS CodeConnections 或 Kubernetes Secrets 存取您的 Git 儲存庫。

the section called “註冊叢集” - 註冊 Argo CD 將部署應用程式的目標叢集。

the section called “專案” - 使用適用於多租戶環境的專案來組織應用程式並強制執行安全界限。

the section called “組態設定” - 使用 ConfigMap `argocd-cm` 設定 Argo CD 行為，包括自訂資源的自訂運作狀態檢查。

the section called “建立應用程式” - 使用自動或手動同步政策，建立從 Git 儲存庫部署的應用程式。

the section called “ApplicationSets” - 使用 ApplicationSets，使用 範本和產生器在多個環境或叢集中部署應用程式。

存取 Argo CD UI

透過 EKS 主控台存取 Argo CD UI：

1. 開啟 Amazon EKS 主控台

2. 選取您的叢集
3. 選擇功能索引標籤
4. 選擇 Argo CD
5. 選擇開啟 Argo CD UI

UI 提供視覺化應用程式拓撲、同步狀態和歷史記錄、資源運作狀態和事件、手動同步控制和應用程式管理。

Argo CD 端點 URL

當您建立 Argo CD 功能時，您會同時收到 Argo CD UI 和 API 的唯一端點。端點主機名稱是確定性的：它使用您已知道的值，例如您的功能名稱、叢集名稱、 AWS 帳戶 ID 和 AWS 區域。如果您刪除功能並使用相同的叢集名稱和功能名稱重新建立它，您會取得相同的端點。由於主機名稱是可預測的，您可以事先建構它。例如，您可以在功能完成佈建之前預先設定 DNS、自動化或 Git Webhook 整合。對於任何功能， `serverUrl`的值`describe-capability`是最終端點。

> **Note**
>
> 由於端點是重複使用的，指向先前功能的整合 - Git Webhook、字符和 DNS 記錄 - 會自動連接到新的功能，而不表示執行個體已變更。重新建立功能後，請重新驗證這些整合，並輪換與先前執行個體共用的任何 Webhook 秘密或字符。

擷取端點

若要取得功能的確切端點，請使用 `describe-capability`命令並讀取 `serverUrl`值。將 *region-code*、*my-cluster* 和 *my-argocd* 取代為您的值：

```
aws eks describe-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-argocd \
  --query 'capability.configuration.argoCd.serverUrl' \
  --output text
```

> **Note**
>
> 欄位`serverUrl`和端點的 DNS 記錄會在功能狀態變更為 後提供`ACTIVE`。

端點格式

Argo CD 端點使用以下格式：

```
https://capability-name-hash-account-id.eks-capabilities.region-code.amazonaws.com
```

主機名稱包含下列元件：

- *capability-name* – Argo CD 功能的名稱，標準化為小寫，每個底線 (_) 以連字號 () 取代-。如果標準化名稱超過 41 個字元，則會截斷為 41 個字元。
- *hash* – 使主機名稱唯一的 8 個字元十六進位值。這是功能名稱和叢集名稱 SHA-256 雜湊的前 8 個字元。名稱會以正斜線 () 聯結 *capability-name/cluster-name* 。使用您在建立 功能時指定的確切名稱。此值僅提供唯一性，並非安全性控制項。
- *account-id* – 您的 12 位數 AWS 帳戶 ID。
- eks-capabilities.*region-code*.amazonaws.com – 您 AWS 區域的 EKS 功能服務網域。

例如，111122223333在 us-west-2 區域中 my-cluster帳戶中叢集my-argocd上名為 的功能具有類似如下的端點：

```
https://my-argocd-dc855fdf-111122223333.eks-capabilities.us-west-2.amazonaws.com
```

主機名稱使用標準化功能名稱，但雜湊使用您指定的原始名稱。例如，叢集Payments_GitOps上的 功能會在主機名稱payments-gitops中my-cluster標準化為 ，其雜湊來自原始字串 Payments_GitOps/my-cluster：

```
https://payments-gitops-4ce0b382-111122223333.eks-capabilities.us-west-2.amazonaws.com
```

> **Note**
>
> 使用describe-capability輸出來確認 AWS 區域的確切端點。

您可以自行計算##元件。sha256sum 在 Linux 或 macOS shasum -a 256上使用 。下列命令會在叢集 my-argocd上使用範例功能my-cluster。取代您在建立 功能時指定的確切名稱：

```
# Linux
printf '%s' "my-argocd/my-cluster" | sha256sum | cut -c1-8
```

```
# macOS
printf '%s' "my-argocd/my-cluster" | shasum -a 256 | cut -c1-8
```

這兩個命令都會傳回 `dc855fdf`。

端點存取和安全性

端點主機名稱不是秘密，而且知道它不會授予對 Argo CD 執行個體的存取權。主機名稱會公開已在功能 Amazon Resource Name (ARN) 中出現的相同非秘密資訊。`create-capability`、 `describe-capability`和 `list-capabilities`操作會傳回此 ARN. AWS Identity Center 身分驗證和 Argo CD 角色型存取控制 (RBAC) 控制對 Argo CD UI 和 API 的存取。

如果您設定私有端點存取，主機名稱只會解析為與您的 VPC 端點相關聯的私有 IP 地址。您只能從 VPC 或連線網路存取端點。如需詳細資訊，請參閱the section called “私有端點存取”。

上游文件

如需 Argo CD 功能的詳細資訊：

- Argo CD 文件 - 完整使用者指南
- Application Spec - 完整應用程式 API 參考
- ApplicationSet 指南 - ApplicationSet 模式和範例
- Argo CD GitHub - 原始程式碼和範例

設定儲存庫存取

部署應用程式之前，請設定 Argo CD 以存取您的 Git 儲存庫和 Helm Chart 登錄檔。Argo CD 支援 GitHub、GitLab、Bitbucket、 AWS CodeCommit 和 AWS ECR 的多種身分驗證方法。

> **Note**
>
> 對於直接 AWS 服務整合 (ECR Helm Chart、CodeCommit 儲存庫和 CodeConnections)，您可以直接在應用程式資源中參考它們，而無需建立儲存庫組態。功能角色必須具有必要的 IAM 許可。如需詳細資訊，請參閱 the section called “設定 許可”。

先決條件

- 建立 Argo CD 功能的 EKS 叢集

- 包含 Kubernetes 資訊清單的 Git 儲存庫
- `kubectl` 設定為與您的叢集通訊

> **Note**
>
> AWS CodeConnections 可以連線至位於 AWS 雲端或內部部署的 Git 伺服器。如需詳細資訊，請參閱 AWS CodeConnections。

身分驗證方法

Method	使用案例	所需的 IAM 許可
直接整合 AWS 服務		
CodeCommit：	直接與 AWS CodeCommit Git 儲存庫整合。不需要儲存庫組態。	`codecommit:GitPull`
CodeConnections	使用受管身分驗證連線至 GitHub、GitLab 或 Bitbucket。需要連線設定。	`codeconnections:UseConnection`
ECR OCI 成品	直接整合 OCI Helm Chart 和資訊清單映像的 AWS ECR。不需要儲存庫組態。	`arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryPullOnly`
具有登入資料的儲存庫組態		
AWS Secrets Manager（使用者名稱/權杖）	存放個人存取字符或密碼。在不使用 Kubernetes 存取的情況下啟用登入資料輪換。	`arn:aws:iam::aws:policy/AWSSecretsManagerClientReadOnlyAccess`
AWS Secrets Manager (SSH 金鑰）	使用 SSH 金鑰身分驗證。在不使用 Kubernetes 存取的情況下啟用登入資料輪換。	`arn:aws:iam::aws:policy/AWSSecretsManagerClientReadOnlyAccess`
AWS Secrets Manager (GitHub 應用程式）	使用私有金鑰進行 GitHub 應用程式身分驗證。在不使用 Kubernetes 存取的情況下啟用登入資料輪換。	`arn:aws:iam::aws:policy/AWSSecretsManagerClientReadOnlyAccess`

Method	使用案例	所需的 IAM 許可
Kubernetes 秘密	使用叢集內秘密的標準 Argo CD 方法	無 (EKS Access Entry 使用 Kubernetes RBAC 處理的許可)

直接存取 AWS 服務

對於 AWS 服務，您可以直接在應用程式資源中參考它們，而無需建立儲存庫組態。功能角色必須具有必要的 IAM 許可。

CodeCommit 儲存庫

直接在應用程式中參考 CodeCommit 儲存庫：

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: my-app
  namespace: argocd
spec:
  source:
    repoURL: https://git-codecommit.region.amazonaws.com/v1/repos/repository-name
    targetRevision: main
    path: kubernetes/manifests
```

必要的功能角色許可：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "codecommit:GitPull",
      "Resource": "arn:aws:codecommit:region:account-id:repository-name"
    }
  ]
}
```

CodeConnections

透過 CodeConnections 參考 GitHub、GitLab 或 Bitbucket 儲存庫。儲存庫 URL 格式衍生自 CodeConnections 連線 ARN。

儲存庫 URL 格式為：

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: my-app
  namespace: argocd
spec:
  source:
    repoURL: https://codeconnections.region.amazonaws.com/git-http/account-
id/region/connection-id/owner/repository.git
    targetRevision: main
    path: kubernetes/manifests
```

必要的功能角色許可：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "codeconnections:UseConnection",
      "Resource": "arn:aws:codeconnections:region:account-id:connection/connection-id"
    }
  ]
}
```

ECR Helm Chart

ECR 會將 Helm Chart 儲存為 OCI 成品。Argo CD 支援兩種參考方式：

Helm 格式 （ 建議用於 Helm Chart)：

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
```

```
  name: my-app-helm
  namespace: argocd
spec:
  source:
    repoURL: account-id.dkr.ecr.region.amazonaws.com/repository-name
    targetRevision: chart-version
    chart: chart-name
    helm:
      valueFiles:
        - values.yaml
```

注意：使用 Helm 格式時，請勿包含 `oci://` 字首。使用 `chart` 欄位來指定圖表名稱。

OCI 格式（適用於具有 Kubernetes 資訊清單的 OCI 成品）：

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: my-app-oci
  namespace: argocd
spec:
  source:
    repoURL: oci://account-id.dkr.ecr.region.amazonaws.com/repository-name
    targetRevision: artifact-version
    path: path-to-manifests
```

注意：使用 OCI 格式時包含字`oci://`首。使用 `path` 欄位而非 `chart`。

必要的功能角色許可 - 連接 受管政策：

```
arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryPullOnly
```

此政策包含必要的 ECR 許可：`ecr:GetAuthorizationToken`、 `ecr:BatchGetImage`和 `ecr:GetDownloadUrlForLayer`。

使用 AWS Secrets Manager

將儲存庫登入資料儲存在 Secrets Manager 中，並在 Argo CD 儲存庫組態中參考它們。使用 Secrets Manager 可啟用自動憑證輪換，而不需要 Kubernetes RBAC 存取 - 憑證可以使用 Secrets Manager 的 IAM 許可輪換，Argo CD 會自動讀取更新的值。

> **Note**
>
> 對於跨多個儲存庫重複使用登入資料（例如，GitHub 組織下的所有儲存庫），請使用儲存庫登入資料範本搭配 `argocd.argoproj.io/secret-type: repo-creds`。這可提供比建立個別儲存庫秘密更好的 UX。如需詳細資訊，請參閱 Argo CD 文件中的儲存庫登入資料。

使用者名稱和字符身分驗證

對於具有個人存取字符或密碼的 HTTPS 儲存庫：

在 Secrets Manager 中建立秘密：

```
aws secretsmanager create-secret \
  --name argocd/my-repo \
  --description "GitHub credentials for Argo CD" \
  --secret-string '{"username":"your-username","token":"your-personal-access-token"}'
```

選用的 TLS 用戶端憑證欄位（適用於私有 Git 伺服器）：

```
aws secretsmanager create-secret \
  --name argocd/my-private-repo \
  --secret-string '{
    "username":"your-username",
    "token":"your-token",

 "tlsClientCertData":"LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCi4uLgotLS0tLUVORCBDRVJUSUZJQ0FURS0tL

 "tlsClientCertKey":"LS0tLS1CRUdJTiBQUklWQVRFIEtFWS0tLS0tCi4uLgotLS0tLUVORCBQUklWQVRFIEtFWS0tLS
  }'
```

> **Note**
>
> `tlsClientCertData` 和 `tlsClientCertKey`值必須為 base64 編碼。

建立參考 Secrets Manager 的儲存庫秘密：

```
apiVersion: v1
kind: Secret
metadata:
```

```
  name: my-repo
  namespace: argocd
  labels:
    argocd.argoproj.io/secret-type: repository
stringData:
  type: git
  url: https://github.com/your-org/your-repo
  secretArn: arn:aws:secretsmanager:us-west-2:111122223333:secret:argocd/my-repo-AbCdEf
  project: default
```

SSH 金鑰身分驗證

對於以 SSH 為基礎的 Git 存取，請將私有金鑰儲存為純文字 （ 而非 JSON) ：

使用 SSH 私有金鑰建立秘密：

```
aws secretsmanager create-secret \
  --name argocd/my-repo-ssh \
  --description "SSH key for Argo CD" \
  --secret-string "-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAABlwAAAAdzc2gtcn
...
-----END OPENSSH PRIVATE KEY-----"
```

建立 SSH 的儲存庫秘密：

```
apiVersion: v1
kind: Secret
metadata:
  name: my-repo-ssh
  namespace: argocd
  labels:
    argocd.argoproj.io/secret-type: repository
stringData:
  type: git
  url: git@github.com:your-org/your-repo.git
  secretArn: arn:aws:secretsmanager:us-west-2:111122223333:secret:argocd/my-repo-ssh-
AbCdEf
  project: default
```

GitHub 應用程式身分驗證

對於使用私有金鑰的 GitHub 應用程式身分驗證：

使用 GitHub 應用程式登入資料建立秘密：

```
aws secretsmanager create-secret \
  --name argocd/github-app \
  --description "GitHub App credentials for Argo CD" \
  --secret-string '{

 "githubAppPrivateKeySecret":"LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVktLS0tLQouLi4KLS0tLS1FTkQgUlNBI
    "githubAppID":"123456",
    "githubAppInstallationID":"12345678"
  }'
```

> **Note**
>
> `githubAppPrivateKeySecret` 值必須為 base64 編碼。

GitHub Enterprise 的選用欄位：

```
aws secretsmanager create-secret \
  --name argocd/github-enterprise-app \
  --secret-string '{

 "githubAppPrivateKeySecret":"LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVktLS0tLQouLi4KLS0tLS1FTkQgUlNBI
    "githubAppID":"123456",
    "githubAppInstallationID":"12345678",
    "githubAppEnterpriseBaseUrl":"https://github.example.com/api/v3"
  }'
```

為 GitHub 應用程式建立儲存庫秘密：

```
apiVersion: v1
kind: Secret
metadata:
  name: my-repo-github-app
  namespace: argocd
  labels:
    argocd.argoproj.io/secret-type: repository
stringData:
  type: git
  url: https://github.com/your-org/your-repo
```

```
  secretArn: arn:aws:secretsmanager:us-west-2:111122223333:secret:argocd/github-app-
AbCdEf
  project: default
```

儲存庫登入資料範本

對於跨多個儲存庫重複使用登入資料（例如，GitHub 組織或使用者下的所有儲存庫），請使用儲存庫登入資料範本搭配 `argocd.argoproj.io/secret-type: repo-creds`。這可提供比為每個儲存庫建立個別儲存庫秘密更好的 UX。

建立儲存庫登入資料範本：

```
apiVersion: v1
kind: Secret
metadata:
  name: github-org-creds
  namespace: argocd
  labels:
    argocd.argoproj.io/secret-type: repo-creds
stringData:
  type: git
  url: https://github.com/your-org
  secretArn: arn:aws:secretsmanager:us-west-2:111122223333:secret:argocd/github-org-
AbCdEf
```

此登入資料範本適用於符合 URL 字首 的所有儲存庫`https://github.com/your-org`。然後，您可以在應用程式中參考此組織下的任何儲存庫，而無需建立其他秘密。

如需詳細資訊，請參閱 Argo CD 文件中的儲存庫登入資料。

> **Important**
>
> 確保您的 IAM 功能角色已`arn:aws:iam::aws:policy/AWSSecretsManagerClientReadOnlyAccess`連接 受管政策或同等許可，包括 `secretsmanager:GetSecretValue`和 KMS 解密許可。如需 IAM 政策組態the section called “Argo CD 考量事項”，請參閱 。

使用 AWS CodeConnections

如需 CodeConnections 整合，請參閱 the section called “ AWS CodeConnections”。

CodeConnections 為 GitHub、GitLab 和 Bitbucket 提供受管身分驗證，而不儲存登入資料。

使用 Kubernetes 秘密

使用標準 Argo CD 方法將登入資料直接儲存在 Kubernetes 中。

對於具有個人存取字符的 HTTPS：

```
apiVersion: v1
kind: Secret
metadata:
  name: my-repo
  namespace: argocd
  labels:
    argocd.argoproj.io/secret-type: repository
stringData:
  type: git
  url: https://github.com/your-org/your-repo
  username: your-username
  password: your-personal-access-token
```

對於 SSH：

```
apiVersion: v1
kind: Secret
metadata:
  name: my-repo-ssh
  namespace: argocd
  labels:
    argocd.argoproj.io/secret-type: repository
stringData:
  type: git
  url: git@github.com:your-org/your-repo.git
  sshPrivateKey: |
    -----BEGIN OPENSSH PRIVATE KEY-----
    ... your private key ...
    -----END OPENSSH PRIVATE KEY-----
```

CodeCommit 儲存庫

For AWS CodeCommit，授予您的 IAM 功能角色 CodeCommit 許可 (`codecommit:GitPull`)。

設定儲存庫：

```
apiVersion: v1
kind: Secret
metadata:
  name: codecommit-repo
  namespace: argocd
  labels:
    argocd.argoproj.io/secret-type: repository
stringData:
  type: git
  url: https://git-codecommit.us-west-2.amazonaws.com/v1/repos/my-repo
  project: default
```

如需詳細的 IAM 政策組態，請參閱 the section called “Argo CD 考量事項”。

驗證儲存庫連線

透過設定 → 儲存庫下的 Argo CD UI 檢查連線狀態。UI 會顯示連線狀態和任何身分驗證錯誤。

儲存庫秘密不包含狀態資訊。

其他資源

- the section called “註冊叢集” - 註冊部署的目標叢集
- the section called “建立應用程式” - 建立您的第一個應用程式
- the section called “Argo CD 考量事項” - IAM 許可和安全性組態
- 私有儲存庫 - 上游儲存庫組態參考

註冊目標叢集

註冊叢集，讓 Argo CD 將應用程式部署到叢集。您可以註冊 Argo CD 正在執行的相同叢集（本機叢集）或不同帳戶或區域中的遠端叢集。註冊叢集後，它將保持未知連線狀態，直到您在該叢集內建立應用程式為止。若要在註冊叢集後建立 Argo CD 應用程式，請參閱 the section called “建立應用程式”。

先決條件

- 建立 Argo CD 功能的 EKS 叢集
- `kubectl` 設定為與您的叢集通訊
- 對於遠端叢集：適當的 IAM 許可和存取項目

註冊本機叢集

若要將應用程式部署至執行 Argo CD 的相同叢集，請將其註冊為部署目標。

> **Important**
>
> Argo CD 功能不會自動註冊本機叢集。您必須明確註冊，才能將應用程式部署到相同的叢集。您可以使用叢集名稱`in-cluster`，與大多數線上 Argo CD 範例相容。

> **Note**
>
> EKS 存取項目會自動為具有 Argo CD 功能角色的本機叢集建立，但預設不會授予 Kubernetes RBAC 許可。這遵循最低權限原則 - 您必須根據您的使用案例明確設定 Argo CD 所需的許可。例如，如果您只使用此叢集做為 Argo CD 中樞來管理遠端叢集，則不需要任何本機部署許可。如需組態選項，請參閱下方的存取項目 RBAC 要求一節。

使用 Argo CD CLI：

```
argocd cluster add <cluster-context-name> \
  --aws-cluster-name arn:aws:eks:us-west-2:111122223333:cluster/my-cluster \
  --name local-cluster
```

使用 Kubernetes 秘密：

```
apiVersion: v1
kind: Secret
metadata:
  name: local-cluster
  namespace: argocd
  labels:
    argocd.argoproj.io/secret-type: cluster
stringData:
  name: local-cluster
  server: arn:aws:eks:us-west-2:111122223333:cluster/my-cluster
  project: default
```

套用組態：

```
kubectl apply -f local-cluster.yaml
```

> **Note**
>
> 在 `server` 欄位中使用 EKS 叢集 ARN，而不是 Kubernetes API 伺服器 URL。受管功能需要 ARNs才能識別叢集。`kubernetes.default.svc` 不支援預設值。

註冊遠端叢集

若要部署至遠端叢集：

步驟 1：在遠端叢集上建立存取項目

將 *region-code* 取代為遠端叢集所在的 AWS 區域，將 *remote-cluster* 取代為遠端叢集的名稱，並將 ARN 取代為 Argo CD 功能角色 ARN。

```
aws eks create-access-entry \
  --region region-code \
  --cluster-name remote-cluster \
  --principal-arn arn:aws:iam::111122223333:role/ArgoCDCapabilityRole \
  --type STANDARD
```

步驟 2：將存取政策與 Kubernetes RBAC 許可建立關聯

Access Entry 需要 Argo CD 的 Kubernetes RBAC 許可才能部署應用程式。若要快速入門，您可以使用 `AmazonEKSClusterAdminPolicy`：

```
aws eks associate-access-policy \
  --region region-code \
  --cluster-name remote-cluster \
  --principal-arn arn:aws:iam::111122223333:role/ArgoCDCapabilityRole \
  --policy-arn arn:aws:eks::aws:cluster-access-policy/AmazonEKSClusterAdminPolicy \
  --access-scope type=cluster
```

> **Important**
>
> `AmazonEKSClusterAdminPolicy` 提供完整的叢集管理員存取權（相當於 `system:masters`）。這對於開始使用很方便，但不應用於生產環境。對於生產環境，請建立

存取項目與自訂 Kubernetes 群組的關聯，並建立適當的角色或 ClusterRole 繫結，以使用更嚴格的許可。如需最低權限組態，請參閱下方的生產設定一節。

步驟 3：在 Argo CD 中註冊叢集

使用 Argo CD CLI：

```
argocd cluster add <cluster-context-name> \
  --aws-cluster-name arn:aws:eks:us-west-2:111122223333:cluster/remote-cluster \
  --name remote-cluster
```

使用 Kubernetes 秘密：

```
apiVersion: v1
kind: Secret
metadata:
  name: remote-cluster
  namespace: argocd
  labels:
    argocd.argoproj.io/secret-type: cluster
stringData:
  name: remote-cluster
  server: arn:aws:eks:us-west-2:111122223333:cluster/remote-cluster
  project: default
```

套用組態：

```
kubectl apply -f remote-cluster.yaml
```

跨帳戶叢集

若要部署到不同 AWS 帳戶中的叢集：

1. 在目標帳戶中，使用來源帳戶中的 Argo CD IAM 功能角色 ARN 做為委託人，在目標 EKS 叢集上建立存取項目
2. 將存取政策與適當的 Kubernetes RBAC 許可建立關聯
3. 使用其 EKS 叢集 ARN 在 Argo CD 中註冊叢集

不需要額外的 IAM 角色建立或信任政策組態—EKS 存取項目會處理跨帳戶存取。

叢集 ARN 格式包含 區域，因此跨區域部署使用與相同區域部署相同的程序。

驗證叢集註冊

檢視已註冊的叢集：

```
kubectl get secrets -n argocd -l argocd.argoproj.io/secret-type=cluster
```

或在設定 → 叢集下的 Argo CD UI 中檢查叢集狀態。

私有叢集

Argo CD 功能提供全私有 EKS 叢集的透明存取，而不需要 VPC 對等互連或專用聯網組態。

AWS 會自動管理 Argo CD 功能和私有遠端叢集之間的連線。

只要使用私有叢集的 ARN 註冊 - 不需要額外的聯網設定。

存取項目 RBAC 要求

當您建立 Argo CD 功能時，會自動為功能角色建立 EKS 存取項目，但預設不會授予 Kubernetes RBAC 許可。此刻意設計遵循最低權限原則 - 不同的使用案例需要不同的許可。

例如：* 如果您僅使用叢集做為 Argo CD 中樞來管理遠端叢集，則不需要本機部署許可 * 如果您在本機部署應用程式，則需要整個叢集的讀取存取和特定命名空間的寫入存取 * 如果您需要建立 CRDs，則需要額外的叢集管理許可

您必須根據您的需求明確設定 Argo CD 所需的許可。

Argo CD 的最低許可

Argo CD 需要兩種類型的許可才能在沒有錯誤的情況下運作：

讀取許可（全叢集）：Argo CD 必須能夠跨叢集讀取以下項目的所有資源類型和自訂資源定義 (CRDs)：

- 資源探索和運作狀態檢查
- 偵測所需和實際狀態之間的偏離
- 在部署之前驗證資源

寫入許可（命名空間特定）：Argo CD 需要建立、更新和刪除應用程式中所定義資源的許可：

- 部署應用程式工作負載（部署、服務、ConfigMaps 等）
- 套用自訂資源 (CRDs)
- 管理應用程式生命週期

快速設定

若要快速入門、測試或開發環境，請使用 AmazonEKSClusterAdminPolicy：

```
aws eks associate-access-policy \
  --region region-code \
  --cluster-name my-cluster \
  --principal-arn arn:aws:iam::111122223333:role/ArgoCDCapabilityRole \
  --policy-arn arn:aws:eks::aws:cluster-access-policy/AmazonEKSClusterAdminPolicy \
  --access-scope type=cluster
```

> **Important**
>
> AmazonEKSClusterAdminPolicy 提供完整的叢集管理員存取權（相當於 system:masters），包括能夠建立 CRDs、修改整個叢集的資源，以及部署到任何命名空間。這對於開發和 POCs 來說很方便，但不應用於生產環境。對於生產，請使用下列最低權限設定。

最低權限的生產設定

對於生產環境，建立自訂 Kubernetes RBAC，以授予：

- 整個叢集對所有 資源的讀取存取權（適用於探索和運作狀態檢查）
- 命名空間特定的寫入存取（適用於部署）

步驟 1：將存取項目與自訂 Kubernetes 群組建立關聯

```
aws eks associate-access-policy \
  --region region-code \
  --cluster-name my-cluster \
  --principal-arn arn:aws:iam::111122223333:role/ArgoCDCapabilityRole \
  --policy-arn arn:aws:eks::aws:cluster-access-policy/AmazonEKSEditPolicy \
  --access-scope type=namespace,namespaces=app-namespace
```

步驟 2：建立 ClusterRole 以進行讀取存取

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: argocd-read-all
rules:
# Read access to all resources for discovery and health checks
- apiGroups: ["*"]
  resources: ["*"]
  verbs: ["get", "list", "watch"]
```

步驟 3：建立角色以寫入存取應用程式命名空間

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: argocd-deploy
  namespace: app-namespace
rules:
# Full access to deploy application resources
- apiGroups: ["*"]
  resources: ["*"]
  verbs: ["*"]
```

步驟 4：將角色繫結至 Kubernetes 群組

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: argocd-read-all
subjects:
- kind: Group
  name: eks-access-entry:arn:aws:iam::111122223333:role/ArgoCDCapabilityRole
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: argocd-read-all
  apiGroup: rbac.authorization.k8s.io
---
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: argocd-deploy
  namespace: app-namespace
subjects:
- kind: Group
  name: eks-access-entry:arn:aws:iam::111122223333:role/ArgoCDCapabilityRole
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: argocd-deploy
  apiGroup: rbac.authorization.k8s.io
```

Note

存取項目的群組名稱格式`eks-access-entry:`後面接著主體 ARN。針對 Argo CD 應部署應用程式的每個命名空間重複 RoleBinding。

Important

Argo CD 必須能夠讀取叢集中的所有資源類型，以進行運作狀態檢查和探索，即使它只部署到特定的命名空間。如果沒有整個叢集的讀取存取權，Argo CD 會在檢查應用程式運作狀態時顯示錯誤。

使用專案限制叢集存取

透過在 中設定允許的目標叢集和命名空間，使用專案來控制應用程式可以部署到哪些叢集和命名空間`spec.destinations`：

```
apiVersion: argoproj.io/v1alpha1
kind: AppProject
metadata:
  name: production
  namespace: argocd
spec:
  destinations:
  - server: arn:aws:eks:us-west-2:111122223333:cluster/prod-cluster
    namespace: '*'
```

```
  - server: arn:aws:eks:eu-west-1:111122223333:cluster/prod-eu-cluster
    namespace: '*'
  sourceRepos:
  - 'https://github.com/example/production-apps'
```

如需詳細資訊，請參閱the section called “專案”。

其他資源

- the section called “專案” - 組織應用程式並強制執行安全界限
- the section called “建立應用程式” - 部署您的第一個應用程式
- the section called “ApplicationSets” - 使用 ApplicationSets 部署到多個叢集
- the section called “Argo CD 考量事項” - 多叢集模式和跨帳戶設定
- 宣告式叢集設定 - 上游叢集組態參考

使用 Argo CD 專案

Argo CD Projects (AppProject) 提供應用程式的邏輯分組和存取控制。專案定義應用程式可以使用哪些 Git 儲存庫、目標叢集和命名空間，在共用 Argo CD 執行個體中啟用多租戶和安全性界限。

何時使用專案

使用專案來：

- 依團隊、環境或業務單位分隔應用程式
- 限制團隊可以從中部署的儲存庫
- 限制團隊可以部署到哪些叢集和命名空間
- 強制執行資源配額和允許的資源類型
- 使用護欄提供自助式應用程式部署

預設專案

每個 Argo CD 功能都包含一個`default`專案，允許存取所有儲存庫、叢集和命名空間。雖然適用於初始測試，但請建立具有明確生產使用限制的專用專案。

如需預設專案組態及其限制方式的詳細資訊，請參閱 Argo CD 文件中的預設專案。

建立專案

透過將 AppProject 資源套用至叢集來建立專案。

範例：團隊特定專案

```
apiVersion: argoproj.io/v1alpha1
kind: AppProject
metadata:
  name: team-a
  namespace: argocd
spec:
  description: Applications for Team A

  # Source repositories this project can deploy from
  sourceRepos:
    - 'https://github.com/my-org/team-a-*'
    - 'https://github.com/my-org/shared-libs'


  # Source namespaces (required for EKS capability)
  sourceNamespaces:
    - argocd
    - team-a-dev
    - team-a-prod

  # Destination clusters and namespaces
  destinations:
    - name: dev-cluster
      namespace: team-a-dev
    - name: prod-cluster
      namespace: team-a-prod

  # Allowed resource types
  clusterResourceWhitelist:
    - group: ''
      kind: Namespace

  namespaceResourceWhitelist:
    - group: 'apps'
      kind: Deployment
    - group: ''
      kind: Service
    - group: ''
```

```
      kind: ConfigMap
```

套用專案：

```
kubectl apply -f team-a-project.yaml
```

專案組態

來源儲存庫

控制此專案中的應用程式可以使用哪些 Git 儲存庫：

```
spec:
  sourceRepos:
    - 'https://github.com/my-org/app-*'  # Wildcard pattern
    - 'https://github.com/my-org/infra'  # Specific repo
```

您可以使用萬用字元和否定模式 (! 字首） 來允許或拒絕特定儲存庫。如需詳細資訊，請參閱 Argo CD 文件中的管理專案。

來源命名空間

使用 EKS Argo CD 功能時，自訂 AppProject 定義中需要 `spec.sourceNamespaces` 欄位。此欄位指定哪些命名空間可以包含參考此專案的應用程式或 ApplicationSets：

> **Important**
>
> 這是 EKS Argo CD 功能的必要欄位，不同於 OSS Argo CD，其中此欄位是選用的。

預設 AppProject 行為

`default` AppProject 會自動在 中包含`argocd`命名空間`sourceNamespaces`。如果您需要在其他命名空間中建立應用程式或 ApplicationSets，請修改 `sourceNamespaces` 欄位以新增這些命名空間：

```
apiVersion: argoproj.io/v1alpha1
kind: AppProject
metadata:
  name: default
  namespace: argocd
spec:
  sourceNamespaces:
```

```
    - argocd           # Already included by default
    - team-a-apps      # Add additional namespaces as needed
    - team-b-apps
```

自訂 AppProject 組態

建立自訂 AppProject 時，您必須手動包含argocd系統命名空間，以及您計劃建立應用程式或 ApplicationSets 的任何其他命名空間：

```
apiVersion: argoproj.io/v1alpha1
kind: AppProject
metadata:
  name: team-a-project
  namespace: argocd
spec:
  description: Applications for Team A

  # Required: Manually specify all namespaces
  sourceNamespaces:
    - argocd           # ArgoCD system namespace (required)
    - team-a-dev       # Custom namespace for dev Applications
    - team-a-prod      # Custom namespace for prod Applications

  # Source repositories this project can deploy from
  sourceRepos:
    - 'https://github.com/my-org/team-a-*'

  # Destination restrictions
  destinations:
    - namespace: 'team-a-*'
      server: arn:aws:eks:us-west-2:111122223333:cluster/my-cluster  # Use cluster ARN
 from: aws eks describe-cluster
```

> **Note**
>
> 如果您省略來自 的命名空間sourceNamespaces，在該命名空間中建立的應用程式或 ApplicationSets 將無法參考此專案，導致部署失敗。

目的地限制

限制應用程式可以部署的位置：

```
spec:
  destinations:
    - name: prod-cluster  # Specific cluster by name
      namespace: production
    - name: '*'  # Any cluster
      namespace: team-a-*  # Namespace pattern
```

> **Important**
>
> 使用特定叢集名稱和命名空間模式，而不是生產專案的萬用字元。這可防止意外部署到未經授權的叢集或命名空間。

您可以使用萬用字元和否定模式來控制目的地。如需詳細資訊，請參閱 Argo CD 文件中的管理專案。

資源限制

控制可以部署哪些 Kubernetes 資源類型：

叢集範圍的資源：

```
spec:
  clusterResourceWhitelist:
    - group: ''
      kind: Namespace
    - group: 'rbac.authorization.k8s.io'
      kind: Role
```

命名空間範圍的資源：

```
spec:
  namespaceResourceWhitelist:
    - group: 'apps'
      kind: Deployment
    - group: ''
      kind: Service
    - group: ''
      kind: ConfigMap
    - group: 's3.services.k8s.aws'
      kind: Bucket
```

使用封鎖清單來拒絕特定資源：

```
spec:
  namespaceResourceBlacklist:
    - group: ''
      kind: Secret  # Prevent direct Secret creation
```

將應用程式指派給專案

建立應用程式時，請在 spec.project 欄位中指定專案：

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: my-app
  namespace: argocd
spec:
  project: team-a  # Assign to team-a project
  source:
    repoURL: https://github.com/my-org/my-app
    path: manifests
  destination:
    name: prod-cluster
    namespace: team-a-prod
```

沒有指定專案的應用程式會使用default專案。

專案角色和 RBAC

專案可以為精細存取控制定義自訂角色。將專案角色映射至您功能組態中的 AWS Identity Center 使用者和群組，以控制誰可以同步、更新或刪除應用程式。

範例：具有開發人員和管理角色的專案

```
apiVersion: argoproj.io/v1alpha1
kind: AppProject
metadata:
  name: team-a
  namespace: argocd
spec:
  sourceRepos:
```

```
    - '*'
  destinations:
    - name: '*'
      namespace: 'team-a-*'

  roles:
    - name: developer
      description: Developers can sync applications
      policies:
        - p, proj:team-a:developer, applications, sync, team-a/*, allow
        - p, proj:team-a:developer, applications, get, team-a/*, allow
      groups:
        - team-a-developers

    - name: admin
      description: Admins have full access
      policies:
        - p, proj:team-a:admin, applications, *, team-a/*, allow
      groups:
        - team-a-admins
```

如需專案角色、CI/CD 管道的 JWT 權杖和 RBAC 組態的詳細資訊，請參閱 Argo CD 文件中的專案角色。

常見模式

以環境為基礎的專案

為每個環境建立個別專案：

```
apiVersion: argoproj.io/v1alpha1
kind: AppProject
metadata:
  name: production
  namespace: argocd
spec:
  sourceRepos:
    - 'https://github.com/my-org/*'
  destinations:
    - name: prod-cluster
      namespace: '*'
  # Strict resource controls for production
  clusterResourceWhitelist: []
```

```
  namespaceResourceWhitelist:
    - group: 'apps'
      kind: Deployment
    - group: ''
      kind: Service
```

以團隊為基礎的專案

使用專用專案隔離團隊：

```
apiVersion: argoproj.io/v1alpha1
kind: AppProject
metadata:
  name: platform-team
  namespace: argocd
spec:
  sourceRepos:
    - 'https://github.com/my-org/platform-*'
  destinations:
    - name: '*'
      namespace: 'platform-*'
  # Platform team can manage cluster resources
  clusterResourceWhitelist:
    - group: '*'
      kind: '*'
```

多叢集專案

使用一致的政策部署到多個叢集：

```
apiVersion: argoproj.io/v1alpha1
kind: AppProject
metadata:
  name: global-app
  namespace: argocd
spec:
  sourceRepos:
    - 'https://github.com/my-org/global-app'
  destinations:
    - name: us-west-cluster
      namespace: app
    - name: eu-west-cluster
      namespace: app
```

```
    - name: ap-south-cluster
      namespace: app
```

最佳實務

從限制性專案開始：從縮小的許可開始，並視需要擴展，而不是從廣泛的存取開始。

使用命名空間模式：在命名空間限制（例如 `team-a-*`) 中利用萬用字元來允許彈性，同時維持邊界。

獨立生產專案：使用專用專案進行生產，搭配更嚴格的控制和手動同步政策。

文件專案目的：使用 `description` 欄位來說明每個專案的用途，以及應該使用它的人員。

定期檢閱專案許可：定期稽核專案，以確保限制仍符合團隊需求和安全性要求。

其他資源

- the section called “設定 許可” - 設定 RBAC 和 Identity Center 整合
- the section called “建立應用程式” - 在專案中建立應用程式
- the section called “ApplicationSets” - 將 ApplicationSets 與多叢集部署的專案搭配使用
- Argo CD 專案文件 - 完整上游參考

設定 Argo CD 設定

Argo CD 的 EKS 功能提供全受管 Argo CD 體驗。上游 Argo CD 提供許多選用設定和功能，而 功能支援其中一部分。對於支援的設定，您可以透過叢集中的 ConfigMap，以與上游 Argo CD `argocd-cm` 相同的方式進行設定。功能會從該 ConfigMap 讀取支援的欄位，並將其套用至受管 Argo CD 執行個體。

下列各節說明如何為支援的設定設定設定 `argocd-cm` ConfigMap。

先決條件

設定 Argo CD 設定之前，您必須擁有：

- 建立 Argo CD 功能的 EKS 叢集（請參閱 the section called “建立 Argo CD 功能”)
- 在 功能中為 Argo CD 設定的命名空間（預設為`argocd`命名空間）
- 設定為與您的叢集通訊的 `kubectl` CLI

設定 argocd-cm ConfigMap

若要設定支援的 Argo CD 設定，請在叢集`argocd-cm`中建立名為 的 ConfigMap。受管功能會從此 ConfigMap 讀取支援的設定，並將其套用至受管 Argo CD 執行個體。如需 功能支援的設定及其套用方式，請參閱 the section called “支援的設定”。

建立具有下列需求的 ConfigMap：

- 為 ConfigMap 命名`argocd-cm`。
- 在 功能中為 Argo CD 設定的命名空間中建立它（您在建立 功能時在 Argo CD 組態中設定的命名空間）。根據預設，這是`argocd`命名空間。
- 套用標籤 `app.kubernetes.io/part-of: argocd`。此標籤是必要的，符合上游 Argo CD 行為。
- 使用與上游 Argo CD 相同的欄位格式和金鑰。

下列範例顯示 ConfigMap 結構，其設定會在 Argo CD UI 上顯示橫幅。`data` 以相同方式新增 下的其他支援設定。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: argocd-cm
  namespace: argocd
  labels:
    app.kubernetes.io/part-of: argocd
data:
  ui.bannercontent: "Production cluster"
```

Important

ConfigMap 不是安全存放區。請勿在 ConfigMap `argocd-cm` 中放置秘密、登入資料或其他敏感資訊。

功能如何套用您的設定

您可以在自己的叢集中建立 `argocd-cm` ConfigMap 來設定 Argo CD。此功能會將 ConfigMap 中支援的設定套用至受管 Argo CD 執行個體。它只會套用支援的設定，並忽略您設定的任何其他欄位或功能。任何未列在 中的設定the section called “支援的設定”都不受支援，而且沒有任何效果。

該功能會驗證您設定的值。如果值無效或格式不正確，則 功能會忽略該值，並繼續執行該設定的預設組態。ConfigMap 中的錯誤不會破壞您的受管 Argo CD 執行個體。

> **Note**
>
> 此功能會從叢集中的 `argocd-cm` ConfigMap 套用組態。具有此 ConfigMap 寫入存取權的任何主體都可以變更受管 Argo CD 執行個體的組態。叢集的 Kubernetes 角色型存取控制 (RBAC) 管理對 ConfigMap 的存取，而不是控制功能資源的 IAM 許可。作為安全最佳實務，將修改 Argo CD 命名空間中物件的許可授予信任的使用者和服務帳戶。由於 Kubernetes RBAC 會依資源類型範圍許可，因此您仍然可以授予其他使用者所需的存取權。例如，您可以允許開發人員建立和管理應用程式，但不能修改 ConfigMaps。這可防止他們變更`argocd-cm`組態。
> 如需 Argo CD 功能的共同責任模型、Kubernetes RBAC 和命名空間隔離的詳細資訊，請參閱 the section called “EKS 功能的考量事項”。若要控制 Argo CD 中的存取，請參閱 the section called “設定 許可”。

支援的設定

下列各節列出 受管功能支援的`argocd-cm`設定，依類別分組。每個設定都使用與上游 Argo CD 相同的金鑰和格式。每個資料表的套用值方式欄會顯示您的值是否附加至或覆寫功能的預設組態。如需每個設定的完整說明，請參閱 Argo CD 文件網站上的 argocd-cm ConfigMap 參考。

使用者界面

這些設定會自訂 Argo CD UI。

設定	說明	如何套用您的值
`ui.bannercontent`	在整個 UI 中顯示的橫幅文字，例如環境識別符或維護通知。	覆寫
`ui.bannerurl`	橫幅連結的 URL，例如 Runbook 或 Wiki 頁面。	覆寫
`ui.bannerpermanent`	設定為 `true`以防止使用者關閉橫幅。	覆寫
`ui.bannerposition`	橫幅出現的位置：`bottom`、 `top`或 `both`。	覆寫

設定	說明	如何套用您的值
ui.cssurl	用於品牌或樣式的自訂 CSS 檔案 URL。CSS 會在您的瀏覽器中執行。	覆寫

資源設定

這些設定控制功能如何監看、比較和顯示 Argo CD 管理的資源。

設定	說明	如何套用您的值
resource.customizations.ignoreDifferences.<group>_<kind>	當 Argo CD 針對資源類型比較 Git 與叢集時要忽略的欄位，例如由 Horizontal Pod Autoscaler 管理的複本計數。	附加
resource.customizations.ignoreDifferences.all	比較 Git 與叢集時要忽略的欄位，套用至所有資源類型。	附加
resource.customizations.ignoreResourceUpdates.<group>_<kind>	Argo CD 在決定更新事件是否應觸發調節時忽略的欄位，以降低負載。事件仍然發生，Argo CD 只會忽略這些欄位的變更。	附加
resource.customizations.ignoreResourceUpdates.all	Argo CD 在處理更新事件時忽略的欄位，套用至所有資源類型。	附加
resource.customizations.knownTypeFie	自訂資源的欄位類型（清單、映射或基本），讓 Argo CD 計算準確的差異，而不是顯示全欄位取代。	附加

設定	說明	如何套用您的值
lds.<group>_<kind>		
resource.customizations.health.<group>_<kind>	資源類型的自訂運作狀態檢查，定義為 Lua 指令碼。功能包括 ACK 和 kro 資源的內建運作狀態檢查。請參閱 the section called “自訂運作狀態檢查”。	覆寫
resource.exclusions	Argo CD 未監看的資源類型，可改善高流失類型的效能。	附加
resource.inclusions	Argo CD 監看的資源類型。設定時，Argo CD 只會監看列出的類型。	附加
resource.compareoptions	控制 Argo CD 如何計算差異的選項，例如 ignoreAggregatedRoles 。	覆寫
resource.respectRBAC	控制器是否只監控其具有 RBAC 讀取許可的資源。接受 normal或 strict。	覆寫
resource.customLabels	要在 UI 資源檢視中顯示的其他資源標籤。	覆寫
resource.includeEventLabelKeys	要複製到 Argo CD 產生的 Kubernetes 事件的應用程式和專案標籤。	覆寫
resource.excludeEventLabelKeys	從 Argo CD 產生的 Kubernetes 事件中排除的標籤。	覆寫
resource.sensitive.mask.annotations	當 UI 或 CLI 顯示秘密時要遮罩的註釋。	覆寫

儲存庫和工具設定

這些設定控制 Argo CD 用來轉譯資訊清單的資訊清單工具。

設定	說明	如何套用您的值
`kustomize.enable`	Kustomize 是否已啟用為資訊清單來源類型。	覆寫
`helm.enable`	Helm 是否啟用為資訊清單來源類型。	覆寫
`jsonnet.enable`	Jsonnet 是否啟用為資訊清單來源類型。	覆寫
`kustomize.buildOptions`	傳遞至每個 的全域命令列旗標`kustomize build`。功能支援一部分旗標。請參閱 the section called “支援的 kustomize.buildOptions 旗標”。	覆寫

支援的 kustomize.buildOptions 旗標

對於 `kustomize.buildOptions`， 功能會將您的值篩選為一組安全、支援的旗標。它不支援允許組建讀取任意檔案或執行任意程式碼的旗標。它會個別捨棄任何不支援的旗標或無效的值，並套用其餘支援的旗標。您可以撰寫 `--flag value`或 `--flag=value`形式的旗標。

旗標	支援的值	備註
`--reorder`	`legacy`, `none`	僅變更轉譯 YAML 的順序。
`--enable-helm`	Boolean	從 路徑執行受管 Helm 二進位檔。
`--enable-managedby-label`	Boolean	僅新增標籤。

此功能會捨棄任何其他旗標，包括 `--load-restrictor`、 `--enable-exec`和 `--enable-alpha-plugins`。

自訂運作狀態檢查

Argo CD 會評估其部署之資源的運作狀態。對於 部署和服務等標準 Kubernetes 資源，Argo CD 具有內建的運作狀態邏輯。對於 Argo CD 無法辨識的自訂資源，它沒有內建的運作狀態邏輯，也不會報告運作狀態。

當自訂資源沒有運作狀態檢查時，Argo CD 會報告沒有運作狀態，並將其排除在應用程式的整體運作狀態之外。因此，應用程式可以報告Healthy，即使其資源仍在佈建或失敗。這也表示同步波可以在這些資源就緒之前進行，因為同步排序取決於報告的運作狀態。

透過自訂運作狀態檢查，您可以定義自訂資源的運作狀態邏輯，因此 Argo CD 會正確報告準確的運作狀態和序列部署。您可以使用相同的組態金鑰，以您在上游 Argo CD 中相同的方式定義自訂運作狀態檢查。現有的上游指令碼和社群範例適用於 Argo CD 的 EKS 功能，無需修改。

ACK 和 kro 的內建運作狀態檢查

Argo CD 的 EKS 功能包含 AWS Kubernetes (ACK) 和 kro (Kube Resource Orchestrator) 資源控制器的內建運作狀態檢查。這些資源會報告準確的運作狀態，無需額外的組態。

若要變更 功能如何評估 ACK 或 kro 資源的運作狀態，您可以定義該資源類型的自訂運作狀態檢查。您為資源類型定義的自訂運作狀態檢查會覆寫該類型的內建運作狀態檢查。

撰寫自訂運作狀態檢查

使用下列格式的金鑰，將 Lua 指令碼新增至 argocd-cm ConfigMap 來定義自訂運作狀態檢查：

```
resource.customizations.health.<group>_<kind>
```

以自訂資源的 API 群組取代 *<group>*，並以其類型取代 *<kind>*。例如，具有 API 群組example.com和種類的自訂資源的金鑰Database為resource.customizations.health.example.com_Database。

Lua 指令碼可透過 全域obj變數存取資源物件。指令碼必須傳回 欄位status設定為Healthy、Degraded、 Progressing或 之一的資料表Suspended。指令碼也可以設定選用message欄位，以提供描述性狀態訊息。

下列範例 ConfigMap 會定義Database自訂資源的運作狀態檢查。當資源的狀態階段為Healthy時Ready，指令碼會回報資源，Progressing否則會回報資源：

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: argocd-cm
  namespace: argocd
  labels:
    app.kubernetes.io/part-of: argocd
data:
```

```
  resource.customizations.health.example.com_Database: |
    hs = {}
    hs.status = "Progressing"
    hs.message = "Waiting for the resource to become ready"
    if obj.status ~= nil then
      if obj.status.phase == "Ready" then
        hs.status = "Healthy"
        hs.message = "Database is ready"
      end
    end
    return hs
```

如需運作狀態檢查指令碼格式、內建運作狀態檢查清單，以及您可以調整的社群範例的詳細資訊，請參閱 Argo CD 文件網站上的 Resource Health。

安全與限制

透過 受管功能，您的自訂運作狀態檢查指令碼會在隔離、全受管的運算中執行。每個功能都會隔離執行環境，而且無法存取叢集的資料或 AWS APIs。您不會佈建、修補或操作執行環境的任何部分。

當您撰寫自訂運作狀態檢查以搭配 EKS 功能使用時，請注意下列事項：

- 標準 Lua 程式庫無法使用。useOpenLibs 選項一律停用，這是上游 Argo CD 中的預設值。指令碼無法存取作業系統或檔案系統。如果您從依賴標準 Lua 程式庫的自我管理 Argo CD 遷移指令碼，則可能不會在 功能中以相同的方式執行。建議您在開發環境中測試運作狀態檢查指令碼，然後再在生產環境中使用它們。

如果運作狀態評估暫時無法使用， 功能會報告受影響的自訂資源，Progressing而不是移除其運作狀態。這可讓受影響的資源在應用程式的運作狀態中可見，直到評估復原為止。

驗證自訂運作狀態檢查

套用或更新 argocd-cm ConfigMap 之後，請確認運作狀態檢查為作用中：

1. 在 Argo CD UI 中，選擇包含您定義運作狀態檢查之自訂資源的應用程式。確認資源報告指令碼傳回的運作狀態。或者，執行argocd app get *<application-name>* 並檢閱資源的運作狀態。
2. 如果資源未報告預期的運作狀態，請確認下列事項：
   - ConfigMap 名為 ，argocd-cm位於 功能中為 Argo CD 設定的命名空間中。
   - ConfigMap 具有必要的app.kubernetes.io/part-of: argocd標籤。
   - 運作狀態檢查金鑰<group>_<kind>針對資源類型使用正確的 。

建立應用程式

應用程式代表目標叢集中的部署。每個應用程式都會定義來源 (Git 儲存庫） 和目的地 （叢集和命名空間）。套用時，Argo CD 會將 Git 儲存庫中資訊清單指定的資源建立到叢集中的命名空間。應用程式通常會指定工作負載部署，但可以管理目的地叢集中可用的任何 Kubernetes 資源。

先決條件

- 建立 Argo CD 功能的 EKS 叢集
- 已設定儲存庫存取 （請參閱 the section called “設定儲存庫”)
- 目標叢集已註冊 （請參閱 the section called “註冊叢集”)
- `kubectl` 設定為與您的叢集通訊

建立基本應用程式

定義從 Git 儲存庫部署的應用程式：

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: guestbook
  namespace: argocd
spec:
  project: default
  source:
    repoURL: https://github.com/argoproj/argocd-example-apps
    targetRevision: HEAD
    path: guestbook
  destination:
    name: in-cluster
    namespace: default
```

> **Note**
>
> `destination.name` 使用 與您在註冊叢集時使用的叢集名稱 （例如`in-cluster`本機叢集的)。`destination.server` 欄位也適用於 EKS 叢集 ARNs，但建議使用叢集名稱以提高可讀性。

套用應用程式：

```
kubectl apply -f application.yaml
```

檢視應用程式狀態：

```
kubectl get application guestbook -n argocd
```

來源組態

Git 儲存庫：

```
spec:
  source:
    repoURL: https://github.com/example/my-app
    targetRevision: main
    path: kubernetes/manifests
```

特定 Git 標籤或遞交：

```
spec:
  source:
    targetRevision: v1.2.0  # or commit SHA
```

Helm Chart：

```
spec:
  source:
    repoURL: https://github.com/example/helm-charts
    targetRevision: main
    path: charts/my-app
    helm:
      valueFiles:
      - values.yaml
      parameters:
      - name: image.tag
        value: v1.2.0
```

含有外部 Git 儲存庫值的 Helm Chart（多來源模式）：

```
spec:
  sources:
```

```
  - repoURL: https://github.com/example/helm-charts
    targetRevision: main
    path: charts/my-app
    helm:
      valueFiles:
      - $values/environments/production/values.yaml
  - repoURL: https://github.com/example/config-repo
    targetRevision: main
    ref: values
```

如需詳細資訊，請參閱 Argo CD 文件中的來自外部 Git 儲存庫的 Helm 值檔案。

ECR 的 Helm Chart：

```
spec:
  source:
    repoURL: oci://account-id.dkr.ecr.region.amazonaws.com/repository-name
    targetRevision: chart-version
    chart: chart-name
```

如果功能角色具有必要的 ECR 許可，則直接使用儲存庫，不需要儲存庫組態。如需詳細資訊，請參閱 the section called “設定儲存庫”。

CodeCommit 的 Git 儲存庫：

```
spec:
  source:
    repoURL: https://git-codecommit.region.amazonaws.com/v1/repos/repository-name
    targetRevision: main
    path: kubernetes/manifests
```

如果功能角色具有必要的 CodeCommit 許可，則直接使用儲存庫，不需要儲存庫組態。如需詳細資訊，請參閱 the section called “設定儲存庫”。

CodeConnections 中的 Git 儲存庫：

```
spec:
  source:
    repoURL: https://codeconnections.region.amazonaws.com/git-http/account-
id/region/connection-id/owner/repository.git
    targetRevision: main
```

```
    path: kubernetes/manifests
```

儲存庫 URL 格式衍生自 CodeConnections 連線 ARN。如果功能角色具有必要的 CodeConnections 許可，且已設定連線，則會直接使用儲存庫，而且不需要儲存庫組態。如需詳細資訊，請參閱 the section called “設定儲存庫”。

Kustomize：

```
spec:
  source:
    repoURL: https://github.com/example/kustomize-app
    targetRevision: main
    path: overlays/production
    kustomize:
      namePrefix: prod-
```

同步政策

控制 Argo CD 同步應用程式的方式。

手動同步（預設）：

應用程式需要手動核准才能同步：

```
spec:
  syncPolicy: {}  # No automated sync
```

手動觸發同步：

```
kubectl patch application guestbook -n argocd \
  --type merge \
  --patch '{"operation": {"initiatedBy": {"username": "admin"}, "sync": {}}}'
```

自動同步：

偵測到 Git 變更時，應用程式會自動同步：

```
spec:
  syncPolicy:
    automated: {}
```

自我修復：

自動將手動變更還原至叢集：

```
spec:
  syncPolicy:
    automated:
      selfHeal: true
```

啟用時，Argo CD 會還原直接對叢集所做的任何手動變更，確保 Git 仍是事實來源。

剔除：

自動刪除從 Git 移除的資源：

```
spec:
  syncPolicy:
    automated:
      prune: true
```

> **Warning**
>
> 刪除 會從叢集刪除資源。在生產環境中謹慎使用 。

合併自動同步：

```
spec:
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
    syncOptions:
    - CreateNamespace=true
```

重試組態：

設定失敗同步的重試行為：

```
spec:
```

```
  syncPolicy:
    retry:
      limit: 5  # Number of failed sync attempts; unlimited if less than 0
      backoff:
        duration: 5s  # Amount to back off (default unit: seconds, also supports "2m",
 "1h")
        factor: 2  # Factor to multiply the base duration after each failed retry
        maxDuration: 3m  # Maximum amount of time allowed for the backoff strategy
```

這對於依賴於先建立CRDs 的資源，或使用可能無法立即使用 CRD 的 kro 執行個體時特別有用。

同步選項

其他同步組態：

如果命名空間不存在，請建立命名空間：

```
spec:
  syncPolicy:
    syncOptions:
    - CreateNamespace=true
```

略過缺少資源的試轉：

套用依賴尚不存在CRDs 的資源時很有用 （ 例如 kro 執行個體 ）：

```
spec:
  syncPolicy:
    syncOptions:
    - SkipDryRunOnMissingResource=true
```

這也可以套用到使用資源本身標籤的特定資源。

在套用之前驗證資源：

```
spec:
  syncPolicy:
    syncOptions:
    - Validate=true
```

僅套用不同步：

```
spec:
  syncPolicy:
    syncOptions:
    - ApplyOutOfSyncOnly=true
```

進階同步功能

Argo CD 支援複雜部署的進階同步功能：

- 同步波 – 使用`argocd.argoproj.io/sync-wave`註釋控制資源建立順序
- 同步掛鉤 - 使用`argocd.argoproj.io/hook`註釋同步之前或之後執行任務 (PreSync、PostSync、SyncFail)
- 資源運作狀態評估 - 應用程式特定資源的自訂運作狀態檢查

如需詳細資訊，請參閱 Argo CD 文件中的同步波和資源勾點。

忽略差異

防止 Argo CD 同步由其他控制器管理的特定欄位（例如 HPA 管理複本）：

```
spec:
  ignoreDifferences:
  - group: apps
    kind: Deployment
    jsonPointers:
    - /spec/replicas
```

如需忽略模式和欄位排除的詳細資訊，請參閱 Argo CD 文件中的差異自訂。

多環境部署

將相同的應用程式部署到多個環境：

開發：

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: my-app-dev
  namespace: argocd
spec:
```

```
  project: default
  source:
    repoURL: https://github.com/example/my-app
    targetRevision: develop
    path: overlays/development
  destination:
    name: dev-cluster
    namespace: my-app
```

生產：

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: my-app-prod
  namespace: argocd
spec:
  project: default
  source:
    repoURL: https://github.com/example/my-app
    targetRevision: main
    path: overlays/production
  destination:
    name: prod-cluster
    namespace: my-app
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
```

監控和管理應用程式

檢視應用程式狀態：

```
kubectl get application my-app -n argocd
```

存取 Argo CD UI：

透過 EKS 主控台開啟 Argo CD UI，以檢視應用程式拓撲、同步狀態、資源運作狀態和部署歷史記錄。如需 UI 存取指示the section called “使用 Argo CD”，請參閱 。

轉返應用程式：

使用 Argo CD UI、Argo CD CLI 或將應用程式規格`targetRevision`中的 更新為先前的 Git 遞交或標籤，轉返至先前的修訂。

使用 Argo CD CLI：

```
argocd app rollback argocd/my-app <revision-id>
```

> **Note**
>
> 搭配 受管功能使用 Argo CD CLI 時，請指定命名空間字首為 的應用程式`namespace/appname`。

如需詳細資訊，請參閱 Argo CD 文件中的 argocd 應用程式轉返。

其他資源

- the section called “專案” - 使用適用於多租戶環境的專案組織應用程式
- the section called “ApplicationSets” - 使用 範本部署到多個叢集
- 應用程式規格 - 完整應用程式 API 參考
- 同步選項 - 進階同步組態

使用 ApplicationSets

ApplicationSets 會從 範本產生多個應用程式，讓您能夠使用單一資源定義，在多個叢集、環境或命名空間之間部署相同的應用程式。

先決條件

- 建立 Argo CD 功能的 EKS 叢集
- 已設定儲存庫存取（請參閱 the section called “設定儲存庫”）
- `kubectl` 設定為與您的叢集通訊

> **Note**
>
> ApplicationSets 不需要多個目標叢集。您可以使用叢集產生器以外的產生器（例如清單、git 或矩陣產生器），在沒有遠端叢集的情況下部署應用程式。

ApplicationSets 的運作方式

ApplicationSets 使用產生器產生參數，然後將這些參數套用至應用程式範本。每組產生的參數都會建立一個應用程式。

EKS 部署的常見產生器：

- 列出產生器 - 明確定義每個環境的叢集和參數
- 叢集產生器 - 自動部署到所有已註冊的叢集
- Git 產生器 - 從儲存庫結構產生應用程式
- 矩陣產生器 - 合併多維部署的產生器
- 合併產生器 - 從多個產生器合併參數

如需完整的產生器參考，請參閱 ApplicationSet 文件。

列出產生器

使用明確組態部署到多個叢集：

```
apiVersion: argoproj.io/v1alpha1
kind: ApplicationSet
metadata:
  name: guestbook-all-clusters
  namespace: argocd
spec:
  generators:
  - list:
      elements:
      - environment: dev
        replicas: "2"
      - environment: staging
        replicas: "3"
      - environment: prod
        replicas: "5"
  template:
    metadata:
      name: 'guestbook-{{environment}}'
    spec:
      project: default
      source:
```

```
        repoURL: https://github.com/example/guestbook
        targetRevision: HEAD
        path: 'overlays/{{environment}}'
      destination:
        name: '{{environment}}-cluster'
        namespace: guestbook
      syncPolicy:
        automated:
          prune: true
          selfHeal: true
```

> **Note**
>
> 搭配使用 `destination.name`與叢集名稱，以提高可讀性。如有需要，`destination.server` 欄位也可使用 EKS 叢集 ARNs。

這會建立三個應用程式：`guestbook-dev`、 `guestbook-staging`和 `guestbook-prod`。

叢集產生器

自動部署到所有已註冊的叢集：

```
apiVersion: argoproj.io/v1alpha1
kind: ApplicationSet
metadata:
  name: cluster-addons
  namespace: argocd
spec:
  generators:
  - clusters: {}
  template:
    metadata:
      name: '{{name}}-addons'
    spec:
      project: default
      source:
        repoURL: https://github.com/example/cluster-addons
        targetRevision: HEAD
        path: addons
      destination:
        server: '{{server}}'
```

```
        namespace: kube-system
      syncPolicy:
        automated:
          prune: true
          selfHeal: true
```

這會自動為每個註冊的叢集建立 應用程式。

篩選叢集：

使用 matchLabels 來包含特定叢集，或使用 matchExpressions 來排除叢集：

```
spec:
  generators:
  - clusters:
      selector:
        matchLabels:
          environment: production
        matchExpressions:
        - key: skip-appset
          operator: DoesNotExist
```

Git 產生器

Git 產生器會根據儲存庫結構建立應用程式：

- 目錄產生器 - 將每個目錄部署為單獨的應用程式（適用於微服務）
- 檔案產生器 - 從參數檔案產生應用程式（適用於多租用戶部署）

範例：微服務部署

```
apiVersion: argoproj.io/v1alpha1
kind: ApplicationSet
metadata:
  name: microservices
  namespace: argocd
spec:
  generators:
  - git:
      repoURL: https://github.com/example/microservices
      revision: HEAD
```

```
      directories:
      - path: services/*
  template:
    metadata:
      name: '{{path.basename}}'
    spec:
      project: default
      source:
        repoURL: https://github.com/example/microservices
        targetRevision: HEAD
        path: '{{path}}'
      destination:
        name: my-cluster
        namespace: '{{path.basename}}'
      syncPolicy:
        automated:
          prune: true
          selfHeal: true
        syncOptions:
        - CreateNamespace=true
```

如需 Git 產生器和檔案型組態的詳細資訊，請參閱 Argo CD 文件中的 Git 產生器。

矩陣產生器

結合多個產生器以跨多個維度部署（環境 × 叢集）：

```
apiVersion: argoproj.io/v1alpha1
kind: ApplicationSet
metadata:
  name: multi-env-multi-cluster
  namespace: argocd
spec:
  generators:
  - matrix:
      generators:
      - list:
          elements:
          - environment: dev
          - environment: staging
          - environment: prod
      - clusters:
          selector:
```

```
            matchLabels:
              region: us-west-2
  template:
    metadata:
      name: 'app-{{environment}}-{{name}}'
    spec:
      project: default
      source:
        repoURL: https://github.com/example/app
        targetRevision: HEAD
        path: 'overlays/{{environment}}'
      destination:
        name: '{{name}}'
        namespace: 'app-{{environment}}'
```

如需合併產生器的詳細資訊，請參閱 Argo CD 文件中的矩陣產生器。

多區域部署

部署到跨多個區域的叢集：

```
apiVersion: argoproj.io/v1alpha1
kind: ApplicationSet
metadata:
  name: global-app
  namespace: argocd
spec:
  generators:
  - list:
      elements:
      - clusterName: prod-us-west
        region: us-west-2
      - clusterName: prod-us-east
        region: us-east-1
      - clusterName: prod-eu-west
        region: eu-west-1
  template:
    metadata:
      name: 'app-{{region}}'
    spec:
      project: default
      source:
        repoURL: https://github.com/example/app
```

```
        targetRevision: HEAD
        path: kubernetes
        helm:
          parameters:
          - name: region
            value: '{{region}}'
      destination:
        name: '{{clusterName}}'
        namespace: app
      syncPolicy:
        automated:
          prune: true
          selfHeal: true
```

管理 ApplicationSets

檢視 ApplicationSets 和產生的應用程式：

```
kubectl get applicationsets -n argocd
kubectl get applications -n argocd -l argocd.argoproj.io/application-set-
name=<applicationset-name>
```

更新 ApplicationSet：

修改 ApplicationSet 規格並重新套用。Argo CD 會自動更新所有產生的應用程式：

```
kubectl apply -f applicationset.yaml
```

刪除 ApplicationSet：

```
kubectl delete applicationset <name> -n argocd
```

> **Warning**
>
> 刪除 ApplicationSet 會刪除所有產生的應用程式。如果這些應用程式有 `prune: true`，其資源也會從目標叢集中刪除。
> 若要在刪除 ApplicationSet 時保留已部署的資源，請在 ApplicationSet 規格`true`中將`.syncPolicy.preserveResourcesOnDeletion`設定為 。如需詳細資訊，請參閱 Argo CD 文件中的應用程式刪除和資源刪除。

> **Important**
>
> Argo CD 的 ApplicationSets 功能具有您在使用 ApplicationSets 之前應注意的安全考量。如需詳細資訊，請參閱 Argo CD 文件中的 ApplicationSet Security。

其他資源

- the section called “專案” - 使用專案組織 ApplicationSets
- the section called “建立應用程式” - 了解應用程式組態
- ApplicationSet 文件 - 完整的產生器參考和模式
- 產生器參考 - 詳細的產生器規格

設定 Argo CD 的私有端點存取

根據預設，您可以透過公有網際網路存取 Argo CD 端點。您可以設定私有端點存取，讓 VPC 和 Argo CD 端點之間的流量保持在 AWS 網路中，而無需透過公有網際網路。

當您啟用私有存取時， 會 AWS 更新網域名稱系統 (DNS)，以便 Argo CD 端點主機名稱解析為與您的 VPC 端點相關聯的私有 IP 地址。您必須擁有這些私有 IPs網路連線能力 （ 透過 VPC、虛擬私有網路 (VPN) 連線或 AWS Direct Connect)，才能存取 Argo CD 端點。

私有存取的運作方式

當您為 Argo CD 功能設定私有存取時：

1. 您可以在 VPC 中為`com.amazonaws.`*`region-code`*`.eks-capabilities`服務建立 VPC 端點。
2. 您可以在建立或更新 Argo CD 功能時提供 VPC 端點 ID。
3. AWS 設定 DNS 記錄，以便 Argo CD 端點主機名稱解析為與您的 VPC 端點相關聯的私有 IP 地址。
4. Argo CD 端點的所有流量都會私下流經 VPC 端點。

先決條件

- 具有作用中 Argo CD 功能的 Amazon Elastic Kubernetes Service (Amazon EKS) 叢集 （ 請參閱 the section called “建立 Argo CD 功能”)
- 熟悉 Amazon VPC 概念，包括子網路、安全群組和界面 VPC 端點

- 在您想要私有存取的可用區域中具有子網路的 VPC
- AWS 命令列界面 (AWS CLI) 版本 2.12.3 或更新版本

步驟 1：為 eks-capabilities 建立 VPC 端點

建立 com.amazonaws.*region-code*.eks-capabilities服務的介面 VPC 端點。

Console

1. 開啟 Amazon VPC 主控台。
2. 在左側導覽窗格中選擇 Endpoints (端點)。
3. 選擇建立端點。
4. 針對服務類別，選擇 AWS 服務。
5. 在搜尋欄位中，輸入 eks-capabilities並選取 com.amazonaws.*region-code*.eks-capabilities。
6. 針對 VPC，選取您要從中存取 Argo CD 端點的 VPC。
7. 對於子網路，請在您要私有存取的每個可用區域中選取子網路。
8. 針對安全群組，選取允許來自用戶端網路的 HTTPS （連接埠 443) 傳入流量的安全群組。
9. 選擇建立端點。
10. 請記下 VPC 端點 ID （例如 vpce-0123456789abcdef0)。

CLI

將 *region-code* 取代為您的 AWS 區域、將 *vpc-id* 取代為您的 VPC ID、將 *subnet-id-1* *subnet-id-2* 取代為不同可用區域中的子網路 IDs，並將 *sg-id* 取代為允許 HTTPS 傳入流量的安全群組。

```
aws ec2 create-vpc-endpoint \
  --region region-code \
  --vpc-endpoint-type Interface \
  --service-name com.amazonaws.region-code.eks-capabilities \
  --vpc-id vpc-id \
  --subnet-ids subnet-id-1 subnet-id-2 \
  --security-group-ids sg-id
```

記下回應中的 VpcEndpointId。

> **Important**
>
> 確定連接到 VPC 端點的安全群組允許傳入 HTTPS （連接埠 443) 流量。流量來源必須包含您的使用者或應用程式存取 Argo CD 端點的所有網路。

步驟 2：在 Argo CD 功能上設定私有存取

在 VPC 端點可用後，設定 Argo CD 功能來使用它。您可以在建立新功能或更新現有功能時設定私有存取。

建立具有私有存取權的新功能

在 create-capability命令networkAccess中包含 。將 *region-code*、*my-cluster* 和其他預留位置取代為您的值：

```
aws eks create-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-argocd \
  --type ARGOCD \
  --role-arn arn:aws:iam::111122223333:role/ArgoCDCapabilityRole \
  --configuration '{
    "argoCd": {
      "awsIdc": {
        "idcInstanceArn": "arn:aws:sso:::instance/ssoins-1234567890abcdef",
        "idcRegion": "idc-region-code"
      },
      "rbacRoleMappings": [{
        "role": "ADMIN",
        "identities": [{
          "id": "user-id",
          "type": "SSO_USER"
        }]
      }],
      "networkAccess": {
        "vpceIds": ["vpce-0123456789abcdef0"]
      }
    }
```

```
  }'
```

更新現有功能以啟用私有存取

將 *region-code*、*my-cluster*、*my-argocd* 和 VPC 端點 ID 取代為您的值：

```
aws eks update-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-argocd \
  --configuration '{
    "argoCd": {
      "networkAccess": {
        "vpceIds": ["vpce-0123456789abcdef0"]
      }
    }
  }'
```

等待功能更新完成：

```
aws eks describe-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-argocd \
  --query 'capability.status' \
  --output text
```

狀態會在更新完成ACTIVE時傳回 。

步驟 3：驗證 DNS 解析和連線

功能啟用並設定私有存取後，請確認 Argo CD 端點主機名稱解析為私有 IP 地址。

從任何網路執行下列nslookup命令：

```
nslookup your-argocd-endpoint.eks-capabilities.region-code.amazonaws.com
```

回應應該會顯示私有 IP 地址（例如，RFC 1918 範圍內的地址，例如 10.x.x.x或 172.16.x.x)。

如果端點未解析為私有 IPs，請確認：

- 功能狀態為 ACTIVE（在功能變為作用中後AWS 建立 DNS 記錄）
- VPC 端點處於 available 狀態
- 功能組態中的 VPC 端點 ID 符合您建立的端點

> **Note**
>
> 設定私有存取後，Argo CD 端點主機名稱會解析為私有 IP 地址。若要連線，您必須透過 VPC、VPN 連線或 AWS Direct Connect 與這些私有 IPs 建立網路連線。

停用私有存取

若要停用私有存取並還原公有端點存取，請更新 功能以移除 VPC IDs：

```
aws eks update-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-argocd \
  --configuration '{
    "argoCd": {
      "networkAccess": {
        "vpceIds": []
      }
    }
  }'
```

考量事項

- 啟用私有存取後，只能從 VPC 內或與其連線的網路（例如，透過 VPN 或 AWS Direct Connect) 存取 Argo CD 端點。
- VPC 端點必須保持 available 狀態，才能私有存取 函數。如果您刪除 VPC 端點，您會失去 Argo CD 端點的存取權，直到您建立新的端點並更新功能為止。
- AWS 目前不支援eks-capabilities服務端點的 VPC 端點政策。

Argo CD 考量事項

本主題涵蓋使用 Argo CD 的 EKS 功能的重要考量，包括規劃、許可、身分驗證和多叢集部署模式。

規劃

部署 Argo CD 之前，請考慮下列事項：

儲存庫策略：判斷您的應用程式資訊清單將存放在何處 (CodeCommit、GitHub、GitLab、Bitbucket)。規劃不同環境的儲存庫結構和分支策略。

RBAC 策略：規劃哪些團隊或使用者應具有管理員、編輯器或檢視器存取權。將這些映射到 AWS Identity Center 群組或 Argo CD 角色。

多叢集架構：判斷您是否將從單一 Argo CD 執行個體管理多個叢集。考慮為 Argo CD 使用專用管理叢集。

應用程式組織：規劃如何建構應用程式和 ApplicationSets。考慮使用專案，依團隊或環境組織應用程式。

同步政策：決定應用程式是否應自動同步或需要手動核准。自動化同步對於開發很常見，對於生產則是手動。

許可

如需 IAM 功能角色、信任政策和安全最佳實務的詳細資訊，請參閱 the section called “功能 IAM 角色”和 the section called “EKS 功能的考量事項”。

IAM 功能角色概觀

當您建立 Argo CD 功能資源時，您會提供 IAM 功能角色。與 ACK 不同，Argo CD 主要管理 Kubernetes 資源，而不是直接管理 AWS 資源。不過，下列項目需要 IAM 功能角色：

- 在 CodeCommit 中存取私有 Git 儲存庫
- 與 AWS Identity Center 整合以進行身分驗證
- 在 AWS Secrets Manager 中存取秘密（如果已設定）
- 跨叢集部署至其他 EKS 叢集

CodeCommit 整合

如果您使用的是 CodeCommit 儲存庫，請連接具有讀取許可的政策：

```
{
  "Version": "2012-10-17",
  "Statement": [
```

```
    {
      "Effect": "Allow",
      "Action": [
        "codecommit:GitPull"
      ],
      "Resource": "*"
    }
  ]
}
```

Important

對於生產用途，請將 `Resource` 欄位限制為特定儲存庫 ARNs，而不是使用 `"*"`。
範例：

```
"Resource": "arn:aws:codecommit:us-west-2:111122223333:my-app-repo"
```

這會將 Argo CD 功能的存取權限制為僅需要管理的儲存庫。

Secrets Manager 整合

如果您要將儲存庫登入資料儲存在 Secrets Manager 中，請連接 受管政策以進行讀取存取：

```
arn:aws:iam::aws:policy/AWSSecretsManagerClientReadOnlyAccess
```

此政策包含必要的許可：`secretsmanager:GetSecretValue`、`secretsmanager:DescribeSecret`和 KMS 解密許可。

基本設定

對於具有公有 Git 儲存庫的基本 Argo CD 功能，除了信任政策之外，不需要額外的 IAM 政策。

身分驗證

AWS Identity Center 整合

Argo CD 受管功能直接與 AWS Identity Center （先前稱為 AWS SSO) 整合，可讓您使用現有的身分提供者進行身分驗證。

當您設定 AWS Identity Center 整合時：

1. 使用者透過 EKS 主控台存取 Argo CD UI
2. 他們使用 AWS Identity Center 進行身分驗證（可聯合到您的公司身分提供者）
3. AWS Identity Center 提供使用者和群組資訊給 Argo CD
4. Argo CD 會根據您的組態，將使用者和群組映射至 RBAC 角色
5. 使用者只會看到他們有權存取的應用程式和資源

使用 Identity Center 許可集簡化存取

AWS Identity Center 在使用 Argo CD 時提供兩種不同的身分驗證路徑：

Argo CD API 身分驗證： Identity Center 為 Argo CD UI 和 API 提供 SSO 身分驗證。這是透過 Argo CD 功能的 RBAC 角色映射來設定。

EKS 叢集存取：Argo CD 功能使用客戶提供的 IAM 角色，透過存取項目向 EKS 叢集進行驗證。您可以手動設定這些存取項目來新增或移除許可。

您可以使用 Identity Center 許可集來簡化身分管理，方法是允許單一身分同時存取 Argo CD 和 EKS 叢集。這透過要求您在兩個系統中僅管理一個身分，而不是為 Argo CD 存取和叢集存取維護單獨的登入資料，來減少額外負荷。

RBAC 角色映射

Argo CD 具有內建角色，您可以映射到 AWS Identity Center 使用者和群組：

ADMIN：完整存取所有應用程式和設定。可以建立、更新和刪除應用程式。可以管理 Argo CD 組態。

EDITOR：可以建立和修改應用程式。無法變更 Argo CD 設定或刪除應用程式。

VIEWER：唯讀存取應用程式。可以檢視應用程式狀態和歷史記錄。無法進行變更。

> **Note**
>
> 角色名稱區分大小寫，且必須為大寫 (ADMIN、EDITOR、VIEWER)。

> **Important**
>
> EKS 功能與 AWS Identity Center 整合支援每個 Argo CD 功能最多 1，000 個身分。身分可以是使用者或群組。

多叢集部署

Argo CD 受管功能支援多叢集部署，可讓您從單一 Argo CD 執行個體管理跨開發、預備和生產叢集的應用程式。

多叢集的運作方式

當您向 Argo CD 註冊其他叢集時：

1. 您可以建立依 ARN 參考目標 EKS 叢集的叢集秘密
2. 您可以建立以不同叢集為目標的應用程式或 ApplicationSets
3. Argo CD 會連線至每個叢集以部署和監看資源
4. 您可以從單一 Argo CD UI 檢視和管理所有叢集

多叢集的先決條件

註冊其他叢集之前：

- 在目標叢集上為 Argo CD 功能角色建立存取項目
- 確保 Argo CD 功能與目標叢集之間的網路連線
- 驗證存取目標叢集的 IAM 許可

註冊叢集

在 `argocd` 命名空間中使用 Kubernetes Secrets 註冊叢集。

取得目標叢集 ARN。將 *`region-code`* 取代為目標叢集所在的 AWS 區域，並將 *`target-cluster`* 取代為目標叢集的名稱。

```
aws eks describe-cluster \
  --region region-code \
  --name target-cluster \
  --query 'cluster.arn' \
  --output text
```

使用叢集 ARN 建立叢集秘密：

```
apiVersion: v1
```

```
kind: Secret
metadata:
  name: target-cluster
  namespace: argocd
  labels:
    argocd.argoproj.io/secret-type: cluster
type: Opaque
stringData:
  name: target-cluster
  server: arn:aws:eks:us-west-2:111122223333:cluster/target-cluster
  project: default
```

Important

在 `server` 欄位中使用 EKS 叢集 ARN，而不是 Kubernetes API 伺服器 URL。受管功能需要 ARNs才能識別目標叢集。

套用秘密：

```
kubectl apply -f cluster-secret.yaml
```

在目標叢集上設定存取項目

目標叢集必須具有存取項目，以授予 Argo CD 功能角色部署應用程式的許可。將 *region-code* 取代為目標叢集所在的 AWS 區域，將 *target-cluster* 取代為目標叢集的名稱，並將 ARN 取代為 Argo CD 功能角色 ARN。

```
aws eks create-access-entry \
  --region region-code \
  --cluster-name target-cluster \
  --principal-arn arn:aws:iam::111122223333:role/ArgoCDCapabilityRole \
  --type STANDARD \
  --kubernetes-groups system:masters
```

Note

對於生產用途，請考慮使用更嚴格的 Kubernetes 群組，而非 `system:masters`。

私有叢集存取

Argo CD 受管功能可以部署到全私有 EKS 叢集，而不需要 VPC 對等互連或專用聯網組態。 會自動 AWS 管理 Argo CD 功能與私有遠端叢集之間的連線。確保您的儲存庫存取控制和 Argo CD RBAC 政策已正確設定。

跨帳戶部署

對於跨帳戶部署，將 Argo CD IAM 功能角色從來源帳戶新增至目標叢集的 EKS 存取項目：

1. 在目標帳戶中，在目標 EKS 叢集上建立存取項目
2. 使用來源帳戶中的 Argo CD IAM 功能角色 ARN 做為委託人
3. 為存取項目設定適當的 Kubernetes RBAC 許可
4. 使用其 EKS 叢集 ARN 在 Argo CD 中註冊目標叢集

不需要額外的 IAM 角色建立或信任政策組態—EKS 存取項目會處理跨帳戶存取。

最佳實務

使用宣告性來源做為事實來源：將所有應用程式資訊清單存放在宣告性來源 (Git 儲存庫、Helm 登錄檔或 OCI 映像），啟用版本控制、稽核追蹤和協同合作。

實作適當的 RBAC：使用 AWS Identity Center 整合來控制誰可以存取和管理 Argo CD 中的應用程式。Argo CD 支援對應用程式（部署、Pod、ConfigMaps、秘密）內資源的精細存取控制。

使用 ApplicationSets 進行多環境部署：使用 ApplicationSets 在具有不同組態的多個叢集或命名空間中部署應用程式。

生命週期管理

應用程式同步政策

控制 Argo CD 同步應用程式的方式：

手動同步：應用程式需要手動核准才能同步變更。建議用於生產環境。

自動同步：偵測到 Git 變更時，應用程式會自動同步。常見於開發和預備環境。

自我修復：自動還原對叢集所做的手動變更。確保叢集狀態符合 Git。

刪除：自動刪除從 Git 移除的資源。請謹慎使用 ，因為這可能會刪除資源。

應用程式運作狀態

Argo CD 會持續監控應用程式運作狀態：

- 運作狀態良好：所有資源都如預期般執行
- 進行中：正在建立或更新資源
- 降級：某些資源運作狀態不佳
- 已暫停：應用程式已暫停
- 遺失：叢集中缺少資源

同步視窗

設定同步時段以控制應用程式可以同步的時間：

- 僅允許在維護時段進行同步
- 上班時間的區塊同步
- 排程特定時間的自動同步
- 在您需要進行變更並停止任何同步的情況下 （突破性案例），使用同步時段

Webhook 組態可加快同步速度

根據預設，Argo CD 每 6 分鐘輪詢一次 Git 儲存庫，以偵測變更。如需更靈敏的部署，請將 Git Webhook 設定為在推送變更時觸發立即同步。

Webhooks 提供數種優點：

- 推送程式碼時的立即同步回應 （秒與分鐘）
- 減少輪詢負荷並改善系統效能
- 更有效率地使用 API 速率限制
- 透過更快的意見回饋獲得更好的使用者體驗

Webhook 端點

Webhook URL 遵循模式 `${serverUrl}/api/webhook`，其中 `serverUrl`是您的 Argo CD 伺服器 URL。若要尋找您的伺服器 URL，請參閱 the section called “Argo CD 端點 URL”。

例如，如果您的 Argo CD 伺服器 URL 為 `https://my-argocd-dc855fdf-111122223333.eks-capabilities.us-west-2.amazonaws.com`，則 Webhook URL 為：

```
https://my-argocd-dc855fdf-111122223333.eks-capabilities.us-west-2.amazonaws.com/api/webhook
```

依 Git 提供者設定 Webhook

GitHub：在您的儲存庫設定中，使用 Argo CD Webhook URL 新增 Webhook。將內容類型設定為 `application/json`，然後選取「僅推送事件」。

GitLab：在您的專案設定中，使用 Argo CD Webhook URL 新增 Webhook。啟用「推送事件」和選用的「標籤推送事件」。

Bitbucket：在您的儲存庫設定中，使用 Argo CD Webhook URL 新增 Webhook。選取「儲存庫推送」作為觸發。

CodeCommit：建立在 CodeCommit 儲存庫狀態變更時觸發的 Amazon EventBridge 規則，並將通知傳送至 Argo CD Webhook 端點。

如需詳細 Webhook 組態指示，請參閱 Argo CD Webhook 組態。

> **Note**
>
> Webhook 補充輪詢 - 它們不會取代它。Argo CD 會繼續輪詢儲存庫做為備用機制，以防 Webhook 通知遺失。

後續步驟

- the section called “使用 Argo CD” - 了解如何建立和管理 Argo CD 應用程式
- the section called “疑難排解” - 對 Argo CD 問題進行故障診斷
- the section called “使用 功能” - 管理您的 Argo CD 功能資源

對 Argo CD 功能的問題進行故障診斷

> **Note**
>
> EKS 功能是完全受管的，並在叢集外部執行。您無法直接存取控制器命名空間。您可以設定控制器日誌交付，以掌握控制器行為。請參閱 the section called “功能控制器日誌”。故障診斷著重於功能運作狀態、應用程式狀態和組態。

功能為 ACTIVE，但應用程式未同步

如果您的 Argo CD 功能顯示ACTIVE狀態，但應用程式未同步，請檢查功能運作狀態和應用程式狀態。

檢查功能運作狀態：

您可以在 EKS 主控台或使用 AWS CLI 檢視功能運作狀態和狀態問題。

主控台：

1. 在以下網址開啟 Amazon EKS 主控台：https://console.aws.amazon.com/eks/home#/clusters。
2. 選取您的叢集名稱。
3. 選擇可觀測性索引標籤。
4. 選擇監控叢集。
5. 選擇功能索引標籤以檢視所有功能的運作狀態和狀態。

AWS CLI：

```
# View capability status and health
aws eks describe-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-argocd

# Look for issues in the health section
```

常見原因：

- 未設定儲存庫：Git 儲存庫未新增至 Argo CD

- 驗證失敗：SSH 金鑰、字符或 CodeCommit 登入資料無效
- 未建立應用程式：叢集中不存在應用程式資源
- 同步政策：需要手動同步（未啟用自動同步）
- IAM 許可：缺少 CodeCommit 或 Secrets Manager 的許可

檢查應用程式狀態：

```
# List applications
kubectl get application -n argocd

# View sync status
kubectl get application my-app -n argocd -o jsonpath='{.status.sync.status}'

# View application health
kubectl get application my-app -n argocd -o jsonpath='{.status.health}'
```

檢查應用程式條件：

```
# Describe application to see detailed status
kubectl describe application my-app -n argocd

# View application health
kubectl get application my-app -n argocd -o jsonpath='{.status.health}'
```

應用程式停滯在「進行中」狀態

如果應用程式顯示 `Progressing`但從未達到 `Healthy`，請檢查應用程式的資源狀態和事件。

檢查資源運作狀態：

```
# View application resources
kubectl get application my-app -n argocd -o jsonpath='{.status.resources}'

# Check for unhealthy resources
kubectl describe application my-app -n argocd | grep -A 10 "Health Status"
```

常見原因：

- 部署未就緒：Pod 無法啟動或整備探查失敗

- 資源相依性：等待其他資源準備就緒的資源
- 映像提取錯誤：無法存取容器映像
- 資源不足：叢集缺乏 Pod 的 CPU 或記憶體

驗證目標叢集組態（適用於多叢集設定）：

```
# List registered clusters
kubectl get secret -n argocd -l argocd.argoproj.io/secret-type=cluster

# View cluster secret details
kubectl get secret cluster-secret-name -n argocd -o yaml
```

儲存庫身分驗證失敗

如果 Argo CD 無法存取您的 Git 儲存庫，請驗證身分驗證組態。

對於 CodeCommit 儲存庫：

驗證 IAM 功能角色具有 CodeCommit 許可：

```
# View IAM policies
aws iam list-attached-role-policies --role-name my-argocd-capability-role
aws iam list-role-policies --role-name my-argocd-capability-role

# Get specific policy details
aws iam get-role-policy --role-name my-argocd-capability-role --policy-name policy-name
```

角色需要儲存庫的`codecommit:GitPull`許可。

對於私有 Git 儲存庫：

確認已正確設定儲存庫登入資料：

```
# Check repository secret exists
kubectl get secret -n argocd repo-secret-name -o yaml
```

確保秘密包含正確的身分驗證憑證 (SSH 金鑰、字符或使用者名稱/密碼）。

對於使用 Secrets Manager 的儲存庫：

```
# Verify IAM Capability Role has Secrets Manager permissions
aws iam list-attached-role-policies --role-name my-argocd-capability-role

# Test secret retrieval
aws secretsmanager get-secret-value --secret-id arn:aws:secretsmanager:region-
code:111122223333:secret:my-secret
```

多叢集部署問題

如果應用程式未部署到遠端叢集，請驗證叢集註冊和存取組態。

檢查叢集註冊：

```
# List registered clusters
kubectl get secret -n argocd -l argocd.argoproj.io/secret-type=cluster

# Verify cluster secret format
kubectl get secret CLUSTER_SECRET_NAME -n argocd -o yaml
```

確保 `server` 欄位包含 EKS 叢集 ARN，而不是 Kubernetes API URL。

驗證目標叢集存取項目：

在目標叢集上，檢查 Argo CD 功能角色是否具有存取項目：

```
# List access entries (run on target cluster or use AWS CLI)
aws eks list-access-entries --cluster-name target-cluster

# Describe specific access entry
aws eks describe-access-entry \
  --cluster-name target-cluster \
  --principal-arn arn:aws:iam::111122223333:role/my-argocd-capability-role
```

檢查跨帳戶的 IAM 許可：

對於跨帳戶部署，請確認 Argo CD 功能角色在目標叢集上有存取項目。受管功能使用 EKS 存取項目進行跨帳戶存取，而非 IAM 角色假設。

如需多叢集組態的詳細資訊，請參閱the section called “註冊叢集”。

增加應用程式同步時間

如果您的應用程式正在同步，但花費的時間超過預期，請使用下列診斷步驟來識別原因。

檢查上次同步時間

透過檢閱應用程式上次同步的時間來確認延遲：

```
# View last sync time for all applications
kubectl get application -n argocd -o jsonpath='{range .items[*]}{.metadata.name}{"\t"}
{.status.operationState.finishedAt}{"\n"}{end}'

# View last sync time for a specific application
kubectl get application my-app -n argocd -o
 jsonpath='{.status.operationState.finishedAt}'
```

檢查應用程式條件

檢閱協調佇列延遲的應用程式條件：

```
# Check conditions on an application
kubectl get application my-app -n argocd -o jsonpath='{.status.conditions}'
```

檢查 targetRevision 組態

使用 的應用程式會在每次遞交至儲存庫時使資訊清單快取`targetRevision: HEAD`失效，這會減慢同步時間：

```
# List applications using HEAD as targetRevision
kubectl get application -n argocd -o jsonpath='{range .items[?
(@.spec.source.targetRevision=="HEAD")]}{.metadata.name}{"\n"}{end}'
```

常見原因

- 無 Webhook 組態：如果沒有 Webhook，Argo CD 會以預設間隔 6 分鐘輪詢儲存庫。這會延遲新遞交的偵測。
- targetRevision 設定為 HEAD：對儲存庫的每個遞交都會使資訊清單快取失效。然後，Argo CD 會在每次對帳時重新產生資訊清單。
- 大型或複雜的 Git 儲存庫：由於要處理的檔案和範本數量， Monorepos 或複雜的 Helm Chart 會導致資訊清單產生緩慢。

- 單一應用程式中大量的 Kubernetes 資源：管理許多資源的應用程式會導致叢集快取同步緩慢，因為 Argo CD 必須追蹤每個資源的狀態。

緩解措施

- 設定 Git Webhook：Webhooks 會在推送變更時立即通知 Argo CD，繞過預設輪詢間隔。如需組態步驟，請參閱 the section called “Argo CD 考量事項”。
- 使用特定分支名稱或遞交 SHAs：`targetRevision`設定為分支名稱或遞交 SHA，而不是`HEAD`在同步之間保留資訊清單快取。
- 分割大型單一儲存庫：將大型儲存庫分割成較小的聚焦儲存庫，以減少資訊清單產生時間。
- 減少每個應用程式的資源：將具有許多 Kubernetes 資源的應用程式分割成多個較小的應用程式，以減少叢集快取同步時間。
- 啟用控制器日誌交付：控制器日誌提供對調校行為和佇列處理的可見性。如需組態步驟，請參閱 the section called “功能控制器日誌”。

應用程式重複同步或無法同步

如果您的應用程式同步，然後立即變成 `OutOfSync`，或者它卡在同步迴圈中，原因通常在 Git 定義和叢集中存在的內容之間偏離。從基準診斷開始。

收集診斷資訊

```
# View current sync and health status
argocd app get my-app

# Show exact fields that differ between Git and live state
argocd app diff my-app

# Check whether the app has ever reached a stable state
argocd app history my-app
```

`argocd app diff` 命令是最有用的起點。它確切顯示哪些欄位導致應用程式出現不同步。

自我管理憑證會造成漂移

cert-manager、OPA Gatekeeper 和 KEDA 等控制器會在執行時間產生憑證。這些執行時間值不在 Git 中，因此 Argo CD 會在每次對帳時偵測偏離。

這些症狀包括：

- 應用程式同步，然後立即顯示 OutOfSync
- 差異顯示 Webhook caBundle 欄位或 TLS Secret data 欄位的變更

若要解決此問題，請ignoreDifferences為受影響的欄位新增 ，並在同步選項RespectIgnoreDifferences中啟用 ：

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: my-app
spec:
  ignoreDifferences:
    - group: admissionregistration.k8s.io
      kind: ValidatingWebhookConfiguration
      jsonPointers:
        - /webhooks/0/clientConfig/caBundle
    - group: ""
      kind: Secret
      jsonPointers:
        - /data/tls.crt
        - /data/tls.key
  syncPolicy:
    syncOptions:
      - RespectIgnoreDifferences=true
```

自我修復會中斷啟動緩慢的工作負載

啟用 selfHeal 時，Argo CD 會在偵測到偏離時重新同步應用程式。如果您的工作負載需要 30–60 秒才能啟動，則在工作負載變成 之前會觸發自我修復Healthy。prune 啟用 後，這可能會縮減部分啟動的資源。

若要解決此問題，請先修正基礎偏離（請參閱憑證案例）。如果偏離不是原因，請考慮針對您僅透過 Git 管理的工作負載停用自我修復：

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: my-app
spec:
```

```
  syncPolicy:
    automated:
      selfHeal: false
      prune: false
```

> **Note**
>
> 自我修復退避計時是一種執行個體層級控制器設定。如果您需要調整自我修復計時，而不是停用它，請開啟 AWS 支援案例。

ApplicationSet 或資源擁有權衝突

如果兩個應用程式或 ApplicationSets 管理相同的 Kubernetes 資源，Argo CD 會顯示 `SharedResourceWarning`。資源永遠不會達到穩定狀態。當共用資源名稱未針對每個環境或叢集設定範圍時，通常會發生這種情況。

若要解決此問題：

- 讓每個擁有者的預期資源是唯一的。將環境或叢集尾碼新增至資源名稱。
- 重新命名 ApplicationSet 時，`preserveResourcesOnDeletion: true`請先設定 以避免現有資源的破壞性縮減：

```
apiVersion: argoproj.io/v1alpha1
kind: ApplicationSet
metadata:
  name: my-appset
spec:
  syncPolicy:
    preserveResourcesOnDeletion: true
```

從資源定稿器卡住刪除

如果應用程式卡在 `Terminating` 狀態或顯示「待刪除的 N 個物件」，則`resources-finalizer.argocd.argoproj.io`定稿器會封鎖移除，直到所有受管資源刪除為止。具有自己無法處理的定稿器的受管資源會無限期地封鎖刪除。

若要確認，請列出具有刪除時間戳記但尚未移除的資源：

```
kubectl get all -n my-namespace -o json | \
```

```
  jq '.items[] | select(.metadata.deletionTimestamp != null) | {name: .metadata.name,
kind: .kind, finalizers: .metadata.finalizers}'
```

若要解決此問題：

- 確定擁有封鎖定稿器的控制器運作狀態良好且正在執行中。
- 如果擁有的控制器運作狀態良好，但尚未處理定案者，請從停滯的資源中移除封鎖定案者。

使用上一個命令列印的最終化器清單，將清單設定為您要保留的最終化器，並只保留封鎖的清單。將`finalizer-a`和 取代`finalizer-b`為這些名稱：

```
kubectl patch resource-kind
            resource-name -n my-namespace \
  --type merge -p '{"metadata":{"finalizers":["finalizer-a","finalizer-b"]}}'
```

如果封鎖定案者是資源上的唯一定案者，請傳遞空清單：`--type merge -p '{"metadata": {"finalizers":[]}}'`。

> **Warning**
>
> 明確設定定案者清單，而不是依位置移除項目。資源可以從多個控制器攜帶定稿器。定案者未按保證順序排列。移除第一個項目可能會刪除錯誤的定稿器，這會保留封鎖的定稿器，而資源仍然卡住。
> 移除定稿器也會略過擁有控制器會執行的任何清除，這可能會讓 AWS 資源落後，而不會在叢集中記錄它們。只有在您確認擁有的控制器無法處理定案者之後，才能執行此操作。

失敗的同步不會自動重試相同的修訂

同步至特定修訂失敗後，Argo CD 不會自動重試相同的修訂。這通常是因為資訊清單瑕疵，例如`ComparisonError`來自重複環境變數索引鍵的 。

檢查應用程式狀態以確認：

```
argocd app get my-app
# Look for: Operation: Sync  Phase: Failed  Revision: <sha>
```

若要解決此問題，請修正 Git 儲存庫中的資訊清單瑕疵，並推送新的遞交。或者，觸發手動同步：

```
argocd app sync my-app
```

Monorepo 遞交流失觸發條件廣泛的再生

如果許多應用程式在同一個儲存庫HEAD上追蹤，則所有應用程式對該儲存庫的任何遞交HEAD都會變更。這會為每個應用程式觸發資訊清單重新產生，即使其檔案未變更也一樣。如需targetRevision和 快取的詳細資訊，請參閱此頁面上的「增加應用程式同步時間」一節。

若要僅將重新產生範圍限定為每個應用程式使用的檔案，請新增manifest-generate-paths註釋：

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: my-app
  annotations:
    argocd.argoproj.io/manifest-generate-paths: /apps/my-app
spec:
  source:
    repoURL: https://github.com/my-org/my-monorepo.git
    targetRevision: HEAD
    path: apps/my-app
```

使用此註釋，Argo CD 只會在指定路徑下的檔案變更時重新產生資訊清單。對於跨應用程式使用的共用程式庫，您可以指定多個以分號 () 分隔的路徑;。

盡可能將 釘選targetRevision至分支名稱或標籤，而非 HEAD。

Kubernetes 預設和變動 Webhook 會導致假體差異

如果您的應用程式在同步OutOfSync後立即顯示，請檢查從未設定的欄位差異（例如terminationGracePeriodSeconds、 dnsPolicy或 /spec/replicas)。Kubernetes API 伺服器或變動 Webhook 會在套用時新增這些欄位。

若要解決由其他控制器管理的欄位（例如 HPA 管理擴展/spec/replicas時），請新增ignoreDifferences：

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: my-app
```

```
spec:
  ignoreDifferences:
    - group: apps
      kind: Deployment
      jsonPointers:
        - /spec/replicas
  syncPolicy:
    syncOptions:
      - RespectIgnoreDifferences=true
```

對於 Kubernetes 預設或變動 Webhook 新增的欄位，您可以在應用程式上啟用伺服器端差異：

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: my-app
  annotations:
    argocd.argoproj.io/compare-options: ServerSideDiff=true,IncludeMutationWebhook=true
```

伺服器端 diff 會針對每個資源執行試轉套用，這會增加 Kubernetes API 伺服器的負載。在廣泛啟用之前，先在少數應用程式上測試此項目。

高流失控制器擁有的資源

有些控制器會產生大量短期或經常更新的資源。範例包括 Karpenter 節點物件、Cilium 身分和端點物件，以及 Kyverno 政策報告。如果這些資源產生大量監看事件並導致同步流失，您可以透過排除這些資源類型或篩選監看事件來減少負載。這些變更需要執行個體層級控制器組態。

在 受管功能上，開啟 AWS 支援案例，請求這些資源類型的資源排除或監看事件篩選。

最佳實務

- 先使用應用程式差異：執行 `argocd app diff`做為任何重複同步問題的第一個診斷步驟。它會顯示偏離的確切原因。
- 偏好窄型 ignoreDifferences：以特定資源類型的特定欄位為目標。避免可遮罩實際組態偏離的廣泛忽略規則。
- 配對 ignoreDifferences with RespectIgnoreDifferences：一律新增`RespectIgnoreDifferences=true`同步選項。如果沒有它，同步仍會覆寫忽略的欄位。
- 讓資源名稱保持唯一：範圍每個環境和叢集的資源名稱，以避免應用程式或 ApplicationSets 之間的所有權衝突。

- 請謹慎使用 prune 和 selfHeal：請勿在需要很長時間才能啟動的工作負載上啟用兩者。自我修復可以在資源正常運作之前將其銷毀。
- Pin targetRevision 和範圍資訊清單路徑：對於大型共用儲存庫中的應用程式，請使用分支或標籤，而不是 HEAD並新增`manifest-generate-paths`註釋。

聯絡 AWS Support 的時機

在下列情況下開啟 AWS 支援案例：

- 執行個體層級控制器調校似乎是必要的（處理器計數、自我修復時間或資源排除）。
- 儲存庫伺服器或控制器容量似乎不足以滿足您的應用程式計數。
- 工作負載組態、偏離、擁有權或定案者不會解釋行為。

在支援案例中包含受影響應用程式的 `argocd app get`和 `argocd app diff` 輸出。

後續步驟

- the section called “Argo CD 考量事項” - Argo CD 考量事項和最佳實務
- the section called “使用 Argo CD” - 建立和管理 Argo CD 應用程式
- the section called “註冊叢集” - 設定多叢集部署
- the section called “故障診斷功能” - 一般功能故障診斷指引

比較 Argo CD 與自我管理 Argo CD 的 EKS 功能

EKS 的 Argo CD 功能提供在 EKS 中執行的全受管 Argo CD 體驗。如需 EKS 功能與自我管理解決方案的一般比較，請參閱 the section called “考量事項”。本主題著重於 Argo CD 的特定差異，包括身分驗證、多叢集管理和上游功能支援。

與上游 Argo CD 的差異

Argo CD 的 EKS 功能是以上游 Argo CD 為基礎，但其存取、設定和與 AWS 服務整合的方式有所不同。

RBAC 和身分驗證： 功能隨附三個 RBAC 角色（管理員、編輯器、檢視器），並使用 AWS Identity Center 進行身分驗證，而非 Argo CD 的內建身分驗證。透過 功能的 `rbacRoleMapping` 參數設定角色映射，將 Identity Center 群組映射至 Argo CD 角色，而不是透過 Argo CD 的 `argocd-rbac-cm`

ConfigMap。Argo CD UI 使用自己的直接 URL 託管（在叢集的功能索引標籤下的 EKS 主控台中找到），而 API 存取使用透過 IAM 的 AWS 身分驗證和授權。

叢集組態： 功能不會自動設定本機叢集或hub-and-spoke式拓撲。您可以設定部署目標叢集和 EKS 存取項目。此功能僅支援使用 EKS 叢集 ARNs（不支援 Kubernetes API 伺服器 URLs) 做為部署目標的 Amazon EKS 叢集。此功能不會自動新增本機叢集 (`kubernetes.default.svc`) 做為部署目標，若要部署至建立此功能的相同叢集，請使用其 ARN 明確註冊該叢集。

簡化的遠端叢集存取： 功能使用 EKS 存取項目授予遠端叢集的 Argo CD 存取，以簡化多叢集部署，無需設定服務帳戶的 IAM 角色 (IRSA) 或設定跨帳戶 IAM 角色假設。此功能也提供全私有 EKS 叢集的透明存取，而不需要 VPC 對等互連或特殊聯網組態，可自動AWS 管理 Argo CD 功能與私有遠端叢集之間的連線。

直接 AWS 服務整合： 功能透過功能角色的 IAM 許可，提供與 AWS 服務的直接整合。您可以直接在應用程式資源中參考 CodeCommit 儲存庫、ECR Helm Chart 和 CodeConnections，而無需建立儲存庫組態。這可簡化身分驗證，無需管理 AWS 服務的個別登入資料。如需詳細資訊，請參閱 the section called “設定儲存庫”。

命名空間支援： 功能需要您指定單一命名空間，其中必須建立 Argo CD Application、ApplicationSet 和 AppProject 自訂資源。

> **Note**
>
> 此命名空間限制僅適用於 Argo CD 自己的自訂資源 (Application、ApplicationSet、AppProject)。您的應用程式工作負載可以部署到任何目標叢集中的任何命名空間。例如，如果您使用命名空間 建立 功能`argocd`，則必須在`argocd`命名空間中建立所有應用程式 CRs，但這些應用程式可以將工作負載部署到 `default`、`staging`、`production`或任何其他命名空間。

> **Note**
>
> 受管功能對 CLI 用量和 AppProject 組態有特定需求：
>
> - 使用 Argo CD CLI 時，請使用命名空間字首指定應用程式： `argocd app sync namespace/appname`
> - AppProject 資源必須指定 `.spec.sourceNamespaces`來定義專案可以監看哪些命名空間的應用程式（通常設定為您在建立 功能時指定的命名空間）

- 資源追蹤註釋使用 格式： `namespace_appname:group/kind:namespace/name`

不支援的功能：下列功能不適用於 受管功能：

- 用於產生自訂資訊清單的 Config Management 外掛程式 (CMPs)
- 通知控制器
- 自訂 SSO 供應商（僅支援 AWS Identity Center，包括透過 AWS Identity Center 的第三方聯合身分）
- UI 擴充功能和自訂橫幅
- 直接存取 `argocd-params`和大多數組態 ConfigMaps（支援`argocd-cm`欄位子集。 請參閱 the section called “組態設定”)
- 修改同步逾時（修正為 120 秒）

相容性：Application 和 ApplicationSets 的運作方式與上游 Argo CD 相同，而不會變更資訊清單。該功能使用相同的 Kubernetes APIs 和 CRDs，因此 等工具`kubectl`的運作方式相同。功能完全支援應用程式和 ApplicationSets、GitOps 工作流程與自動同步、多叢集部署、同步政策（自動化、剪下、自我修復）、同步波和勾點、標準 Kubernetes 資源的運作狀態評估、轉返功能、Git 儲存庫來源 (HTTPS 和 SSH)、Helm、Kutomize 和純 YAML 資訊清單、GitHub 應用程式憑證、多租用專案，以及資源排除和包含項目。

搭配 受管功能使用 Argo CD CLI

Argo CD CLI 的運作方式與大多數操作的上游 Argo CD 相同，但身分驗證和叢集註冊不同。

先決條件

按照上游安裝說明安裝 Argo CD CLI。

組態

使用環境變數設定 CLI：

1. 從 EKS 主控台（叢集的功能索引標籤下）或使用 AWS CLI 取得 Argo CD 伺服器 URL。必須移除字`https://`首：

```
export ARGOCD_SERVER=$(aws eks describe-capability \
```

```
  --cluster-name my-cluster \
  --capability-name my-argocd \
  --query 'capability.configuration.argoCd.serverUrl' \
  --output text \
  --region region-code | sed 's|^https://||')
```

2. 從 Argo CD UI 產生帳戶字符 (設定 → 帳戶 → 管理員 → 產生新字符)，然後將其設定為環境變數：

```
export ARGOCD_AUTH_TOKEN="your-token-here"
```

> **Important**
>
> 此組態使用管理員帳戶字符進行初始設定和開發工作流程。對於生產使用案例，請使用專案範圍的角色和字符來遵循最低權限原則。如需設定專案角色和 RBAC 的詳細資訊，請參閱 the section called “設定 許可”。

1. 設定所需的 gRPC 選項：

```
export ARGOCD_OPTS="--grpc-web"
```

設定這些環境變數後，您可以使用 Argo CD CLI，無需 `argocd login`命令。

主要差異

受管功能具有下列 CLI 限制：

- `argocd admin` 不支援 命令 （需要直接 Pod 存取）
- `argocd login` 不支援 （請改用 帳戶或專案字符）
- `argocd cluster add` 需要具有 EKS 叢集 ARN 的 `--aws-cluster-name`旗標

範例：註冊叢集

註冊 EKS 叢集以進行應用程式部署：

```
# Get the cluster ARN
CLUSTER_ARN=$(aws eks describe-cluster \
```

```
  --name my-cluster \
  --query 'cluster.arn' \
  --output text)

# Register the cluster
argocd cluster add $CLUSTER_ARN \
  --aws-cluster-name $CLUSTER_ARN \
  --name in-cluster \
  --project default
```

如需完整的 Argo CD CLI 文件，請參閱 Argo CD CLI 參考。

遷移路徑

您可以從自我管理的 Argo CD 遷移到 受管功能：

1. 檢閱您目前的 Argo CD 組態是否有不支援的功能（通知控制器、CMPs、自訂運作狀態檢查、UI 延伸）
2. 將您的自我管理 Argo CD 控制器擴展至零個複本，以防止衝突
3. 在叢集上建立 Argo CD 功能資源
4. 匯出現有的應用程式、ApplicationSets 和 AppProjects
5. 遷移儲存庫登入資料、叢集秘密和儲存庫登入資料範本（已複製）
6. 如果使用 GPG 金鑰、TLS 憑證或 SSH 已知主機，請一併遷移這些組態
7. 更新`destination.server`欄位以使用叢集名稱或 EKS 叢集 ARNs
8. 將它們套用至受管 Argo CD 執行個體
9. 驗證應用程式是否正確同步
10. 停用自我管理的 Argo CD 安裝

受管功能使用相同的 Argo CD APIs 和資源定義，因此您現有的資訊清單會以最少的修改運作。

後續步驟

- the section called “建立 Argo CD 功能” - 建立 Argo CD 功能資源
- the section called “使用 Argo CD” - 部署您的第一個應用程式
- the section called “Argo CD 考量事項” - 設定 AWS Identity Center 整合

資源合成與 kro (Kube Resource Orchestrator)

kro (Kube Resource Orchestrator) 是一種開放原始碼的 Kubernetes 原生專案，可讓您使用簡單且直接的組態來定義自訂 Kubernetes APIs。使用 kro，您可以輕鬆設定新的自訂 APIs，以建立一組 Kubernetes 物件及其之間的邏輯操作。

透過 EKS 功能，kro 完全由 管理 AWS，無需在叢集上安裝、維護和擴展 kro 控制器。

kro 的運作方式

kro 推出名為 (RGD) 的自訂資源定義 `ResourceGraphDefinition`(CRD)，可簡單且簡化地建立自訂 Kubernetes APIs。當您建立 時`ResourceGraphDefinition`，kro 會使用原生 Kubernetes 延伸模組來建立和管理叢集中的新 APIs。從這個單一資源規格中，kro 會根據您的規格為您建立新的 CRD 並註冊，並調整以管理新定義的自訂資源。

RGDs可以包含多個資源，而 kro 會判斷相互依存性和資源排序，因此您不需要。您可以使用簡單的語法，將組態從一個資源注入到另一個資源，大幅簡化合成並消除叢集中對「黏附」運算子的需求。透過 kro，您的自訂資源可以包含原生 Kubernetes 資源，以及安裝在叢集中的任何自訂資源定義 (CRDs)。

kro 支援單一主要資源類型：

- ResourceGraphDefinition (RGD)：定義 Kubernetes 自訂資源，封裝一或多個基礎原生或自訂 Kubernetes 資源

除了此資源之外，kro 還將建立和管理使用它建立的自訂資源的生命週期，以及其所有組成資源。

kro 與 AWS Controllers for Kubernetes (ACK) 無縫整合，可讓您使用 AWS 資源編寫工作負載資源，以建立更高階的抽象概念。這可讓您建立自己的雲端建置區塊，簡化資源管理，並根據組織標準，使用預設和不可變的組態設定啟用可重複使用的模式。

kro 的優點

kro 可讓平台團隊建立自訂 Kubernetes APIs，將多個資源組成更高階的抽象概念。這可讓開發人員使用簡單、標準化和版本控制的自訂資源來部署複雜的應用程式，進而簡化資源管理。您可以定義常用資源組合的可重複使用模式，讓整個組織建立一致的資源。

kro 在 Kubernetes 中使用通用表達式語言 (CEL)，在資源之間傳遞值並整合條件式邏輯，提供資源合成的彈性。您可以將 ACK 管理的 Kubernetes 資源 AWS 和資源組合為統一APIs，以啟用完整的應用程式和基礎設施定義。

kro 透過 Kubernetes 資訊清單支援宣告式組態，讓 GitOps 工作流程和基礎設施成為與現有開發程序無縫整合的程式碼實務。作為 EKS 受管功能的一部分，kro 完全由 管理 AWS，無需在叢集上安裝、設定和維護 kro 控制器。

範例：建立 ResourceGraphDefinition

下列範例顯示使用 部署和服務建立 Web 應用程式的簡單 `ResourceGraphDefinition` ：

```
apiVersion: kro.run/v1alpha1
kind: ResourceGraphDefinition
metadata:
  name: web-application
spec:
  schema:
    apiVersion: v1alpha1
    kind: WebApplication
    spec:
      name: string
      replicas: integer | default=3
  resources:
    - id: deployment
      template:
        apiVersion: apps/v1
        kind: Deployment
        metadata:
          name: ${schema.spec.name}
        spec:
          replicas: ${schema.spec.replicas}
    - id: service
      template:
        apiVersion: v1
        kind: Service
        metadata:
          name: ${schema.spec.name}
```

當使用者建立`WebApplication`自訂資源的執行個體時，kro 會自動建立對應的部署和服務資源，並管理其生命週期與自訂資源。

與其他 EKS 受管功能整合

kro 與其他 EKS 受管功能整合。

- AWS Kubernetes (ACK) 的控制器：使用 kro 將 ACK 資源編寫為更高層級的抽象，簡化 AWS 資源管理。
- Argo CD：使用 Argo CD 管理跨多個叢集的 kro 自訂資源部署，為您的平台建置區塊和應用程式堆疊啟用 GitOps 工作流程。

kro 入門

若要開始使用 kro 的 EKS 功能：

1. 透過 AWS 主控台、 AWS CLI 或您偏好的基礎設施做為程式碼工具，在您的 EKS 叢集上建立 kro 功能資源。
2. 建立 ResourceGraphDefinitions (RGDs)，以定義您的自訂 APIs和資源組成。
3. 套用自訂資源的執行個體，以佈建和管理基礎 Kubernetes AWS 和資源。

建立 kro 功能

本主題說明如何在 Amazon EKS 叢集上建立 kro 功能。

先決條件

在建立 kro 功能之前，請確定您有：

- 執行支援 Kubernetes 版本的現有 Amazon EKS 叢集 （支援標準和延伸支援中的所有版本）
- 足夠的 IAM 許可，可在 EKS 叢集上建立功能資源
- （針對 CLI/eksctl) 安裝和設定適當的 CLI 工具

> **Note**
>
> 與 ACK 和 Argo CD 不同， kro 不需要信任政策以外的其他 IAM 許可。 kro 完全在您的叢集內運作，而且不會進行 AWS API 呼叫。不過，您仍然需要為 IAM 能力角色提供適當的信任政策。如需為 kro 設定 Kubernetes RBAC 許可的資訊，請參閱 the section called “設定 許可”。

選擇您的工具

您可以使用、 AWS 管理主控台 AWS CLI 或 eksctl 建立 kro 功能：

- the section called “主控台” - 使用 主控台進行引導式體驗
- the section called “ AWS CLI” - 使用 AWS CLI 進行指令碼編寫和自動化
- the section called “eksctl” - 使用 eksctl 獲得 Kubernetes 原生體驗

當您建立 kro 功能時會發生什麼情況

當您建立 kro 功能時：

1. EKS 會建立 kro 功能服務，並將其設定為監控和管理叢集中的資源
2. 您的叢集中已安裝自訂資源定義 (CRDs)
3. 會自動為您的 IAM 功能角色建立存取項目`AmazonEKSKROPolicy`，其中 授予管理 ResourceGraphDefinitions 及其執行個體的許可 （ 請參閱 the section called “EKS 功能的考量事項”)
4. 該功能假設您提供的 IAM 功能角色 （ 僅用於信任關係 ）
5. kro 開始觀察`ResourceGraphDefinition`資源及其執行個體
6. 功能狀態從 變更為 `CREATING` `ACTIVE`

啟用後，您可以建立 ResourceGraphDefinitions 來定義自訂 APIs並建立這些 APIs的執行個體。

> **Note**
>
> 自動建立的存取項目包含 `AmazonEKSKROPolicy`，授予 kro 許可來管理 ResourceGraphDefinitions 及其執行個體。若要允許 kro 建立 ResourceGraphDefinitions 中定義的基礎 Kubernetes 資源 （ 例如部署、服務或 ACK 資源 ），您必須設定其他存取項目政策。若要進一步了解存取項目以及如何設定其他許可，請參閱 the section called “設定 許可”和 the section called “EKS 功能的考量事項”。

後續步驟

建立 kro 功能之後：

- the section called “kro 概念” - 了解 kro 概念和資源合成

使用主控台建立 kro 功能

使用 AWS 管理主控台在 Amazon EKS 叢集上建立 kro 功能。

建立 kro 功能

1. 在以下網址開啟 Amazon EKS 主控台：https://console.aws.amazon.com/eks/home#/clusters。
2. 選取您的叢集名稱以開啟叢集詳細資訊頁面。
3. 選擇功能索引標籤。
4. 在左側導覽中，選擇 kro (Kube Resource Orchestrator)。
5. 選擇建立 kro 功能。
6. 對於 IAM 功能角色：
   - 如果您已經有 IAM 功能角色，請從下拉式清單中選取它
   - 如果您需要建立角色，請選擇建立 kro 角色

     這會在預先填入信任政策的新標籤中開啟 IAM 主控台。此角色不需要額外的 IAM 許可，因為 kro 完全在您的叢集中運作。

     建立角色後，返回 EKS 主控台並自動選取角色。

     > **Note**
     >
     > 與 ACK 和 Argo CD 不同， kro 不需要信任政策以外的其他 IAM 許可。 kro 完全在您的叢集內運作，而且不會進行 AWS API 呼叫。

7. 選擇建立。

功能建立程序開始。

驗證功能是否作用中

1. 在功能索引標籤上，檢視 kro 功能狀態。
2. 等待狀態從 變更為 `CREATING` `ACTIVE`。
3. 啟用後，此功能即可使用。

如需功能狀態和故障診斷的資訊，請參閱 the section called “使用 功能”。

授予管理 Kubernetes 資源的許可

當您建立 kro 功能時，系統會自動使用 建立 EKS 存取項目`AmazonEKSKROPolicy`，允許 kro 管理 ResourceGraphDefinitions 及其執行個體。不過，預設不會授予許可來建立 ResourceGraphDefinitions 中定義的基礎 Kubernetes 資源 ConfigMaps 等）。 ResourceGraphDefinitions

此刻意設計遵循最低權限原則 - 不同的 ResourceGraphDefinitions 需要不同的許可。您必須根據 ResourceGraphDefinitions 將管理的資源，明確設定 kro 所需的許可。

若要快速入門、測試或開發環境，請使用 `AmazonEKSClusterAdminPolicy`：

1. 在 EKS 主控台中，導覽至叢集的存取索引標籤。
2. 在存取項目下，尋找 kro 功能角色的項目 （它將具有您先前建立的角色 ARN)。
3. 選擇存取項目以開啟其詳細資訊。
4. 在存取政策區段中，選擇關聯存取政策。
5. `AmazonEKSClusterAdminPolicy` 從政策清單中選取 。
6. 針對存取範圍，選取叢集。
7. 選擇關聯。

> **Important**
>
> `AmazonEKSClusterAdminPolicy` 授予建立和管理所有 Kubernetes 資源的廣泛許可，包括跨所有命名空間建立任何資源類型的能力。這對於開發和 POCs 來說很方便，但不應用於生產環境。對於生產，請建立自訂 RBAC 政策，僅授予 ResourceGraphDefinitions 將管理之特定資源所需的許可。如需設定最低權限許可的指引，請參閱 the section called “設定 許可”和 the section called “EKS 功能的考量事項”。

確認可用的自訂資源

功能處於作用中狀態後，請確認叢集中是否有可用的 kro 自訂資源。

使用主控台

1. 在 Amazon EKS 主控台中導覽至您的叢集
2. 選擇資源索引標籤

3. 選擇延伸模組
4. 選擇 CustomResourceDefinitions

您應該會看到列出的ResourceGraphDefinition資源類型。

使用 kubectl

```
kubectl api-resources | grep kro.run
```

您應該會看到列出的ResourceGraphDefinition資源類型。

後續步驟

- the section called “kro 概念” - 了解 kro 概念和資源合成
- the section called “使用 功能” - 管理您的 kro 功能資源

使用 CLI 建立 kro AWS 功能

使用 CLI 在 Amazon EKS 叢集上建立 kro AWS 功能。此程序會逐步引導您建立 IAM 角色，並驗證功能是否處於作用中狀態。

先決條件

- AWS CLI – 版本 2.12.3 或更新版本。若要檢查您的版本，請執行 aws --version。如需詳細資訊，請參閱《 AWS 命令列界面使用者指南》中的安裝。
- **kubectl** ：命令列工具，適用於使用 Kubernetes 叢集。如需詳細資訊，請參閱the section called “設定 kubectl 和 eksctl”。

步驟 1：建立 IAM 功能角色

建立信任政策檔案：

```
cat > kro-trust-policy.json << 'EOF'
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
```

```
      "Principal": {
        "Service": "capabilities.eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ]
    }
  ]
}
EOF
```

建立 IAM 角色：

```
aws iam create-role \
  --role-name KROCapabilityRole \
  --assume-role-policy-document file://kro-trust-policy.json
```

> **Note**
>
> 與 ACK 和 Argo CD 不同，kro 不需要額外的 IAM 許可。 kro 完全在您的叢集內運作，而且不會進行 AWS API 呼叫。只有建立與 EKS 功能服務的信任關係時，才需要此角色。

步驟 2：建立 kro 功能

在叢集上建立 kro 功能資源。將 *region-code* 取代為您的叢集所在的 AWS 區域 （例如 us-west-2），並將 *my-cluster* 取代為您的叢集名稱。

```
aws eks create-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-kro \
  --type KRO \
  --role-arn arn:aws:iam::$(aws sts get-caller-identity --query Account --output
 text):role/KROCapabilityRole \
  --delete-propagation-policy RETAIN
```

命令會立即傳回，但當 EKS 建立所需的功能基礎設施和元件時，功能需要一些時間才會變成作用中。EKS 會在建立叢集時，在叢集中安裝與此功能相關的 Kubernetes 自訂資源定義。

> **Note**
>
> 如果您收到叢集不存在或您沒有許可的錯誤，請驗證：
>
> - 叢集名稱正確
> - 您的 AWS CLI 已設定為正確的區域
> - 您擁有必要的 IAM 許可

步驟 3：確認功能處於作用中狀態

等待 功能變成作用中。將 *region-code* 取代為您的叢集所在的 AWS 區域，並將 *my-cluster* 取代為您的叢集名稱。

```
aws eks describe-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-kro \
  --query 'capability.status' \
  --output text
```

當狀態顯示 時，此功能已就緒ACTIVE。

您也可以檢視完整的功能詳細資訊：

```
aws eks describe-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-kro
```

步驟 4：授予管理 Kubernetes 資源的許可

當您建立 kro 功能時，系統會自動使用 建立 EKS 存取項目AmazonEKSKROPolicy，允許 kro 管理 ResourceGraphDefinitions 及其執行個體。不過，預設不會授予許可來建立 ResourceGraphDefinitions 中定義的基礎 Kubernetes 資源 ConfigMaps 等）。 ResourceGraphDefinitions

此刻意設計遵循最低權限原則 - 不同的 ResourceGraphDefinitions 需要不同的許可。例如：* 僅建立 ConfigMaps 和 Secrets 的 ResourceGraphDefinition 需要與建立部署和服務者不同的許可 * 建立 ACK 資源的 ResourceGraphDefinition 需要這些特定自訂資源的許可 * 有些 ResourceGraphDefinitions 可能只讀取現有資源，而不建立新的資源

您必須根據 ResourceGraphDefinitions 將管理的資源，明確設定 kro 所需的許可。

快速設定

若要快速入門、測試或開發環境，請使用 AmazonEKSClusterAdminPolicy：

取得功能角色 ARN：

```
CAPABILITY_ROLE_ARN=$(aws eks describe-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-kro \
  --query 'capability.roleArn' \
  --output text)
```

關聯叢集管理員政策：

```
aws eks associate-access-policy \
  --region region-code \
  --cluster-name my-cluster \
  --principal-arn $CAPABILITY_ROLE_ARN \
  --policy-arn arn:aws:eks::aws:cluster-access-policy/AmazonEKSClusterAdminPolicy \
  --access-scope type=cluster
```

> **Important**
>
> AmazonEKSClusterAdminPolicy 授予建立和管理所有 Kubernetes 資源的廣泛許可，包括跨所有命名空間建立任何資源類型的能力。這對於開發和 POCs 來說很方便，但不應用於生產環境。對於生產，請建立自訂 RBAC 政策，僅授予 ResourceGraphDefinitions 將管理之特定資源所需的許可。如需設定最低權限許可的指引，請參閱 the section called “設定 許可”和 the section called “EKS 功能的考量事項”。

步驟 5：確認可用的自訂資源

功能處於作用中狀態後，請確認叢集中是否有可用的 kro 自訂資源：

```
kubectl api-resources | grep kro.run
```

您應該會看到列出的ResourceGraphDefinition資源類型。

後續步驟

- the section called “kro 概念” - 了解 kro 概念和資源合成
- the section called “使用 功能” - 管理您的 kro 功能資源

使用 eksctl 建立 kro 功能

使用 eksctl 在 Amazon EKS 叢集上建立 kro 功能。

> **Note**
> 下列步驟需要 eksctl 版本 `0.215.0` 或更新版本。若要檢查您的版本，請執行 `eksctl version`。

步驟 1：建立 IAM 功能角色

建立信任政策檔案：

```
cat > kro-trust-policy.json << 'EOF'
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "capabilities.eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ]
    }
  ]
}
EOF
```

建立 IAM 角色：

```
aws iam create-role \
  --role-name KROCapabilityRole \
```

```
  --assume-role-policy-document file://kro-trust-policy.json
```

> **Note**
>
> 與 ACK 和 Argo CD 不同， kro 不需要信任政策以外的其他 IAM 許可。 kro 完全在您的叢集內運作，而且不會進行 AWS API 呼叫。

步驟 2：建立 kro 功能

使用 eksctl 建立 kro 功能。將 *region-code* 取代為您的叢集所在的 AWS 區域，並將 *my-cluster* 取代為您的叢集名稱。

```
eksctl create capability \
  --region region-code \
  --cluster my-cluster \
  --name my-kro \
  --type KRO \
  --role-arn arn:aws:iam::111122223333:role/KROCapabilityRole
```

命令會立即傳回，但 功能需要一些時間才會變成作用中。

步驟 3：確認功能處於作用中狀態

檢查功能狀態。將 *region-code* 取代為您的叢集所在的 AWS 區域，並將 *my-cluster* 取代為您的叢集名稱。

```
eksctl get capability \
  --region region-code \
  --cluster my-cluster \
  --name my-kro
```

當狀態顯示 時，此功能已就緒`ACTIVE`。

步驟 4：授予管理 Kubernetes 資源的許可

根據預設，kro 只能建立和管理 ResourceGraphDefinitions 及其執行個體。若要允許 kro 建立和管理 ResourceGraphDefinitions 中定義的基礎 Kubernetes 資源，請將`AmazonEKSClusterAdminPolicy`存取政策與功能的存取項目建立關聯。

取得功能角色 ARN：

```
CAPABILITY_ROLE_ARN=$(aws eks describe-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-kro \
  --query 'capability.roleArn' \
  --output text)
```

關聯叢集管理員政策：

```
aws eks associate-access-policy \
  --region region-code \
  --cluster-name my-cluster \
  --principal-arn $CAPABILITY_ROLE_ARN \
  --policy-arn arn:aws:eks::aws:cluster-access-policy/AmazonEKSClusterAdminPolicy \
  --access-scope type=cluster
```

> **Important**
>
> AmazonEKSClusterAdminPolicy 授予建立和管理所有 Kubernetes 資源的廣泛許可，旨在簡化入門。對於生產用途，請建立更嚴格的 RBAC 政策，僅授予 ResourceGraphDefinitions 將管理之特定資源所需的許可。如需設定最低權限許可的指引，請參閱 the section called “設定 許可”和 the section called “EKS 功能的考量事項”。

步驟 5：確認可用的自訂資源

功能處於作用中狀態後，請確認叢集中是否有可用的 kro 自訂資源：

```
kubectl api-resources | grep kro.run
```

您應該會看到列出的ResourceGraphDefinition資源類型。

後續步驟

- the section called “kro 概念” - 了解 kro 概念和資源合成
- the section called “使用 功能” - 管理您的 kro 功能資源

kro 概念

kro 可讓平台團隊建立自訂 Kubernetes APIs，將多個資源組成更高階的抽象概念。本主題會逐步解說實際範例，然後說明使用 kro 的 EKS 功能時需要了解的核心概念。

kro 入門

建立 kro 功能後（請參閱 the section called “建立 kro 功能”），您可以在叢集中使用 ResourceGraphDefinitions 開始建立自訂 APIs。

以下是建立簡單 Web 應用程式抽象化的完整範例：

```
apiVersion: kro.run/v1alpha1
kind: ResourceGraphDefinition
metadata:
  name: webapplication
spec:
  schema:
    apiVersion: v1alpha1
    kind: WebApplication
    group: kro.run
    spec:
      name: string | required=true
      image: string | default="nginx:latest"
      replicas: integer | default=3
  resources:
  - id: deployment
    template:
      apiVersion: apps/v1
      kind: Deployment
      metadata:
        name: ${schema.spec.name}
      spec:
        replicas: ${schema.spec.replicas}
        selector:
          matchLabels:
            app: ${schema.spec.name}
        template:
          metadata:
            labels:
              app: ${schema.spec.name}
          spec:
            containers:
```

```
            - name: app
              image: ${schema.spec.image}
              ports:
              - containerPort: 80
  - id: service
    template:
      apiVersion: v1
      kind: Service
      metadata:
        name: ${schema.spec.name}
      spec:
        selector:
          app: ${schema.spec.name}
        ports:
        - protocol: TCP
          port: 80
          targetPort: 80
```

套用此 ResourceGraphDefinition 後，應用程式團隊可以使用簡化的 API 建立 Web 應用程式：

```
apiVersion: kro.run/v1alpha1
kind: WebApplication
metadata:
  name: my-app
spec:
  name: my-app
  replicas: 5
```

kro 會自動使用適當的組態建立部署和服務。由於 `image` 未指定，因此會使用結構描述`nginx:latest`中的預設值。

核心概念

> **Important**
>
> kro 會在建立時間驗證 ResourceGraphDefinitions，而不是在執行時間。當您建立 RGD 時，kro 會驗證 CEL 語法、根據實際 Kubernetes 結構描述類型檢查表達式、驗證欄位是否存在，以及偵測循環相依性。這表示在部署任何執行個體之前，建立 RGD 時會立即發現錯誤。

ResourceGraphDefinition

ResourceGraphDefinition (RGD) 透過指定下列項目來定義自訂 Kubernetes API：

- 結構描述 - 使用 SimpleSchema 格式的 API 結構 （ 欄位名稱、類型、預設值、驗證 ）
- 資源 - 要建立的基礎 Kubernetes 或 AWS 資源的範本
- 相依性 - 資源彼此的關係 （ 從欄位參考自動偵測 ）

當您套用 RGD 時，kro 會在叢集中註冊新的自訂資源定義 (CRD)。然後，應用程式團隊可以建立自訂 API 的執行個體，然後 kro 會處理建立和管理所有基礎資源。

如需詳細資訊，請參閱 kro 文件中的 ResourceGraphDefinition 概觀。

SimpleSchema 格式

SimpleSchema 提供簡單的方式來定義 API 結構描述，而不需要 OpenAPI 知識：

```
schema:
  apiVersion: v1alpha1
  kind: Database
  spec:
    name: string | required=true description="Database name"
    size: string | default="small" enum=small,medium,large
    replicas: integer | default=1 minimum=1 maximum=5
```

SimpleSchema `integer`支援 `string`、`boolean`、 和 `number`類型，具有 `required`、、`default``minimum`/`maximum`、 `enum`和 等限制`pattern`條件。

如需詳細資訊，請參閱 kro 文件中的 SimpleSchema。

CEL 表達式

kro 使用通用表達式語言 (CEL) 動態參考值，並新增條件式邏輯。CEL 表達式包裝在 ${和 中，}可用兩種方式使用：

獨立表達式 - 整個欄位值是單一表達式：

```
spec:
  replicas: ${schema.spec.replicaCount}  # Expression returns integer
  labels: ${schema.spec.labelMap}        # Expression returns object
```

表達式結果會取代整個欄位值，且必須符合欄位的預期類型。

字串範本 - 內嵌在字串中的一或多個表達式：

```
metadata:
  name: "${schema.spec.prefix}-${schema.spec.name}"  # Multiple expressions
  annotation: "Created by ${schema.spec.owner}"      # Single expression in string
```

字串範本中的所有表達式都必須傳回字串。使用 `string()`轉換其他類型："replicas-${string(schema.spec.count)}"。

欄位參考 - 使用 存取執行個體規格值`schema.spec`：

```
template:
  metadata:
    name: ${schema.spec.name}-deployment
    namespace: ${schema.metadata.namespace}  # Can also reference metadata
  spec:
    replicas: ${schema.spec.replicas}
```

選用欄位存取 - `?`用於可能不存在的欄位：

```
# For ConfigMaps or Secrets with unknown structure
value: ${configmap.data.?DATABASE_URL}

# For optional status fields
ready: ${deployment.status.?readyReplicas > 0}
```

如果 欄位不存在，則表達式會傳回 `null`而非失敗。

條件式資源 - 只有在符合條件時才包含資源：

```
resources:
- id: ingress
  includeWhen:
    - ${schema.spec.enableIngress == true}
  template:
    # ... ingress configuration
```

`includeWhen` 欄位接受布林表達式的清單。所有條件都必須為 true，才能建立資源。目前，`includeWhen` 只能參考 `schema.spec` 欄位。

轉換 - 使用三元運算子和函數轉換值：

```
template:
  spec:
    resources:
      requests:
        memory: ${schema.spec.size == "small" ? "512Mi" : "2Gi"}

    # String concatenation
    image: ${schema.spec.registry + "/" + schema.spec.imageName}

    # Type conversion
    port: ${string(schema.spec.portNumber)}
```

跨資源參考 - 來自其他資源的參考值：

```
resources:
- id: bucket
  template:
    apiVersion: s3.services.k8s.aws/v1alpha1
    kind: Bucket
    spec:
      name: ${schema.spec.name}-data

- id: configmap
  template:
    apiVersion: v1
    kind: ConfigMap
    data:
      BUCKET_NAME: ${bucket.spec.name}
      BUCKET_ARN: ${bucket.status.ackResourceMetadata.arn}
```

當您在 CEL 表達式中參考另一個資源時，它會自動建立相依性。 kro 會確保先建立參考的資源。

如需詳細資訊，請參閱 kro 文件中的 CEL 表達式。

資源相依性

kro 會自動從 CEL 表達式推斷相依性 - 您不會指定順序，而是描述關係。當一個資源使用 CEL 表達式參考另一個資源時，kro 會建立相依性並決定正確的建立順序。

```
resources:
```

```
- id: bucket
  template:
    apiVersion: s3.services.k8s.aws/v1alpha1
    kind: Bucket
    spec:
      name: ${schema.spec.name}-data

- id: notification
  template:
    apiVersion: s3.services.k8s.aws/v1alpha1
    kind: BucketNotification
    spec:
      bucket: ${bucket.spec.name}  # Creates dependency: notification depends on bucket
```

表達式${bucket.spec.name}會建立相依性。 kro 會建立所有資源及其相依性的定向無環圖 (DAG)，然後運算拓撲順序以進行建立。

建立順序：依拓撲順序建立資源 （依存性優先）。

平行建立：同時建立沒有相依性的資源。

刪除順序：以反向拓撲順序刪除資源 （先依存）。

循環相依性：不允許 - Kro 在驗證期間拒絕具有循環相依性的 ResourceGraphDefinitions。

您可以檢視計算的建立順序：

```
kubectl get resourcegraphdefinition my-rgd -o jsonpath='{.status.topologicalOrder}'
```

如需詳細資訊，請參閱 kro 文件中的圖形推論。

使用 ACK 編寫

kro 與 ACK 的 EKS 功能無縫搭配運作，以使用 Kubernetes AWS 資源編寫資源：

```
resources:
# Create {aws} S3 bucket with ACK
- id: bucket
  template:
    apiVersion: s3.services.k8s.aws/v1alpha1
    kind: Bucket
    spec:
      name: ${schema.spec.name}-files
```

```
# Inject bucket details into Kubernetes ConfigMap
- id: config
  template:
    apiVersion: v1
    kind: ConfigMap
    data:
      BUCKET_NAME: ${bucket.spec.name}
      BUCKET_ARN: ${bucket.status.ackResourceMetadata.arn}

# Use ConfigMap in application deployment
- id: deployment
  template:
    apiVersion: apps/v1
    kind: Deployment
    spec:
      template:
        spec:
          containers:
          - name: app
            envFrom:
            - configMapRef:
                name: ${config.metadata.name}
```

此模式可讓您建立 AWS 資源、擷取其詳細資訊 (ARNs、URLs、端點），並將它們插入您的應用程式組態，全都以單一單位管理。

如需更多合成模式和進階範例，請參閱 the section called “考量事項”。

後續步驟

- the section called “考量事項” - 了解 EKS 特定的模式、RBAC 以及與 ACK 和 Argo CD 的整合
- kro 文件 - 完整的 kro 文件，包括進階 CEL 表達式、驗證模式和故障診斷

設定 kro 許可

與 ACK 和 Argo CD 不同，kro 不需要 IAM 許可。 kro 完全在您的 Kubernetes 叢集內運作，而且不會進行 AWS API 呼叫。使用標準 Kubernetes RBAC 控制對 kro 資源的存取。

許可如何使用 kro

kro 使用兩種具有不同範圍的 Kubernetes 資源類型：

ResourceGraphDefinitions：定義自訂 APIs叢集範圍資源。通常由設計和維護組織標準的平台團隊管理。

執行個體：從 ResourceGraphDefinitions 建立的命名空間範圍自訂資源。可由具有適當 RBAC 許可的應用程式團隊建立。

根據預設，kro 功能具有透過AmazonEKSKROPolicy存取項目政策管理 ResourceGraphDefinitions 及其執行個體的許可。不過，kro 需要額外的許可，才能建立和管理 ResourceGraphDefinitions 中定義的基礎 Kubernetes 資源（例如部署、服務或 ACK 資源）。您必須透過存取項目政策或 Kubernetes RBAC 授予這些許可。如需授予這些許可的詳細資訊，請參閱 the section called “eksctl”、 the section called “ AWS CLI”或 the section called “主控台”。

平台團隊許可

平台團隊需要許可才能建立和管理 ResourceGraphDefinitions。

平台團隊的範例 ClusterRole：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: kro-platform-admin
rules:
- apiGroups: ["kro.run"]
  resources: ["resourcegraphdefinitions"]
  verbs: ["*"]
```

綁定平台團隊成員：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: platform-team-kro-admin
subjects:
- kind: Group
  name: platform-team
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: kro-platform-admin
  apiGroup: rbac.authorization.k8s.io
```

應用程式團隊許可

應用程式團隊需要許可，才能在其命名空間中建立自訂資源的執行個體。

應用程式團隊的角色範例：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: kro-app-developer
  namespace: my-app
rules:
- apiGroups: ["kro.run"]
  resources: ["webapps", "databases"]
  verbs: ["create", "get", "list", "update", "delete", "patch"]
```

繫結至應用程式團隊成員：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: app-team-kro-developer
  namespace: my-app
subjects:
- kind: Group
  name: app-team
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: kro-app-developer
  apiGroup: rbac.authorization.k8s.io
```

> **Note**
>
> 角色 (`kro.run`在此範例中） 中的 API 群組必須符合 ResourceGraphDefinition 結構描述中`apiVersion`定義的 。

唯讀存取

授予唯讀存取權以檢視 ResourceGraphDefinitions 和執行個體，無需修改許可。

唯讀 ClusterRole：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: kro-viewer
rules:
- apiGroups: ["kro.run"]
  resources: ["resourcegraphdefinitions"]
  verbs: ["get", "list", "watch"]
```

執行個體的唯讀角色：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: kro-instance-viewer
  namespace: my-app
rules:
- apiGroups: ["kro.run"]
  resources: ["webapps", "databases"]
  verbs: ["get", "list", "watch"]
```

多命名空間存取

授予應用程式團隊使用 ClusterRoles 搭配 RoleBindings 存取多個命名空間的權限。

用於多命名空間存取的 ClusterRole：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: kro-multi-namespace-developer
rules:
- apiGroups: ["kro.run"]
  resources: ["webapps"]
  verbs: ["create", "get", "list", "update", "delete"]
```

繫結至特定命名空間：

```
apiVersion: rbac.authorization.k8s.io/v1
```

```
kind: RoleBinding
metadata:
  name: app-team-dev-access
  namespace: development
subjects:
- kind: Group
  name: app-team
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: kro-multi-namespace-developer
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: app-team-staging-access
  namespace: staging
subjects:
- kind: Group
  name: app-team
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: kro-multi-namespace-developer
  apiGroup: rbac.authorization.k8s.io
```

最佳實務

最低權限原則：僅授予每個團隊責任所需的最低許可。

使用群組而非個別使用者：將角色繫結至群組，而非個別使用者，以便於管理。

個別的平台和應用程式考量：平台團隊管理 ResourceGraphDefinitions、應用程式團隊管理執行個體。

命名空間隔離：使用命名空間隔離不同的團隊或環境，具有 RBAC 控制對每個命名空間的存取。

稽核的唯讀存取權：為安全與合規團隊提供稽核目的的唯讀存取權。

後續步驟

- the section called “kro 概念” - 了解 kro 概念和資源合成
- the section called “EKS 功能的考量事項” - 檢閱 功能的安全最佳實務

EKS 的 kro 考量

檢閱使用 kro 的 EKS 受管功能的重要考量，包括何時使用資源合成、RBAC 模式，以及與其他 EKS 功能的整合。

何時使用 kro

kro 旨在建立可重複使用的基礎設施模式和自訂 APIs，以簡化複雜的資源管理。

當您需要下列項目時，請使用 kro：

- 使用簡化APIs 為應用程式團隊建立自助式平台
- 標準化跨團隊的基礎設施模式（資料庫 + 備份 + 監控）
- 管理資源相依性，並在資源之間傳遞值
- 建置可隱藏實作複雜性的自訂抽象概念
- 將多個 ACK 資源組成更高階的建置區塊
- 為複雜的基礎設施堆疊啟用 GitOps 工作流程

以下情況請勿使用 kro：

- 管理簡單、獨立的資源（直接使用 ACK 或 Kubernetes 資源）
- 您需要動態執行時間邏輯 (kro 是宣告式而非必要)
- 資源沒有相依性或共用組態

kro 擅長建立「鋪設路徑」- 有意見且可重複使用的模式，可讓團隊輕鬆正確部署複雜的基礎設施。

RBAC 模式

kro 可讓建立 ResourceGraphDefinitions 的平台團隊和建立執行個體的應用程式團隊之間分離疑慮。

平台團隊責任

平台團隊會建立和維護定義自訂 APIs ResourceGraphDefinitions (RGDs)。

所需的許可：

- 建立、更新、刪除 ResourceGraphDefinitions
- 管理基礎資源類型（部署、服務、ACK 資源）
- 存取將使用 RGDs的所有命名空間

平台團隊的範例 ClusterRole：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: platform-kro-admin
rules:
- apiGroups: ["kro.run"]
  resources: ["resourcegraphdefinitions"]
  verbs: ["*"]
- apiGroups: ["*"]
  resources: ["*"]
  verbs: ["get", "list", "watch"]
```

如需詳細的 RBAC 組態，請參閱 the section called “設定 許可”。

應用程式團隊的責任

應用程式團隊會建立 RGDs定義的自訂資源執行個體，而不需要了解基礎複雜性。

所需的許可：

- 建立、更新、刪除自訂資源的執行個體
- 對其命名空間的讀取存取權
- 無法存取基礎資源或 RGDs

優點：

- 團隊使用簡單、高階 APIs
- 平台團隊控制實作詳細資訊
- 降低組態錯誤的風險
- 加快新團隊成員的加入速度

與其他 EKS 功能整合

編寫 ACK 資源

與 ACK 結合以建立基礎設施模式時，kro 特別強大。

常見模式：

- 具有儲存體的應用程式：S3 儲存貯體 + SQS 佇列 + 通知組態
- 資料庫堆疊：RDS 執行個體 + 參數群組 + 安全群組 + Secrets Manager 秘密
- 網路：VPC + 子網路 + 路由表 + 安全群組 + NAT 閘道
- 使用儲存體運算：EC2 執行個體 + EBS 磁碟區 + IAM 執行個體描述檔

kro 會處理相依性排序、在資源之間傳遞值（例如 ARNs 和連線字串），並以單一單位管理完整生命週期。

如需編寫 ACK 資源的範例，請參閱 the section called “ACK 概念”。

使用 Argo CD 的 GitOps

使用適用於 Argo CD 的 EKS 功能，從 Git 儲存庫部署 RGDs 和執行個體。

儲存庫組織：

- 平台儲存庫：包含由平台團隊管理的 ResourceGraphDefinitions
- 應用程式儲存庫：包含應用程式團隊管理的自訂資源執行個體
- 共用儲存庫：包含小型組織的 RGDs 和執行個體

考量：

- 在執行個體之前部署 RGDs(Argo CD 同步波可協助您）
- 為平台和應用程式團隊使用單獨的 Argo CD 專案
- 平台團隊控制 RGD 儲存庫存取
- 應用程式團隊具有 RGD 定義的唯讀存取權

如需 Argo CD 的詳細資訊，請參閱 the section called “使用 Argo CD”。

組織 ResourceGraphDefinitions

依用途、複雜性和擁有權來組織 RGDs。

依用途：

- 基礎設施：資料庫堆疊、聯網、儲存

- 應用程式：Web 應用程式、APIs、批次任務
- 平台：共用服務、監控、記錄

根據複雜性：

- 簡單：具有最少相依性的 2-3 個資源
- 中等：具有某些相依性的 5-10 個資源
- 複雜：超過 10 個具有複雜相依性的資源

命名慣例：

- 使用描述性名稱：`webapp-with-database`、 `s3-notification-queue`
- 在名稱中包含版本以進行重大變更： `webapp-v2`
- 對相關 RGDs 使用一致的字首：`platform-` 、 `app-`

命名空間策略：

- RGDs是叢集範圍（非命名空間）
- 執行個體為命名空間
- 在 RGDs 中使用命名空間選擇器來控制可以建立執行個體的位置

版本控制和更新

規劃 RGD 演變和執行個體遷移。

RGD 更新：

- 非重大變更：就地更新 RGD（新增選用欄位、使用 includeWhen 的新資源）
- 重大變更：使用不同名稱建立新的 RGD (webapp-v2)
- 棄用：使用註釋標記舊 RGDs、傳達遷移時間表

執行個體遷移：

- 使用更新的 RGD 建立新的執行個體
- 驗證新執行個體是否正常運作

- 刪除舊執行個體
- kro 會自動處理基礎資源更新

最佳實務：

- 先測試非生產環境中的 RGD 變更
- 在 RGD 名稱中使用語意版本控制進行主要變更
- 文件重大變更和遷移路徑
- 為應用程式團隊提供遷移範例

驗證和測試

在部署到生產環境之前驗證 RGDs。

驗證策略：

- 結構描述驗證：kro 會自動驗證 RGD 結構
- 乾執行執行個體：在開發命名空間中建立測試執行個體
- 整合測試：確認編寫的資源可一起運作
- 政策強制執行：使用許可控制器強制執行組織標準

要測試的常見問題：

- 資源相依性和排序
- 在資源之間傳遞的值 (CEL 表達式)
- 條件式資源包含 (includeWhen)
- 基礎資源的狀態傳播
- 執行個體建立的 RBAC 許可

上游文件

如需使用 kro 的詳細資訊：

- kro 入門 - 建立 ResourceGraphDefinitions
- CEL 表達式 - 撰寫 CEL 表達式

- kro Guides - 進階合成模式
- 故障診斷 - 故障診斷和偵錯

後續步驟

- the section called “設定 許可” - 為平台和應用程式團隊設定 RBAC
- the section called “kro 概念” - 了解 kro 概念和資源生命週期
- the section called “疑難排解” - 對 kro 問題進行故障診斷
- the section called “ACK 概念” - 了解用於合成的 ACK 資源
- the section called “使用 Argo CD” - 使用 GitOps 部署 RGDs 和執行個體

對 kro 功能的問題進行故障診斷

> **Note**
>
> EKS 功能是完全受管的，並在叢集外部執行。您無法直接存取控制器命名空間。您可以設定控制器日誌交付，以掌握控制器行為。請參閱 the section called “功能控制器日誌”。故障診斷著重於功能運作狀態、RBAC 組態和資源狀態。

功能為 ACTIVE，但 ResourceGraphDefinitions 無法運作

如果您的 kro 功能顯示ACTIVE狀態，但 ResourceGraphDefinitions 未建立基礎資源，請檢查功能運作狀態、RBAC 許可和資源狀態。

檢查功能運作狀態：

您可以在 EKS 主控台或使用 AWS CLI 檢視功能運作狀態和狀態問題。

主控台：

1. 在以下網址開啟 Amazon EKS 主控台：https://console.aws.amazon.com/eks/home#/clusters。
2. 選取您的叢集名稱。
3. 選擇可觀測性索引標籤。
4. 選擇監控叢集。
5. 選擇功能索引標籤以檢視所有功能的運作狀態和狀態。

AWS CLI：

```
# View capability status and health
aws eks describe-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-kro

# Look for issues in the health section
```

常見原因：

- 缺少 RBAC 許可：kro 缺少建立基礎 Kubernetes 資源的許可
- 無效的 CEL 表達式：ResourceGraphDefinition 中的語法錯誤
- 資源相依性：相依資源未就緒
- 結構描述驗證：執行個體不符合 RGD 結構描述要求

驗證 RBAC 許可：

```
# Check if capability has cluster admin policy
kubectl get accessentry -A | grep kro
```

如果功能沒有所需的許可，請將 AmazonEKSClusterAdminPolicy與 kro 功能的存取項目建立關聯，或建立更嚴格的 RBAC 政策以供生產使用。如需詳細資訊，請參閱 the section called “設定 許可”。

檢查 ResourceGraphDefinition 狀態：

```
# List all RGDs
kubectl get resourcegraphdefinition

# Describe specific RGD
kubectl describe resourcegraphdefinition my-rgd

# Check for validation errors
kubectl get resourcegraphdefinition my-rgd -o jsonpath='{.status.conditions}'
```

ResourceGraphDefinitions 有三個金鑰狀態條件：

- ResourceGraphAccepted - RGD 是否通過驗證 (CEL 語法、類型檢查、欄位存在)

- KindReady - 是否產生並註冊自訂 API 的 CRD
- ControllerReady - kro 是否正在主動監看自訂 API 的執行個體

如果 ResourceGraphAccepted是 False，請檢查條件訊息是否有驗證錯誤，例如未知欄位、類型不相符或循環相依性。

已建立執行個體，但基礎資源未顯示

如果自訂資源執行個體存在，但未建立基礎 Kubernetes 資源 （ 部署、服務、ConfigMaps)，請確認 kro 具有許可並檢查合成錯誤。

檢查執行個體狀態：

```
# Describe the instance (replace with your custom resource kind and name)
kubectl describe custom-kind my-instance

# View instance events
kubectl get events --field-selector involvedObject.name=my-instance

# Check instance status conditions
kubectl get custom-kind my-instance -o jsonpath='{.status.conditions}'

# Check instance state
kubectl get custom-kind my-instance -o jsonpath='{.status.state}'
```

執行個體具有顯示高階狀態state的欄位：

- ACTIVE - 執行個體已成功執行
- IN_PROGRESS - 正在處理或調校執行個體
- FAILED - 執行個體無法協調
- DELETING - 正在刪除執行個體
- ERROR - 處理期間發生錯誤

執行個體也有四個狀態條件：

- InstanceManaged - 已正確設定定案者和標籤
- GraphResolved - 建立執行期圖表並解析資源
- ResourcesReady - 所有已建立並就緒的資源

- Ready - 整體執行個體運作狀態 （只有在所有子條件皆為 True時才會變成 True)

專注於 Ready條件，以判斷執行個體的運作狀態。如果 Ready是 False，請檢查子條件以識別哪個階段失敗。

驗證 RBAC 許可：

kro 功能需要許可才能建立 ResourceGraphDefinitions 中定義的基礎 Kubernetes 資源。

```
# Check if the capability has the AmazonEKSClusterAdminPolicy
kubectl get accessentry -A | grep kro
```

如果缺少許可，請將 AmazonEKSClusterAdminPolicy與 kro 功能的存取項目建立關聯，或建立更嚴格的 RBAC 政策以供生產使用。如需詳細資訊，請參閱 the section called “設定 許可”。

CEL 表達式錯誤

CEL 表達式錯誤會在 ResourceGraphDefinition 建立時間發現，而不是在建立執行個體時。 kro 會驗證所有 CEL 語法、針對 Kubernetes 結構描述類型檢查表達式，並在建立 RGD 時驗證欄位是否存在。

常見的 CEL 驗證錯誤：

- 未定義的欄位參考：參考結構描述或資源中不存在的欄位
- 類型不符：運算式傳回錯誤類型 （例如，預期整數的字串）
- 無效的語法：CEL 表達式中缺少括號、引號或運算子
- 未知資源類型：參考叢集中不存在的 CRD

檢查 RGD 驗證狀態：

```
# Check if RGD was accepted
kubectl get resourcegraphdefinition my-rgd -o jsonpath='{.status.conditions[?
(@.type=="ResourceGraphAccepted")]}'

# View detailed validation errors
kubectl describe resourcegraphdefinition my-rgd
```

如果 ResourceGraphAccepted是 False，則條件訊息會包含驗證錯誤。

有效的 CEL 表達式範例：

```
# Reference schema field
${schema.spec.appName}

# Conditional expression
${schema.spec.replicas > 1}

# String template (expressions must return strings)
name: "${schema.spec.appName}-service"

# Standalone expression (can be any type)
replicas: ${schema.spec.replicaCount}

# Resource reference
${deployment.status.availableReplicas}

# Optional field access (returns null if field doesn't exist)
${configmap.data.?DATABASE_URL}
```

資源相依性無法解析

kro 會自動從 CEL 表達式推斷相依性，並以正確的順序建立資源。如果未如預期般建立資源，請檢查相依性順序和資源準備程度。

檢視計算的建立順序：

```
# See the order kro will create resources
kubectl get resourcegraphdefinition my-rgd -o jsonpath='{.status.topologicalOrder}'
```

這會顯示資源之間根據 CEL 表達式參考計算的順序。

檢查資源準備程度：

```
# View instance status to see which resources are ready
kubectl get custom-kind my-instance -o jsonpath='{.status}'

# Check specific resource status
kubectl get deployment my-deployment -o jsonpath='{.status.conditions}'
```

確認readyWhen條件（若有使用）：

此 readyWhen 欄位為選用。如果未指定，資源會在建立後立即視為就緒。如果您已定義readyWhen條件，請確認它們是否正確檢查資源準備程度：

```
resources:
  - id: deployment
    readyWhen:
      - ${deployment.status.availableReplicas == deployment.spec.replicas}
```

檢查資源事件：

```
# View events for the underlying resources
kubectl get events -n namespace --sort-by='.lastTimestamp'
```

結構描述驗證失敗

如果執行個體因為結構描述驗證錯誤而無法建立，請確認執行個體符合 RGD 結構描述要求。

檢查驗證錯誤：

```
# Attempt to create instance and view error
kubectl apply -f instance.yaml

# View existing instance validation status
kubectl describe custom-kind my-instance | grep -A 5 "Validation"
```

常見的驗證問題：

- 缺少必要欄位：執行個體不提供所有必要的結構描述欄位
- 類型不符：提供預期整數的字串
- 列舉值無效：使用不在允許清單中的值
- 模式不相符：字串不符合 regex 模式

檢閱 RGD 結構描述：

```
# View the schema definition
kubectl get resourcegraphdefinition my-rgd -o jsonpath='{.spec.schema}'
```

確保您的執行個體為所有必要欄位提供正確的類型。

後續步驟

- the section called “考量事項” - kro 考量事項和最佳實務
- the section called “設定 許可” - 為平台和應用程式團隊設定 RBAC
- the section called “kro 概念” - 了解 kro 概念和資源生命週期
- the section called “故障診斷功能” - 一般功能故障診斷指引

比較 kro 與自我管理 kro 的 EKS 功能

適用於 kro 的 EKS 功能提供與自我管理 kro 相同的功能，但具有顯著的操作優勢。如需 EKS 功能與自我管理解決方案的一般比較，請參閱 the section called “考量事項”。

kro 的 EKS 功能使用相同的上游 kro 控制器，並與上游 kro 完全相容。ResourceGraphDefinitions、CEL 表達式和資源合成的運作方式相同。如需完整的 kro 文件和範例，請參閱 kro 文件。

遷移路徑

您可以從自我管理的 kro 遷移到受管功能，無需停機。

> **Important**
>
> 在遷移之前，請確保您的自我管理 kro 控制器執行的版本與 kro 的 EKS 功能相同。在 EKS 主控台或使用 檢查功能版本`aws eks describe-capability`，然後升級自我管理的安裝以符合。這可防止遷移期間的相容性問題。

1. 更新您的自我管理 kro 控制器以`kube-system`用於領導者選擇租用：

```
helm upgrade --install kro \
  oci://ghcr.io/awslabs/kro/kro-chart \
  --namespace kro \
  --set leaderElection.namespace=kube-system
```

   這會將控制器的租用移至 `kube-system`，允許受管功能與其協調。
2. 在叢集上建立 kro 功能（請參閱 the section called “建立 kro 功能”）
3. 受管功能可識別現有的 ResourceGraphDefinitions 和執行個體，接管對帳

4. 逐漸縮減或移除自我管理的 kro 部署：

```
helm uninstall kro --namespace kro
```

此方法可讓兩個控制器在遷移期間安全地共存。受管功能會自動採用 ResourceGraphDefinitions 和先前由自我管理的 kro 管理的執行個體，以確保持續對帳而不會發生衝突。

後續步驟

- the section called “建立 kro 功能” - 建立 kro 功能資源
- the section called “kro 概念” - 了解 kro 概念和資源合成

對 EKS 功能進行故障診斷

> **Note**
>
> EKS 功能是完全受管的，並在叢集外部執行。您無法直接存取控制器命名空間。故障診斷著重於功能運作狀態、資源狀態、組態和控制器日誌。您可以設定控制器日誌交付，以取得控制器行為的可見性。請參閱 the section called “功能控制器日誌”。

一般疑難排解方法

疑難排解 EKS 功能時，請遵循下列一般方法：

1. 檢查功能運作狀態：使用 `aws eks describe-capability` 來檢視功能狀態和運作狀態問題
2. 驗證資源狀態：檢查您建立的狀態條件和事件的 Kubernetes 資源 CRDs)
3. 檢閱控制器日誌：如果已設定日誌交付，請查詢控制器日誌以取得錯誤和調校詳細資訊
4. 檢閱 IAM 許可：確保功能角色具有必要的許可
5. 檢查組態：驗證功能特定的組態是否正確

使用控制器日誌進行故障診斷

如果您已設定控制器日誌交付（請參閱the section called “功能控制器日誌”），您可以查詢日誌以識別對帳錯誤、資源衝突和組態問題。

所有控制器的查詢錯誤

```
fields @timestamp, controller, message, error
| filter level = "error"
| sort @timestamp desc
| limit 50
```

篩選特定 ACK 服務控制器的日誌

使用 controllerGroup 欄位隔離日誌與特定 ACK 服務控制器：

```
fields @timestamp, message, error
| filter controllerGroup = "s3.services.k8s.aws"
| filter level = "error"
| sort @timestamp desc
```

若要依資源類型進一步篩選（例如，僅來自 EC2 控制器的SecurityGroup日誌）：

```
fields @timestamp, message, error
| filter controllerGroup = "ec2.services.k8s.aws"
| filter controllerKind = "SecurityGroup"
| sort @timestamp desc
| limit 100
```

篩選特定 Argo CD 應用程式的日誌

使用 application 欄位隔離特定 Argo CD 應用程式的日誌：

```
fields @timestamp, message, error
| filter application = "my-application"
| sort @timestamp desc
| limit 100
```

追蹤特定資源的對帳

使用 reconcileID 欄位遵循單一調校週期：

```
fields @timestamp, level, message, error
| filter reconcileID = "your-reconcile-id"
```

```
| sort @timestamp asc
```

指出問題的常見日誌模式

- 重複的對帳錯誤 — 控制器無法達到資源所需的狀態。請檢查 `error` 欄位以取得詳細資訊，例如 IAM 許可失敗或資源組態無效。
- 「協調程式錯誤」與 AWS API 錯誤 — 功能角色可能缺少特定 AWS 服務操作的許可。檢閱錯誤訊息並相應地更新 IAM 政策。
- 資源沒有日誌項目 - 如果您沒有看到預期控制器進行協調之資源的日誌，請確認該功能為 ，`ACTIVE`且該資源存在於該功能可存取的命名空間中。

檢查功能運作狀態

所有 EKS 功能都透過 EKS 主控台和 `describe-capability` API 提供運作狀態資訊。

主控台：

1. 在以下網址開啟 Amazon EKS 主控台：https://console.aws.amazon.com/eks/home#/clusters。
2. 選取您的叢集名稱。
3. 選擇可觀測性索引標籤。
4. 選擇監控叢集。
5. 選擇功能索引標籤以檢視所有功能的運作狀態和狀態。

功能索引標籤會顯示：

- 功能名稱和類型
- 目前狀態
- 運作狀態問題，含描述

AWS CLI：

```
aws eks describe-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-capability-name
```

回應包括：

- 狀態：目前功能狀態 (CREATING、ACTIVE、UPDATING、DELETING、CREATE_FAILED) UPDATE_FAILED
- 運作狀態：運作狀態資訊，包括 功能偵測到的任何問題

常見功能狀態

CREATING：正在設定功能。

ACTIVE：功能正在執行並準備好使用。如果資源未如預期般運作，請檢查資源狀態和 IAM 許可。

正在更新：正在套用組態變更。等待狀態傳回 ACTIVE。

CREATE_FAILED 或 UPDATE_FAILED：設定或更新發生錯誤。如需詳細資訊，請參閱運作狀態區段。常見原因：

- IAM 角色信任政策不正確或遺失
- IAM 角色不存在或無法存取
- 叢集存取問題
- 無效的組態參數

驗證 Kubernetes 資源狀態

EKS 功能會在叢集中建立和管理 Kubernetes 自訂資源定義 (CRDs)。疑難排解時，請檢查您建立的資源狀態：

```
# List resources of a specific type
kubectl get resource-kind -A

# Describe a specific resource to see conditions and events
kubectl describe resource-kind resource-name -n namespace

# View resource status conditions
kubectl get resource-kind resource-name -n namespace -o jsonpath='{.status.conditions}'

# View events related to the resource
kubectl get events --field-selector involvedObject.name=resource-name -n namespace
```

資源狀態條件提供下列相關資訊：

- 資源是否已就緒
- 遇到的任何錯誤
- 目前對帳狀態

檢閱 IAM 許可和叢集存取

許多功能問題來自 IAM 許可問題或缺少叢集存取組態。驗證功能角色許可和叢集存取項目。

檢查 IAM 角色許可

確認功能角色具有必要的許可：

```
# List attached managed policies
aws iam list-attached-role-policies --role-name my-capability-role

# List inline policies
aws iam list-role-policies --role-name my-capability-role

# Get specific policy details
aws iam get-role-policy --role-name my-capability-role --policy-name policy-name

# View the role's trust policy
aws iam get-role --role-name my-capability-role --query 'Role.AssumeRolePolicyDocument'
```

信任政策必須允許`capabilities.eks.amazonaws.com`服務主體：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "capabilities.eks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
```

```
  ]
}
```

檢查 EKS 存取項目和存取政策

所有功能都需要其操作所在叢集上適當的 EKS 存取項目和存取政策。

驗證存取項目是否存在：

```
aws eks list-access-entries \
  --cluster-name my-cluster \
  --region region-code
```

在清單中尋找功能角色 ARN。如果遺失，則 功能無法存取叢集。

檢查連接到項目的存取政策：

```
aws eks list-associated-access-policies \
  --cluster-name my-cluster \
  --principal-arn arn:aws:iam::111122223333:role/my-capability-role \
  --region region-code
```

所有功能都需要適當的存取政策：

- ACK：需要許可才能建立和管理 Kubernetes 資源
- kro：需要建立和管理 Kubernetes 資源的許可
- Argo CD：需要建立和管理應用程式的許可，並且需要遠端目標叢集上的存取項目以進行多叢集部署

針對 Argo CD 多叢集部署：

如果部署到遠端叢集，請確認功能角色在每個目標叢集上有存取項目：

```
# Check Access Entry on target cluster
aws eks describe-access-entry \
  --cluster-name target-cluster \
  --principal-arn arn:aws:iam::111122223333:role/argocd-capability-role \
  --region region-code
```

如果目標叢集上缺少存取項目，Argo CD 就無法將應用程式部署到其中。如需組態詳細資訊the section called “註冊叢集”，請參閱 。

功能特定的疑難排解

如需每個功能類型特有的詳細故障診斷指引：

- the section called “疑難排解” - 疑難排解 ACK 資源建立、IAM 許可和跨帳戶存取
- the section called “疑難排解” - 疑難排解應用程式同步、儲存庫身分驗證和多叢集部署
- the section called “疑難排解” - 疑難排解 ResourceGraphDefinitions、CEL 表達式和 RBAC 許可

所有功能的常見問題

功能卡在 CREATING 狀態

如果功能保持 CREATING 狀態的時間超過預期：

1. 在主控台 (可觀測性 > 監控叢集 > 功能索引標籤） 或使用 CLI AWS 檢查功能運作狀態是否有特定問題：

```
aws eks describe-capability \
  --region region-code \
  --cluster-name my-cluster \
  --capability-name my-capability-name \
  --query 'capability.health'
```

2. 驗證 IAM 角色是否存在，並具有正確的信任政策
3. 確保您的叢集可存取且運作狀態良好
4. 檢查是否有任何叢集層級的問題可能會阻礙功能設定

未建立或更新的資源

如果功能是 ，ACTIVE但資源未建立或更新：

1. 檢查資源狀態是否有錯誤情況
2. 驗證特定 AWS 服務 (ACK) 或儲存庫 (Argo CD) 的 IAM 許可
3. 檢查建立基礎資源的 RBAC 許可 (kro)
4. 檢閱驗證錯誤的資源規格

功能運作狀態顯示問題

如果 `describe-capability` 顯示運作狀態問題：

1. 仔細閱讀問題描述 - 它們通常指出特定問題
2. 解決根本原因 (IAM 許可、組態錯誤等)
3. 問題解決後，此功能會自動復原

後續步驟

- the section called “使用 功能” - 管理功能資源
- the section called “疑難排解” - ACK 特定的疑難排解
- the section called “疑難排解” - Argo CD 特定疑難排解
- the section called “疑難排解” - kro 特定疑難排解
- the section called “EKS 功能的考量事項” - 功能的安全最佳實務

組織與監控叢集資源

本章節包括可協助您管理叢集的以下主題。您也可以使用 檢視 Kubernetes 資源的相關資訊 AWS 管理主控台。

- Kubernetes 儀表板是 Kubernetes 叢集的一般用途、以 Web 為基礎的 UI。它可讓使用者管理叢集中執行的應用程式並對其進行疑難排解，以及管理叢集本身。如需詳細資訊，請參閱 Kubernetes Dashboard GitHub 儲存庫。
- the section called “指標伺服器” – Kubernetes 指標伺服器是叢集中資源用量資料的彙總工具。根據預設，它不會部署在您的叢集中，但是由 Kubernetes 附加元件使用，例如 Kubernetes 儀表板和 the section called “Horizontal Pod Autoscaler”。在本主題中，您將了解如何安裝指標伺服器。
- the section called “使用 Helm 部署應用程式” – Kubernetes 的 Helm 套件管理工具可協助您安裝和管理 Kubernetes 叢集上的應用程式。此主題協助您安裝和執行 Helm 二進位檔，讓您可以在本機電腦上使用 Helm CLI 安裝和管理圖表。
- the section called “標記您的 資源” – 為協助您管理 Amazon EKS 資源，您可以用標籤形式將您自己的中繼資料指派給每個資源。本主題說明標籤並示範如何建立它們。
- the section called “Service Quotas” – AWS 您的帳戶具有每個 AWS 服務的預設配額，先前稱為限制。了解 Amazon EKS 的配額，以及如何提高配額。

監控和最佳化 Amazon EKS 叢集成本

成本監控是管理 Amazon EKS 上 Kubernetes 叢集的重要層面。透過取得叢集成本的可見性，您可以最佳化資源使用率、設定預算，並就您的部署做出資料驅動的決策。Amazon EKS 提供了兩種成本監控解決方案，每種方案都有其獨特的優勢，可協助您有效追蹤和分配成本：

Amazon EKS 的 AWS 帳單拆分成本分配資料：此原生功能與 AWS 帳單主控台無縫整合，讓您能使用與其他 AWS 服務相同的熟悉介面和工作流程來分析和分配成本。透過拆分成本分配，您可直接與其他 AWS 支出一起深入了解您的 Kubernetes 成本，從而更輕鬆地在整個 AWS 環境中全面最佳化成本。您也可利用現有 AWS 帳單功能，例如 Cost Categories 和成本異常偵測，進一步增強您的成本管理功能。如需詳細資訊，請參閱《AWS 帳單使用者指南》中了解拆分成本分配資料。

Kubecost：Amazon EKS 支援作為 Kubernetes 成本監控工具的 Kubecost。Kubecost 提供了功能豐富、Kubernetes 原生的成本監控方法，可按 Kubernetes 資源提供精細的成本拆分、成本最佳化建議以及現成的儀表板和報告。Kubecost 還透過與 AWS 成本和用量報告進行整合來擷取準確的定價資料，確保您能準確檢視 Amazon EKS 成本。了解如何安裝 Kubecost。請參閱 Kubecost AWS Marketplace 頁面，以取得免費 Kubecost 訂閱的相關資訊，

使用分割成本分配在 AWS 帳單中依 Pod 檢視成本

使用 Amazon EKS 的 AWS 分割成本分配資料進行成本監控

您可以使用 Amazon EKS 的 AWS 分割成本分配資料，取得 Amazon EKS 叢集的精細成本可見性。這可讓您分析、最佳化和回收 Kubernetes 應用程式的成本和用量。您可根據 Kubernetes 應用程式消耗的 Amazon EC2 CPU 和記憶體資源，將應用程式成本分配給個別業務單位和團隊。Amazon EKS 的拆分成本分配資料可讓您查看每個 Pod 的成本，並可讓您使用命名空間、叢集和其他 Kubernetes 基本概念來彙總每個 Pod 的成本資料。以下是您可用於分析 Amazon EKS 成本分配資料的 Kubernetes 基本概念範例。

- 叢集名稱
- 部署
- 命名空間
- 節點
- 工作負載名稱
- 工作負載類型

同時支援使用者定義的成本分配標籤。如需使用分割成本分配資料的詳細資訊，請參閱《 AWS 帳單使用者指南》中的了解分割成本分配資料。

設置成本與用量報告

您可以在成本管理主控台、 AWS 命令列界面或 AWS SDKs 中開啟 EKS 的分割成本分配資料。

將下列項目用於拆分成本分配資料：

1. 選擇加入拆分成本分配資料。如需詳細資訊，請參閱《 成本和用量報告使用者指南》中的啟用分割成本分配資料。 AWS
2. 您可將資料包含在新的或現有的報告中。
3. 檢視報告。您可以使用帳單和成本管理主控台，或在 Amazon Simple Storage Service 中檢視報告檔案。

安裝 Kubecost

Amazon EKS 支援 Kubecost，可讓您用於監控按 Kubernetes 資源 (包括 Pod、節點、命名空間和標籤) 細分的成本。本主題涵蓋安裝 Kubecost 和存取 Kubecost 儀表板。

Amazon EKS 提供 AWS 最佳化的 Kubecost 套件，以實現叢集成本可見性。您可以使用現有的 AWS 支援協議來取得支援。如需有關 Kubecost 可用版本的詳細資訊，請參閱 the section called “深入了解 Kubecost”。

> **Note**
>
> Kubecost v3 引進了重大的架構改進，包括大幅提升的效能和增強的自動化功能。進一步了解 Kubecost v3。
> Kubecost v2 推出數個主要新功能。進一步了解 Kubecost v2。

有關 Kubecost 的更多資訊，請參閱 Kubecost 文件和常見問答集。

安裝 Amazon EKS 最佳化 Kubecost 套件

您可使用下列程序之一來安裝 Amazon EKS 最佳化 Kubecost 套件：

- 開始之前，建議您檢閱 Kubecost - 架構概觀，以了解 Kubecost 如何在 Amazon EKS 上運作。
- 如果您是初次使用 Amazon EKS，建議您使用 Amazon EKS 附加元件進行安裝，因為它簡化了 Amazon EKS 最佳化 Kubecost 套件的安裝過程。如需詳細資訊，請參閱使用 Amazon EKS 附加元件在 Amazon EKS 叢集上部署 Kubecost。
- 要自訂安裝，您可使用 Helm 設定您的 Amazon EKS 最佳化 Kubecost 套件。如需詳細資訊，請參閱 Kubecost 文件中的使用 Helm 在 Amazon EKS 叢集上部署 Kubecost。

> **Important**
>
> 對於 Kubecost v3，Helm Chart 位置已變更為 `public.ecr.aws/kubecost/kubecost`。如果您是從 v2 升級，請相應地更新 Helm 儲存庫參考。

> **Note**
>
> 對於使用 Kubecost v3 的多叢集部署，您需要 S3-compatible物件儲存體 （適用於 EKS 客戶的AWS S3) 來進行指標儲存。這會取代 v2 中使用的 Prometheus 相容儲存體。如需詳細資訊，請參閱 Kubecost 文件中的多叢集安裝。

存取 Kubecost 儀表板

Amazon EKS 最佳化 Kubecost 套件設定完成後，您可以存取 Kubecost 儀表板。如需詳細資訊，請參閱the section called “存取 Kubecost 儀表板”。

存取 Kubecost 儀表板

先決條件

1. 確定 Kubecost 相關的 Pod 狀態為「執行中」。

```
kubectl get pods --namespace kubecost
```

存取 Kubecost 儀表板

1. 在您的裝置上，啟用連接埠轉送以公開 Kubecost 儀表板。
   - 如果使用 Helm 安裝 Kubecost v3：

     ```
     kubectl port-forward deployment/kubecost-frontend 9090 --namespace kubecost
     ```

   - 如果使用 Helm 安裝 Kubecost v1 或 v2：

     ```
     kubectl port-forward deployment/kubecost-cost-analyzer 9090 --namespace kubecost
     ```

   - 如果使用 Amazon EKS 附加元件安裝 Kubecost：

     ```
     kubectl port-forward deployment/cost-analyzer 9090 --namespace kubecost
     ```

     或者，您可使用 AWS Load Balancer 控制器來公開 Kubecost，並使用 Amazon Cognito 進行驗證、授權和使用者管理。如需詳細資訊，請參閱如何使用 Application Load Balancer 和 Amazon Cognito 為您的 Kubernetes Web 應用程式驗證使用者。
2. 在您完成上一個步驟的同一部裝置上，開啟 Web 瀏覽器並輸入下列位址。

   ```
   http://localhost:9090
   ```

   瀏覽器中會顯示 Kubecost 的「概觀」頁面。根據您的叢集大小，Kubecost 可能需要 5-10 分鐘（或更多）才能收集指標。您可查看您的 Amazon EKS 支出，包括累積叢集成本、相關聯的 Kubecost 資產成本和每月彙總支出。

3. 要在叢集層級追蹤成本，請標記您的 Amazon EKS 資源以進行計費。如需詳細資訊，請參閱the section called “標記您的資源以便計費”。
   - Cost allocation (成本分配) – 檢視過去七天內每個命名空間和其他維度的每月 Amazon EKS 成本和累計成本。這有助於了解應用程式的哪些部分產生了 Amazon EKS 支出。
   - 資產 – 檢視與您的 Amazon EKS 資源相關聯的 AWS 基礎設施資產成本。

深入了解 Kubecost

Amazon EKS 提供 AWS 最佳化的 Kubecost 套件，以實現叢集成本可見性。Amazon EKS 支援 Kubecost，可讓您用於監控按 Kubernetes 資源 (包括 Pod、節點、命名空間和標籤) 細分的成本。

本主題涵蓋了 Kubecost 的可用版本，以及不同可用層級之間的差異。EKS 支援 Kubecost 第 1 版、第 2 版和第 3 版。每個版本都有不同的層級可供選擇。您可為 Amazon EKS 叢集使用 Amazon EKS 最佳化 Kubecost 套件，無需額外費用。您可能需要支付使用相關 AWS 服務的費用，例如 Amazon Managed Service for Prometheus。此外，您可以使用現有的 AWS 支援協議來取得支援。

身為 Kubernetes 平台管理員和財務負責人，您可使用 Kubecost 視覺化 Amazon EKS 費用明細、分配成本，以及向應用程式團隊等組織單位收取費用。您可以根據內部團隊和業務單位的實際 AWS 帳單，提供透明且準確的成本資料。此外，您還可以根據他們的基礎設施環境和叢集內的使用模式，獲得客製化的成本最佳化建議。如需 Kubecost 的詳細資訊，請參閱 Kubecost 文件。

自訂套件的 Kubecost 和免費版本的 Kubecost (也稱為 OpenCost) 有何區別？

AWS 和 Kubecost 合作提供自訂版本的 Kubecost。此版本提供商業功能的子集，不收取額外費用。如需 Kubecost 自訂套件中包含的功能，請參閱下表。

Kubecost v3

Kubecost v2 和 v3 之間的差異是什麼？

Kubecost 3.0 是主要的架構升級，可提供大幅提升的效能、增強的可擴展性和主動最佳化功能。最重要的變更是遷移至 ClickHouse 資料庫，從 2.8 版取代 DuckDB，可提供更快的查詢和更可靠的大規模效能。Kubecost 3.0 也推出整合 Kubecost 和 Cloudability 功能的統一代理程式，消除 Prometheus 相依性並減少記憶體使用量，同時維持 OpenCost 相容性。

> **Important**
>
> 升級至 v3 之前，請先檢閱 Kubecost 文件。從 v2 遷移需要仔細規劃，並可能影響轉換期間的報告可用性。Helm Chart 位置已變更為 `public.ecr.aws/kubecost/kubecost`。

v3 中的關鍵架構改進：

- ClickHouse 資料庫：取代 DuckDB 以獲得更快的查詢和更佳的可擴展性
- 統一代理程式：結合 Kubecost 和 Cloudability 功能，消除 Prometheus 相依性
- 適用於多叢集的 S3-Compatible儲存體：對於多叢集部署，v3 使用 S3-compatible物件儲存體（適用於 EKS 客戶的AWS S3)，而不是 Amazon Managed Service for Prometheus 等 Prometheus 相容儲存體。FinOps 代理程式會從 Kubernetes API 提取指標並推送至 S3-compatible儲存體，然後彙總器會提取該資料、執行衍生步驟，並在前端顯示結果。如需詳細資訊，請參閱 Kubecost 文件中的多叢集安裝和次要叢集指南。
- 減少記憶體足跡：大幅降低資源需求，同時維護功能
- 簡化架構：適用於獨立擴展和改善彈性的單一容器 Pod 拓撲
- 增強自動化：具有多叢集意識和自訂設定檔的自動化容器請求大小

v3 中的 Amazon EKS 最佳化套件優點：

Amazon EKS 最佳化 Kubecost 套件繼續免費提供，並免除 Kubecost v3 免費方案中引進的新 100，000 USD 花費限制。無論支出層級為何，EKS 使用者都會保留所有 Kubernetes 支出功能的完整存取權。

核心功能對比：

功能	Kubecost 免費方案 3.0	Amazon EKS 最佳化 Kubecost 套件 3.0	Kubecost Enterprise 3.0
叢集成本能見度	無限制叢集，30 天內以 100，000 USD 的固定支出	沒有支出限制的統一多叢集	跨無限數量環境 (即多重雲端) 的統一且無限數量的叢集
資料庫後端	ClickHouse（本機）	ClickHouse 搭配S3-compatible儲存	ClickHouse 搭配自訂資料庫選項
效能	與 v2 相比，查詢速度大幅加快	與 v2 相比，查詢速度大幅加快	與 v2 相比，查詢速度大幅加快
記憶體使用量	減少與 v2（無 Prometheus 相依性）	減少與 v2（無 Prometheus 相依性）	減少與 v2（無 Prometheus 相依性）

功能	Kubecost 免費方案 3.0	Amazon EKS 最佳化 Kubecost 套件 3.0	Kubecost Enterprise 3.0
自動化容器請求大小	可用 （僅限 250 個核心）	無核心限制提供	無核心限制提供
花費限制	30 天內 100，000 USD	沒有花費限制	沒有花費限制
多叢集自動化	有限	使用安全訊息的完整多叢集意識	使用安全訊息的完整多叢集意識

Kubecost v2

Kubecost v1 和 v2 之間有什麼差異？

Kubecost 2.0 是先前版本的重大升級，包含主要的新功能，包括全新的 API 後端。請注意，分配和資產 API 完全向後相容。請檢閱 Kubecost 文件，確保順利轉換。有關增強功能的完整清單，請參閱 Kubecost v2.0 公告和完整版本備註。

> **Important**
>
> 升級前，請檢閱 Kubecost 文件。升級可能會影響報告的可用性。

核心功能對比：

功能	Kubecost 免費方案 2.0	Amazon EKS 最佳化 Kubecost 套件 2.0	Kubecost Enterprise 2.0
叢集成本能見度	無限制叢集，最多 250 個核心	與 Amazon Managed Service for Prometheus 整合時，具有統一的多叢集且無核心限制	跨無限數量環境 (即多重雲端) 的統一且無限數量的叢集
部署	使用者託管	使用者託管	使用者託管、Kubecost 託管 (專用租用戶)、SaaS

功能	Kubecost 免費方案 2.0	Amazon EKS 最佳化 Kubecost 套件 2.0	Kubecost Enterprise 2.0
支援的資料庫	本機 Prometheus	Amazon Managed Service for Prometheus 或本機 Prometheus	任何 Prometheus 版本和自訂資料庫
資料庫保留支援 (原始指標)	15 天	無限的歷史記錄資料	無限的歷史記錄資料
Kubecost API 和 UI 保留 (ETL)	15 天	15 天	無限制
混合雲端能見度	-	Amazon EKS 和 Amazon EKS Anywhere 叢集	多重雲端和混合雲端
警示和週期性報告	僅主要叢集支援，限 250 個核心	效率警示、預算警示、支出變更警示，以及跨所有叢集更多支援的功能	效率警示、預算警示、支出變更警示，以及跨所有叢集更多支援的功能
已儲存的報告	-	使用 15 天指標的報告	使用無限期歷史記錄資料和指標的報告
雲端計費整合	僅主要叢集支援，限 250 個核心	的自訂定價支援 AWS（包括多個叢集和多個帳戶）	支援任何雲端的自訂定價
Savings 建議	僅主要叢集支援，限 250 個核心	主要叢集洞察，但沒有 250 個核心限制	多重叢集深入解析
治理：稽核	-	-	稽核歷史記錄成本事件
單一登入 (SSO) 支援	-	支援 Amazon Cognito	Okta、Auth 0、PingID、KeyCloak 以及任何其他自訂項目

功能	Kubecost 免費方案 2.0	Amazon EKS 最佳化 Kubecost 套件 2.0	Kubecost Enterprise 2.0
角色型存取控制 (RBAC) 與 SAML 2.0	-	-	Okta、Auth 0、PingID、KeyCloak 以及任何其他自訂項目
企業培訓和入門	-	-	全方位服務培訓和 FinOps 入門
團隊	-	-	是

新功能：

以下功能有指標限制：

- Kubecost 彙總工具
- 網路監控
- Kubecost 動作
- 集合
- 異常偵測
- 容器請求資源大小調整
- Kubecost 預測
- 用於篩選和彙總的自動完成

指標限制：

指標	Kubecost 免費方案 2.0	Amazon EKS 最佳化 Kubecost 套件 2.0	Kubecost Enterprise 2.0
叢集大小	無限制叢集，最多 250 個核心	無限制	無限制
指標保留	15 天	15 天	無限制

指標	Kubecost 免費方案 2.0	Amazon EKS 最佳化 Kubecost 套件 2.0	Kubecost Enterprise 2.0
多叢集支援	不適用	Available	Available
核心限制	每個叢集 250 個核心	無核心限制	無核心限制

Kubecost v1

功能	Kubecost 免費方案	Amazon EKS 最佳化 Kubecost 套件	Kubecost Enterprise
部署	使用者託管	使用者託管	使用者託管或 Kubecos 託管 (SaaS)
支援的叢集數目	無限制	無限制	無限制
支援的資料庫	本機 Prometheus	本機 Prometheus 或 Amazon Managed Service for Prometheus	Prometheus、Amazon Managed Service for Prometheus、Cortex 或 Thanos
資料庫保留支援	15 天	無限的歷史記錄資料	無限的歷史記錄資料
Kubecost API 保留 (ETL)	15 天	15 天	無限的歷史記錄資料
叢集成本能見度	單一叢集	統一的多重叢集	統一的多重叢集
混合雲端能見度	-	Amazon EKS 和 Amazon EKS Anywhere 叢集	多重雲端和混合雲端支援
警示和週期性報告	-	效率警示、預算警示、支出變更警示，以及更多支援的功能	效率警示、預算警示、支出變更警示，以及更多支援的功能

功能	Kubecost 免費方案	Amazon EKS 最佳化 Kubecost 套件	Kubecost Enterprise
已儲存的報告	-	使用 15 天資料的報告	使用無限期歷史記錄資料的報告
雲端計費整合	每一個叢集都需要	的自訂定價支援 AWS（包括多個叢集和多個帳戶）	的自訂定價支援 AWS（包括多個叢集和多個帳戶）
Savings 建議	單一叢集深入解析	單一叢集深入解析	多重叢集深入解析
治理：稽核	-	-	稽核歷史記錄成本事件
單一登入 (SSO) 支援	-	支援 Amazon Cognito	Okta、Auth 0、PingID、KeyCloak
角色型存取控制 (RBAC) 與 SAML **2.0**	-	-	Okta、Auth 0、PingID、Keycloak
企業培訓和入門	-	-	全方位服務培訓和 FinOps 入門

常見問答集

請參閱以下有關搭配使用 Kubecost 與 Amazon EKS 的常見問題和解答。

什麼是 Kubecost API 保留 (ETL) 功能？

Kubecost ETL 功能可彙總並整理指標，藉此顯示各種規模程度的成本能見度 (例如 `namespace-level`、`pod-level`，以及 `deployment-level`)。若為 Amazon EKS 最佳化 Kubecost 套件，客戶可從過去 15 天的指標中取得資料和深入解析。

什麼是提醒和週期性報告功能？ 它包含哪些提醒和報告？

Kubecost 警示可讓團隊即時接收 Kubernetes 支出和雲端支出的更新。週期性報告可讓團隊接收歷史記錄 Kubecost 和雲端支出的自訂檢視。兩者均可使用 Kubecost UI 或 Helm 值來設定。他們支援電子郵件、Slack 和 Microsoft Teams。

儲存的報告包括哪些項目？

Kubecost 儲存的報告是成本和效率指標的預先定義檢視。它們包含依叢集、命名空間、標籤以及更多類別來區分的成本。

什麼是雲端帳單整合？

與 AWS 帳單 APIs整合可讓 Kubecost out-of-cluster (例如 Amazon S3)。此外，它也能讓 Kubecost 協調 Kubecost 具有實際計費資料的叢集內預測，以說明 Spot 使用量、Savings Plans 和企業折扣。

節省成本建議包括哪些項目？

Kubecost 提供深入解析和自動化功能，協助使用者最佳化 Kubernetes 基礎結構和支出。

此功能是否需要付費？

否。您可免費使用 Amazon EKS 最佳化 Kubecost 套件。如果您想要不包含其他 Kubecost 功能，您可以透過 AWS Marketplace 或直接從 Kubecost 購買 Kubecost 的企業授權。

Amazon EKS 最佳化 Kubecost 套件是否提供支援？

是，僅在您使用 Amazon EKS 最佳化 Kubecost 套件時。

如何為 Amazon EKS 最佳化 Kubecost 套件取得支援？

您可以在聯絡 AWS 的支援團隊開立 AWS 支援案例。

我是否需要授權才能使用 Amazon EKS 整合提供的 Kubecost 功能？

否。

我可以將 Kubecost 與 AWS 成本和用量報告整合，以獲得更準確的報告嗎？

是。您可以設定 Kubecost 從 AWS 成本和用量報告中擷取資料，以取得準確的成本可見性，包括折扣、Spot 定價、預留執行個體定價等。如需詳細資訊，請參閱 Kubecost 文件中的 AWS 雲端帳單整合。

此版本是否支援 Amazon EC2 上自我管理 Kubernetes 叢集的成本管理？

否。Amazon EKS 最佳化 Kubecost 套件僅與 Amazon EKS 叢集相容。

Kubecost 是否可以追蹤 AWS Fargate 上的 Amazon EKS 的成本？

Kubecost 會盡最大努力顯示在 Fargate 上使用 Amazon EKS 時的叢集成本情況，但準確度低於在 Amazon EC2 上使用 Amazon EKS。這主要是因為用量計費方式有所不同。在 Fargate 上使用

Amazon EKS 時，您是按耗用的資源付費。若使用 Amazon EC2 節點上的 Amazon EKS，您需要支付佈建資源的費用。Kubecost 會根據節點規格計算 Amazon EC2 節點的成本，包括 CPU、RAM 和暫時性儲存。使用 Fargate 時，系統會根據 Fargate Pod 所請求的資源來計算成本。

如何獲得更新和新版的 Kubecost？

您可使用標準 Helm 升級程序升級您的 Kubecost 版本。對於 Kubecost v3，最新版本可在新的 Helm Chart 位置 取得`public.ecr.aws/kubecost/kubecost`。舊版 (v1 和 v2) 仍可在 Amazon ECR Public Gallery 中使用。

> **Important**
>
> 升級至 Kubecost v3 時，請注意 Helm Chart 位置已從 變更為 `public.ecr.aws/kubecost/cost-analyzer` `public.ecr.aws/kubecost/kubecost`。相應地更新您的 Helm 儲存庫參考。

是否支援 **`kubectl-cost`** CLI？ 如何安裝？

是。`Kubectl-cost` 是 Kubecost 的開放原始碼工具 (Apache 2.0 License)，提供對 Kubernetes 成本分配指標的 CLI 存取。若要安裝 `kubectl-cost`，請參閱 GitHub 上的 Installation (安裝)。

是否支援 Kubecost 使用者介面？ 如何存取它？

Kubecost 提供 Web 儀表板，您可透過 `kubectl` 連接埠轉送、輸入或負載平衡器來存取。您也可以使用 AWS Load Balancer控制器公開 Kubecost，並使用 Amazon Cognito 進行身分驗證、授權和使用者管理。如需詳細資訊，請參閱 AWS 部落格上的如何使用 Application Load Balancer 和 Amazon Cognito 來驗證 Kubernetes Web 應用程式的使用者。

Kubecost v3 中的新 10 萬美金支出限制是否會影響 Amazon EKS 使用者？

否。在 Kubecost v3 免費方案中引入超過 30 天的 100，000 USD 花費限制不適用於 Amazon EKS 最佳化 Kubecost 套件使用者。無論支出層級為何，EKS 使用者都會保留所有 Kubernetes 支出功能的完整存取權。

Kubecost v3 的主要效能改善為何？

Kubecost v3 透過 ClickHouse 資料庫後端引入了顯著的效能改進，與 v2.8 中使用的 DuckDB 相比，它提供了更快的查詢。此外，統一代理程式架構可消除 Prometheus 相依性，減少記憶體使用量，同時維持完整的功能和 OpenCost 相容性。

Kubecost v3 用於多叢集部署的儲存後端為何？

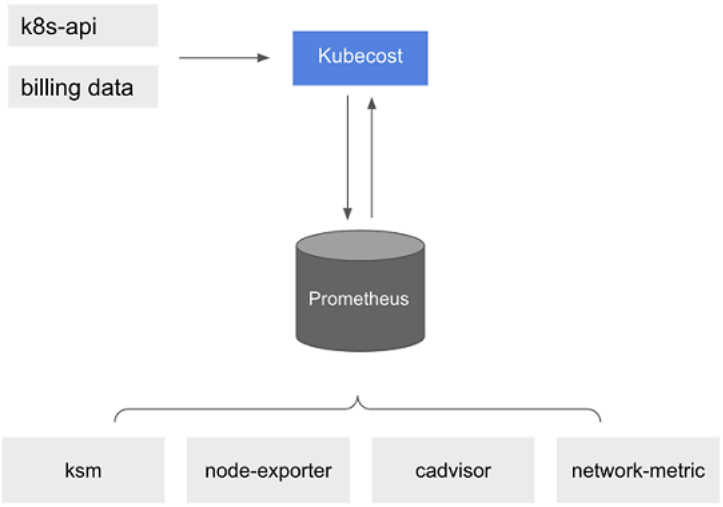
Kubecost v3 S3-compatible物件儲存（適用於 EKS 客戶的AWS S3) 進行多叢集指標儲存，取代 v2 中使用的 Prometheus 相容儲存。FinOps 代理程式會從 Kubernetes API 收集指標，並將其推送至與 S3-compatible儲存。然後，彙總器會擷取此資料、執行成本計算，並在前端顯示結果。如需多叢集設定說明的詳細資訊，請參閱 Kubecost 文件中的多叢集安裝和次要叢集指南。

我可以直接從 Kubecost v1 升級到 v3 嗎？

否。不支援從 v1 直接升級至 v3。您必須先升級至 v2，然後遷移至 v3。檢閱 Kubecost 文件以取得詳細的遷移指導，因為程序需要仔細規劃，並可能影響轉換期間的報告可用性。

其他 Kubecost 功能

- 下列功能可在 Kubecost v1、v2 和 v3 中使用。
  - 匯出成本指標 – 使用 Kubecost 部署 Amazon EKS 最佳化成本監控。在 v1 和 v2 中，Kubecost 與 Prometheus 整合，以進行指標儲存和處理。在 v3 中，Kubecost 使用 ClickHouse 資料庫大幅改善效能，同時維持 OpenCost 相容性。對於 v3 中的多叢集部署，指標存放在 S3-compatible物件儲存體中（適用於 EKS 客戶的AWS S3)，而不是 Prometheus 相容儲存體。Kubecost 會讀取指標、執行成本分配計算，並透過其 APIs和使用者介面提供資料。架構因版本而異，但會維持一致的功能。



    您可以撰寫查詢，將 Kubecost 資料擷取到您目前的商業智慧系統中，以進行進一步分析。您也可將其用作目前 Kubecost 儀表板的資料來源，以顯示您的內部團隊熟悉的 Amazon EKS 叢集成本。若要進一步了解如何撰寫查詢，請參閱 OpenCost 組態文件，或使用 Kubecost GitHub 儲存庫中的範例 Grafana JSON 模型做為參考。

- AWS 成本和用量報告整合 – 為了為您的 Amazon EKS 叢集執行成本分配計算，Kubecost 會從 AWS 價格清單 API 擷取 AWS 服務 AWS 和資源的公有定價資訊。您也可以將 Kubecost 與 AWS 成本和用量報告整合，以提升 AWS 您帳戶特定定價資訊的準確性。這類資訊包括企業折扣方案、預留執行個體使用量、Savings Plans 和 Spot 使用量。若要進一步了解 AWS 成本和用量報告整合的運作方式，請參閱 Kubecost 文件中的AWS 雲端帳單整合。

使用 Kubernetes 指標伺服器檢視資源用量

Kubernetes 指標伺服器是叢集中的資源用量資料的彙整工具，且根據預設，其不會部署於 Amazon EKS 叢集中。如需詳細資訊，請參閱 GitHub 上的 Kubernetes 指標伺服器。Metrics Server 通常由其他 Kubernetes 附加元件使用，例如使用 Horizontal Pod Autoscaler 擴展 Pod 部署或 Kubernetes Dashboard。如需詳細資訊，請參閱 Kubernetes 文件中的資源指標管道。此主題說明如何在 Amazon EKS 叢集上部署 Kubernetes 指標伺服器。

> **Important**
>
> 這些指標適用於時間點分析，並不是歷史分析的準確來源。它們不能作為監控解決方案或其他非自動擴展目的之用。如需監控工具的相關資訊，請參閱 監控叢集。

考量事項

- 如果使用資訊清單將 Kubernetes 指標伺服器手動部署到 Fargate 節點，請將 `metrics-server` 部署設定為使用預設連接埠 `10250` 以外的連接埠。此連接埠會為 Fargate 保留。指標伺服器的 Amazon EKS 附加元件版本已預先設定為使用連接埠 `10251`。
- 請確保安全群組和網路 ACL 允許 `metrics-server` Pod 之間及所有其他節點和 Pod 之間的連接埠 `10250`。Kubernetes 指標伺服器仍會使用連接埠 `10250` 收集叢集中其他端點的指標。如果您在 Fargate 節點上部署，請同時允許設定的替代指標伺服器連接埠和連接埠 `10250`。

使用 Amazon EKS 附加元件部署為社群附加元件

新功能：您現在可以使用 AWS 主控台或 Amazon EKS APIs，將 Metrics Server 部署為社群附加元件。

使用 AWS 主控台部署

1. 在 AWS 主控台中開啟您的 EKS 叢集

2. 從「附加元件」索引標籤中，選取取得更多附加元件。
3. 從「社群附加元件」區段中，選取指標伺服器，然後選取下一步
4. EKS 會為您的叢集決定適當的附加元件版本。您可以使用版本下拉式選單來變更版本。
5. 選取下一步，然後選取建立，以安裝附加元件。

其他資源

進一步了解 the section called “社群附加元件”。

您安裝或更新社群附加元件的方式與安裝或更新其他 Amazon EKS 附加元件相同。

- the section called “建立附加元件”
- the section called “更新附加元件”
- the section called “移除附加元件”

使用資訊清單進行部署

新功能：您現在可以使用 AWS 主控台或 Amazon EKS APIs，將 Metrics Server 部署為社群附加元件。這些資訊清單安裝說明將會封存。

1. 使用下列命令部署指標伺服器：

```
kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/
download/components.yaml
```

如果您使用 Fargate，則需要變更此檔案。在預設組態中，指標伺服器使用連接埠 10250。此連接埠會保留在 Fargate 上。將 components.yaml 中連接埠 10250 的參考取代為另一個連接埠，例如 10251。

2. 使用下列命令確認 `metrics-server` 部署正在執行所需數量的 Pod。

```
kubectl get deployment metrics-server -n kube-system
```

範例輸出如下。

```
NAME             READY   UP-TO-DATE   AVAILABLE   AGE
metrics-server   1/1     1            1           6m
```

3. 透過顯示節點的資源 (CPU/記憶體） 用量，測試指標伺服器是否正常運作。

```
kubectl top nodes
```

4. 如果您收到錯誤訊息 `Error from server (Forbidden)`，則需要更新 Kubernetes RBAC 組態。您的 Kubernetes RBAC 身分需要足夠的許可才能讀取叢集指標。檢閱 GitHub 上的讀取指標所需的最低 Kubernetes API 許可。了解如何授予 AWS IAM 身分，例如角色存取 Kubernetes APIs

在 Amazon EKS 上，使用 Helm 部署應用程式

Kubernetes 的 Helm 套件管理工具可協助您安裝和管理 Kubernetes 叢集上的應用程式。如需詳細資訊，請參閱 Helm 文件。此主題協助您安裝和執行 Helm 二進位檔，讓您可以在本機系統上使用 Helm CLI 安裝和管理圖表。

Important

將 Helm 圖表安裝在 Amazon EKS 叢集上之前，務必設定 `kubectl` 以搭配 Amazon EKS 來運作。若您尚未完成此動作，請參閱the section called “透過 kubectl 存取叢集” 之後再繼續。若下列命令在您的叢集上成功執行，就表示您的設定正確。

```
kubectl get svc
```

1. 為您的用戶端作業系統執行適當的命令。
   - 若您在 macOS 上使用 Homebrew，請透過下列命令安裝這些二進位檔。

     ```
     brew install helm
     ```

   - 如需更多安裝選項，請參閱 Helm 文件中的安裝 Helm。

     Note

     如果您收到必須先安裝 `openssl` 的訊息，您可以使用下列命令進行安裝。

     ```
     sudo yum install openssl
     ```

1. 若要在 中挑選新的二進位檔PATH，請關閉目前的終端機視窗並開啟新的。
2. 查看您安裝的 Helm 版本。

```
helm version --template='{{ .Version }}{{ "\n" }}'
```

範例輸出如下。

```
v3.17.2
```

3. 請確定安裝的版本與您的叢集版本相容。檢查支援的版本差異，以進一步了解。例如，如果您使用執行 `3.17.x`，支援的 Kubernetes 版本應在 `1.29.x`到 的範圍內`1.32.x`。
4. 此時，您可以執行任何 Helm 命令 (例如 `helm install` *`chart-name`*)，在叢集中安裝、修改、刪除或查詢 Helm 圖表。如果您是 Helm 新手，而且尚未安裝任何特定圖表，則您可以：
   - 透過安裝範例圖表進行實驗。請參閱 Helm 快速入門指南中的安裝範例圖表。
   - 建立範例圖表並將其推送到 Amazon ECR。如需詳細資訊，請參閱 Amazon Elastic Container Registry 使用者指南中的推送 Helm Chart。
   - 安裝 eks-charts GitHub 儲存庫或 ArtifactHub 的 Amazon EKS 圖表。

使用標籤組織 Amazon EKS 資源

您可以使用標籤協助您管理 Amazon EKS 資源。本主題提供標籤功能的概觀，並展示您如何建立標籤。

主題

- 標籤基本概念
- 標記您的 資源
- 標籤限制
- 標記您的資源以便計費
- 透過主控台使用標籤
- 透過 CLI、API 或 eksctl 使用標籤

> **Note**
>
> 標籤是一種與 Kubernetes 標籤和註釋分開的中繼資料類型。如需有關這些其他中繼資料類型的詳細資訊，請參閱 Kubernetes 文件中的下列章節：
>
> - 標籤和選取器
> - 註釋

標籤基本概念

標籤是您指派給 AWS 資源的標籤。每個標籤皆包含索引鍵與選用值。

使用標籤，您可以分類 AWS 資源。例如，您可以依用途、擁有者或環境來分類資源。當您有許多相同類型的資源時，您可以依據先前指派給特定資源的標籤來快速識別該資源。例如，您可以為 Amazon EKS 叢集定義一組標籤，協助您追蹤每個叢集的擁有者和堆疊層級。建議您為每個資源類型設計一組一致的標籤金鑰。然後，就可以根據您新增的標籤來搜尋和篩選資源。

新增標籤後，您可以隨時編輯標籤索引鍵和值，或從資源移除標籤。如果您刪除資源，也會刪除任何該資源所使用的標籤。

標籤對 Amazon EKS 來說不具有任何語意意義，並會嚴格解譯為字元字串。您可以將標籤的值設為空白字串。不過，您無法將標籤的值設為 null。若您將與現有標籤具有相同鍵的標籤新增到該資源，則新值會覆寫早前的值。

如果您使用 AWS Identity and Access Management (IAM)，您可以控制 AWS 帳戶中哪些使用者具有管理標籤的許可。

標記您的 資源

以下 Amazon EKS 資源支援標籤：

- 叢集
- 受管節點群組
- Fargate 描述檔

您可以使用以下項目標記這些資源：

- 如果使用 Amazon EKS 主控台，則可以隨時將標籤套用到新資源或現有資源。您可以在相關資源頁面使用 Tags (標籤) 索引標籤進行此操作。如需詳細資訊，請參閱the section called “透過主控台使用標籤”。
- 如果使用 `eksctl`，則可以在使用 `--tags` 選項建立時將標籤套用到資源。
- 如果您使用 AWS CLI、Amazon EKS API 或 AWS SDK，您可以使用相關 API 動作上的 `tags` 參數，將標籤套用至新資源。您也可以使用 `TagResource` API 動作將標籤套用到現有資源。如需詳細資訊，請參閱 TagResource。

當您使用某些資源建立動作時，您也可以在建立資源的同時為資源指定標籤。如果在建立資源時無法套用標籤，則無法建立資源。此機制可確保您要標記的資源是以您指定的標籤建立，不然就根本不會建立。如果您在建立資源時將其標記，則不需要在建立資源之後執行自訂標記指令碼。

標籤不會傳播到與您建立的資源相關聯的其他資源。例如，Fargate 設定檔標籤不會傳播到與 Fargate 設定檔相關聯的其他資源，例如與設定檔一起排程的 Pod。

標籤限制

以下限制適用於標籤：

- 資源最多可與 50 個標籤建立關聯。
- 一個資源的標籤索引鍵不能重複。每個標籤索引鍵都必須是唯一的，而且只能有一個值。
- 索引鍵的長度上限是 128 個 UTF-8 字元。
- 索引鍵的長度上限是 256 個 UTF-8 字元。
- 如果多個 AWS 服務和資源使用您的標記結構描述，請限制您使用的字元類型。某些服務可能對允許的字元設有限制。通常允許的字元為：字母、數字和空格，以及下列字元：`+ - = . _ : / @`。
- 標籤鍵與值皆區分大小寫。
- 不使用 `aws:`、 `AWS:` 或任何大小寫組合作為索引鍵或值的字首。這些僅供保留 AWS 使用。您不可編輯或刪除具有此字首的標籤索引鍵或值。具有此字首的標籤不算在每一資源的標籤數限制內。

標記您的資源以便計費

當您將標籤套用到 Amazon EKS 叢集時，您可以在成本與用量報告中使用標籤來分配成本。成本與用量報告中的計量資料會顯示所有 Amazon ECS 叢集的用量。如需詳細資訊，請參閱《 AWS 帳單使用者指南》中的AWS 成本和用量報告。

AWS 產生的成本分配標籤，特別是 ，`aws:eks:cluster-name`可讓您依 Cost Explorer 中的個別 Amazon EKS 叢集細分 Amazon EC2 執行個體成本。但是，此標籤不會擷取控制平面費用。標籤會自動新增至參與 Amazon EKS 叢集的 Amazon EC2 執行個體中。無論執行個體是使用 Amazon EKS 受管節點群組、Karpenter 或直接使用 Amazon EC2 佈建，都會發生此行為。此特定標籤不會計入 50 個標籤限制。若要使用 標籤，帳戶擁有者必須在 AWS 帳單主控台中或使用 API 來啟用該標籤。當 AWS Organizations 管理帳戶擁有者啟用標籤時，也會為所有組織成員帳戶啟用該標籤。

您也可以根據具有相同標籤索引鍵值的資源來整理您的帳單資訊。例如，您可以使用特定應用程式名稱來標記數個資源，然後整理帳單資訊。這樣一來，您就可以查看該應用程式跨數項服務的總成本。如需使用標籤設定成本分配報告的詳細資訊，請參閱 帳單使用者指南中的每月成本分配報告。 AWS

> **Note**
>
> 如果您剛啟用報告，當月資料會在 24 小時之後提供檢視。

Cost Explorer 是一種報告工具，可作為 AWS 免費方案的一部分使用。您可以使用 Cost Explorer 檢視過去 13 個月的 Amazon EKS 資源圖表。您還可以預測未來三個月可能花費的金額。您可以查看 AWS 一段時間內資源花費的模式。例如，您可以用它來找出需進一步調查的領域，以及查看您可用來了解成本的趨勢。您也可以指定資料的時間範圍，以及根據天或月檢視時間資料。

透過主控台使用標籤

您可以使用 Amazon EKS 主控台，管理與新的或現有的叢集和受管節點群組相關聯的標籤。

當您在 Amazon EKS 主控台中選取資源限定頁面時，該頁面會顯示那些資源的清單。例如，若您從左側導航窗格選取 Clusters (叢集)，主控台會顯示 Amazon EKS 叢集的清單。當您從支援標籤的其中一個清單選取資源時 (例如，特定的叢集)，您便可以在 Tags (標籤) 索引標籤上檢視和管理其標籤。

您也可以在 中使用標籤編輯器 AWS 管理主控台，提供統一的方式來管理您的標籤。如需詳細資訊，請參閱《標籤編輯器使用者指南》中的使用標籤編輯器標記您的 AWS 資源。 AWS

在建立資源時新增標籤

在建立 Amazon EKS 叢集、受管節點群組和 Fargate 描述檔時，您可以將標籤新增至其中。如需詳細資訊，請參閱the section called “建立 叢集”。

在資源上新增和刪除標籤

您可以直接從資源的頁面新增或刪除與叢集相關聯的標籤。

1. 開啟 Amazon EKS 主控台。
2. 在導覽列上，選取要使用的 AWS 區域。
3. 在左側導覽窗格中選擇叢集。
4. 選擇特定叢集。
5. 選擇 Tags (標籤) 索引標籤，然後選擇 Manage tags (管理標籤)。
6. 在 Manage tags (管理標籤) 頁面上，視需要新增或刪除標籤。
   - 若要新增標籤，請選擇 Add tag (新增標籤)。然後指定每個標籤的索引鍵和值。
   - 若要移除標籤，請選擇 Remove tag (移除標籤)。
7. 針對您要新增或刪除的每個標籤重複此程序。
8. 選擇 Update (更新) 以完成操作。

透過 CLI、API 或 `eksctl` 使用標籤

使用下列 AWS CLI 命令或 Amazon EKS API 操作來新增、更新、列出和刪除資源的標籤。您只能使用 `eksctl` 新增標籤，同時使用一個命令建立新資源。

任務	AWS CLI	AWS 適用於 Windows PowerShell 的工具	API 動作
新增或覆寫一或多個標籤。	tag-resource	Add-EKSResourceTag	TagResource
刪除一或多個標籤。	untag-resource	Remove-EKSResourceTag	UntagResource

下列範例示範如何使用 CLI AWS 標記或取消標記資源。

範例 1：標記現有的叢集

以下命令標記現有的叢集。

```
aws eks tag-resource --resource-arn resource_ARN --tags team=devs
```

範例 2：取消標記現有的叢集

以下命令從現有的叢集刪除標籤。

```
aws eks untag-resource --resource-arn resource_ARN --tag-keys tag_key
```

範例 3：列出資源的標籤

以下命令列出與現有資源相關聯的標籤。

```
aws eks list-tags-for-resource --resource-arn resource_ARN
```

當您使用某些資源建立動作時，您可以在建立資源的同時指定標籤。下列動作支援在建立資源時指定標籤。

任務	AWS CLI	AWS 適用於 Windows PowerShell 的工具	API 動作	eksctl
建立 叢集	create-cluster	New-EKSCluster	CreateCluster	`create cluster`
建立受管節點群組*	create-nodegroup	New-EKSNodegroup	CreateNodegroup	`create nodegroup`
建立 Fargate 設定檔	create-fargate-profile	New-EKSFargateProfile	CreateFargateProfile	`create fargatepr ofile`

- 如果還想在建立受管節點群組時為 Amazon EC2 執行個體加上標籤，請使用啟動範本建立受管節點群組。如需詳細資訊，請參閱the section called “標記 Amazon EC2 執行個體”。如果您的執行個體已經存在，則可以手動為執行個體加上標籤。如需詳細資訊，請參閱《Amazon EC2 使用者指南》中的標記您的資源。

檢視與管理 Amazon EKS 和 Fargate 服務配額

Amazon EKS 已與 Service Quotas 整合，此服務 AWS 可讓您從中央位置檢視和管理配額。如需詳細資訊，請參閱Service Quotas 使用者指南中的什麼是 Service Quotas？。透過 Service Quotas 整合，您可以使用 和 CLI 快速查詢 Amazon EKS AWS 管理主控台 和 AWS Fargate AWS 服務配額的值。

在 中檢視 EKS 服務配額 AWS 管理主控台

1. 開啟 Service Quotas 主控台。
2. 在左側導覽窗格中，選擇 AWS 服務。
3. 從 AWS 服務清單中，搜尋並選取 Amazon Elastic Kubernetes Service (Amazon EKS) 或 AWS Fargate。

   在服務配額清單中，您可以看到服務配額名稱、套用值（如果有的話）、 AWS 預設配額，以及配額值是否可以調整。
4. 若要檢視服務配額的其他資訊 (例如說明)，請選擇配額名稱。
5. (選用) 若要請求增加配額，請選取您要增加的配額、選取 Request quota increase (請求增加配額)、輸入或選取必要資訊，然後選取 Request (請求)。

若要使用 進一步使用服務配額 AWS 管理主控台，請參閱 Service Quotas 使用者指南。若要請求提升配額，請參閱《Service Quotas 使用者指南》中的請求提升配額。

使用 CLI 檢視 EKS AWS 服務配額

執行下列命令，以檢視您的 Amazon EKS 配額。

```
aws service-quotas list-aws-default-service-quotas \
    --query 'Quotas[*].
{Adjustable:Adjustable,Name:QuotaName,Value:Value,Code:QuotaCode}' \
    --service-code eks \
    --output table
```

執行下列命令，以檢視您的 Fargate 配額。

```
aws service-quotas list-aws-default-service-quotas \
    --query 'Quotas[*].
{Adjustable:Adjustable,Name:QuotaName,Value:Value,Code:QuotaCode}' \
    --service-code fargate \
    --output table
```

> **Note**
>
> 傳回的配額是可在目前 AWS 區域中此帳戶中的 Fargate 上同時執行的 Amazon ECS 任務或 Amazon EKS Pod 數量。

若要使用 CLI AWS 使用更多服務配額，請參閱 AWS CLI 命令參考中的服務配額。若要請求提高配額，請參閱 CLI 命令參考中的 request-service-quota-increase 命令。 AWS

Amazon EKS 服務配額

AWS 建議使用 AWS 管理主控台 來檢視您目前的配額。如需詳細資訊，請參閱the section called “在中檢視 EKS 服務配額 AWS 管理主控台”。

若要檢視預設 EKS 服務配額，請參閱《 AWS 一般參考》中的 Amazon Elastic Kubernetes Service 端點和配額。

在 Service Quotas 主控台中，Amazon Elastic Kubernetes Service (Amazon EKS) 之下會列出這些服務配額。若要針對顯示為可調整的值申請增加配額，請參閱《Service Quotas 使用指南》中的請求提高配額。

> **Note**
>
> Service Quotas 不支援對以下元件做出調整：* 每個叢集的 Pod 身分識別關聯。若要了解限制相關資訊，請參閱 the section called “Pod 身分識別”。* 適用於遠端節點網路的 CIDR，或是適用於混合節點的遠端 Pod 網路。若要了解限制相關資訊，請參閱 the section called “混合節點”。

AWS Fargate 服務配額

Service Quotas 主控台中的 AWS Fargate 服務會列出數個服務配額。您可以設定警示，在您的用量接近服務配額時發出警示。如需詳細資訊，請參閱the section called “建立 CloudWatch 警示來監控 Fargate 資源用量指標”。

新 AWS 帳戶的初始配額可能較低，可能會隨著時間增加。Fargate 會持續監控每個 AWS 區域中的帳戶用量，然後根據用量自動增加配額。您也可以針對顯示為可調整的值申請增加配額。如需詳細資訊，請參閱「Service Quotas 使用者指南」中的請求提高配額。

AWS 建議使用 AWS 管理主控台 來檢視您目前的配額。如需詳細資訊，請參閱the section called “在中檢視 EKS 服務配額 AWS 管理主控台”。

若要檢視 EKS 服務配額上的預設 AWS Fargate，請參閱《 AWS 一般參考》中的 Fargate 服務配額。

> **Note**
>
> Fargate 還強制執行 Amazon ECS 任務和 Amazon EKS Pod 啟動速率配額。如需詳細資訊，請參閱 Amazon ECS 指南中的 AWS Fargate 限流配額。

Amazon EKS 中的安全

的雲端安全性 AWS 是最高優先順序。身為 AWS 客戶，您可以受益於資料中心和網路架構，這些架構是為了滿足最安全敏感組織的需求而建置。

安全性是 AWS 與您之間的共同責任。共同責任模型 將此描述為雲端的安全和雲端內的安全：

- 雲端的安全性 – AWS 負責保護在 AWS 雲端中執行 AWS 服務的基礎設施。對於 Amazon EKS，AWS 負責 Kubernetes 控制平面，其中包含控制平面節點和etcd資料庫。在 AWS 合規計畫中，第三方稽核員會定期測試並驗證我們的安全功效。若要了解適用於 Amazon EKS 的合規計劃，請參閱合規計劃範圍內的AWS 服務。
- 雲端內部的安全 – 您的責任包含下列領域：
  - 資料平面的安全組態，包括允許流量從 Amazon EKS 控制平面傳遞至客戶 VPC 的安全群組各項組態
  - 節點和容器本身的組態
  - 節點的作業系統 (包括更新和安全修補程式)
  - 其他相關的應用程式軟體：
    - 設定及管理網路控制，例如防火牆規則
    - 搭配 IAM 或另以其他方式管理平台層級身分與存取管理
  - 資料的機密性、公司的要求，以及適用法律和法規

Amazon EKS 獲得多項合規計畫認證，適用於受監管及敏感型應用程式。Amazon EKS 滿足 SOC、PCI、ISO、FedRAMP-Moderate、IRAP、C5、K-ISMS、ENS High、OSPAR、HITRUST CSF 等合規要求，並且是一項符合 HIPAA 資格的服務。如需詳細資訊，請參閱管理存取權。

本文件有助於您了解如何在使用 Amazon EKS 時套用共同責任模型。下列主題說明如何將 Amazon EKS 設定為符合您的安全與合規目標。您也會了解如何使用其他 AWS 服務來協助您監控和保護 Amazon EKS 資源。

> **Note**
>
> Linux 容器由控制群組 (cgroup) 和命名空間組成，這些命名空間有助於限制容器可存取的內容，但是所有容器皆會與主機 Amazon EC2 執行個體共用相同的 Linux 核心。非常不建議您以根使用者身分執行容器 (UID 0)，或授予容器存取主機資源或命名空間 (例如主機網路或主機 PID 命名空間)，因為這樣做會降低容器所提供之隔離的有效性。

主題

- 藉助最佳實務來保護 Amazon EKS 叢集
- 分析 Amazon EKS 中的漏洞
- Amazon EKS 叢集的合規性驗證
- Amazon Elastic Kubernetes Service 的安全考量
- Kubernetes 的安全性考量事項
- Amazon EKS 自動模式的安全考量
- EKS 功能的安全考量
- Amazon EKS 的 Identity and Access Management

藉助最佳實務來保護 Amazon EKS 叢集

請參閱 Amazon EKS 最佳實務指南中的中的安全最佳實務，了解 Amazon EKS 安全最佳實務。

分析 Amazon EKS 中的漏洞

安全性是設定和維護 Kubernetes 叢集和應用程式的重要考量因素。以下列出可供您分析 EKS 叢集安全組態的資源、可供您檢查漏洞的資源，以及可為您執行該分析 AWS 的服務整合。

Center for Internet Security (CIS) 的 Amazon EKS 基準

Center for Internet Security (CIS) Kubernetes 基準提供 Amazon EKS 安全組態的指導。基準化分析：

- 適用於您負責 Kubernetes 元件的安全組態的 Amazon EC2 節點 (受管和自我管理)。
- 提供經社群核准的標準方式，以確保您在使用 Amazon EKS 時能安全地設定 Kubernetes 叢集和節點。
- 由四個部分組成：控制平面記錄設定、節點安全設定、政策和受管服務。
- 支援 Amazon EKS 目前可用的所有 Kubernetes 版本，並且可以使用 kube-bench 予以執行，這是一個標準開源工具，可用於在 Kubernetes 叢集上使用 CIS 基準檢查組態。

若要進一步了解，請參閱 CIS Amazon EKS 基準簡介。

有關使用符合 CIS 基準的 AMI 更新節點群組的自動化 `aws-sample` 管道，請參閱 EKS 最佳化 AMI 強化管道。

Amazon EKS 平台版本

Amazon EKS 平台版本代表叢集控制平面的功能，包括啟用了哪些 Kubernetes API 伺服器旗標，以及目前的 Kubernetes 修補程式版本。新叢集是使用最新的平台版本部署。如需詳細資訊，請參閱 EKS 平台版本。

您可以將 Amazon EKS 叢集更新為較新的 Kubernetes 版本。當 Amazon EKS 中提供新的 Kubernetes 版本時，建議您主動更新叢集，以便使用最新的可用版本。有關 EKS 中 Kubernetes 版本的更多資訊，請參閱 Amazon EKS-Anywhere 支援的版本。

作業系統漏洞清單

AL2023 漏洞清單

在 Amazon Linux 安全中心追蹤 Amazon Linux 2023 的安全或隱私權事件，或訂閱相關聯的 RSS 摘要。安全性和隱私權事件包括問題概觀、受影響的套件，以及更新執行個體以修正問題的指示。

Amazon Linux 2 漏洞清單

可以透過 Amazon Linux 安全中心或是訂閱關聯的 RSS 摘要，追蹤 Amazon Linux 2 的安全或隱私權事件。安全性和隱私權事件包括問題概觀、受影響的套件，以及更新執行個體以修正問題的指示。

使用 Amazon Inspector 進行節點偵測

您可以使用 Amazon Inspector 檢查節點是否有意外的網路存取問題，以及 Amazon EC2 執行個體是否有漏洞。

使用 Amazon GuardDuty 進行叢集與節點偵測

Amazon GuardDuty 是一種威脅偵測服務，可協助保護您的帳戶、容器、工作負載和 AWS 環境中的資料。除了這些功能外，GuardDuty 還提供下列兩種功能，可偵測對 EKS 叢集的潛在威脅：EKS 保護和執行時期監控。

如需詳細資訊，請參閱the section called “Amazon GuardDuty”。

Amazon EKS 叢集的合規性驗證

若要了解 AWS 服務是否在特定合規計劃的範圍內，請參閱AWS 合規計劃範圍內的服務，然後選擇您感興趣的合規計劃。如需一般資訊，請參閱 AWS 合規計劃。

您可以使用 AWS Artifact 下載第三方稽核報告。如需詳細資訊，請參閱以 AWS 成品下載報告。

您使用 AWS 服務的合規責任取決於資料的機密性、您公司的合規目標，以及適用的法律和法規。AWS 提供下列資源來協助合規：

- 安全合規與治理 - 這些解決方案實作指南內容討論了架構考量，並提供部署安全與合規功能的步驟。
- HIPAA 合格服務參考 – 列出 HIPAA 合格服務。並非所有 AWS 服務都符合 HIPAA 資格。
- AWS 合規資源 – 此工作手冊和指南集合可能適用於您的產業和位置。
- AWS 客戶合規指南 – 透過合規的角度了解共同責任模型。本指南摘要說明保護 AWS 服務的最佳實務，並將指引映射至跨多個架構的安全控制（包括國家標準技術研究所 (NIST)、支付卡產業安全標準委員會 (PCI) 和國際標準化組織 (ISO))。
- AWS Config 開發人員指南中的使用規則評估資源 – Config AWS 服務會評估資源組態符合內部實務、產業準則和法規的程度。
- AWS Security Hub – AWS 此服務可讓您全面檢視其中的安全狀態 AWS。Security Hub 使用安全控制來評估您的 AWS 資源，並檢查是否符合安全產業標準和最佳實務。如需支援的服務和控制清單，請參閱 Security Hub 控制參考。
- Amazon GuardDuty – AWS 此服務透過監控您的環境是否有可疑和惡意活動，來偵測對 AWS 您的帳戶、工作負載、容器和資料的潛在威脅。GuardDuty 可滿足特定合規架構所規定的入侵偵測需求，以協助您因應 PCI DSS 等各種不同的合規需求。
- AWS Audit Manager – AWS 此服務可協助您持續稽核 AWS 用量，以簡化您管理風險的方式，以及符合法規和業界標準的方式。

Amazon Elastic Kubernetes Service 的安全考量

下面介紹了雲端安全性考量，因為安全性會對 Amazon EKS 產生影響。

主題

- Amazon EKS 中的基礎設施安全
- 了解 Amazon EKS 叢集中的彈性
- Amazon EKS 中的預防跨服務混淆代理人

Amazon EKS 中的基礎設施安全

作為受管服務，Amazon Elastic Kubernetes Service 受到 AWS 全球網路安全的保護。如需 AWS 安全服務以及 AWS 如何保護基礎設施的相關資訊，請參閱 AWS 雲端安全。若要使用基礎設施安全性的最佳實務來設計您的 AWS 環境，請參閱安全支柱 AWS 架構良好的框架中的基礎設施保護。

您可使用 AWS 發佈的 API 呼叫，透過網路存取 Amazon EKS。使用者端必須支援下列專案：

- Transport Layer Security (TLS)。我們需要 TLS 1.2 並建議使用 TLS 1.3。
- 具備完美轉送私密(PFS)的密碼套件，例如 DHE (Ephemeral Diffie-Hellman)或 ECDHE (Elliptic Curve Ephemeral Diffie-Hellman)。現代系統(如 Java 7 和更新版本)大多會支援這些模式。

此外，請求必須使用存取金鑰 ID 和與 IAM 主體相關聯的私密存取金鑰來簽署。或者，您可以使用 AWS Security Token Service (AWS STS) 來產生暫時安全憑證，以簽署請求。

建立 Amazon EKS 叢集時，為要使用的叢集指定 VPC 子網路。Amazon EKS 需要至少兩個處於可用區域的子網路。我們建議使用具有公有和私有子網路的 VPC，以便 Kubernetes 在公有子網路中建立公有負載平衡器，以平衡在私有子網路中的節點上執行的 Pod 負載。

如需 VPC 考量的詳細資訊，請參閱 the section called “VPC 和子網要求”。

如果您透過 Amazon EKS 入門逐步解說提供的 AWS CloudFormation 範本建立 VPC 和節點群組，您的控制平面和節點安全群組將會設定為建議的設定。

如需安全群組考量的詳細資訊，請參閱 the section called “安全群組要求”。

建立新的叢集時，Amazon EKS 會為您用來與叢集通訊的受管 Kubernetes API 伺服器建立端點 (使用 Kubernetes 管理工具，例如 `kubectl`)。根據預設，此 API 伺服器端點會在網際網路上公開，而其存取則受到 AWS Identity and Access Management (IAM) 及原生 Kubernetes 角色型存取控制 (RBAC) 的共同保護。

您可啟用 Kubernetes API 伺服器的私有存取，讓節點和 API 伺服器間的所有通訊都不會離開 VPC。您可以限制可以從網際網路存取 API 伺服器的 IP 地址，或完全停用對 API 伺服器的網際網路存取。

如需修改叢集端點存取的詳細資訊，請參閱 the section called “修改叢集端點存取”。

您可以搭配 Amazon VPC CNI 或第三方工具 (例如 Project Calico) 來實作 Kubernetes 網路政策。如需有關使用適用於網路政策的 Amazon VPC CNI 的詳細資訊，請參閱 the section called “Kubernetes 政策”。Project Calico 是第三方開放原始碼專案。如需詳細資訊，請參閱 Project Calico 文件。

使用 AWS PrivateLink 存取 Amazon EKS

您可以使用 AWS PrivateLink 在 VPC 和 Amazon Elastic Kubernetes Service 之間建立私有連線。您可以像在 VPC 中一樣存取 Amazon EKS，無需使用網際網路閘道、NAT 裝置、VPN 連接或 AWS Direct Connect 連接。VPC 中的執行個體無需公有 IP 位址，即可存取 Amazon EKS。

您可以透過建立採用 AWS PrivateLink 技術的介面端點來建立此私有連線。我們會在您為介面端點啟用的每個子網中建立端點網路介面。這些是請求者管理的網路介面，可作為目的地為 Amazon EKS 之流量的進入點。

如需詳細資訊，請參閱《 AWS PrivateLink 指南》中的透過 PrivateLink 存取 AWS 服務。 AWS PrivateLink

開始之前

開始之前，請確定您已執行下列任務：

- 《 AWS PrivateLink 指南》中的檢閱使用介面 VPC 端點存取 AWS 服務

考量事項

- 支援與限制：Amazon EKS 介面端點支援透過 VPC 安全地存取全部 Amazon EKS API 動作，但同時有特定限制：不支援存取 Kubernetes API，因為他們具有單獨的私有端點，因此，您不能將 Amazon EKS 設定為僅可透過介面端點存取。IAM 角色用於服務帳戶 (IRSA) 的叢集 OIDC 探索和 JWKS 端點，也會從具有其服務名稱和存取控制行為的獨立介面端點提供；請參閱 the section called “使用 AWS PrivateLink 存取叢集 OIDC 端點”。
- 定價：使用 Amazon EKS 的界面端點會產生 standard AWS PrivateLink 費用：每個可用區域中佈建的每個端點的每小時費用、透過端點的流量的資料處理費用。如需了解詳細資訊，請參閱 AWS PrivateLink 定價。
- 安全與存取權控制：建議透過這些額外的組態來增強安全性及控制存取權 – 藉助 VPC 端點政策，透過介面端點來控制對 Amazon EKS 的存取、關聯安全群組與端點網路介面來管理流量、使用 VPC 流量日誌來擷取及監控傳入與傳出介面端點的 IP 流量，以及可將日誌發布至 Amazon CloudWatch 或 Amazon S3。如需了解詳細資訊，請參閱藉助端點政策來控制對 VPC 端點的存取權，以及藉助 VPC 流量日誌來記錄 IP 流量日誌。
- 連線選項：界面端點使用內部部署存取（使用 AWS Direct Connect 或 AWS Site-to-Site VPN 將內部部署資料中心連接至 VPC 與介面端點）或透過跨 VPC 連線（使用 AWS Transit Gateway 或 VPC 對等互連將其他 VPCs 與介面端點連接至 VPC，將流量保留在 AWS 網路內）提供彈性的連線選項。
- IP 版本支援：2024 年 8 月之前建立的端點僅支援 IPv4 (使用 eks.region.amazonaws.com)。2024 年 8 月之後建立的全新端點則支援雙重堆疊 IPv4 和 IPv6 (例如 eks.region.amazonaws.com、eks.region.api.aws)。
- 區域可用性：EKS API 的 AWS PrivateLink 不適用於亞太區域（馬來西亞）(ap-southeast-5)、亞太區域（泰國）(ap-southeast-7)、墨西哥（中部）(mx-central-1) 和亞太區域（台北）(ap-

east-2) 區域。 AWS PrivateLink 對 eks-auth (EKS Pod Identity) 的支援適用於亞太區域 （馬來西亞） (ap-southeast-5) 區域。叢集 OIDC 端點 (`com.amazonaws.region-code.oidc-eks`) 的 AWS PrivateLink 適用於大多數 AWS 區域。

建立 Amazon VPC 的介面端點

您可以使用 Amazon VPC 主控台或 AWS 命令列界面 (AWS CLI) 來建立 Amazon EKS 的界面端點。如需詳細資訊，請參閱 AWS PrivateLink 指南中的建立 VPC 端點。

使用以下服務名稱為 Amazon EKS 建立介面端點：

EKS API

- com.amazonaws.region-code.eks
- com.amazonaws.region-code.eks-fips (適用於 FIPS 相容端點)

EKS 驗證 API (EKS Pod 身分識別)

- com.amazonaws.region-code.eks-auth

EKS 叢集 OIDC 端點 (IRSA 探索/JWKS)

- com.amazonaws.region-code.oidc-eks

此端點的存取控制行為與 EKS API 端點不同。如需詳細資訊，請參閱the section called “使用 AWS PrivateLink 存取叢集 OIDC 端點”。

EKS 主控台資源檢視

- com.amazonaws.region-code.eks-proxy

此端點服務可支援 AWS 主控台中的叢集資源檢視。它由 AWS受管主控台和服務使用，而不是由您的應用程式直接呼叫（沒有公有 SDK 或 CLI)。當您的 VPC 沒有網際網路輸出，或當您覆寫私有託管`eks.region-code.amazonaws.com`區域中的區域時，請建立此端點。在這些情況下，主控台資源檢視是必要的。

此端點會註冊兩個私有 DNS 名稱：

- `eks-proxy.region-code.api.aws` （雙堆疊）

- `eks-proxy.eks.region-code.amazonaws.com` (IPv4；在IPv4-only的網路上使用)

適用於 Amazon EKS 介面端點的私有 DNS 功能

預設針對 Amazon EKS 和其他 AWS 服務的介面端點啟用的私有 DNS 功能，可使用預設的區域 DNS 名稱來促進安全且私有的 API 請求。此功能可確保透過私有 AWS 網路的界面端點路由 API 呼叫，從而增強安全性和效能。

當您為 Amazon EKS 或其他 AWS 服務建立介面端點時，私有 DNS 功能會自動啟用。如需啟用，需要藉由設定特定屬性來正確設定 VPC：

- enableDnsHostnames：允許 VPC 內的執行個體具有 DNS 主機名稱。
- enableDnsSupport：啟用整個 VPC 的 DNS 解析。

若要了解檢查或修改這些設定的逐步說明，請參閱檢視並更新 VPC 的 DNS 屬性。

DNS 名稱與 IP 位址類型

若啟用私有 DNS 功能，您可使用特定 DNS 名稱來連接至 Amazon EKS，並且這些選項可隨時間推移而演進：

- eks.region.amazonaws.com：傳統 DNS 名稱，僅解析為 2024 年 8 月之前的 IPv4 位址。針對更新至雙重堆疊的現有端點，此名稱解析為 IPv4 與 IPv6 位址。
- eks.region.api.aws：適用於 2024 年 8 月之後建立的新端點，此雙重堆疊 DNS 名稱解析為 IPv4 與 IPv6 位址。

在 2024 年 8 月之後，新介面端點帶有兩個 DNS 名稱，並且您可選擇雙重堆疊 IP 位址類型。若是現有端點，更新至雙重堆疊會修改 eks.region.amazonaws.com，以支援 IPv4 與 IPv6。

使用私有 DNS 功能

一旦設定完成，私有 DNS 功能即可整合至工作流程，提供以下功能：

- API 請求：依據端點的設定，使用預設的區域性 DNS 名稱 `eks.region.amazonaws.com` 或 `eks.region.api.aws`，向 Amazon EKS 發出 API 請求。
- 應用程式相容性：現有應用程式呼叫 EKS API，無須任何變更即可善用此功能。
- 具有雙堆疊的 AWS CLI：若要搭配 CLI AWS 使用雙堆疊端點，請參閱 AWS SDKs和工具參考指南中的雙堆疊和 FIPS 端點組態。

- 自動路由：對 Amazon EKS 預設服務端點進行任何呼叫，都會透過介面端點自動引導，從而確保私有、安全的連接。

使用 AWS PrivateLink 存取叢集 OIDC 端點

每個 Amazon EKS 叢集都會在其 OIDC 發行者 URL 上發佈公有 OpenID Connect (OIDC) 探索文件和 JSON Web 金鑰集 (JWKS)。服務帳戶的 IAM 角色 (IRSA) 使用此端點來發佈叢集的公有簽署金鑰。設定 IRSA 或驗證字符的工具會直接從此端點擷取金鑰。

您可以使用 AWS PrivateLink 從您的 VPC 私下到達此 OIDC 端點，無需網際網路輸出。這適用於需要傳入 OIDC 端點的私有或無輸出 VPCs，例如，建立叢集的 IAM OIDC 身分提供者 (`eksctl`、Terraform 或 AWS CLI)，或在 VPC 內執行您自己的字符驗證程式。先前，到達此端點需要網際網路存取或 DNS 解決方法 （ 請參閱 GitHub 上的 container-roadmap#2038)。

使用下列服務名稱建立叢集 OIDC 端點的介面端點：

- `com.amazonaws.region-code.oidc-eks`

考量事項

- 此端點不支援 VPC 端點政策。OIDC 探索和 JWKS 文件會以匿名方式提供。這是 OpenID Connect 規格的必要項目，也是 IRSA 的運作方式：驗證程式在不顯示登入資料的情況下擷取公有金鑰。由於請求沒有 IAM 主體或動作，因此 VPC 端點政策沒有可評估的項目。此端點僅接受預設的完整存取政策。控制誰可以使用端點網路介面上的安全群組和子網路路由來到達端點。
- 私有連線不是授權界限。透過 AWS PrivateLink 到達 OIDC 端點會將此流量保留在 AWS 網路上 — 它本身不會控制誰可以擔任 IAM 角色。提供的資料是公有金鑰材料。擔任角色的信任政策會強制執行 IRSA 授權，而不是用於到達 OIDC 端點的網路路徑。當工作負載呼叫 時，就會發生此強制執行`sts:AssumeRoleWithWebIdentity`。控制哪些工作負載可以使用角色的信任政策中的 `aud`和 `sub`條件擔任角色；請參閱將 IAM 角色指派給 Kubernetes 服務帳戶。
- AWS STS 會在自己的路徑上驗證權杖。此端點不會影響 AWS STS 驗證 IRSA 字符的方式。當工作負載呼叫 時`AssumeRoleWithWebIdentity`，STS 會從內部擷取叢集的 JWKS AWS，而不是透過 VPC 或此端點。建立此端點或覆寫 VPC 中的 OIDC DNS 不會變更權杖驗證。若要同時保持工作負載的`AssumeRoleWithWebIdentity`呼叫私有，請建立 AWS STS 介面端點並分別設定區域 STS 端點；請參閱設定服務帳戶 AWS 的安全字符服務端點。
- 定價：需支付 Standard AWS PrivateLink 費用。請參閱 AWS PrivateLink 定價。

私有 DNS 和雙堆疊名稱

每個叢集都有一個 OIDC 發行者主機名稱，由叢集的 IP 系列決定。IPv4 叢集預設為使用 `oidc.eks.region-code.amazonaws.com`。IPv6 叢集使用雙堆疊主機名稱 `oidc-eks.region-code.api.aws`，可解析為 IPv4 和 IPv6 地址。 AWS 在中國區域中，雙堆疊主機名稱為 `oidc-eks.region-code.api.amazonwebservices.com.cn`。

介面端點會註冊這兩個主機名稱。啟用私有 DNS 時（預設值），每個主機名稱都會解析為您的介面端點。因此，叢集的發行者 URL 會解析為端點，無論叢集使用`oidc.eks.region-code.amazonaws.com`主機名稱還是雙堆疊主機名稱。遵循字符發行者 URL 來擷取探索文件的用戶端，而且 JWKS 在這兩種情況下都私下到達端點。

了解 Amazon EKS 叢集中的彈性

AWS 全球基礎設施是以 AWS 區域和可用區域為基礎建置。 AWS 區域提供多個實體隔離和隔離的可用區域，這些區域以低延遲、高輸送量和高度備援聯網連接。透過可用區域，您所設計與操作的應用程式和資料庫，就能夠在可用區域之間自動容錯移轉，而不會發生中斷。可用區域的可用性、容錯能力和擴充能力，均較單一或多個資料中心的傳統基礎設施還高。

Amazon EKS 會跨多個 AWS 可用區域執行和擴展 Kubernetes 控制平面，以確保高可用性。Amazon EKS 會根據負載來自動擴展控制平面執行個體，且會偵測及取代狀態不佳的控制平面執行個體，還會對控制平面進行自動化程式修補。啟動版本更新後，Amazon EKS 會更新控制平面，並在更新期間維持控制平面的高可用性。

此控制平面由至少兩個 API 伺服器執行個體和三個執行個體組成，這些`etcd`執行個體在 AWS 區域內的三個可用區域中執行。Amazon EKS：

- 主動監控控制平面執行個體上的負載，並自動進行擴展以確保高效能。
- 自動偵測並取代運作狀態不佳的控制平面執行個體，並視需要跨 AWS 區域內的可用區域重新啟動執行個體。
- 利用 AWS 區域的架構來維持高可用性。因此，Amazon EKS 能夠提供 API 伺服器端點可用性的 SLA。

如需 AWS 區域和可用區域的詳細資訊，請參閱 AWS 全球基礎設施。

Amazon EKS 中的預防跨服務混淆代理人

混淆代理人問題屬於安全性議題，其中沒有執行動作許可的實體可以強制具有更多許可的實體執行該動作。在 中 AWS，跨服務模擬可能會導致混淆代理人問題。在某個服務 (呼叫服務) 呼叫另一個服務 (被

呼叫服務) 時，可能會發生跨服務模擬。可以操縱呼叫服務來使用其許可，以其不應有存取許可的方式對其他客戶的資源採取動作。為了防止這種情況， AWS 提供工具，協助您保護所有 服務的資料，讓服務主體能夠存取您帳戶中的資源。

若要限制 Amazon Elastic Kubernetes Service (Amazon EKS) 為資源提供另一項服務的許可，我們推薦在資源政策中使用 aws:SourceArn、aws:SourceAccount 全域條件內容索引鍵。

aws:SourceArn

使用 aws:SourceArn，僅將一個資源與跨服務存取權相關聯。

aws:SourceAccount

使用 aws:SourceAccount，讓該帳戶中的任何資源都與跨服務使用相關聯。

防範混淆代理人問題的最有效方法是使用 aws:SourceArn 全域條件內容索引鍵，以及資源的完整 ARN。如果不知道資源的完整 ARN，或者如果您指定了多個資源，請使用 aws:SourceArn 全域內容條件索引鍵搭配萬用字元 (*) 來表示 ARN 的未知部分。例如 arn:aws:<servicename>:*:<123456789012>:*。

如果 aws:SourceArn 值不包含帳戶 ID (例如 Amazon S3 儲存貯體 ARN)，您必須同時使用 aws:SourceAccount 和 aws:SourceArn 來限制許可。

Amazon EKS 叢集角色預防跨服務混淆代理人

每個叢集都需要一個 Amazon EKS 叢集 IAM 角色。受 Amazon EKS 管理的 Kubernetes 叢集使用此角色來管理節點，而舊式雲端供應商則使用此角色為服務建立具有 Elastic Load Balancing 的負載平衡器。這些叢集動作只能影響同一帳戶，因此建議您將每個叢集角色限制在該叢集和帳戶。這是 AWS 建議遵循您帳戶中最低權限原則的特定應用程式。

資源 ARN 格式

aws:SourceArn 的值必須是 EKS 叢集的 ARN，格式為 arn:aws:eks:*region*:*account*:cluster/*cluster-name* 。例如， arn:aws: eks:us-west-2:123456789012:cluster/my-cluster。

EKS 叢集角色的信任政策格式

下列範例示範如何使用 Amazon EKS 中的 aws:SourceArn 和 aws:SourceAccount 全域條件內容索引鍵，來預防混淆代理人問題。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:eks:us-west-2:123456789012:cluster/my-cluster"
          },
        "StringEquals": {
            "aws:SourceAccount": "123456789012"
        }
      }
    }
  ]
}
```

Kubernetes 的安全性考量事項

下面介紹了雲端安全性考量事項，因為安全性會對 Amazon EKS 叢集中的 Kubernetes 產生影響。若要深入探究 Kubernetes 中的安全控制與實務，請參閱 Kubernetes 文件中的雲端原生安全性與 Kubernetes。

主題

- 透過 Kubernetes 憑證保護工作負載
- 輪換 EKS 叢集憑證授權單位 (CA)
- 理解 Amazon EKS 所建立的 RBAC 角色及使用者
- 在現有叢集上使用 KMS 加密 Kubernetes 秘密
- 搭配 Amazon EKS Pod 使用 AWS Secrets Manager 秘密
- 針對所有 Kubernetes API 資料的預設封套加密
- 在 Amazon EKS 中強化 Kubernetes RBAC

透過 Kubernetes 憑證保護工作負載

Kubernetes 憑證 API 可實現 X.509 憑證佈建的自動化程序。該 API 具有一個命令行介面，供 Kubernetes API 用戶端從憑證認證機構 (CA) 請求和獲取 X.509 憑證。您可以使用 `CertificateSigningRequest` (CSR) 資源來請求受指示的簽署者簽署憑證。您的請求在被簽署之前會先被核准或拒絕。Kubernetes 支援內建簽署者和具有明確定義行為的自訂簽署者。透過這種方式，客戶可以預測其 CSR 會發生什麼情況。如需憑證簽署的相關資訊，請參閱 Certificate Signing Requests (憑證簽署請求)。

其中一個內建簽署者是 `kubernetes.io/legacy-unknown`。CSR 資源的 `v1beta1` API 接受此傳統未知的簽署者。然而，穩定的 CSR `v1` API 不允許將 `signerName` 設定為 `kubernetes.io/legacy-unknown`。

如果您想使用 Amazon EKS CA 在叢集上產生憑證，則必須使用自訂簽署者。若要使用 CSR `v1` API 版本並產生新憑證，您必須遷移任何現有清單檔案和 API 用戶端。使用現有 `v1beta1` API 建立的現有憑證有效且可正常運作，直到憑證過期為止。這包含下列項目：

- 信任分佈：無。在 Kubernetes 叢集中，此簽署者沒有標準的信任或分佈。
- 允許的主體：任何
- 允許的 x509 延伸項目：接受 subjectAltName 和金鑰使用方式延伸項目，並放棄其他延伸項目
- 允許的金鑰使用方式：不得包括 [「金鑰加密」、「數位簽章」、「伺服器身分驗證」] 以外的用法

> **Note**
>
> 不支援用戶端憑證簽署。

- 過期/憑證生命週期：45 天 （預設和上限）
- 允許/不允許 CA 位元：不允許

使用 signerName 產生 CSR 的範例

這些步驟示範如何使用 產生 DNS `myserver.default.svc` 名稱的服務憑證`signerName: beta.eks.amazonaws.com/app-serving`。將此作為您自己環境的指南。

1. 執行 `openssl genrsa -out myserver.key 2048` 命令以產生 RSA 私有金鑰。

```
openssl genrsa -out myserver.key 2048
```

2. 執行下列命令來產生憑證請求。

```
openssl req -new -key myserver.key -out myserver.csr -subj "/CN=myserver.default.svc"
```

3. 產生 CSR 請求的 base64 值，並將其存放於變數中以供稍後步驟中使用。

```
base_64=$(cat myserver.csr | base64 -w 0 | tr -d "
")
```

4. 執行下列命令以建立名為 mycsr.yaml 的檔案。在下列範例中，beta.eks.amazonaws.com/app-serving 是 signerName。

```
cat >mycsr.yaml <<EOF
apiVersion: certificates.k8s.io/v1
kind: CertificateSigningRequest
metadata:
  name: myserver
spec:
  request: $base_64
  signerName: beta.eks.amazonaws.com/app-serving
  usages:
    - digital signature
    - key encipherment
    - server auth
EOF
```

5. 提交 CSR。

```
kubectl apply -f mycsr.yaml
```

6. 核准服務憑證。

```
kubectl certificate approve myserver
```

7. 驗證憑證已核發。

```
kubectl get csr myserver
```

範例輸出如下。

```
NAME       AGE     SIGNERNAME                            REQUESTOR          CONDITION
```

```
myserver   3m20s   beta.eks.amazonaws.com/app-serving   kubernetes-admin
 Approved,Issued
```

8. 匯出已核發的憑證。

```
kubectl get csr myserver -o jsonpath='{.status.certificate}'| base64 -d >
 myserver.crt
```

輪換 EKS 叢集憑證授權單位 (CA)

在公有金鑰基礎設施 (PKI) 中，憑證授權機構 (CA) 是可發行和簽署數位憑證的受信任實體。這些憑證會建立身分，並使用 TLS (Transport Layer Security) 在系統之間啟用加密通訊。當用戶端連線至伺服器時，伺服器會顯示由 CA 簽署的憑證。用戶端會根據其信任CAs 來驗證伺服器的憑證，再允許連線繼續。

在 Amazon Elastic Kubernetes Service (Amazon EKS) 中，會在叢集建立時為每個 EKS 叢集建立 CA。這遵循與上游 Kubernetes 相同的模型：每個 EKS 叢集都有自己的 CA 來簽署 API 伺服器的憑證。這可讓控制平面元件、工作者節點和用戶端向 API 伺服器進行身分驗證，並建立與 EKS 叢集的加密連線。

這些憑證授權單位 CAs) 具有定義的有效期間。CA 輪換是在到期之前取代 EKS 叢集憑證授權單位的程序，確保您的叢集保持運作和可存取。我們有內建的保護措施，可自動管理此程序。如果您不自行啟動 CA 輪換，我們會自動附加後續 CA，並在傳出 CA 過期之前啟用它，確保您的叢集保持可用。

在 CA 輪換期間，後續 CA 會附加到您的 EKS 叢集。EKS 會自動將後續 CA 分發到所有 AWS 受管元件 （控制平面、EKS Auto Mode 執行個體和 AWS Fargate 節點）。您有責任更新您管理的工作者節點 （非 EKS Auto Mode、非 Fargate) 和外部用戶端 （例如 kubeconfig 檔案和 CI/CD 管道），以便在啟用後續 CA 之前信任後續 CA。啟動後續 CA 之後，EKS 叢集會轉換為使用後續 CA 簽署憑證。然後會淘汰傳出 CA。

每個 EKS 叢集都需要 CA 輪換，因為 CAs 的有效期間有限。有效期間取決於叢集的 CA 建立時間。如需詳細資訊，請參閱常見問答集一節。EKS 保護措施可確保您的叢集本身在整個輪換生命週期內保持可用，但成功的輪換也取決於您更新管理的工作者節點和外部用戶端，以便在啟用後續 CA 之後保持連線。

以下各節中的 APIs、主控台體驗、通知和step-by-step指引旨在支援您完成此程序。共同責任模型章節涵蓋完整的責任範圍。

CA 輪換的運作方式

Amazon EKS 中的 CA 輪換是多階段程序。無論您是否採取行動，我們都有自動保護措施來在整個輪換生命週期中保持 EKS 叢集可用性。成功輪換，其中所有元件都會維持連線能力，因此需要下列各節所述的步驟。

階段 1：附加後續 CA

後繼 CA 會附加至 EKS 叢集。從此時開始，EKS 叢集會同時信任傳出 CA 和後續 CA。憑證會繼續由傳出 CA 發行。不會發生中斷。

只要叢集處於作用中狀態，您可以隨時使用 AWS CLI、EKS API、主控台或基礎設施做為程式碼 (IaC)，例如 AWS CloudFormation，自行附加後續 CA。如果您不這樣做，我們會自動代表您附加一個。

在 EKS （階段 2) 中 AWS 完成對 AWS 受管元件的分發之前，無法啟用後續 CA。這是開始識別需要更新的工作者節點和外部用戶端的正確時機。識別連接到 EKS 叢集 API 伺服器的所有系統可能需要一些時間，特別是在具有多個團隊、CI/CD 管道和監控工具的環境中。儘早開始此程序可讓您在自己的時間軸上協調更新。

階段 2：分發後續 CA

在附加後續 CA 之後， AWS 更新 EKS 叢集中的受管元件 （控制平面、EKS Auto Mode 執行個體和 AWS Fargate 節點），以識別和信任這兩個 CAs。您可以透過 CA 的分佈狀態來追蹤進度。分佈完成之前，無法啟用後續 CA。

完成受管元件的 CA 分佈後，您需負責更新兩個群組：您管理的工作者節點 （非 EKS Auto Mode、非 Fargate)，以及信任後續 CA 的外部用戶端。這可確保他們會在後續 CA 啟用後繼續連線到 API 伺服器。如果在啟用後續 CA 之前未更新任何元件，CA 復原可在您完成其餘更新時還原連線。

階段 3：啟用後續 CA

在我們完成後續 CA 分佈至 EKS 中的所有 AWS 受管元件之後，即可啟用後續 CA。我們建議您在自己的時間軸上啟用後續 CA。給自己足夠的時間來探索和更新您管理的工作者節點和外部用戶端，以信任後續 CA。啟用後續 CA 之後，EKS 叢集會發行後續 CA 簽署的憑證。傳出 CA 仍受信任，但不再用於簽署。復原時段在啟用後有限期間內可用，可讓您視需要還原至傳出 CA。下一節會詳細說明轉返。

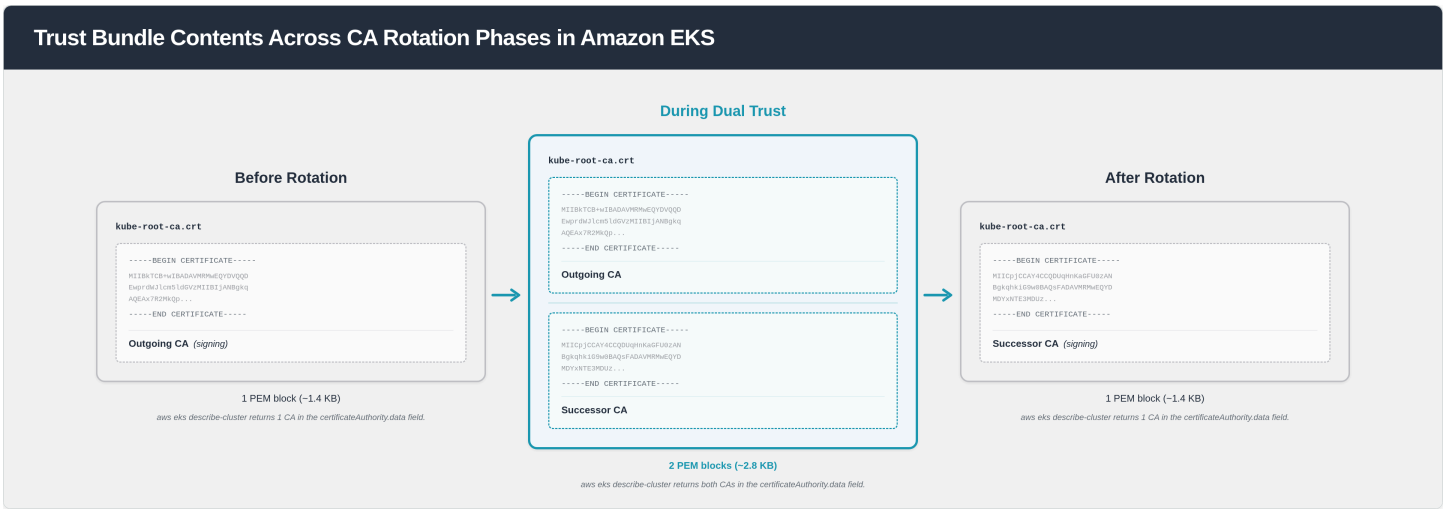
當您確信您管理的工作者節點和用戶端已更新時，您可以自行啟用後續 CA。如果沒有，我們會在到期截止日期接近時自動啟用後續 CA。

雙重信任期間

附加後續 CA （階段 1) 到淘汰傳出 CA 之間的時間稱為雙重信任期間。在此期間，EKS 叢集會同時信任這兩個 CAs。這就是讓輪換不中斷的原因：元件可以遞增更新，因為 EKS 叢集接受任一 CA 簽署的憑證。

雙重信任期間可讓您有時間識別和更新您管理的所有工作者節點和外部用戶端，而無需同時協調所有變更。

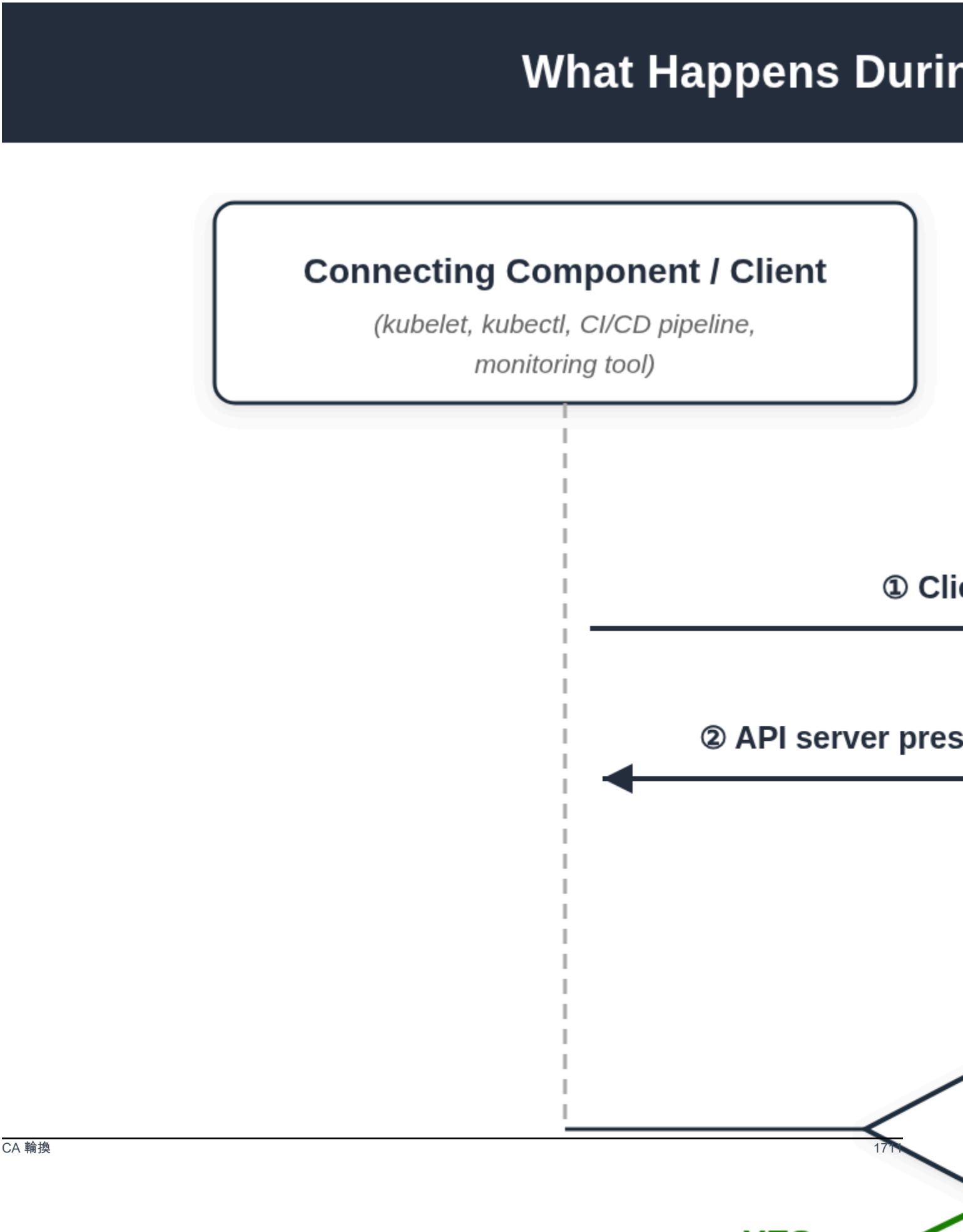
> **Note**
>
> 在雙重信任期間，叢集的信任套件包含兩個 CAs。這是 .pem 編碼信任套件的標準行為。在輪換開始之前，將執行嚴格單一 CA 驗證或 CA 鎖定的應用程式更新為接受多個 CAs。



請務必了解後續 CA 的啟用如何影響連線。當用戶端連線到 API 伺服器時，它會檢查伺服器的憑證是否由其信任的 CA 簽署，以驗證伺服器的身分。

啟用後續 CA 之後，API 伺服器會顯示其由後續 CA 簽署的憑證。已更新其信任套件以包含後續 CA 的用戶端將成功驗證並正常連線。尚未更新其信任套件的用戶端將無法辨識伺服器的憑證，也無法建立連線。

下圖顯示後續 CA 啟用後的 TLS 連線流程。


What Happens Durin
Connecting Component / Client
(kubelet, kubectl, CI/CD pipeline,
monitoring tool)
① Cli
② API server pres
YES

這就是為什麼在啟用後續 CA 之前更新工作者節點和外部用戶端很重要：他們需要其信任套件中的後續 CA 來驗證 API 伺服器的身分和連線。雙重信任期間和 CA 轉返可為您提供時間和安全網路來完成此操作。

Amazon EKS CA Rotation

Stage 1 — Append CA

WHAT HAPPENS

Successor CA added to the cluster. Cluster now tru
CAs. Certificates still issued by outgoing CA.

WHO ACTS

You or AWS (auto-append at T−2 years before CA

共同責任模型

Amazon EKS 中的 CA 輪換遵循廣泛套用的相同共同責任模型 AWS。 AWS 負責雲端基礎設施的安全性和可用性，而且您需負責其中工作負載的安全性和組態。如需如何將共同責任套用至 Amazon EKS 的詳細資訊，請參閱 EKS 安全最佳實務。

在 CA 輪換的情況下，這表示：

AWS 負責

- 更新 EKS 叢集控制平面以信任並從後續 CA 發行憑證
- 更新 EKS Auto Mode 節點以信任後續 CA
- 更新 AWS Fargate 節點以信任後續 CA
- 確保在 EKS 中發佈至 AWS 受管元件完成之前，無法啟用後續 CA
- 在整個輪換生命週期中保留 EKS 叢集可用性
- 在輪換程序的每個階段通知您
- 如果您在 CA 接近過期之前未採取行動，則自動啟動輪換

您負責

- 更新您的外部用戶端（開發人員工作站、CI/CD 管道、監控工具、自動化）以信任後續 CA
- 更新您的工作者節點（受管節點群組、Karpenter 控制節點、自我管理節點、混合節點）以信任後續 CA
- 當您確定您的元件已更新時，啟用後續 CA

我們無法代表您執行這些動作。外部用戶端存在於 AWS 操作界限之外。非由 EKS Auto Mode 或 Fargate 管理的工作者節點會在啟動時間或透過只有您控制的引導程序設定其 CA 信任組態。這與 TLS 信任的運作方式一致：用戶端擁有自己的信任存放區，只有用戶端的管理員可以更新它。

以下各節會在每一端展開：對您 AWS 有何幫助，以及您需要做什麼，以及如何執行的step-by-step指引。


Amazon EKS CA Rotation
AWS manages
EKS cluster control plane
EKS Auto Mode instances
AWS Fargate nodes
CA distribution to managed con
Blocking activation until distrib
Preserving cluster availability th

為您 AWS 做什麼

AWS 會在 EKS 叢集的 CA 輪換生命週期中管理下列項目：

自動建立 CA

如果您未在自己的時間軸上附加後續 CA，我們會在 EKS 叢集的傳出 CA 接近過期時自動附加 CA。這可確保輪換程序從足夠的時間開始，讓您可以在過期截止日期之前探索和更新用戶端。

控制平面更新

我們會自動更新 EKS 叢集控制平面，以信任後續 CA。啟用後續 CA 之後，控制平面會發行後續 CA 簽署的憑證。控制平面不需要您採取任何動作。

EKS Auto 模式和 Fargate 更新

我們會自動更新 EKS Auto Mode 節點和 Fargate Pod，以信任後續 CA。這些元件完全由我們管理，而且在 CA 輪換期間不需要您採取任何動作。

EKS 功能更新

我們會自動更新 EKS 功能 (Kubernetes (ACK)、Argo CD 和 kro (Kube Resource Orchestrator) 的 AWS 控制器），以信任後續 CA。這些受管資源會與您叢集的 API 伺服器通訊，並在 CA 分佈程序中更新。對於 EKS 功能中的受管資源，您不需要採取任何動作。如需詳細資訊，請參閱 EKS 功能。

分佈狀態追蹤

在我們更新 EKS 叢集中的受管元件時，您可以監控 CA 分佈狀態的進度。這會告訴您我們是否已完成輪換。分佈完成之前，無法啟用後續 CA。

內建防護措施

我們有內建的保護措施，可在輪換期間保護您的 EKS 叢集：

- 在 EKS 中所有 AWS 受管元件的分佈完成之前，無法啟用後續 CA
- 當 AWS附加的後續 CA 是叢集上唯一的後續 CA 時，就無法刪除。此保護措施可確保叢集一律具有有效的 CA 路徑，以防止過期。啟用後續 CA 之後，即可刪除傳出 CA。
- 在 CA 到期前達到兩年標記後，無法刪除客戶套用的後續 CA。在此時間點之後，CA 會受到保護，不會遭到刪除，以確保您的叢集一律具有後續 CA，做為過期方法。
- 如果過期截止日期接近且您尚未自行啟用，我們會自動啟用後續 CA

這些保護措施可確保 EKS 叢集在整個輪換生命週期中保持可用性，無論您是否採取行動。

通知

我們在 CA 輪換生命週期的每個階段通知您。通知會透過 AWS Health、Cluster Insights 和電子郵件傳送。每個通知都會告訴您發生了什麼、您需要採取什麼動作（如果有的話），以及 EKS 叢集在輪換時間軸中的位置。

通知	當	代表什麼意思
CA 到期提醒	CA 過期前 2.5 年	EKS 叢集的 CA 具有定義的過期日期。規劃輪換。
已附加後續 CA	當您或 AWS 附加後續 CA 時（在過期前 2 年自動附加）	輪換程序已開始。 AWS 正在將後續 CA 分發給受管元件。
分佈完成	附加後不久（依叢集而異）	AWS 已完成其端。您現在可以更新您管理的工作者節點和外部用戶端。
啟用警告	自動啟用前 60 天	我們很快就會啟用後續 CA。如果您尚未更新元件，請更新元件。
啟用後續 CA	當您或 AWS 啟用時（過期前 6 個月自動啟用）	EKS 叢集現在正在從後續 CA 發行憑證。
最終自動啟用（如果復原）	過期前 45 天	AWS 會啟用後續 CA。沒有可用的 CA 轉返。

> **Note**
>
> 在 2018-2019 中建立的叢集具有不同的通知時間軸。這些叢集將依調整後的排程收到自動通知。

您也可以使用 Amazon EventBridge 設定自己的通知，將 CA 輪換事件整合到現有的監控和提醒工作流程中。

您需要做什麼（以及原因）

成功的 CA 輪換需要您更新 AWS 無法代表您到達的元件。這些分為兩個類別：

外部用戶端

從叢集外部連線至 EKS 叢集 API 伺服器的任何系統。這包括開發人員工作站、CI/CD 管道 (Jenkins、GitHub Actions、GitLab、ArgoCD)、監控和可觀測性工具、自動化指令碼，以及使用 kubeconfig 與 API 伺服器通訊的任何應用程式。

這些系統各自維護自己的信任組態。啟用後續 CA 時，API 伺服器會顯示後續 CA 簽署的憑證。在啟用後續 CA 之前，更新這些用戶端以信任後續 CA，確保它們保持連線。如果錯過用戶端，CA 轉返可以在您完成更新時還原存取權。

工作者節點（非 EKS Auto Mode、非 Fargate)

非由 EKS Auto Mode 或 Fargate 管理的工作者節點會在啟動時間或透過 kubelet 引導程序設定其 CA 信任組態。這些節點需要重新整理，以便他們信任後續 CA。所需的動作取決於工作者節點的類型：

受管節點群組

執行節點群組版本更新，這會觸發節點的滾動取代。具有更新 CA 信任組態的新節點自動引導。

Karpenter 控制的節點

如果啟用偏離偵測，Karpenter 將在設定的偏離時段內循環節點，而新節點將在沒有手動動作的情況下取得後續 CA。如果偏離偵測已停用或設定為長視窗，請將這些節點視為與自我管理節點相同的節點。

自我管理的節點

取代節點，使其以更新的 CA 信任組態引導。這通常涉及使用更新的 CA 資料更新啟動範本，並透過 Auto Scaling 群組觸發滾動替換。

混合節點

更新每個混合節點上的信任組態，以包含後續 CA。特定程序取決於如何啟動混合節點，以及如何管理其信任組態。

您可以使用 Cluster Insights 來識別 EKS 叢集中執行的工作者節點類型。更新每種類型的step-by-step 指引會在下一節中提供。

如果遺漏任何工作者節點，我們會提供 CA 轉返做為安全網路。不過，在啟用後續 CA 之前更新所有工作者節點，可避免完全中斷連線。在啟用後續 CA 之前未更新的節點將失去與 EKS 叢集 API 伺服器的連線，直到替換或執行 CA 轉返為止。

為什麼只有您才能這樣做

外部用戶端存在於 AWS 操作界限之外。在您的公司網路中執行的 CI/CD 管道，開發人員的筆記型電腦，即內部部署託管的監控工具： AWS 沒有機制可以連線到這些系統並更新其信任組態。

非 EKS Auto Mode 工作者節點可透過您擁有的啟動範本、使用者資料指令碼或引導程序來控制其信任組態。更新它們需要取代節點或修改其組態，這兩者都是基礎設施中的動作。

這是 Kubernetes 目前使用的 TLS 信任模型的限制。API 伺服器沒有通訊協定層級機制可查詢用戶端是否已更新其信任套件。伺服器只能在用戶端連線時呈現其憑證。如果用戶端信任簽署它的 CA，連線會成功。如果沒有，則會失敗。沒有允許 代表您 AWS 確認元件整備的啟用前驗證路徑。

何時開始

儘早開始識別您的外部用戶端。這是 CA 輪換中最耗時的部分，特別是在具有多個團隊，可獨立將工作負載部署到 EKS 叢集的環境中。您越早開始用戶端探索，就越能在團隊間無壓力地協調更新。

以下各節提供如何更新每種類型的用戶端和工作者節點的詳細指導。

先決條件

開始 CA 輪換之前，請確認下列事項：

- AWS CLI：2.x 版或更新版本。CA 輪換 APIs可在最新的 CLI AWS 中使用。執行 `aws --version`以檢查。
- 主控台存取：適用於支援區域的 Amazon EKS 主控台提供 CA 輪換。
- 區域可用性：CA 輪換可在支援 Amazon EKS 的所有 AWS 商業區域中使用。

除了管理 EKS 叢集所需的許可之外，不需要額外的 IAM 許可。如果您現在可以為 EKS 叢集呼叫 EKS APIs，則可以執行 CA 輪換。

開始使用

您可以使用 AWS CLI 或 Amazon EKS 主控台執行 CA 輪換。確定您正在執行 AWS CLI 2.x 版或更新版本 (`aws --version` 檢查）。

使用 AWS CLI

下列演練涵蓋使用 CLI 和 EKS API AWS 的end-to-end CA 輪換程序。 APIs

步驟 1：檢查您的作用中 CA

檢視 EKS 叢集上的作用中憑證授權單位。

```
aws eks list-certificate-authorities --cluster-name my-cluster --region us-west-2
```

預期的輸出結果：

```
{
    "certificateAuthorities": [
        {
            "id": "a1b2c3d4-5678-90ab-cdef-example11111",
            "createdAt": "2024-01-15T10:30:00-07:00",
            "createdBy": "EKS",
            "activatedAt": "2024-01-15T10:30:00-07:00",
            "activatedBy": "EKS",
            "signingStatus": "IN_USE",
            "distributionStatus": "COMPLETE"
        }
    ]
}
```

這會顯示建立 EKS 叢集時所建立的 CA。它目前正在使用（簽署憑證）且分佈已完成 (EKS 中的所有 AWS 受管元件都信任它）。

步驟 2：檢視 CA 詳細資訊和過期

```
aws eks describe-certificate-authority --cluster-name my-cluster --certificate-
authority-id a1b2c3d4-5678-90ab-cdef-example11111 --region us-west-2
```

預期的輸出結果：

```
{
    "certificateAuthority": {
        "id": "a1b2c3d4-5678-90ab-cdef-example11111",
        "createdAt": "2024-01-15T10:30:00-07:00",
        "createdBy": "EKS",
        "activatedAt": "2024-01-15T10:30:00-07:00",
        "activatedBy": "EKS",
        "signingStatus": "IN_USE",
        "distributionStatus": "COMPLETE",
```

```
        "validity": {
            "notBefore": "2024-01-15T10:30:00-07:00",
            "notAfter": "2029-01-14T10:30:00-07:00"
        },
        "rollbackAvailable": false
    }
}
```

validity 區塊會顯示 CA 建立的時間 (notBefore) 和過期的時間 (notAfter)。rollbackAvailable指出您是否可以在啟用後續 CA 之後還原至先前的 CA。對於使用 EKS 叢集建立的初始 CA，false這是因為沒有先前的 CA 可還原。

> **Note**
>
> scheduledEvents 區塊 （包含 firstAutoActivation和 finalAutoActivation) 會出現在後續 CA 上，而不是傳出 CA。如果您尚未自行啟用，這些欄位會顯示我們自動啟用後續 CA 的時間。當您在附加後描述後續 CA 時，您會看到這些欄位。

步驟 3：附加後續 CA

```
aws eks create-certificate-authority --cluster-name my-cluster --region us-west-2
```

這會將後續 CA 附加到您的 EKS 叢集。回應包含updateId可用來追蹤進度的 。

步驟 4：追蹤更新

```
aws eks describe-update --name my-cluster --update-id a1b2c3d4-update-id --region us-west-2
```

等待更新狀態為 Successful。

> **Note**
>
> 如果更新狀態顯示 ，UPDATE_FAILED且後續 CA distributionStatus顯示 FAILED，則 CA 建立未成功。使用 刪除失敗的 CA，aws eks delete-certificate-authority並建立新的 CA。對於 AWS啟動的自動輪換， 會在附加新的後續程式之前 AWS 自動偵測和清除失敗CAs，因此在該案例中不需要客戶動作。

步驟 5：驗證分佈狀態

```
aws eks list-certificate-authorities --cluster-name my-cluster --region us-west-2
```

您現在應該會看到兩個 CAs。更新 EKS 叢集中的所有 AWS 受管元件COMPLETE後，後續 CA 將擁有 signingStatus: NOT_USED，distributionStatus並將從 進展IN_PROGRESS至 。

在後續 CA 的分佈狀態為 之前，請勿繼續COMPLETE。

步驟 6：更新您的 kubeconfig

```
aws eks update-kubeconfig --name my-cluster --region us-west-2
```

這會將您的本機 kubeconfig 更新為信任這兩個 CAs。之後，您的 kubectl 命令會在後續 CA 啟用後繼續運作。

步驟 7：更新您管理的工作者節點和外部用戶端

這會在下一節中詳細說明。在您管理的所有工作者節點和外部用戶端更新為信任後續 CA 之後，請繼續啟用。

步驟 8：啟用後續 CA

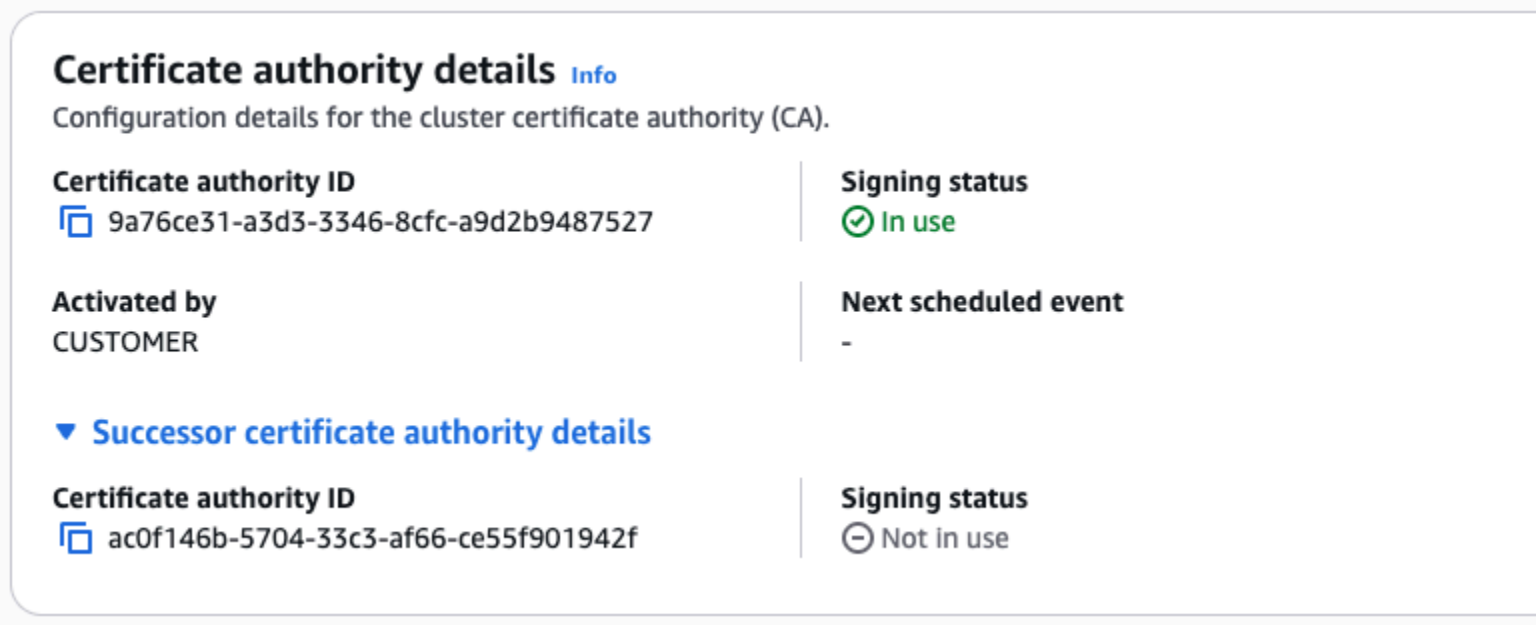
```
aws eks activate-certificate-authority --cluster-name my-cluster --certificate-
authority-id <successor-ca-id> --region us-west-2
```

啟用後續 CA 後，EKS 叢集會發行後續 CA 簽署的憑證。驗證連線，以確認所有元件都如預期般運作。

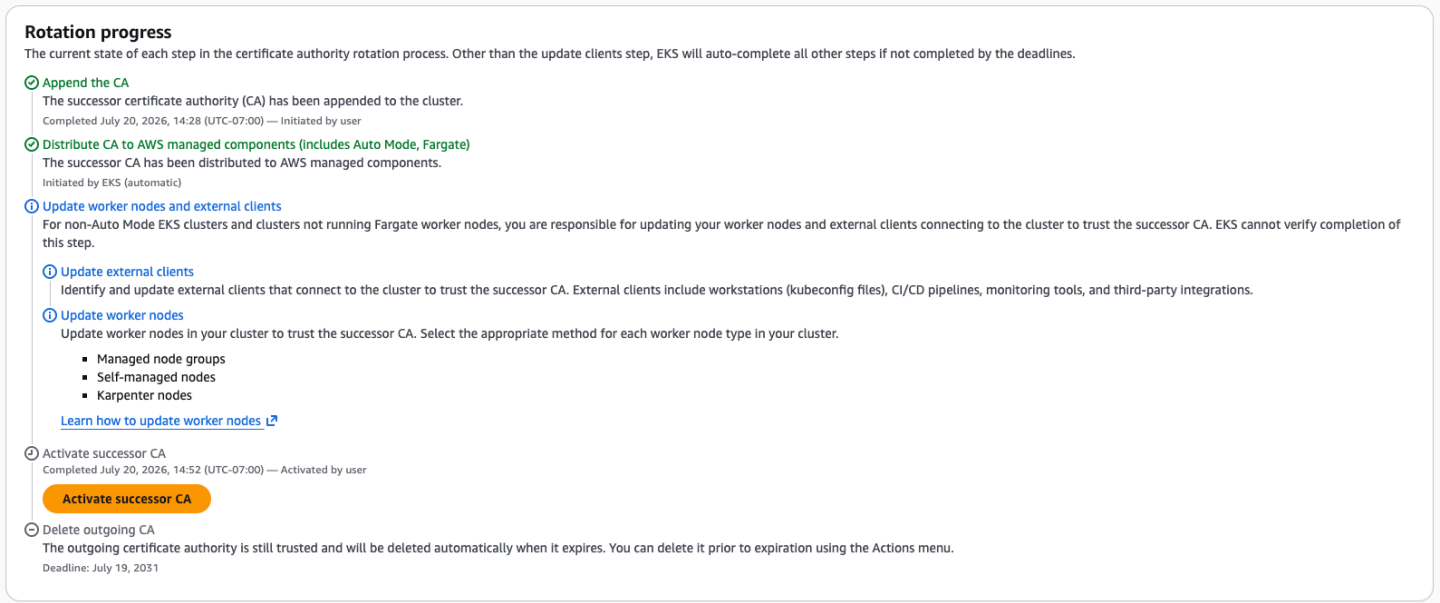
使用 Amazon EKS 主控台

Amazon EKS 主控台提供 CA 輪換的引導式體驗。您可以直接從主控台檢視 CA 狀態、附加後續 CA、監控分佈進度，以及啟用後續 CA。

下圖顯示在作用中輪換期間 Amazon EKS 主控台中的憑證授權機構詳細資訊檢視。作用中 CA 和後續 CA 都會顯示其簽署狀態、過期日期和過期天數。



下圖顯示 Amazon EKS 主控台中的輪換進度檢視。輪換程序的每個步驟都會以其目前狀態顯示，包括附加、分佈、更新工作者節點和外部用戶端、啟用和刪除傳出 CA。



更新您的 Kubernetes 用戶端

> **Important**
>
> 在雙重信任期間，叢集的信任套件包含兩個 CA 憑證。兩個 base64 編碼 CAs 的合併大小約為 2.8 KB （或使用 gzip 壓縮約 1.9 KB)。對於在 EC2 啟動範本中提供自訂使用者資料的工作者

節點，請確認您的總使用者資料大小不超過 EC2 使用者資料限制 16KB。如果您現有的使用者資料接近此限制，請考慮使用 gzip 壓縮使用者資料內容，以減少大小。

在附加後續 CA 且 AWS 完成對 EKS 叢集 (`distributionStatus: COMPLETE`) 中受管元件的分發之後，您需要更新自己的元件以信任後續 CA。在此內容中，「用戶端」是連線到 EKS 叢集 API 伺服器的任何系統。這包括本機 kubectl 組態、CI/CD 管道、監控工具、自動化指令碼和工作者節點。

EKS 叢集（現在包含目前和後續 CA) 的更新 CAs 資料可以使用下列方式擷取：

```
aws eks describe-cluster --name my-cluster --region us-west-2 --query
 'cluster.certificateAuthority.data' --output text
```

使用此值來更新下列小節中每個用戶端類型的信任組態。

Kubeconfig（開發人員工作站、CI/CD 管道、自動化）

執行下列動作以更新本機 kubeconfig：

```
aws eks update-kubeconfig --name my-cluster --region us-west-2
```

這會自動擷取最新的 CA 資料並更新您的 kubeconfig。使用此 kubeconfig 的任何系統都會信任這兩個 CAs。

對於產生自己的 kubeconfig 的 CI/CD 管道和自動化（例如，直接使用 EKS APIs 或將 kubeconfig 儲存為秘密），請使用從 擷取的值更新 `certificate-authority-data` 欄位`describe-cluster`。

受管節點群組

執行節點群組版本更新，以觸發節點的滾動取代：

```
aws eks update-nodegroup-version --cluster-name my-cluster --nodegroup-name my-
nodegroup --region us-west-2
```

具有更新 CA 資料的新節點自動引導。滾動替換可確保節點一次替換一個節點，而不會中斷執行中的工作負載。

如果您透過 Terraform 或 CloudFormation 管理節點群組，CA 輪換不會在您的 IaC 狀態建立偏離。CA 生命週期是透過與您的叢集資源組態分開的專用 EKS APIs 進行管理。如需 CA 輪換如何以程式碼形式與基礎設施互動的詳細資訊，請參閱基礎設施即程式碼一節。

具有自訂 AMI 的自訂啟動範本

如果您的節點群組是使用自訂 AMI 部署， AWS 不會合併使用者資料。您負責提供正確的引導組態，包括更新的 CA 信任套件。CA 輪換不會為您更新使用者資料，沒有後續 CA 的節點無法加入叢集。

1. 擷取更新的 CA 資料 （包含傳出和後續 CAs的合併信任套件）：

```
aws eks describe-cluster --name my-cluster --region us-west-2 --query
 'cluster.certificateAuthority.data' --output text
```

2. 更新啟動範本使用者資料中的 CA 資料。您指定 CA 資料的方式取決於您的作業系統和引導機制，並符合您最初提供的方式。如需自訂受管節點的詳細資訊，請參閱使用啟動範本自訂受管節點。
3. 使用更新的使用者資料建立新的啟動範本版本，然後將節點群組更新為該啟動範本版本。這會回收節點，以便它們使用後續 CA 引導：

```
aws eks update-nodegroup-version --cluster-name my-cluster --nodegroup-name my-
nodegroup --launch-template id=<lt-id>,version=<new-version> --region us-west-2
```

   如需將節點群組更新為新啟動範本版本的詳細資訊，請參閱更新叢集的受管節點群組。

驗證節點是否在更新的啟動範本上執行

在繼續啟用之前，請確認受管節點群組中的所有節點都在最新的啟動範本版本上執行。此版本必須包含更新的 CA 信任套件。

1. 取得節點群組的啟動範本。如果 `describe-nodegroup`傳回`launchTemplate`欄位，請直接使用它：

```
aws eks describe-nodegroup --cluster-name my-cluster --nodegroup-name my-nodegroup --
region us-west-2 --query 'nodegroup.launchTemplate'
```

   如果它未傳回`launchTemplate`欄位， 會在內部 AWS 管理啟動範本。改為透過 Auto Scaling 群組尋找：

```
ASG=$(aws eks describe-nodegroup --cluster-name my-cluster --nodegroup-name my-
nodegroup --region us-west-2 --query 'nodegroup.resources.autoScalingGroups[0].name'
 --output text)
aws autoscaling describe-auto-scaling-groups --auto-scaling-group-
names "$ASG" --region us-west-2 --query 'AutoScalingGroups[0].
```

```
{LaunchTemplate: LaunchTemplate, MixedInstancesPolicy:
 MixedInstancesPolicy.LaunchTemplate.LaunchTemplateSpecification}'
```

> **Note**
>
> 啟動範本可能低於`LaunchTemplate`或`MixedInstancesPolicy`取決於 Auto Scaling 群組組態。

2. 解碼啟動範本使用者資料。使用上一個步驟的啟動範本 ID 和版本。確認使用者資料中的 CA 資料符合 傳回的合併信任套件`describe-cluster`。保留 CA 資料的欄位取決於您的作業系統和引導機制：

```
aws ec2 describe-launch-template-versions --launch-template-
id <lt-id> --versions <version> --region us-west-2 --query
 'LaunchTemplateVersions[0].LaunchTemplateData.UserData' --output text | base64 --
decode
```

3. 確認升級後所有工作者節點都在最新的啟動範本版本上執行。描述節點群組的 Auto Scaling 群組。比較每個執行個體的啟動範本版本與群組目前的啟動範本版本。每個`InService`執行個體都必須在目前的版本上。滾動取代會耗盡處於 `Terminating` 狀態的執行個體。您可以忽略這些執行個體：

```
ASG=$(aws eks describe-nodegroup --cluster-name my-cluster --nodegroup-name my-
nodegroup --region us-west-2 --query 'nodegroup.resources.autoScalingGroups[0].name'
 --output text)
aws autoscaling describe-auto-scaling-groups --auto-scaling-group-names "$ASG" --
region us-west-2 --query 'AutoScalingGroups[0].Instances[].{InstanceId: InstanceId,
 LifecycleState: LifecycleState, LaunchTemplateVersion: LaunchTemplate.Version}' --
output table
```

4. 確認取代節點運作狀態良好。節點群組中的每個節點都必須是 `Ready`，確認 kubelet 已使用更新的 CA 信任套件建立與 API 伺服器的連線：

```
kubectl get nodes -l eks.amazonaws.com/nodegroup=my-nodegroup
```

Karpenter 控制的節點

如果在 Karpenter NodePool 中啟用偏離偵測，Karpenter 將自動偵測節點是否使用過時的 CA 資料執行，並在設定的中斷時段內循環它們。新節點無需手動動作即可取得後續 CA。

確認您的 NodePool 組態中已啟用偏離偵測：

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: default
spec:
  disruption:
    consolidationPolicy: WhenEmptyOrUnderutilized
    budgets:
      - nodes: "10%"
  template:
    spec:
      nodeClassRef:
        group: karpenter.k8s.aws
        kind: EC2NodeClass
        name: default
```

如果disruption使用合併政策設定 ，則偏離偵測預設為作用中。Karpenter 將取代偏離其所需狀態的節點，其中包括 CA 資料變更。

如果啟用偏離偵測，請確認您的中斷預算允許在後續 CA 啟用日期之前取代所有 Karpenter 控制的節點。如果預算太嚴格 （例如，具有低取代百分比的狹窄維護時段），則並非所有節點都會及時取代。

如果偏離偵測已停用，或您的中斷預算限制取代超過輪換時間軸的時段，您可以透過封鎖和耗盡節點來手動觸發節點取代：

```
kubectl cordon <node-name>
kubectl drain <node-name> --ignore-daemonsets --delete-emptydir-data
```

Karpenter 將佈建替代節點，以使用更新後的 CA 資料引導。

自我管理的節點

對於自我管理節點，您需要更新啟動範本中的 CA 資料或節點在引導期間使用的使用者資料指令碼：

1. 擷取更新的 CA 資料：

```
aws eks describe-cluster --name my-cluster --region us-west-2 --query
 'cluster.certificateAuthority.data' --output text
```

2. 使用更新的 CA 資料值更新您的啟動範本 （或使用者資料）。
3. 透過 Auto Scaling 群組觸發節點的滾動取代 （例如，執行個體重新整理）。

新節點將使用更新的 CA 資料引導，並信任目前和後續 CAs。

AWS Fargate Pod (EKS Fargate 啟動類型）

AWS EKS 叢集中的 Fargate Pod 與其他 EKS 資料平面啟動模式（受管節點群組、自我管理節點、Karpenter 控制節點）的運作方式不同。

當 Pod 連線到 API 伺服器時，會發生兩件事：Pod 會使用其服務帳戶字符（向 API 伺服器提供其身分）進行身分驗證，而 Pod 透過檢查伺服器憑證是否由其信任的 CA 簽署來驗證 API 伺服器身分。存放在 Pod 環境中的 CA 資料是實現此驗證的原因。如果 API 伺服器開始呈現由 Pod 不信任的後續 CA 簽署的憑證，Pod 會拒絕連線。

在其他 EKS 資料平面啟動模式中（受管節點群組、自我管理節點、Karpenter 控制節點），CA 資料會存在於節點上。當您取代節點時，新的節點引導會使用更新的 CA 資料。排程在該新節點上的 Pod 會收到更新的 CA 資料，允許他們驗證 API 伺服器，無論哪個 CA 簽署其憑證。

在 Fargate 中，每個 Pod 都會使用自己的 kubelet 程序，在自己的專用運算環境中執行。在建立 Pod 時，此 kubelet 引導具有 CA 資料。下方沒有共用節點，而且您無法直接存取基礎運算。

在 CA 輪換期間，EKS 中的 AWS Fargate 節點不需要客戶動作。 AWS 自然會透過其修補程序回收 Fargate 節點中的 Pod。在附加後續 CA 之後，作為此程序的一部分，會回收預先存在的 Fargate Pod。他們無需您採取任何動作即可信任後續 CA。由於 會在 AWS EKS 中的任何起始 CA 啟用時間軸之前 AWS 附加後續 CA，因此 Fargate Pod 將回收，並在 AWS 啟用後續 CA 的時間之前信任後續 CA。有內建的保護措施，可防止啟用後續 CA，直到 Fargate Pod 完成回收。

這表示 EKS 叢集的受管資料平面選項 (EKS Auto Mode 和 Fargate) 具有相同的 CA 輪換使用者體驗：工作者節點不需要客戶動作。您仍需負責更新連線至 API 伺服器的任何外部用戶端。

這是邊緣案例。鑑於輪換時間表（後續 CA 在過期前幾年附加），在絕大多數情況下，自然修補週期會在啟動後續 CA 之前順利完成。在客戶嘗試提早啟用的罕見情況下，防護措施做為預防措施存在。

外部用戶端（監控工具、第三方整合）

使用 kubeconfig 或憑證信任組態連線到 EKS 叢集 API 伺服器的任何應用程式或工具都需要更新為更新的 CA 資料。其中包含：

- 監控和可觀測性工具 (Datadog、Prometheus、Grafana 代理程式）
- 在叢集外部執行的 GitOps 控制器 (ArgoCD、Flux)
- 呼叫 Kubernetes API 的自訂自動化或指令碼
- 儲存`certificate-authority-data`為靜態值的任何系統

對於這些項目，請將儲存的 CA 資料取代為來自 的更新值`describe-cluster`。

如何確認用戶端已更新

更新用戶端後，請確認用戶端仍可與 API 伺服器通訊：

```
kubectl get nodes
```

如果命令成功，您的 kubeconfig 會信任作用中的 CA 資料。啟用後續 CA 之後，請執行相同的命令以確認持續連線。

基礎設施即程式碼

您可以與現有基礎設施一起執行 CA 輪換做為程式碼 (IaC)，而無需建立偏離或需要變更 IaC 組態。

為什麼 CA 輪換不會影響您的 IaC 狀態

CA 生命週期是透過與 EKS `create-certificate-authority`叢集資源組態完全分開的專用 EKS APIs (`activate-certificate-authority`、、`delete-certificate-authority`) 進行管理。無論您是由您啟動還是由 自動啟動 CA 輪換 AWS，都不會修改 EKS 叢集資源上 IaC 工具追蹤的任何屬性。

這表示：

- 在附加或啟用 CA 之後套用或更新 IaC 堆疊，不會偵測到偏離或嘗試協調 CA 狀態
- 透過 CLI 或主控台啟動的 CA 輪換操作不會與 IaC 受管叢集資源衝突

傳回`certificateAuthority.data`的欄位`describe-cluster`是唯讀輸出。它反映了目前的合併信任套件（兩個 CAs雙重信任期間），但不是可設定的屬性。IaC 工具不會將其追蹤為要協調的項目。

每個 CA 記錄上的屬性欄位 (`createdBy`、`activatedBy`) 可讓您區分啟動的操作和自動 AWS 啟動的操作，以支援稽核和變更管理工作流程。

搭配 CA 輪換使用 CloudFormation

您可以在 `AWS::EKS::Cluster` 資源上使用 WriteOnly 屬性，透過 CloudFormation 觸發 CA 輪換。此屬性會觸發啟用，但不會存放在堆疊的狀態，因此後續堆疊更新時不會嘗試還原或停用。

```
# Phase 1: Add to existing stack that manages your cluster
Resources:
```

```
    NewCA:
        Type: AWS::EKS::CertificateAuthority
        Properties:
            ClusterName: my-cluster

    MyCluster:
        Type: AWS::EKS::Cluster
        Properties:
            Name: my-cluster
```

第一個堆疊更新會將後續 CA 附加到您的叢集。等待後續 CA 的分佈狀態到達，COMPLETE然後再繼續。

```
# Phase 2: After distribution completes, update your existing Cluster resource to
 activate
Resources:
    NewCA:
        Type: AWS::EKS::CertificateAuthority
        Properties:
            ClusterName: my-cluster

    # Update your existing Cluster resource to activate
    MyCluster:
        Type: AWS::EKS::Cluster
        Properties:
            Name: my-cluster
            ActiveCertificateAuthorityId: !GetAtt NewCA.Id
```

第二個堆疊更新會觸發後續 CA 的啟用。由於 ActiveCertificateAuthorityId 是 WriteOnly 屬性，因此不會在讀取時傳回，而且如果作用中 CA 在 CloudFormation 外部變更（例如，透過自動啟用方式），CloudFormation 將不會偵測偏離 AWS。

重要：CA 輪換 CloudFormation 整合遵循與典型 CloudFormation 資源不同的模式。EKS 中的憑證授權機構不是具有自己的 ARN 的獨立資源。它在叢集的憑證生命週期中存在，並透過叢集本身授權，類似於如何透過其父角色 (AWS::IAM::RolePolicy) 授權 IAM 角色政策，或透過其執行個體 () 授權 EIP 關聯AWS::EC2::EIPAssociation。透過 CloudFormation 管理 CA 輪換的客戶應注意此差異。

受管節點群組和 IaC

使用更新的 CA 信任組態執行節點群組版本更新以重新整理節點是一種操作動作。新節點會使用來自叢集的作用中 CA 信任資料自動引導。如果您的 IaC 範本未在啟動範本或使用者資料中硬式編碼 CA 資料，則不需要變更範本。

CA 轉返

啟用後續 CA 之後，如果您發現您管理的工作者節點（非 EKS Auto Mode、非 Fargate) 或外部用戶端發生連線問題，則可以轉返到先前的 CA。轉返會將先前的 CA 重新啟用為 EKS 叢集的簽署授權。

當 CA 轉返可用時

CA 復原可在 CA 啟用後使用，只要：

- CA 啟用是由客戶起始或第一次自動啟用 AWS（大約在傳出 CA 過期前 6 個月）
- 轉返視窗尚未過期

您可以使用 傳回的 `rollbackAvailable` 欄位，隨時檢查 CA 轉返是否可用`describe-certificate-authority`：

```
aws eks describe-certificate-authority --cluster-name my-cluster --certificate-
authority-id <ca-id> --region us-west-2
```

```
{
    "certificateAuthority": {
        "id": "a1b2c3d4-5678-90ab-cdef-example22222",
        "signingStatus": "IN_USE",
        "distributionStatus": "COMPLETE",
        "rollbackAvailable": true
    }
}
```

當 CA 轉返不可用時

最終自動啟用後，無法使用 CA 轉返。如果 最後一次 AWS 啟用後續 CA (CA 過期前 45 天），輪換必須繼續進行。只有在第一次自動啟用之前已復原時，才會發生最終自動啟用。它作為最後的防護措施存在，以確保 EKS 叢集不會在沒有有效 CA 的情況下達到 CA 過期。

如何轉返

若要轉返，請重新啟用先前的 CA：

```
aws eks activate-certificate-authority --cluster-name my-cluster --certificate-
authority-id <previous-ca-id> --region us-west-2
```

CA 復原期間會發生什麼情況

- 先前的 CA 會繼續簽署 EKS 叢集的簽署憑證
- 接續 CA 會保留在信任套件中（這兩個 CAs 仍然受信任）
- 已更新以信任後續 CA 的工作者節點和用戶端將繼續運作（他們信任這兩個 CAs)
- 尚未更新的工作者節點和用戶端將恢復正常操作 (API 伺服器正在呈現由他們已經信任的 CA 簽署的憑證）
- 工作者節點上的 Kubelet 程序會透過其內建的重試迴圈自動重新連線

何時考慮 CA 轉返

CA 轉返是一種安全機制，適用於後續 CA 啟用顯示您未事先發現的連線問題的情況：

- 在更新階段期間未識別的外部用戶端在後續 CA 啟用後失去連線
- 監控或可觀測性工具無法驗證新憑證
- CI/CD 管道因為使用硬式編碼憑證信任組態而中斷

轉返後，您可以保留完整的雙信任期，以在重新啟用後續 CA 之前識別和修正問題。

不含 CA 轉返的復原

如果您在更新受管節點群組之前啟用後續 CA，且 CA 復原時段不再可用（例如，在最終自動啟用截止日期之後），您可以透過對受影響的節點群組執行滾動更新來復原。不過，初始滾動更新嘗試將會失敗，因為中斷連線的節點無法從 API 伺服器接收 Pod 移出命令。

復原步驟

1. 識別 NotReady 節點：

```
kubectl get nodes
```

2. 列出每個 NotReady 節點上的 Pod：

```
kubectl get pods --all-namespaces --field-selector spec.nodeName=<node-name>
```

3. 強制刪除每個 NotReady 節點上的所有 Pod：

```
kubectl delete pod --force --grace-period=0 -n <namespace> <pod-name>
```

4. 重試滾動更新：

```
aws eks update-nodegroup-version --cluster-name <cluster> --nodegroup-name
 <nodegroup> --region <region>
```

5. 驗證節點復原：

```
kubectl get nodes
```

> **Important**
>
> 強制刪除 Pod 會無可追溯地終止工作負載。狀態工作負載可能會遺失資料。在 Auto Scaling 群組終止容器之前，容器可能會繼續在中斷連線的執行個體上執行。這是當 CA 轉返視窗不再可用時的最後手段。

考量和限制

區域可用性

CA 輪換可在支援 Amazon EKS 的所有 AWS 商業區域中使用。

最多兩個 CAs

EKS 叢集隨時最多可以有兩個 CAs：作用中 CA 和一個後續 CA。在先前的輪換完成之前，您無法附加第二個後續 CA。

無憑證撤銷

CA 輪換不支援撤銷個別憑證。這與上游 Kubernetes 一致，上游 Kubernetes 不會實作憑證撤銷 (CRL 或 OCSP)。輪換會取代整個 CA，從信任套件中移除傳出 CA 簽署的所有憑證，自然會使其失效。

AWS無法刪除 附加CAs

如果 AWS 自動附加後續 CA，則無法刪除它。此保護措施可確保輪換程序不會因意外刪除而中斷。只要客戶附加CAs 不是作用中簽署 CA，就可以將其刪除。

CA 有效期間

使用叢集建立的原始 CA 具有 10 年的有效期。透過輪換程序建立的後續 CAs具有 5 年有效期間。您叢集的所有未來 CAs都將遵循 5 年有效期間。使用 檢查您的 CA 過期`describe-certificate-authority`。

雙重信任期間的 EC2 使用者資料大小

在雙重信任期間，由於包含兩個 CA 憑證 （合併約 2.8 KB，或使用 gzip 壓縮約 1.9 KB)，叢集的信任套件大小會增加。對於在 EC2 啟動範本中提供自訂使用者資料的工作者節點，請確認您的總使用者資料大小不超過 EC2 使用者資料限制 16KB。如果您現有的使用者資料接近此限制，新增第二個 CA 可能會導致啟動範本建立失敗，並防止佈建新的節點。請考慮使用 gzip 壓縮您的使用者資料內容，以減少大小。

CA 輪換期間的版本升級

EKS 叢集版本升級和 CA 輪換是獨立的操作。不過，您無法同時執行兩者。如果 CA 輪換操作正在進行中，版本升級將被拒絕，直到 CA 操作完成，反之亦然。

自攜 CA (BYOCA)

目前不支援使用您自己的 AWS 私有 CA 來備份 EKS 叢集的憑證。

常見問答集

如何開始使用 CA 輪換？

若要開始使用，請執行 `aws eks list-certificate-authorities --cluster-name my-cluster`以檢視作用中的 CA 及其過期時間。如果您準備好開始輪換，請執行 `aws eks create-certificate-authority --cluster-name my-cluster` 以附加後續 CA。入門一節涵蓋完整的step-by-step程序。您也可以透過 Amazon EKS 主控台執行 CA 輪換。

CA 輪換是否有任何相關成本？

否。所有 EKS 叢集可免費使用 CA 輪換。

如果我未在 CA 過期之前輪換 CA，會發生什麼情況？

我們有自動保護措施，可防止 EKS 叢集在沒有有效 CA 的情況下達到 CA 過期。如果您不自行啟動 CA 輪換，我們會在過期之前自動附加並啟用後續 CA，確保您的叢集保持可用。

不過，成功輪換也需要您更新管理的工作者節點（非 EKS Auto Mode、非 Fargate) 和外部用戶端，以信任後續 CA。如果這些元件未在後續 CA 啟用之前更新，它們將失去與 API 伺服器的連線。

在 Kubernetes 中，如果 CA 未在到期之前輪換，則該 CA 簽署的所有憑證都會失效。任何用戶端都無法再連線 API 伺服器，而且叢集無法使用。

我有多少時間可以完成 CA 輪換？

完成 CA 輪換的整體時間取決於 AWS 自動化防護措施和您自己的更新程序。

AWS 提供其管理項目的最終時間表。後繼 CA 會在傳出 CA 到期前約 2 年附加。如果您不自行啟用後續 CA，我們會在過期前約 6 個月自動啟用它。如果您在自動啟用後轉返，我們會在過期前 45 天執行最終自動啟用。這些保護措施可確保您的 EKS 叢集保持可用，無論您是否採取行動。

成功的 CA 輪換也取決於您更新管理的工作者節點（非 EKS Auto Mode、非 Fargate) 和外部用戶端，以在啟用後續 CA 之前信任後續 CA。這需要多長時間，取決於資料平面的工作者節點組態、外部用戶端使用量，以及執行這些元件的探索和更新所需的時間。

CA 輪換是否會導致我的叢集停機？

否。您的 EKS 叢集在整個 CA 輪換生命週期中仍然可用。控制平面會繼續在每個階段提供請求。在雙重信任期間，傳出 CA 和後續 CAs都會同時受信任，允許元件遞增更新，而不會中斷叢集操作。如果您因為發現用戶端問題而轉返到先前的 CA， 會在傳出 CA 過期前約 45 天 AWS 執行最終轉返，作為保護。

請務必區分 EKS 叢集（控制平面）和您的資料平面元件。控制平面由 完全管理 AWS ，並在整個輪換過程中保持可用。

對於 EKS Auto Mode 和 Fargate， 會自動 AWS 更新工作者節點。在這些資料平面啟動模式中，工作者節點沒有連線中斷的風險。您仍需負責更新連線到 API 伺服器的任何外部用戶端。

對於受管節點群組、自我管理節點、Karpenter 控制執行個體（未啟用偏離偵測）和混合節點，在啟用後續 CA 之前，您必須負責取代或更新這些節點。如果未取代這些節點，這些節點會在後續 CA 啟動後失去與控制平面的連線，即使控制平面本身仍能完全運作。

我的工作負載是否會在 CA 輪換期間中斷？

執行中的工作負載 (Pod) 不會因 CA 輪換本身而中斷。Pod 透過叢集網路彼此通訊，這不受 CA 變更影響。CA 用於元件與 API 伺服器之間的通訊，而非 pod-to-pod流量。

如果您的工作者節點需要替換為更新它們以信任後續 CA 的一部分（例如，執行滾動更新的受管節點群組，或 Karpenter 替換漂移節點），這些節點上的 Pod 將重新排程為正常節點替換程序的一部分。

這是節點替換期間的標準 Kubernetes 行為，不是 CA 輪換的副作用。確保您已針對關鍵工作負載設定 Pod 中斷預算 (PDBs)，以控制節點取代期間如何移出 Pod。

我叢集的 CA 何時過期？

您可以使用 AWS CLI、EKS APIs 或 Amazon EKS 主控台來檢查叢集的 CA 何時到期。例如，您可以執行下列動作：

```
aws eks describe-certificate-authority --cluster-name my-cluster
 --certificate-authority-id <ca-id> --region us-west-2 --query
 'certificateAuthority.validity.notAfter'
```

您可以執行 來尋找 CA 的 IDaws eks list-certificate-authorities --cluster-name my-cluster。

我叢集 CA 的有效期間是多久？

使用叢集建立的原始 CA 具有 10 年的有效期。透過輪換程序建立的後續 CAs具有 5 年有效期間。您叢集的所有未來 CAs都將遵循 5 年有效期間。您可以使用 檢查 CA 的過期時間describe-certificate-authority。

我可以先自行啟動 CA 輪換 AWS ，再自動啟動嗎？

是。您可以隨時使用 附加後續 CAaws eks create-certificate-authority --cluster-name my-cluster。您不需要等待 AWS 來啟動輪換。提早開始可讓您有更多時間根據自己的排程來識別和更新工作者節點和外部用戶端。

啟用後續 CA 後是否可以轉返？

是，只要轉返視窗尚未過期，CA 轉返會在啟用後續 CA 之後提供。CA 轉返會將先前的 CA 重新啟用為簽署授權。在最終自動啟用後 (CA 過期前 45 天） 無法使用此功能。您可以使用 檢查 CA 上的 rollbackAvailable 欄位describe-certificate-authority。如需詳細資訊，請參閱 CA 轉返一節。

如何知道啟用後續 CA 是安全的？

當您管理的所有工作者節點（非 EKS Auto Mode、非 Fargate) 和外部用戶端已更新以信任後續 CA 時，可以安全地啟用後續 CA。您可以透過確認每個用戶端都可以使用更新的信任組態與 API 伺服器成功通訊來驗證這一點。 AWS 不提供所有用戶端都就緒的單一指標，因為它無法在您的外部系統中看到。提早開始探索程序，給自己時間來識別所有用戶端。

如何監控多個叢集的 CA 輪換進度？

您可以使用 CLI AWS 或 EKS API，以程式設計方式完成此作業。針對每個叢集使用 `aws eks list-certificate-authorities` 。下列欄位提供機群層級監控的情境感知：

- `signingStatus`：指出 CA 是否正在主動簽署憑證 (NOT_USED、ACTIVATING、IN_USE)
- `distributionStatus`：指出 是否 AWS 已完成將 CA 分配至受管元件 (IN_PROGRESS、COMPLETE、FAILED、DELETING)
- `rollbackAvailable`：指出啟用後續 CA 後是否可使用 CA 轉返
- `createdBy` / `activatedBy`：區分客戶起始和 AWS起始的操作 (CUSTOMER、EKS)
- `scheduledEvents.firstAutoActivation` / `scheduledEvents.finalAutoActivation`：顯示即將到來的 AWS 自動啟用日期

下列範例指令碼會檢查叢集清單中的 CA 輪換狀態：

```
#!/bin/bash
CLUSTERS=("cluster-1" "cluster-2" "cluster-3")
REGION="us-west-2"

for CLUSTER in "${CLUSTERS[@]}"; do
  echo "--- $CLUSTER ---"
  aws eks list-certificate-authorities \
    --cluster-name "$CLUSTER" \
    --region "$REGION" \
    --query 'certificateAuthorities[].
{Id:id,Signing:signingStatus,Distribution:distributionStatus,Expiry:validity.notAfter}'
 \
    --output table
done
```

您可以擴展此範圍以涵蓋多個區域和帳戶，並篩選已附加後續 CA、正在等待您的動作或即將接近自動啟用截止日期的叢集。

通知也會在輪換生命週期的每個階段，透過 AWS 運作狀態和電子郵件傳遞每個叢集。

如果我在更新所有用戶端之前啟用後續 CA，會發生什麼情況？

任何尚未更新以信任後續 CA 的用戶端都會失去與 EKS 叢集的連線。啟用後續 CA 之後，API 伺服器會呈現後續 CA 簽署的憑證。不信任用戶端的 TLS 驗證會失敗且無法連線。如果發生這種情況，您可以轉返到先前的 CA （如果轉返視窗仍然可用），以在修復剩餘的用戶端時還原連線。

如果我錯過 CA 過期截止日期，會發生什麼情況？

AWS 可防止這種情況發生。自動防護可確保您的 EKS 叢集不會在沒有有效 CA 的情況下達到 CA 過期。如果您尚未這樣做，我們將附加後續 CA，並在過期前約 6 個月自動將其啟用。如果您在第一次自動啟用後轉返， 會在過期前 45 天 AWS 執行最終轉返。您的叢集將保持可用。

不過，如果您管理的工作者節點 （非 EKS Auto Mode、非 Fargate) 和外部用戶端在自動啟用發生時尚未更新以信任後續 CA，這些元件將失去與 API 伺服器的連線。

在 CA 輪換期間，是否可以失去對叢集的存取權？

您的 EKS 叢集 （控制平面） 在整個 CA 輪換生命週期保持可用。 AWS 保護可確保叢集本身不會無法使用。

不過，如果您管理的個別用戶端在啟用後續 CA 之前未更新為信任後續 CA，則可能會失去存取權。例如，如果您的 kubeconfig、CI/CD 管道或監控工具仍然只參考傳出 CA，這些用戶端在啟用後續 CA 之後將無法連線。如果發生這種情況，CA 轉返可以在您更新受影響的用戶端時還原存取權。

我需要重新啟動 Pod 嗎？

執行中的工作負載 Pod 不會直接受到 CA 輪換的影響。每個節點上的 kubelet 會處理 API 伺服器通訊，因此只要您的節點已更新，工作負載 Pod 就會繼續執行而不會中斷。不過，在啟用後續 CA 之後，使用 client-go 與 API 伺服器通訊的叢集內控制器和運算子可能需要重新啟動，因為 client-go 不會動態重新讀取 CA 信任套件。對於 AWS Fargate 節點， 會透過自然 Pod 回收程序自動 AWS 處理更新。

我的信任套件是否會在輪換期間變更？

是。在 CA 輪換期間，叢集的信任套件會同時包含兩個憑證授權單位：傳出 CA 和後續 CA。這是雙信任期間預期的行為，也是輪換程序維持所有元件連線的方式。

應用程式和用戶端應設定為信任 CA 套件，而不是釘選到單一 CA 憑證。CA 鎖定 （不建議對單一 CA 進行嚴格驗證），因為它會在信任套件更新時導致失敗。這適用於連線到 EKS 叢集 API 伺服器的任何用戶端 TLS 組態。

為什麼我的通知時間表與本文件所述不同？

如果您的叢集是在 2018-2019 中建立，您的叢集將在調整後的時間軸上收到自動通知。您的第一個通知將包含叢集特定的相關日期和後續步驟。標準通知里程碑是根據叢集的 CA 過期日期來計算。對於此範圍內的叢集，這些計算日期早於此功能的可用性，因此會套用調整後的排程。

理解 Amazon EKS 所建立的 RBAC 角色及使用者

當您建立 Kubernetes 叢集時，會在該叢集上建立數個預設 Kubernetes 身分，以便 Kubernetes 能正常運作。Amazon EKS 會為其每個預設元件建立 Kubernetes 身分。這些身分為叢集元件提供 Kubernetes 角色型存取控制 (RBAC)。如需詳細資訊，請參閱 Kubernetes 文件中的使用 RBAC 授權。

當您將選用的附加元件安裝到叢集時，可能會將其他 Kubernetes 身分新增至叢集。如需有關本主題未討論之身分的詳細資訊，請參閱附加元件的文件。

您可以使用 AWS 管理主控台 或 `kubectl`命令列工具，在叢集上檢視 Amazon EKS 建立的 Kubernetes 身分清單。所有的使用者身分都會出現在 kube 稽核日誌中，可供您透過 Amazon CloudWatch 加以利用。

AWS 管理主控台

先決條件

您使用的 IAM 主體必須具有所需許可中所述的許可。

使用 檢視 Amazon EKS 建立的身分 AWS 管理主控台

1. 開啟 Amazon EKS 主控台。
2. 在 Clusters (叢集) 清單中，選擇包含您要檢視之身分的叢集。
3. 選擇 Resources (資源) 標籤。
4. 在 Resource types (資源類型) 下選擇 Authorization (授權)。
5. 選擇 ClusterRoles、ClusterRoleBindings、Roles 或 RoleBindings。所有前面帶有 eks 的資源皆是由 Amazon EKS 所建立。其他由 Amazon EKS 建立的身分資源包含：
   - 名為 aws-node 的 ClusterRole 和 ClusterRoleBinding。aws-node 資源支援 Kubernetes 專用 Amazon VPC CNI 外掛程式，Amazon EKS 會在所有叢集上安裝。
   - 名為 vpc-resource-controller-role 的 ClusterRole，以及名為 vpc-resource-controller-rolebinding 的 ClusterRoleBinding。這些資源支援 Amazon VPC 資源控制器，Amazon EKS 會在所有叢集上安裝該控制器。

   除了您在主控台中看到的資源之外，您的叢集上還存在下列特殊使用者身分，但在叢集的組態中不會顯示出來：
   - **`eks:cluster-bootstrap`**：用於叢集引導期間的 `kubectl` 操作。
   - **`eks:support-engineer`**：用於叢集管理操作。

6. 選擇特定資源以檢視其相關詳細資訊。依預設，您可以在 Structured view (結構化檢視) 中檢視資訊。您可以在詳細資訊頁面的右上角，選擇 Raw view (原始檢視) 以查看資源的所有資訊。

kubectl

先決條件

您用來列出叢集上 Kubernetes 資源的實體 (AWS Identity and Access Management (IAM) 或 OpenID Connect (OIDC)) 必須由 IAM 或 OIDC 身分提供者驗證。必須授予實體在叢集上針對 `Role`、`ClusterRole`、`RoleBinding` 和 `ClusterRoleBinding` 資源使用 Kubernetes `get` 和 `list` 動詞的許可，以便與實體互相搭配使用。如需有關授予 IAM 實體叢集存取權的詳細資訊，請參閱the section called “Kubernetes API 存取”。如需有關授予經由您的 OIDC 提供商驗證之實體叢集存取權的詳細資訊，請參閱 the section called “連結 OIDC 提供商”。

使用 **`kubectl`** 檢視 Amazon EKS 所建立的身分

執行您要查看的資源類型的命令。所有前面帶有 eks 的傳回資源皆是由 Amazon EKS 所建立。除了命令輸出中傳回的資源之外，您的叢集上還存在下列特殊使用者身分，但在叢集的組態中不會顯示出來：

- **`eks:cluster-bootstrap`**：用於叢集引導期間的 `kubectl` 操作。
- **`eks:support-engineer`**：用於叢集管理操作。

ClusterRoles：`ClusterRoles` 設定範圍為您的叢集內，因此授予角色的任何許可皆會套用至叢集上任何 Kubernetes 命名空間中的資源。

下列命令會傳回叢集上由 Amazon EKS 建立的所有 Kubernetes `ClusterRoles`。

```
kubectl get clusterroles | grep eks
```

除了在前面加上 的輸出中`ClusterRoles`傳回的 之外`eks:`，還`ClusterRoles`存在以下項目。

- **`aws-node`** – 此 `ClusterRole` 支援 Kubernetes 專用 Amazon VPC CNI 外掛程式，Amazon EKS 會在所有叢集上安裝。
- **`vpc-resource-controller-role`**：此 `ClusterRole` 支援 Amazon VPC 資源控制器，Amazon EKS 會在所有叢集上安裝該控制器。

若要查看 `ClusterRole` 的規格，請將以下命令中的 *`ek: k8s-metrics`* 取代為先前命令輸出中傳回的 `ClusterRole`。下列範例會傳回 *`eks:k8s-metrics`* `ClusterRole` 的規格。

```
kubectl describe clusterrole eks:k8s-metrics
```

範例輸出如下。

```
Name:         eks:k8s-metrics
Labels:       <none>
Annotations:  <none>
PolicyRule:
  Resources         Non-Resource URLs  Resource Names  Verbs
  ---------         -----------------  --------------  -----
                    [/metrics]         []              [get]
  endpoints         []                 []              [list]
  nodes             []                 []              [list]
  pods              []                 []              [list]
  deployments.apps  []                 []              [list]
```

ClusterRoleBindings：ClusterRoleBindings 設定範圍為您的叢集內。

下列命令會傳回叢集上由 Amazon EKS 建立的所有 Kubernetes ClusterRoleBindings。

```
kubectl get clusterrolebindings | grep eks
```

除了輸出中傳回的 ClusterRoleBindings 之外，還存在以下 ClusterRoleBindings。

- **aws-node** – 此 ClusterRoleBinding 支援 Kubernetes 專用 Amazon VPC CNI 外掛程式，Amazon EKS 會在所有叢集上安裝。
- **vpc-resource-controller-rolebinding**：此 ClusterRoleBinding 支援 Amazon VPC 資源控制器，Amazon EKS 會在所有叢集上安裝該控制器。

若要查看 ClusterRoleBinding 的規格，請將以下命令中的 *ek: k8s-metrics* 取代為先前命令輸出中傳回的 ClusterRoleBinding。下列範例會傳回 *eks:k8s-metrics* ClusterRoleBinding 的規格。

```
kubectl describe clusterrolebinding eks:k8s-metrics
```

範例輸出如下。

```
Name:         eks:k8s-metrics
```

```
Labels:       <none>
Annotations:  <none>
Role:
  Kind:  ClusterRole
  Name:  eks:k8s-metrics
Subjects:
  Kind  Name             Namespace
  ----  ----             ---------
  User  eks:k8s-metrics
```

角色：Roles 設定範圍為 Kubernetes 命名空間內。所有 Amazon EKS 所建立的 Roles 設定範圍皆為 kube-system 命名空間內。

下列命令會傳回叢集上由 Amazon EKS 建立的所有 Kubernetes Roles。

```
kubectl get roles -n kube-system | grep eks
```

若要查看 Role 的規格，請將以下命令中的 *ek: k8s-metrics* 取代為先前命令輸出中傳回之 Role 的名稱。下列範例會傳回 *eks:k8s-metrics* Role 的規格。

```
kubectl describe role eks:k8s-metrics -n kube-system
```

範例輸出如下。

```
Name:         eks:k8s-metrics
Labels:       <none>
Annotations:  <none>
PolicyRule:
  Resources         Non-Resource URLs  Resource Names              Verbs
  ---------         -----------------  --------------              -----
  daemonsets.apps   []                 [aws-node]                  [get]
  deployments.apps  []                 [vpc-resource-controller]  [get]
```

RoleBindings：RoleBindings 設定範圍為 Kubernetes 命名空間內。所有 Amazon EKS 所建立的 RoleBindings 設定範圍皆為 kube-system 命名空間內。

下列命令會傳回叢集上由 Amazon EKS 建立的所有 Kubernetes RoleBindings。

```
kubectl get rolebindings -n kube-system | grep eks
```

若要查看 RoleBinding 的規格，請將以下命令中的 *ek: k8s-metrics* 取代為先前命令輸出中傳回的 RoleBinding。下列範例會傳回 *eks:k8s-metrics* RoleBinding 的規格。

```
kubectl describe rolebinding eks:k8s-metrics -n kube-system
```

範例輸出如下。

```
Name:         eks:k8s-metrics
Labels:       <none>
Annotations:  <none>
Role:
  Kind:  Role
  Name:  eks:k8s-metrics
Subjects:
  Kind  Name             Namespace
  ----  ----             ---------
  User  eks:k8s-metrics
```

在現有叢集上使用 KMS 加密 Kubernetes 秘密

> **Important**
>
> 此程序僅適用於執行 Kubernetes 版本 1.27 或更低版本的 EKS 叢集。如果您執行的是 Kubernetes 版本 1.28 或更新版本，則根據預設，會使用封套加密來保護 Kubernetes 秘密。如需詳細資訊，請參閱the section called “針對所有 Kubernetes API 資料的預設封套加密”。

如果您啟用秘密加密，則會使用您選取的 AWS KMS 金鑰來加密 Kubernetes 秘密。KMS 金鑰必須滿足以下條件：

- 對稱
- 可加密和解密資料
- 在與叢集相同的 AWS 區域中建立
- 如果 KMS 金鑰是在其他帳戶中建立，則 IAM p主體必須具有 KMS 金鑰的存取權。

如需詳細資訊，請參閱 AWS Key Management Service 開發人員指南中的允許其他帳戶中的 IAM 主體使用 KMS 金鑰。

> **Warning**
>
> 啟用秘密加密後，您無法予以停用。此動作不可復原。

eksctl

此程序僅適用於執行 Kubernetes 版本 1.27 或更低版本的 EKS 叢集。如需詳細資訊，請參閱the section called “針對所有 Kubernetes API 資料的預設封套加密”。

您可以在以兩種方式啟用加密：

- 使用單一命令將加密新增至叢集。

  若要自動重新加密您的秘密，請執行下列命令。

  ```
  eksctl utils enable-secrets-encryption \
      --cluster my-cluster \
      --key-arn arn:aws: kms:region-code:account:key/key
  ```

  若要退出自動重新加密您的密碼，請執行下列命令。

  ```
  eksctl utils enable-secrets-encryption \
      --cluster my-cluster \
      --key-arn arn:aws: kms:region-code:account:key/key \
      --encrypt-existing-secrets=false
  ```

- 使用 `kms-cluster.yaml` 檔案將加密新增到您的叢集中。

  ```
  apiVersion: eksctl.io/v1alpha5
  kind: ClusterConfig

  metadata:
    name: my-cluster
    region: region-code

  secretsEncryption:
    keyARN: arn:aws: kms:region-code:account:key/key
  ```

  若要讓您的秘密自動重新加密，請執行下列命令。

```
eksctl utils enable-secrets-encryption -f kms-cluster.yaml
```

若要退出自動重新加密您的密碼，請執行下列命令。

```
eksctl utils enable-secrets-encryption -f kms-cluster.yaml --encrypt-existing-
secrets=false
```

AWS 管理主控台

a. 此程序僅適用於執行 Kubernetes 版本 1.27 或更低版本的 EKS 叢集。如需詳細資訊，請參閱the section called “針對所有 Kubernetes API 資料的預設封套加密”。
b. 開啟 Amazon EKS 主控台。
c. 選擇您要向其新增 KMS 加密的叢集。
d. 選擇 Overview (概觀) 索引標籤 (預設選取此項)。
e. 向下捲動到 Secrets encryption (私密加密) 區段，然後選擇 Enable (啟用)。
f. 從下拉式清單中選取金鑰，然後選擇 Enable (啟用) 按鈕。如果未列出任何金鑰，您必須先建立一個。如需詳細資訊，請參閱建立金鑰。
g. 選擇 Confirm (確認) 按鈕以使用選擇的金鑰。

AWS CLI

a. 此程序僅適用於執行 Kubernetes 版本 1.27 或更低版本的 EKS 叢集。如需詳細資訊，請參閱the section called “針對所有 Kubernetes API 資料的預設封套加密”。
b. 使用下列 CLI AWS 命令，將秘密加密組態與叢集建立關聯。使用自己的取代範例值。

```
aws eks associate-encryption-config \
    --cluster-name my-cluster \
    --encryption-config '[{"resources":["secrets"],"provider":
{"keyArn":"arn:aws: kms:region-code:account:key/key"}}]'
```

範例輸出如下。

```
{
  "update": {
    "id": "3141b835-8103-423a-8e68-12c2521ffa4d",
    "status": "InProgress",
    "type": "AssociateEncryptionConfig",
    "params": [
```

```
      {
        "type": "EncryptionConfig",
        "value": "[{\"resources\":[\"secrets\"],\"provider\":{\"keyArn\":
\"arn:aws: kms:region-code:account:key/key\"}}]"
      }
    ],
    "createdAt": 1613754188.734,
    "errors": []
  }
}
```

c. 您可以使用下列命令來監控您的加密更新的狀態。使用之前輸出傳回的特定 cluster name 和 update ID。顯示 Successful 狀態時，即表示更新已完成。

```
aws eks describe-update \
    --region region-code \
    --name my-cluster \
    --update-id 3141b835-8103-423a-8e68-12c2521ffa4d
```

範例輸出如下。

```
{
  "update": {
    "id": "3141b835-8103-423a-8e68-12c2521ffa4d",
    "status": "Successful",
    "type": "AssociateEncryptionConfig",
    "params": [
      {
        "type": "EncryptionConfig",
        "value": "[{\"resources\":[\"secrets\"],\"provider\":{\"keyArn\":
\"arn:aws: kms:region-code:account:key/key\"}}]"
      }
    ],
    "createdAt": 1613754188.734,
    "errors": []
  }
}
```

d. 若要驗證您的叢集中是否已啟用加密，請執行 describe-cluster 命令。回應包含 EncryptionConfig 字串。

```
aws eks describe-cluster --region region-code --name my-cluster
```

在叢集上啟用加密之後，您須使用新金鑰來加密所有現有的密碼：

> **Note**
>
> 如果您使用 `eksctl`，則僅當您選擇退出自動重新加密秘密時，才需要執行以下命令。

```
kubectl get secrets --all-namespaces -o json | kubectl annotate --overwrite -f - kms-encryption-timestamp="time value"
```

> **Warning**
>
> 如果針對現有叢集啟用秘密加密，且使用的 KMS 金鑰已被刪除，則無法復原叢集的路徑。若刪除 KMS 金鑰，即會永久使叢集處於降級狀態。如需詳細資訊，請參閱刪除 AWS KMS 金鑰。

> **Note**
>
> 根據預設，`create-key`命令會使用金鑰政策來建立對稱加密 KMS 金鑰，讓帳戶根管理員能夠存取 AWS KMS 動作和資源。如果您想要縮小許可範圍，請確保要呼叫 `create-cluster` API 之主體的政策上允許 `kms:DescribeKey` 和 `kms:CreateGrant` 動作。
> 對於使用 KMS 封套加密的叢集，需要 `kms:CreateGrant` 許可。CreateCluster 動作不支援條件 `kms:GrantIsForAWSResource`，並且不應在 KMS 政策中使用該條件來控制執行 CreateCluster 的使用者的 `kms:CreateGrant` 許可。

搭配 Amazon EKS Pod 使用 AWS Secrets Manager 秘密

若要顯示來自 Secrets Manager 的秘密，以及參數存放區的參數，作為掛載於 Amazon EKS Pod 的檔案，您可以將 AWS Secrets and Configuration Provider (ASCP) 用於 Kubernetes Secrets Store CSI Driver。

透過 ASCP，您可以在 Secrets Manager 中存放和管理您的秘密，然後透過在 Amazon EKS 上執行的工作負載擷取這些秘密。您可以使用 IAM 角色和政策，來限制對叢集中特定 Kubernetes Pod 的秘密存取。ASCP 會擷取 Pod 身分識別並交換 IAM 角色的身分識別。ASCP 會假設 Pod 的 IAM 角色，然後可以從已授權該角色的 Secrets Manager 擷取秘密。

如果您使用 Secrets Manager 自動輪換您的秘密，您也可以使用 Secrets Store CSI Driver 輪換調解器功能，以確保您從 Secrets Manager 擷取最新的秘密。

> **Note**
>
> 不支援 AWS Fargate (Fargate) 節點群組。

如需詳細資訊，請參閱《AWS Secrets Manager 使用者指南》中的使用 Amazon EKS 中的 Secrets Manager 秘密。

針對所有 Kubernetes API 資料的預設封套加密

Amazon Elastic Kubernetes Service (Amazon EKS) 會針對執行 Kubernetes 版本 1.28 或更新版本的 EKS 叢集中的所有 Kubernetes API 資料，提供預設封套加密。

封套加密可保護您使用 Kubernetes API 伺服器存放的資料。例如，封套加密適用於 Kubernetes 叢集的組態，例如 `ConfigMaps`。封套加密不適用於節點或 EBS 磁碟區上的資料。EKS 先前支援加密 Kubernetes 秘密，而現在此封套加密會延伸至所有 Kubernetes API 資料。

這項作法可提供受管的預設體驗，為您的 Kubernetes 應用程式實作深度防禦功能，且您不需要採取任何行動。

Amazon EKS 使用 AWS Key Management Service (KMS) 搭配 Kubernetes KMS 提供者 v2 搭配 Amazon Web Services 擁有的金鑰，提供多一層安全性，以及讓您從 AWS KMS 使用自己的客戶受管金鑰 (CMK) 的選項。

了解封套加密

信封加密是使用資料加密金鑰 (DEK) 在傳送到資料存放區 （ 等 ） 之前加密純文字資料，然後使用存放在遠端、集中管理 KMS 系統 (AWS KMS) 中的根 KMS 金鑰加密 DEK 的程序。這是一種深度防禦策略，因為它會使用加密金鑰 (DEK) 保護資料，然後使用單獨、安全儲存的加密金鑰 (稱為金鑰加密金鑰 (KEK)) 保護 DEK，以增添另一層安全性。

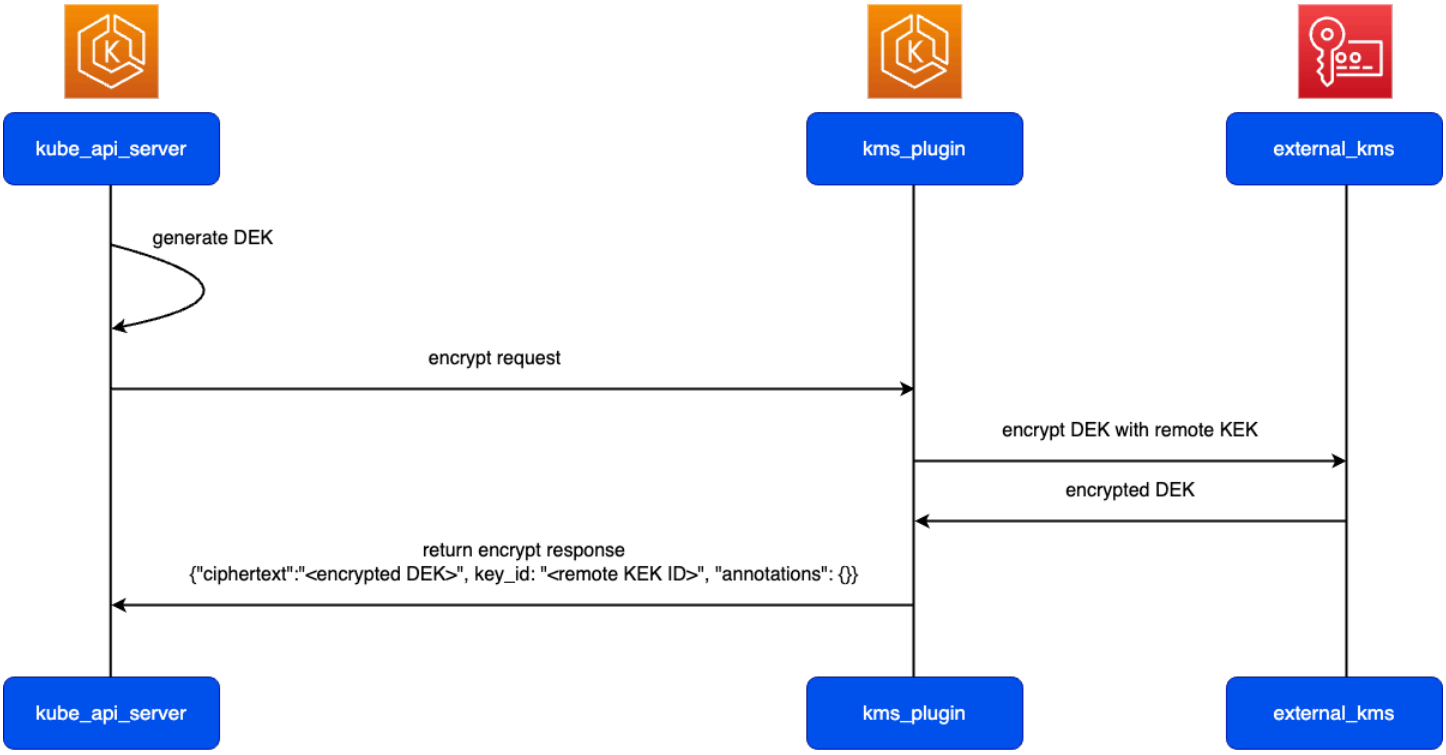
Amazon EKS 如何使用 KMS v2 和 AWS KMS 啟用預設信封加密

Amazon EKS 使用 KMS v2 來實作受管 Kubernetes 控制平面中所有 API 資料的預設封套加密，然後再將其保留在 etcd 資料庫中。啟動時，叢集 API 伺服器會從與隨機產生的資料結合的秘密種子，產生資料加密金鑰 (DEK)。此外，在啟動時，API 伺服器會呼叫 KMS 外掛程式，使用來自 KMS 的遠端金鑰加密金鑰 (KEK) 來加密 AWS DEK 種子。這是一種一次性呼叫，會在 API 伺服器啟動和 KEK 輪換時執行。然後，API 伺服器會快取加密的 DEK 種子。在此之後，API 伺服器會使用快取的 DEK 種子，

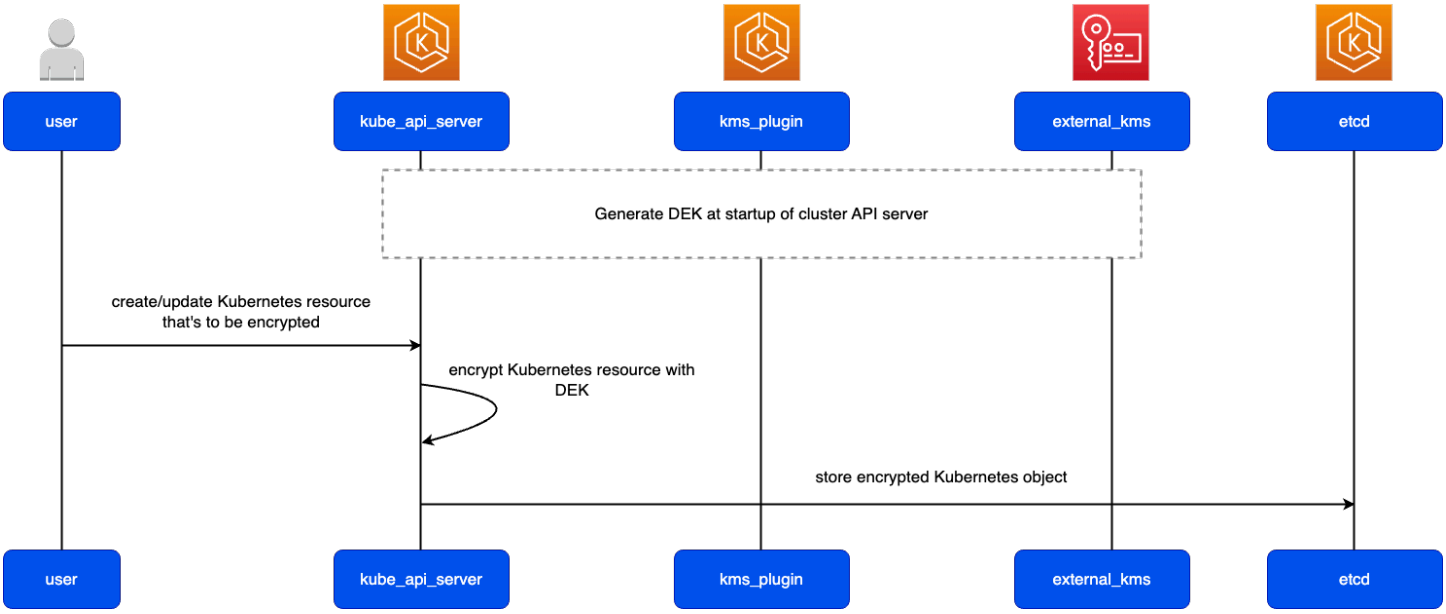
根據金鑰衍生函數 (KDF) 產生其他單次使用 DEK。然後，每個產生的 DEK 在儲存到 etcd 之前，僅可使用一次來加密單一 Kubernetes 資源。透過使用 KMS v2 中的加密的快取 DEK 種子，在 API 伺服器上加密 Kubernetes 資源的程序將實現更高效能且更具成本效益。

根據預設，此 KEK 由 擁有 AWS，但您可以選擇從 AWS KMS 自攜。

以下圖表說明 API 伺服器啟動時 DEK 的產生和加密過程。



以下高階圖表說明 Kubernetes 資源在儲存到 etcd 之前對其進行加密的過程。



常見問答集

預設封套加密如何提升 EKS 叢集的安全狀態？

此功能可減少未加密的中繼資料和客戶內容的表面積和時間期間。使用預設封套加密時，中繼資料和客戶內容在儲存到 etcd 之前，在 kube-apiserver 的記憶體中僅會處於暫時未加密狀態。kube-apiserver 的記憶體可透過 Nitro 系統進行保護。Amazon EKS 僅會針對受管 Kubernetes 控制平面使用 Nitro 型 EC2 執行個體。這些執行個體具有安全控制設計，可防止任何系統或人員存取其記憶體。

要獲得此功能，我需要執行哪個 Kubernetes 版本？

若要啟用預設封套加密，Amazon EKS 叢集必須執行 Kubernetes 版本 1.28 或更新版本。

如果我執行的 Kubernetes 叢集版本不支援此功能，我的資料是否仍然安全？

是。 AWS安全是我們的最高優先順序。我們所有的數位轉型和創新皆會遵循最高的安全操作實務，並且我們始終致力於提高該標準。

無論執行的 Kubernetes 版本為何，對於每個 EKS 叢集，都會以磁碟層級加密儲存在 etcd 中的所有資料。EKS 使用根金鑰來產生磁碟區加密金鑰，且這些金鑰由 EKS 服務管理。此外，每個 Amazon EKS 叢集都會使用叢集特定的虛擬機器在隔離的 VPC 中執行。因為此架構以及我們的操作安全性實務，Amazon EKS 已達到多項合規評分和標準，包括 SOC 1、SOC 2、SOC 3、PCI-DSS、ISO 和 HIPAA 資格。這些合規評分和標準適用於所有 EKS 叢集，而無論是否使用預設封套加密。

封套加密如何在 Amazon EKS 中運作？

啟動時，叢集 API 伺服器會從與隨機產生的資料結合的秘密種子，產生資料加密金鑰 (DEK)。此外，在啟動時，API 伺服器會呼叫 KMS 外掛程式，以使用來自 KMS 的遠端金鑰加密金鑰 (KEK) 來加密 AWS DEK。這是一種一次性呼叫，會在 API 伺服器啟動和 KEK 輪換時執行。然後，API 伺服器會快取加密的 DEK 種子。在此之後，API 伺服器會使用快取的 DEK 種子，根據金鑰衍生函數 (KDF) 產生其他單次使用 DEK。然後，每個產生的 DEK 在儲存到 etcd 之前，僅可使用一次來加密單一 Kubernetes 資源。

請務必注意，從 API 伺服器發出其他呼叫，以驗證 AWS KMS 整合的運作狀態和正常功能。這些額外的運作狀態檢查會顯示在您的 AWS CloudTrail 中。

我是否需要執行任何操作或變更此功能的任何許可，以確保在 EKS 叢集中正常運作？

否，您無需執行任何動作。在 Amazon EKS 中，封套加密現在已成為預設組態，並且所有執行 Kubernetes 版本 1.28 或更新版本的所有叢集均已啟用該組態。 AWS KMS 整合是由 管理的 Kubernetes API 伺服器建立 AWS。這表示您不需要設定任何許可，即可開始為叢集使用 KMS 加密。

如何知道我的叢集是否已啟用預設封套加密？

如果您進行移轉以使用自己的 CMK，則會看到與叢集相關聯的 KMS 金鑰 ARN。此外，您可以檢視與叢集 CMK 使用相關聯的 AWS CloudTrail 事件日誌。

如果您的叢集使用 AWS 擁有的金鑰，則會在 EKS 主控台中詳細說明 （金鑰的 ARN 除外）。

是否可以 AWS 存取 Amazon EKS 中用於預設信封加密的 AWS 擁有金鑰？

No.在 Amazon EKS 中 AWS 具有嚴格的安全控制，可防止任何人存取任何用於保護編碼資料庫中資料的純文字加密金鑰。這些安全措施也會套用至 AWS 擁有的 KMS 金鑰。

我現有的 EKS 叢集是否已啟用預設封套加密？

如果您使用 Kubernetes 版本 1.28 版或更新版本執行 Amazon EKS 叢集，則會啟用所有 Kubernetes API 資料的封套加密。對於現有的叢集，Amazon EKS 會使用 `eks:kms-storage-migrator` RBAC ClusterRole 將先前未在 etcd 中進行封套加密的資料移轉至這一新的加密狀態。

如果我已在 EKS 叢集中為秘密啟用封套加密，這表示什麼？

如果您在 KMS 中已有用於對 Kubernetes 秘密進行封套加密的客戶自管金鑰 (CMK)，則該相同的金鑰將用作叢集中所有 Kubernetes API 資料類型的封套加密的 KEK。

使用預設封套加密執行 EKS 叢集是否需要任何額外費用？

如果您使用 Amazon Web Services 擁有的金鑰進行預設封套加密，則受管 Kubernetes 控制平面不會產生任何額外費用。根據預設，每個執行 Kubernetes 1.28 版或更新版本的 EKS 叢集都會使用 Amazon Web Services 擁有的金鑰。不過，如果您使用自己的 AWS KMS 金鑰，則適用一般 KMS 定價。

使用我自己的 AWS KMS 金鑰來加密叢集中的 Kubernetes API 資料需要多少成本？

為儲存您建立或匯入 KMS 的任何自訂金鑰，您每月需支付 1 美元。加密和解密請求的 KMS 費用。每個帳戶每月都有一個免費方案，其中已涵蓋 20,000 個請求，而對於超出每月免費方案的部分，每 10,000 個請求須支付 0.03 美元。這適用於 帳戶的所有 KMS 用量，因此在叢集上使用您自己的 AWS KMS 金鑰的成本將受到您帳戶中其他叢集或 AWS 資源使用此金鑰的影響。

既然我的客戶自管金鑰 (CMK) 已被用於對所有 Kubernetes API 資料 (而不僅僅是秘密) 進行封套加密，我的 KMS 費用是否會提高？

否。我們使用 KMS v2 實作可大幅減少對 AWS KMS 發出的呼叫次數。無論 EKS 叢集中要加密或解密的其他 Kubernetes 資料為何，這都會降低與您的 CMK 相關聯的費用。

如上所述，使用遠端 KEK 加密 Kubernetes API 伺服器後，用於加密 Kubernetes 資源的產生 DEK 種子會以本機方式儲存在該伺服器的快取中。如果加密的 DEK 種子不在 API 伺服器的快取中，API 伺服器將呼叫 AWS KMS 來加密 DEK 種子。然後，API 伺服器會快取加密的 DEK 種子，以供未來在叢集中使用，而無需呼叫 KMS。同樣地，對於解密請求，API 伺服器將呼叫 AWS KMS 進行第一個解密請求，之後將快取解密的 DEK 種子並用於未來的解密操作。

如需詳細資訊，請參閱 GitHub 上 Kubernetes 增強功能中的 KEP-3299: KMS v2 改進。

我是否可以將相同的 CMK 金鑰用於多個 Amazon EKS 叢集？

是。若要再次使用金鑰，您可以在建立過程中建立 ARN 與叢集的關聯，進而將其連結至相同區域中的叢集中。不過，如果您針對多個 EKS 叢集使用相同的 CMK，則應採取必要的措施，以防止任意停用 CMK。否則，與多個 EKS 叢集相關聯的已停用 CMK 會對叢集產生更廣泛的影響，具體取決於該金鑰。

如果在啟用預設封套加密之後，我的 CMK 無法使用，那麼我的 EKS 叢集會發生什麼狀況？

如果您停用 KMS 金鑰，則其無法在任何密碼編譯操作中使用。如果無法存取現有的 CMK，API 伺服器將無法加密和保留任何新建立的 Kubernetes 物件，並且也無法解密儲存在 etcd 中的任何先前加密的 Kubernetes 物件。如果停用 CMK，叢集將立即處於運作狀態不佳/降級狀態，而此時我們將無法履行我們的服務承諾，直到您重新啟用相關聯的 CMK 為止。

停用 CMK 時，您將會收到相關通知，其中包括 EKS 叢集運作狀態降級，以及在停用 CMK 後 30 天內需要重新啟用 CMK 以確保成功還原 Kubernetes 控制平面資源。

我該如何保護 EKS 叢集免受已停用/已刪除 CMK 的影響？

為了保護您的 EKS 叢集免於發生此類狀況，您的金鑰管理員應使用具有最低權限原則的 IAM 政策來管理對 KMS 金鑰操作的存取，進而任何降低與任意停用或刪除 EKS 叢集相關聯的金鑰的風險。此外，您可以將 CloudWatch 警示設定為接收有關 CMK 狀態的通知。

如果我重新啟用 CMK，我的 EKS 叢集是否會還原？

為確保成功還原 EKS 叢集，強烈建議在停用 CMK 後的前 30 天內重新啟用 CMK。不過，成功還原 EKS 叢集還將取決於它是否會因叢集處於運作狀態不佳/降級狀態期間可能進行的自動 Kubernetes 升級而導致其出現任何 API 重大變更。

停用 CMK 後，為什麼我的 EKS 叢集會處於運作狀態不佳/降級狀態？

EKS 控制平面的 API 伺服器會使用 DEK 金鑰 (該金鑰已在 API 伺服器的記憶體中經過加密和快取)，以便在建立/更新操作期間加密所有物件，然後再將其儲存在 etcd。當從 etcd 擷取現有物件時，API 伺服器會使用相同的快取 DEK 金鑰並解密 Kubernetes 資源物件。如果您停用 CMK，API 伺服器將

不會立即受到任何影響，因為 API 伺服器的記憶體中有快取的 DEK 金鑰。不過，當 API 伺服器執行個體重新啟動時，它不會有快取的 DEK，而且需要呼叫 AWS KMS 以進行加密和解密操作。如果沒有 CMK，此程序將會失敗並顯示 KMS_KEY_DISABLED 錯誤代碼，從而導致 API 伺服器無法成功開機。

如果我刪除我的 CMK，那麼我的 EKS 叢集會發生什麼狀況？

刪除與 EKS 叢集相關聯的 CMK 金鑰，將會導致其運作狀態降級且無法復原。如果沒有叢集的 CMK，API 伺服器將無法再加密和保留任何新的 Kubernetes 物件，並且也無法解密儲存在 etcd 資料庫中的任何先前加密的 Kubernetes 物件。只有在您確定不再需要使用 EKS 叢集時，才應繼續刪除 EKS 叢集的 CMK 金鑰。

請注意，如果未找到 CMK (KMS_KEY_NOT_FOUND) 或對與您的叢集相關聯的 CMK 的授與已被撤銷 (KMS_GRANT_REVOKED)，則您的叢集將無法復原。如需叢集運作狀態和錯誤代碼的詳細資訊，請參閱叢集運作狀態FAQs和具有解析路徑的錯誤代碼。

如果因為我停用或刪除我的 CMK 而導致 EKS 叢集降級/運作狀態不佳，我是否仍要為此付費？

是。雖然在停用 CMK 的情況下，EKS 控制平面將無法使用，但仍 AWS 會執行分配給 EKS 叢集的專用基礎設施資源，直到客戶將其刪除為止。此外，在這種情況下，我們的服務承諾將不適用，因為它是客戶的自願行為或不作為，會導致您的 EKS 叢集無法保持正常的運作狀態以及正常運作。

當我的 EKS 叢集因停用的 CMK 而處於運作狀態不佳/降級狀態時，其是否可以自動升級？

是。不過，如果您的叢集有已停用的 CMK，則您將有 30 天的時間重新啟用。在此 30 天期間內，您的 Kubernetes 叢集不會自動升級。不過，如果此時間期限已過且您尚未重新啟用 CMK，則叢集將依照 EKS 中的 Kubernetes 版本生命週期，自動升級至標準支援的下一個版本 (n+1)。

當您發現受影響的叢集時，我們強烈建議快速重新啟用已停用的 CMK。請務必注意，雖然 EKS 會自動升級這些受影響的叢集，但無法保證它們會成功復原，特別是在叢集經歷過多次自動升級的情況下，因為這可能包括 Kubernetes API 的變更，以及 API 伺服器引導程序中的非預期行為。

我是否可以使用 KMS 金鑰別名？

是。Amazon EKS 支援使用 KMS 金鑰別名。別名是指 Amazon Web Service KMS 金鑰的易記名稱。例如，別名可讓您將 KMS 金鑰稱為 my-key 而不是 **1234abcd-12ab-34cd-56ef-1234567890ab**。

我是否仍然可以使用我自己的 Kubernetes 備份解決方案來備份和還原我的叢集資源？

是。您可以使用 Kubernetes 備份解決方案 (例如 Velero) 進行 Kubernetes 叢集災難復原、資料移轉和資料保護。如果您執行 Kubernetes 備份解決方案，而該解決方案可透過 API 伺服器存取叢集資源，則

該應用程式擷取的任何資料都會在到達用戶端之前解密。這可讓您復原另一個 Kubernetes 叢集中的叢集資源。

在 Amazon EKS 中強化 Kubernetes RBAC

Kubernetes 角色型存取控制 (RBAC) 控制可在叢集內執行的動作身分。許多叢集元件，包括 CSI 驅動程式和安裝為 DaemonSets 的其他附加元件，都需要廣泛的 許可才能運作。檢閱和界定這些許可會減少任何意外存取的潛在範圍。

本主題說明常見叢集元件的許可考量，以及建議的控制項。

DaemonSet 服務帳戶許可

DaemonSet Pods 會在叢集中的每個節點上執行，因此其服務帳戶字符和這些字符授予的 RBAC 許可會出現在每個節點上。

節點上的未經授權程序可能可以存取在相同節點上執行之其他 Pod 的服務帳戶字符，包括 DaemonSet Pod。授予 DaemonSet 服務帳戶的 RBAC 許可在叢集中的每個節點上都相同。

通常部署為 DaemonSets 的元件包括：

- CSI 節點驅動程式 (`ebs-csi-node`、`efs-csi-node`、`mountpoint-s3-csi-node`)
- Amazon VPC CNI 外掛程式 (`aws-node`)
- `kube-proxy`

如果 DaemonSet Pod 透過 AWS EKS Pod Identity 或 IAM Roles for Service Accounts (IRSA) 擁有 IAM 登入資料，則在同一節點上取得容器外部存取權的程序也可能存取這些登入資料。這會將 Kubernetes RBAC 的影響範圍延伸至授予 DaemonSet IAM 角色的任何 AWS API 許可。

> **Important**
>
> 檢閱許可時，請將每個 DaemonSet 服務帳戶的 Kubernetes RBAC 許可和 IAM 許可視為可從叢集中的每個節點存取。

CSI 驅動程式 RBAC 範圍

CSI 驅動程式通常會持有廣泛的 RBAC 授權，因為它們與節點、持久性磁碟區和儲存 APIs 互動。

節點物件許可

CSI 驅動程式可能需要 RBAC 許可才能修改節點物件，以支援污點移除或其他節點管理任務等功能。由於 Kubernetes RBAC 限制，這些許可適用於叢集中的所有節點物件，而不只是驅動程式執行所在的本機節點。

對於 EBS CSI 驅動程式，Helm Chart 提供參數 (`node.serviceAccount.disableMutation`)，從`ebs-csi-node`服務帳戶移除節點修改許可。啟用此功能會停用污點移除功能。

服務帳戶字符公開

CSI 驅動程式 Pod 可能會使用投影服務帳戶字符進行身分驗證。在未經授權的程序在其容器外取得存取權的節點上，這些字符可以透過容器檔案系統或 kubelet API 存取。如果服務帳戶也透過 EKS Pod Identity 或 IRSA 與 IAM 角色相關聯，則可以使用公開的字符來取得 AWS IAM 憑證。

建議的控制項

將 RBAC 範圍設為最低權限

- 檢閱繫結至 CSI 驅動程式和 DaemonSet 服務帳戶的 ClusterRoles。移除工作負載不需要的許可。
- 對於 EBS CSI 驅動程式，`true`如果您不使用污點移除功能，請將 `node.serviceAccount.disableMutation`設定為 。
- 使用 `kubectl auth can-i --list --as=system:serviceaccount:NAMESPACE:SERVICE_ACCOUNT`來稽核有效許可。

強制執行 Pod 安全標準

使用內建 Pod 安全許可控制器或政策引擎套用 Kubernetes Pod 安全標準。至少，強制執行整個`baseline`設定檔叢集和工作負載命名空間的`restricted`設定檔。這限制了在系統命名空間之外建立特權容器的能力。

使用網路政策

套用網路政策，將 CSI 驅動程式和 DaemonSet Pod 的輸出限制為僅需要的端點 （ 例如 Kubernetes API 伺服器 AWS 和服務端點 ）。這可減少可能的動作範圍。

監控 RBAC 活動

啟用 Kubernetes 稽核記錄，並監控來自 DaemonSet 服務帳戶的非預期 API 呼叫。尋找：

- 來自 CSI 驅動程式服務帳戶的節點修改

- 在系統命名空間中建立 Pod
- 秘密上的異常 `get`或 `list`呼叫

如需詳細資訊，請參閱the section called “控制平面日誌”。

Amazon EKS 自動模式的安全考量

本主題描述 Amazon EKS 自動模式的安全架構、控制項和最佳實務。隨著組織大規模部署容器化應用程式，維護強大的安全狀態變得日益複雜。EKS Auto Mode 實作自動化安全控制，並與 AWS 安全服務整合，以協助您保護叢集基礎設施、工作負載和資料。透過內建的安全功能，例如強制執行的節點生命週期管理和自動化修補程式部署，EKS 自動模式協助您維持安全最佳實務，同時減少營運開銷。

在繼續本主題之前，請確定您已熟悉基本的 EKS 自動模式概念，並已檢閱在叢集上啟用 EKS 自動模式的先決條件。有關 Amazon EKS 安全性的常規資訊，請參閱 安全。

Amazon EKS 自動模式建立在 Amazon EKS 現有安全基礎之上，同時為 EC2 受管執行個體引入了額外的自動化安全控制項。

API 安全性和驗證

Amazon EKS Auto Mode 使用 AWS 平台安全機制來保護和驗證對 Amazon EKS API 的呼叫。

- 透過與 IAM 身分整合的 EKS 存取項目來保護對 Kubernetes API AWS 的存取。
  - 如需詳細資訊，請參閱the section called “存取項目”。
- 客戶可以透過設定 EKS 存取項目來實作對 Kubernetes API 端點的精細存取控制。

網路安全

Amazon EKS 自動模式支援多層網路安全：

- VPC 整合
  - 在 Amazon 虛擬私有雲端 (VPC) 中操作
  - 支援自訂 VPC 組態與子網路佈局
  - 啟用叢集元件之間的私有聯網
  - 如需詳細資訊，請參閱管理 Amazon 虛擬私有雲的安全責任
- 網路政策

- Kubernetes 網路政策的原生支援
- 能夠定義精細網路流量規則
- 如需詳細資訊，請參閱the section called “Kubernetes 政策”

EC2 受管執行個體安全性

Amazon EKS 自動模式透過以下安全控制項來操作 EC2 受管執行個體：

EC2 安全性

- EC2 受管執行個體維持 Amazon EC2 的安全功能。
- 有關 EC2 受管執行個體的更多資訊，請參閱 Amazon EC2 中的安全性。

執行個體生命週期管理

由 EKS Auto Mode 操作的 EC2 受管執行個體的生命週期上限為 21 天。Amazon EKS 自動模式會自動終止超過此生命週期的執行個體。此生命週期限制有助於防止組態偏離，並維持安全狀態。

資料保護

- Amazon EC2 執行個體儲存體已加密，屬於直接連接到執行個體的儲存體。如需詳細資訊，請參閱 Amazon EC2 中的資料保護。
- EKS 自動模式在建立時管理連接到 EC2 執行個體的磁碟區，包括根磁碟區和資料磁碟區。EKS 自動模式不會全權管理使用 Kubernetes 持久性儲存功能建立的 EBS 磁碟區。

修補管理

- Amazon EKS 自動模式會自動將修補程式套用至受管執行個體。
- 修補程式包括：
  - 作業系統更新
  - 安全性修補程式
  - Amazon EKS 自動模式元件

> **Note**
>
> 客戶仍負責保護與更新在這些執行個體上執行的工作負載。

存取控制

- 對執行個體的直接存取受到限制：
  - 無法使用 SSH 存取。
  - AWS Systems Manager Session Manager (SSM) 存取無法使用。
- 管理操作透過 Amazon EKS API 與 Kubernetes API 執行。

自動化資源管理

Amazon EKS 自動模式不會全權管理透過 Kubernetes 持久儲存功能建立的 Amazon Elastic Block Store (Amazon EBS) 磁碟區。EKS 自動模式也不會管理彈性負載平衡器 (ELB)。Amazon EKS 自動模式會為這些資源自動執行常規任務。

儲存體安全性

- AWS 建議您為 Kubernetes 持久性儲存功能佈建的 EBS 磁碟區啟用加密。如需詳細資訊，請參閱the section called “建立 StorageClass”。
- 使用 AWS KMS 進行靜態加密
- 您可以設定 AWS 帳戶，強制加密您建立的新 EBS 磁碟區和快照副本。如需詳細資訊，請參閱《Amazon EBS 使用者指南》中的依預設啟用 Amazon EBS 加密。
- 如需詳細資訊，請參閱 Amazon EBS 中的安全性。

負載平衡器安全性

- Elastic Load Balancer 的自動化組態
- 透過 AWS Certificate Manager 整合進行 SSL/TLS 憑證管理
- 用於負載平衡器存取控制的安全群組自動化
- 如需詳細資訊，請參閱 Elastic Load Balancing 中的安全性。

安全最佳實務

下列章節描述 Amazon EKS 自動模式的安全最佳實務。

- 定期檢閱 AWS IAM 政策和 EKS 存取項目。
- 為工作負載實作最低權限存取模式。
- 透過 AWS CloudTrail 和 Amazon CloudWatch 監控叢集活動。如需詳細資訊，請參閱the section called “ AWS CloudTrail”及the section called “Amazon CloudWatch”。
- 使用 AWS Security Hub 進行安全狀態評估。
- 實作適合您工作負載的 Pod 安全標準。

EKS 功能的安全考量

本主題涵蓋 EKS 功能的重要安全考量，包括 IAM 角色組態、Kubernetes 許可，以及多叢集部署和跨帳戶 AWS 資源管理的架構模式。

EKS 功能使用 IAM 角色、EKS 存取項目和 Kubernetes RBAC 的組合，以提供對 AWS 服務、叢集內 Kubernetes 資源以及與 AWS CodeConnections、 AWS Secrets Manager 和其他 AWS 服務的整合的安全存取。

功能 IAM 角色

當您建立功能時，您會提供 IAM 功能角色，讓 EKS 用來代表您執行動作。此角色必須：

- 與叢集和功能資源位於相同的 AWS 帳戶
- 擁有允許`capabilities.eks.amazonaws.com`服務主體擔任角色的信任政策
- 根據您的需求，擁有適用於 功能類型和使用案例的 IAM 許可。如需必要 IAM 許可的詳細資訊，請參閱 the section called “ AWS CodeConnections”、 the section called “ AWS Secrets Manager”和 the section called “設定 許可”

最佳實務是考慮特定使用案例所需的權限範圍，並僅授予滿足需求所需的許可。例如，使用適用於 Kube Resource Orchestrator 的 EKS 功能時，不需要 IAM 許可，而使用適用於 Kubernetes 的 AWS 控制器的 EKS 功能時，您可以授予一或多個 AWS 服務的完整存取權。

> **Important**
>
> 雖然某些使用案例可能需要使用廣泛的管理權限，但請遵循最低權限原則，只授予特定使用案例所需的最低 IAM 許可，使用 ARNs存取，而不是使用萬用字元許可。

如需建立和設定功能 IAM 角色的詳細資訊，請參閱 the section called “功能 IAM 角色”。

EKS 存取項目

當您使用 IAM 角色建立功能時，Amazon EKS 會自動為叢集上的該角色建立存取項目。此存取項目會授予功能基準 Kubernetes 許可來運作。

> **Note**
>
> 系統會為建立 功能的叢集建立存取項目。對於遠端叢集的 Argo CD 部署，您必須在具有 Argo CD 功能適當許可的叢集上建立存取項目，以部署和管理應用程式。

存取項目包括：

- 作為委託人的 IAM 角色 ARN
- 授予基準 Kubernetes 許可的功能特定存取項目政策
- 根據功能類型的適當範圍（全叢集或命名空間範圍）

> **Note**
>
> 對於 Argo CD，命名空間範圍許可會授予功能組態中指定的命名空間（預設為 `argocd`)。

依功能區分的預設存取項目政策

每個功能類型都會授予功能角色所需的許可，並設定不同的預設存取項目政策，如下所示：

kro

- `arn:aws:eks::aws:cluster-access-policy/AmazonEKSKROPolicy`（叢集範圍）

  准許監看和管理 ResourceGraphDefinitions，並建立 RGDs定義的自訂資源執行個體。

ACK

- arn:aws:eks::aws:cluster-access-policy/AmazonEKSACKPolicy（叢集範圍）

  准許在所有命名空間中建立、讀取、更新和刪除 ACK 自訂資源。

Argo CD

- arn:aws:eks::aws:cluster-access-policy/AmazonEKSArgoCDClusterPolicy（叢集範圍）

  授予 Argo CD 的叢集層級許可，以探索資源和管理叢集範圍物件。

- arn:aws:eks::aws:cluster-access-policy/AmazonEKSArgoCDPolicy（命名空間範圍）

  授予 Argo CD 部署和管理應用程式的命名空間層級許可。範圍為功能組態中指定的命名空間（預設為 argocd)。

如需更多詳細資訊the section called “檢閱存取政策”，請參閱 。

其他 Kubernetes 許可

除了預設存取項目政策之外，某些功能可能需要額外的 Kubernetes 許可。您可以使用下列其中一種方式授予這些許可：

- 存取項目政策：將其他受管政策與存取項目建立關聯
- Kubernetes RBAC：為功能的 Kubernetes 使用者建立Role或ClusterRole繫結

ACK 秘密讀取器許可

有些 ACK 控制器需要讀取 Kubernetes 秘密，才能擷取資料庫密碼等敏感資料。下列 ACK 控制器需要秘密讀取存取權：

- acm, acmpca, documentdb, memorydb, mq, rds, secretsmanager

若要授予秘密讀取許可：

1. 將arn:aws:eks::aws:cluster-access-policy/AmazonEKSSecretReaderPolicy存取項目政策與功能的存取項目建立關聯
2. 將政策範圍限定在特定命名空間，其中 ACK 資源會參考秘密，或授予整個叢集的存取權

> **Important**
>
> 秘密讀取許可的範圍僅限於您在關聯存取項目政策時指定的命名空間。這可讓您限制 功能可存取的秘密。

kro 任意資源許可

根據預設，kro 可以監看和管理 ResourceGraphDefinitions (RGDs) 及其執行個體。若要管理 部署、服務或 ConfigMaps 等合成資源，請設定其他 Kubernetes 許可。

若要授予 kro 建立資源的許可：

選項 1：存取項目政策

將預先定義的存取項目政策，例如 `AmazonEKSAdminPolicy`或 `AmazonEKSEditPolicy` 與功能的存取項目建立關聯。

選項 2：Kubernetes RBAC

建立`ClusterRoleBinding`授予功能 Kubernetes 使用者必要許可的 ：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: kro-cluster-admin
subjects:
- kind: User
  name: arn:aws:sts::111122223333:assumed-role/my-kro-role/KRO
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: cluster-admin
  apiGroup: rbac.authorization.k8s.io
```

> **Note**
>
> kro 的 Kubernetes 使用者名稱遵循 模式： `arn:aws:sts::ACCOUNT_ID:assumed-role/ROLE_NAME/KRO`
> 工作階段名稱 `/KRO`（大寫） 是由 EKS kro 功能自動設定。

功能所需的 IAM 許可

kro (Kube Resource Orchestrator)

不需要 IAM 許可。您可以建立沒有連接政策的功能角色。 kro 只需要 Kubernetes RBAC 許可。

ACK (Kubernetes 的AWS 控制器)

需要許可來管理 ACK 將建立和管理 AWS 的資源。您應該根據您的需求，將許可範圍限定為特定服務、動作和資源。如需設定 ACK 許可的詳細資訊，包括使用 IAM 角色選取器的生產最佳實務，請參閱 the section called “設定 許可”。

Argo CD

預設不需要 IAM 許可。以下項目可能需要選用許可：

- AWS Secrets Manager：如果在 Secrets Manager 中存放 Git 儲存庫登入資料
- AWS CodeConnections：如果使用 CodeConnections for Git 儲存庫身分驗證
- Amazon ECR：如果使用 Amazon ECR 中以 OCI 格式存放的 Helm Chart

安全最佳實務

IAM 最低權限

僅授予您的 功能資源使用案例所需的許可。這並不表示您無法視需要將廣泛的管理許可授予您的 功能。在這種情況下，您應該適當地控管對這些資源的存取。

功能角色：

- ACK：可能的話，根據使用案例和需求，將 IAM 許可限制為您的團隊所需的特定 AWS 服務和資源
- Argo CD：限制對特定 Git 儲存庫和 Kubernetes 命名空間的存取
- kro：需要信任政策的功能角色，但不需要 IAM 許可 （ 僅使用叢集 RBAC)

範例：不是 `"Resource": "*"`，而是指定特定資源或資源群組的模式。

```
"Resource": [
  "arn:aws:s3:::my-app-*",
  "arn:aws:rds:us-west-2:111122223333:db:prod-*"
]
```

使用 IAM 條件金鑰進一步限制存取：

```
"Condition": {
  "StringEquals": {
    "aws:ResourceTag/Environment": "production"
  }
}
```

如需其他 IAM 組態資訊，請參閱每個功能的考量區段。

Argo CD 秘密的命名空間隔離

受管 Argo CD 功能可存取其設定命名空間中的所有 Kubernetes 秘密（預設：argocd)。若要維持最佳安全狀態，請遵循下列命名空間隔離實務：

- 在 Argo CD 命名空間中僅保留與 Argo CD 相關的秘密
- 避免將不相關的應用程式秘密存放在與 Argo CD 相同的命名空間
- 對 Argo CD 操作不需要的應用程式秘密使用單獨的命名空間

此隔離可確保 Argo CD 的秘密存取僅限於 Git 儲存庫身分驗證和其他 Argo CD 特定操作所需的登入資料。

Kubernetes RBAC

控制哪些使用者和服務帳戶可以建立和管理功能資源。最佳實務是在具有適當 RBAC 政策的專用命名空間中部署功能資源。

範例：使用 ACK 的 RBAC 角色，允許在app-team命名空間中管理 S3 儲存貯體資源：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: ack-s3-manager
  namespace: app-team
rules:
- apiGroups: ["s3.services.k8s.aws"]
  resources: ["buckets"]
  verbs: ["get", "list", "create", "update", "delete"]
```

稽核記錄

CloudTrail：所有 EKS 功能 API 操作（建立、更新、刪除）都會記錄到 AWS CloudTrail。

啟用 CloudTrail 記錄以追蹤：

- 建立或修改功能的人員
- 當功能組態變更時
- 使用哪些功能角色

網路存取和 VPC 端點

私有 Argo CD API 存取

您可以將一或多個 VPC 端點與託管 Argo CD 端點建立關聯，以限制對 Argo CD API 伺服器的存取。這可從 VPC 內啟用私有連線，而不會周遊公有網際網路。VPC 端點可讓您存取 Argo CD Web UI 和 Argo CD API（包括 CLI 存取）。

> **Note**
>
> 連線至託管 Argo CD API 端點的 VPC 端點（使用 eks-capabilities.*region*.amazonaws.com）不支援 VPC 端點政策。

部署至私有叢集

Argo CD 功能可以將應用程式部署到全私有 EKS 叢集，藉由消除對 VPC 互連或複雜聯網組態的需求，提供顯著的操作優勢。不過，在設計此架構時，請考慮 Argo CD 會從 Git 儲存庫提取組態（可能是公有的），並將其套用至您的私有叢集。

確保您：

- 針對敏感工作負載使用私有 Git 儲存庫
- 實作適當的 Git 儲存庫存取控制和身分驗證
- 在合併之前，透過提取請求來檢閱和核准變更
- 考慮使用 Argo CD 的同步視窗來控制何時可以進行部署
- 監控 Argo CD 稽核日誌是否有未經授權的組態變更

合規

EKS 功能受到完整管理，並具有 Amazon EKS 的合規認證。

如需最新的合規資訊，請參閱AWS 合規計劃範圍內的服務。

後續步驟

- the section called “設定 許可” - 設定 ACK 的 IAM 許可
- the section called “設定 許可” - 為 kro 設定 Kubernetes RBAC
- the section called “設定 許可” - 設定 Argo CD 的 Identity Center 整合
- the section called “故障診斷功能” - 對安全性和許可問題進行故障診斷

Amazon EKS 的 Identity and Access Management

AWS Identity and Access Management (IAM) 是一種 AWS 服務，可協助管理員安全地控制對 AWS 資源的存取。IAM 管理員可以控制驗證 (已登入) 和授權 (具有許可) 來使用 Amazon EKS 資源。IAM 是一項服務 AWS ，您可以免費使用。

目標對象

使用 AWS Identity and Access Management (IAM) 的方式會有所不同，取決於您在 Amazon EKS 中執行的工作。

服務使用者 – 如果您使用 Amazon EKS 執行任務，您的管理員會為您提供您需要的憑證和許可。隨著您為了執行作業而使用的 Amazon EKS 功能數量變多，您可能會需要額外的許可。了解存取許可的管理方式可協助您向管理員請求正確的許可。如果您無法存取 Amazon EKS 中的功能，請參閱 the section called “疑難排解”。

服務管理員 – 若您在公司負責管理 Amazon EKS 資源，您應該具備服務使用的完整存取權限。您的任務是判斷服務使用者應存取的 Amazon EKS 功能和資源。接著，您必須將請求提交給您的 IAM 管理員，來變更您服務使用者的許可。檢閱此頁面上的資訊，了解 IAM 的基本概念。若要進一步了解貴公司可搭配 Amazon EKS 使用 IAM 的方式，請參閱 the section called “Amazon EKS 和 IAM”。

IAM 管理員 – 如果您是 IAM 管理員，建議您掌握如何撰寫政策以管理 Amazon EKS 存取權的詳細資訊。若要檢視您可以在 IAM 中使用 Amazon EKS 身分型政策範例，請參閱 the section called “身分型政策”。

使用身分驗證

身分驗證是您 AWS 使用身分憑證登入 的方式。您必須以 AWS 帳戶根使用者、IAM 使用者或擔任 IAM 角色身分進行身分驗證 （ 登入 AWS)。

您可以使用透過身分來源提供的登入資料，以聯合身分 AWS 的形式登入 。 AWS IAM Identity Center (IAM Identity Center) 使用者、您公司的單一登入身分驗證，以及您的 Google 或 Facebook 登入資料，都是聯合身分的範例。您以聯合身分登入時，您的管理員先前已設定使用 IAM 角色的聯合身分。當您使用聯合 AWS 身分存取 時，您會間接擔任角色。

根據您的使用者類型，您可以登入 AWS 管理主控台 或 AWS 存取入口網站。如需登入的詳細資訊 AWS，請參閱 AWS 登入使用者指南中的如何登入 AWS 您的帳戶。

如果您以 AWS 程式設計方式存取 ， AWS 會提供軟體開發套件 (SDK) 和命令列界面 (CLI)，以使用您的登入資料以密碼編譯方式簽署您的請求。如果您不使用 AWS 工具，則必須自行簽署請求。如需使用建議方法自行簽署請求的詳細資訊，請參閱《IAM 使用者指南》中的簽署 AWS API 請求。

無論您使用何種身分驗證方法，您可能都需要提供額外的安全性資訊。例如， AWS 建議您使用多重要素驗證 (MFA) 來提高帳戶的安全性。若要進一步了解，請參閱《 AWS IAM Identity Center 使用者指南》中的多重要素驗證和《IAM 使用者指南》中的在 中使用多重要素驗證 (MFA) AWS。

AWS 帳戶根使用者

當您建立 AWS 帳戶時，您會從一個登入身分開始，該身分可完整存取帳戶中的所有 AWS 服務和資源。此身分稱為 AWS 帳戶根使用者，可透過使用您用來建立帳戶的電子郵件地址和密碼登入來存取。強烈建議您不要以根使用者處理日常作業。保護您的根使用者憑證，並將其用來執行只能由根使用者執行的任務。如需這些任務的完整清單，了解需以根使用者登入的任務，請參閱《IAM 使用者指南》中的需要根使用者憑證的任務。

IAM 使用者和群組

IAM 使用者是您 AWS 帳戶中的身分，具有單一人員或應用程式的特定許可。建議您盡可能依賴臨時憑證，而不是擁有建立長期憑證 (例如密碼和存取金鑰) 的 IAM 使用者。但是如果特定使用案例需要擁有長期憑證的 IAM 使用者，建議您輪換存取金鑰。如需更多資訊，請參閱《IAM 使用者指南》中的為需要長期憑證的使用案例定期輪換存取金鑰。

IAM 群組是一種指定 IAM 使用者集合的身分。您無法以群組身分登入。您可以使用群組來一次為多名使用者指定許可。群組可讓管理大量使用者許可的程序變得更為容易。例如，您可以擁有一個名為 IAMAdmins 的群組，並給予該群組管理 IAM 資源的許可。

使用者與角色不同。使用者只會與單一人員或應用程式建立關聯，但角色的目的是在由任何需要它的人員取得。使用者擁有永久的長期憑證，但角色僅提供暫時憑證。如需進一步了解，請參閱IAM 使用者指南中的建立 IAM 使用者 (而非角色) 的時機。

IAM 角色

IAM 角色是您 AWS 帳戶中具有特定許可的身分。它類似 IAM 使用者，但不與特定的人員相關聯。您可以透過切換角色暫時在 中擔任 AWS 管理主控台 IAM 角色。 https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_use_switch-role-console.html您可以透過呼叫 CLI AWS 或 AWS API 操作或使用自訂 URL 來擔任角色。如需使用角色的方法詳細資訊，請參閱 IAM 使用者指南中的使用 IAM 角色。

使用暫時憑證的 IAM 角色在下列情況中非常有用：

- 聯合身分使用者存取 – 如需向聯合身分指派許可，請建立角色，並為角色定義許可。當聯合身分進行身分驗證時，該身分會與角色建立關聯，並獲授予由角色定義的許可。如需有關聯合角色的相關資訊，請參閱 IAM 使用者指南中的為第三方身分提供者建立角色。如果您使用 IAM Identity Center，則需要設定許可集。為控制身分驗證後可以存取的內容，IAM Identity Center 將許可集與 IAM 中的角色相關聯。如需許可集的資訊，請參閱《 AWS IAM Identity Center 使用者指南》中的許可集。
- 暫時 IAM 使用者許可 – IAM 使用者或角色可以擔任 IAM 角色來暫時針對特定任務採用不同的許可。
- 跨帳戶存取權：您可以使用 IAM 角色，允許不同帳戶中的某人 (信任的主體) 存取您帳戶的資源。角色是授予跨帳戶存取權的主要方式。不過，對於某些 AWS 服務，您可以將政策直接連接到資源（而不是使用角色做為代理）。如需了解使用角色和資源型政策進行跨帳戶存取之間的差異，請參閱《IAM 使用者指南》中的 IAM 中的跨帳戶資源存取。
- 跨服務存取 – 有些 AWS 服務使用其他服務中的功能 AWS 。例如，當您在服務中進行呼叫時，該服務通常會在 Amazon EC2 中執行應用程式或將物件儲存在 Amazon Simple Storage Service (Amazon S3) 中。服務可能會使用呼叫主體的許可、使用服務角色或使用服務連結角色來執行此作業。
  - 轉送存取工作階段 (FAS) – 當您使用 IAM 使用者或角色在其中執行動作時 AWS，您會被視為委託人。使用某些服務時，您可能會執行某個動作，進而在不同服務中啟動另一個動作。FAS 使用呼叫 AWS 服務的委託人許可，結合請求 AWS 服務，向下游服務提出請求。只有當服務收到需要與其他 AWS 服務或資源互動才能完成的請求時，才會提出 FAS 請求。在此情況下，您必須具有執行這兩個動作的許可。如需提出 FAS 請求時的政策詳細資訊，請參閱轉發存取工作階段。
  - 服務角色 – 服務角色是服務擔任的 IAM 角色，可代表您執行動作。IAM 管理員可以從 IAM 內建立、修改和刪除服務角色。如需詳細資訊，請參閱《IAM 使用者指南》中的建立角色以將許可委派給 AWS 服務。
  - 服務連結角色 – 服務連結角色是連結至 服務的一種 AWS 服務角色。服務可以擔任代表您執行動作的角色。服務連結角色會顯示在您的帳戶中 AWS ，並由服務擁有。IAM 管理員可以檢視，但不能編輯服務連結角色的許可。

- 在 Amazon EC2 上執行的應用程式 – 您可以使用 IAM 角色來管理在 EC2 執行個體上執行之應用程式的臨時登入資料，以及提出 AWS CLI 或 AWS API 請求。這是在 EC2 執行個體內儲存存取金鑰的較好方式。若要將 AWS 角色指派給 EC2 執行個體並將其提供給其所有應用程式，您可以建立連接至執行個體的執行個體描述檔。執行個體設定檔包含該角色，並且可讓 EC2 執行個體上執行的程式取得暫時憑證。如需詳細資訊，請參閱 IAM 使用者指南中的利用 IAM 角色來授予許可給 Amazon EC2 執行個體上執行的應用程式。

如需了解是否要使用 IAM 角色或 IAM 使用者，請參閱 IAM 使用者指南中的建立 IAM 角色 (而非使用者) 的時機。

使用政策管理存取權

您可以透過建立政策並將其連接到身分或資源 AWS 來控制 AWS 中的存取。政策是 中的物件， AWS 當與身分或資源建立關聯時， 會定義其許可。當委託人 （使用者、根使用者或角色工作階段） 發出請求時， 會 AWS 評估這些政策。政策中的許可決定是否允許或拒絕請求。大多數政策會以 JSON 文件 AWS 形式存放在 中。如需 JSON 政策文件結構和內容的詳細資訊，請參閱《IAM 使用者指南》中的 JSON 政策概觀。

管理員可以使用 AWS JSON 政策來指定誰可以存取內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

預設情況下，使用者和角色沒有許可。若要授予使用者對其所需資源執行動作的許可，IAM 管理員可以建立 IAM 政策。然後，管理員可以將 IAM 政策新增至角色，使用者便能擔任這些角色。

IAM 政策定義該動作的許可，無論您使用何種方法來執行操作。例如，假設您有一個允許 `iam:GetRole` 動作的政策。具有該政策的使用者可以從 AWS 管理主控台、 CLI 或 AWS API AWS 取得角色資訊。

身分型政策

身分型政策是可以附加到身分 (例如 IAM 使用者、使用者群組或角色) 的 JSON 許可政策文件。這些政策可控制身分在何種條件下能對哪些資源執行哪些動作。若要了解如何建立身分類型政策，請參閱 IAM 使用者指南中的建立 IAM 政策。

身分型政策可進一步分類成內嵌政策或受管政策。內嵌政策會直接嵌入單一使用者、群組或角色。受管政策是獨立的政策，您可以連接到 AWS 帳戶中的多個使用者、群組和角色。受管政策包括 AWS 受管政策和客戶受管政策。如需了解如何在受管政策及內嵌政策間選擇，請參閱 IAM 使用者指南中的在受管政策和內嵌政策間選擇。

資源型政策

資源型政策是附加到資源的 JSON 政策文件。資源型政策的最常見範例是 IAM 角色信任政策和 Amazon S3 儲存貯體政策。在支援資源型政策的服務中，服務管理員可以使用它們來控制對特定資源的存取權限。對於附加政策的資源，政策會定義指定的主體可以對該資源執行的動作以及在何種條件下執行的動作。您必須在資源型政策中指定主體。委託人可以包括帳戶、使用者、角色、聯合身分使用者 AWS 或服務。

資源型政策是位於該服務中的內嵌政策。您無法在以資源為基礎的政策中使用來自 IAM 的 AWS 受管政策。

存取控制清單 (ACL)

存取控制清單 (ACL) 可控制哪些主體 (帳戶成員、使用者或角色) 擁有存取某資源的許可。ACL 類似於資源型政策，但它們不使用 JSON 政策文件格式。

Amazon S3、 AWS WAF 和 Amazon VPC 是支援 ACLs的服務範例。如需進一步了解 ACL，請參閱《Amazon Simple Storage Service 開發人員指南》中的存取控制清單 (ACL) 概觀。

其他政策類型

AWS 支援其他較不常見的政策類型。這些政策類型可設定較常見政策類型授予您的最大許可。

- 許可界限 – 許可範圍是一種進階功能，可供您設定身分型政策能授予 IAM 實體 (IAM 使用者或角色) 的最大許可。您可以為實體設定許可界限。所產生的許可會是實體的身分類型政策和其許可界限的交集。會在 `Principal` 欄位中指定使用者或角色的資源型政策則不會受到許可界限限制。所有這類政策中的明確拒絕都會覆寫該允許。如需許可界限的詳細資訊，請參閱《IAM 使用者指南》中的 IAM 實體許可界限。
- 服務控制政策 (SCPs) – SCPs是 JSON 政策，可指定 AWS Organizations 中組織或組織單位 (OU) 的最大許可。 AWS Organizations 是一項服務，用於分組和集中管理您企業擁有的多個 AWS 帳戶。若您啟用組織中的所有功能，您可以將服務控制政策 (SCP) 套用到任何或所有帳戶。SCP 會限制成員帳戶中實體的許可，包括每個 AWS 帳戶根使用者。如需 Organizations 和 SCPs的詳細資訊，請參閱 AWS 《 Organizations 使用者指南》中的服務控制政策。
- 工作階段政策 – 工作階段政策是一種進階政策，您可以在透過撰寫程式的方式建立角色或聯合身分使用者的暫時工作階段時，做為參數傳遞。所產生工作階段的許可會是使用者或角色的身分型政策和工作階段政策的交集。許可也可以來自資源型政策。所有這類政策中的明確拒絕都會覆寫該允許。如需詳細資訊，請參《IAM 使用者指南》中的工作階段政策。

多種政策類型

當多種類型的政策適用於請求時，產生的許可會更複雜而無法理解。若要了解如何 AWS 在涉及多個政策類型時決定是否允許請求，請參閱《IAM 使用者指南》中的政策評估邏輯。

Amazon EKS 如何搭配 IAM 運作

在您使用 IAM 管理對 Amazon EKS 的存取權之前，您應該了解哪些 IAM 功能可以與 Amazon EKS 搭配使用。若要全面了解 Amazon EKS 和其他 AWS 服務如何與 IAM 搭配使用，請參閱《IAM 使用者指南》中的AWS 與 IAM 搭配使用的 服務。

主題

- Amazon EKS 身分型政策
- Amazon EKS 資源型政策
- 以 Amazon EKS 標籤為基礎的授權
- Amazon EKS IAM 角色

Amazon EKS 身分型政策

使用 IAM 身分型政策，您可以指定允許或拒絕的動作和資源，以及在何種條件下允許或拒絕動作。Amazon EKS 支援特定動作、資源和條件索引鍵。若要了解您在 JSON 政策中使用的所有元素，請參閱《IAM 使用者指南》中的 IAM JSON 政策元素參考。

動作

管理員可以使用 AWS JSON 政策來指定誰可以存取內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

JSON 政策的 `Action` 元素描述您可以用來允許或拒絕政策中存取的動作。政策動作通常具有與相關聯 AWS API 操作相同的名稱。有一些例外狀況，例如沒有相符的 API 操作的僅限許可動作。也有一些作業需要政策中的多個動作。這些額外的動作稱為相依動作。

政策會使用動作來授予執行相關聯動作的許可。

Amazon EKS 中的政策動作會在動作之前使用下列字首：`eks:`。例如，若要授予某人取得 Amazon EKS 叢集描述性資訊的許可，請在其政策中加入 `DescribeCluster` 動作。政策陳述式必須包含 `Action` 或 `NotAction` 元素。

若要在單一陳述式中指定多個動作，請用逗號分隔，如下所示：

```
"Action": ["eks:action1", "eks:action2"]
```

您也可以使用萬用字元 (*) 來指定多個動作。例如，若要指定開頭是 Describe 文字的所有動作，請包含以下動作：

```
"Action": "eks:Describe*"
```

若要查看 Amazon EKS 動作的清單，請參閱服務授權參考中的 Amazon Elastic Kubernetes Service 定義的動作。

Resources

管理員可以使用 AWS JSON 政策來指定誰可以存取內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

Resource JSON 政策元素可指定要套用動作的物件。陳述式必須包含 Resource 或 NotResource 元素。最佳實務是使用其 Amazon Resource Name (ARN) 來指定資源。您可以針對支援特定資源類型的動作 (稱為資源層級許可) 來這麼做。

對於不支援資源層級許可的動作 (例如列出操作)，請使用萬用字元 (*) 來表示陳述式適用於所有資源。

```
"Resource": "*"
```

Amazon EKS 叢集資源有以下 ARN。

```
 arn:aws: eks:region-code:account-id:cluster/cluster-name
```

如需 ARNs 格式的詳細資訊，請參閱 Amazon 資源名稱 (ARNs) AWS 和服務命名空間。

例如，若要在您的陳述式中指定具有 *my-cluster* 名稱的叢集，請使用以下 ARN：

```
"Resource": "arn:aws: eks:region-code:111122223333:cluster/my-cluster"
```

若要指定屬於特定帳戶和 AWS 區域的所有叢集，請使用萬用字元 (*)：

```
"Resource": "arn:aws: eks:region-code:111122223333:cluster/*"
```

有些 Amazon EKS 動作無法對特定資源執行，例如用來建立資源的動作。在這些情況下，您必須使用萬用字元 (*)。

```
"Resource": "*"
```

若要查看 Amazon EKS 資源類型及其 ARN 的詳細資訊，請參閱服務授權參考中的 Amazon Elastic Kubernetes Service 定義的資源。若要了解您可以使用哪些動作指定每個資源的 ARN，請參閱 Amazon Elastic Kubernetes Service 定義的動作。

條件索引鍵

Amazon EKS 會定義自己的一組條件索引鍵，也支援使用一些全域條件金鑰。若要查看所有 AWS 全域條件金鑰，請參閱《IAM 使用者指南》中的AWS 全域條件內容金鑰。

將 OpenID Connect 提供商關聯到您的叢集時，您可以設定條件索引鍵。如需詳細資訊，請參閱the section called “範例 IAM 政策”。

所有 Amazon EC2 操作都支援 `aws:RequestedRegion` 和 `ec2:Region` 條件索引鍵。如需詳細資訊，請參閱範例：限制對特定 AWS 區域的存取。

如需 Amazon EKS 條件金鑰的清單，請參閱服務授權參考中的 Amazon Elastic Kubernetes Service 定義的條件。若要了解您可以搭配哪些動作和資源使用條件索引鍵，請參閱 Amazon Elastic Kubernetes Service 定義的動作。

範例

若要檢視 Amazon EKS 身分型政策的範例，請參閱 the section called “身分型政策”。

當您建立 Amazon EKS 叢集時，建立該叢集的 IAM 主體會由 Amazon EKS 控制平面中叢集的角色型存取控制 (RBAC) 組態來自動授予 `system:masters` 許可。此主體不會出現在任何可見的組態中，因此請務必記下最初建立叢集的主體。若要賦予其他 IAM 主體與叢集互動的能力，您必須編輯 Kubernetes 中的 `aws-auth ConfigMap` 並用 `group` 的名稱建立一個 Kubernetes `rolebinding` 或 `clusterrolebinding`，其名稱是您在 `aws-auth ConfigMap` 中指定的名稱。

如需使用 ConfigMap 代理程式組態檔案的詳細資訊，請參閱 the section called “Kubernetes API 存取”。

Amazon EKS 資源型政策

Amazon EKS 不支援資源型政策。

以 Amazon EKS 標籤為基礎的授權

您可以將標籤連接至 Amazon EKS 資源，或是在請求中將標籤傳遞至 Amazon EKS。若要根據標籤控制存取，請使用 `aws:ResourceTag/key-name` 、`aws:RequestTag/key-name` 或 `aws:TagKeys` 條件索引鍵，在政策的條件元素中，提供標籤資訊。如需標記 Amazon EKS 資源的詳細資訊，請參閱 the section called “標記您的 資源”。如需您在條件索引鍵下可使用哪些標籤動作的詳細資訊，請參閱《服務授權參考》中的 Amazon EKS 定義的動作。

Amazon EKS IAM 角色

IAM 角色是您 AWS 帳戶中具有特定許可的實體。

將暫時憑證與 Amazon EKS 搭配使用

您可以搭配聯合使用暫時憑證、擔任 IAM 角色，或是擔任跨帳戶角色。您可以透過呼叫 AssumeRole 或 GetFederationToken 等 AWS STS API 操作來取得臨時安全登入資料。

Amazon EKS 支援使用臨時憑證。

服務連結角色

服務連結角色可讓 AWS 服務存取其他服務中的資源，以代表您完成 動作。服務連結角色會顯示在您的 IAM 帳戶中，並由該服務所擁有。管理員可以檢視，但不能編輯服務連結角色的許可。

Amazon EKS 支援服務連結角色。如需建立或管理 Amazon EKS 服務連結角色的詳細資訊，請參閱 the section called “服務連結角色”。

服務角色

此功能可讓服務代表您擔任服務角色。此角色可讓服務存取其他服務中的資源，以代表您完成動作。服務角色會出現在您的 IAM 帳戶中，且由該帳戶所擁有。這表示 IAM 管理員可以變更此角色的許可。不過，這樣可能會破壞此服務的功能。

Amazon EKS 支援服務角色。如需詳細資訊，請參閱the section called “叢集 IAM 角色”及the section called “節點 IAM 角色”。

在 Amazon EKS 中選擇 IAM 角色

當您在 Amazon EKS 中建立叢集資源時，您必須選擇一個角色，以允許 Amazon EKS 代表您存取數個其他 AWS 資源。如果您之前已建立服務角色，則 Amazon EKS 會提供一個角色清單供您選擇。請務

必選擇連接 Amazon EKS 受管政策的角色。如需詳細資訊，請參閱the section called “檢查現有的叢集角色”及the section called “檢查現有的節點角色”。

Amazon EKS 身分型政策範例

根據預設，IAM 使用者和角色不具備建立或修改 Amazon EKS 資源的許可。他們也無法使用 、 AWS CLI AWS 管理主控台或 AWS API 執行任務。IAM 管理員必須建立 IAM 政策，授予使用者和角色在指定資源上執行特定 API 作業的所需許可。管理員接著必須將這些政策連接至需要這些許可的 IAM 使用者或群組。

若要了解如何使用這些範例 JSON 政策文件建立 IAM 身分型政策，請參閱《IAM 使用者指南》中的在 JSON 標籤上建立政策。

當您建立 Amazon EKS 叢集時，建立該叢集的 IAM 主體會由 Amazon EKS 控制平面中叢集的角色型存取控制 (RBAC) 組態來自動授予 `system:masters` 許可。此主體不會出現在任何可見的組態中，因此請務必記下最初建立叢集的主體。若要賦予其他 IAM 主體與叢集互動的能力，您必須編輯 Kubernetes 中的 `aws-auth ConfigMap` 並用 `group` 的名稱建立一個 Kubernetes `rolebinding` 或 `clusterrolebinding`，其名稱是您在 `aws-auth ConfigMap` 中指定的名稱。

如需使用 ConfigMap 代理程式組態檔案的詳細資訊，請參閱 the section called “Kubernetes API 存取”。

主題

- 政策最佳實務
- 使用 Amazon EKS 主控台
- 允許 IAM 使用者檢視他們自己的許可
- 在 AWS 雲端上建立 Kubernetes 叢集
- 在 Outpost 上建立本機 Kubernetes 叢集
- 更新 Kubernetes 叢集
- 所有叢集的清單或描述
- 取消叢集更新 （ 轉返 ）

政策最佳實務

身分型政策會判斷您帳戶中的某個人員是否可以建立、存取或刪除 Amazon EKS 資源。這些動作可能會對您的帳戶產生成本 AWS 。當您建立或編輯身分型政策時，請遵循下列準則及建議事項：

- 開始使用 AWS 受管政策並邁向最低權限許可 – 若要開始將許可授予您的使用者和工作負載，請使用將許可授予許多常見使用案例的 AWS 受管政策。它們可在您的帳戶中使用 AWS 。我們建議您定義特定於使用案例 AWS 的客戶受管政策，以進一步減少許可。如需更多資訊，請參閱《IAM 使用者指南》中的 AWS 受管政策或任務職能的AWS 受管政策。
- 套用最低權限許可 – 設定 IAM 政策的許可時，請僅授予執行任務所需的許可。為實現此目的，您可以定義在特定條件下可以對特定資源採取的動作，這也稱為最低權限許可。如需使用 IAM 套用許可的更多相關資訊，請參閱《IAM 使用者指南》中的 IAM 中的政策和許可。
- 使用 IAM 政策中的條件進一步限制存取權 – 您可以將條件新增至政策，以限制動作和資源的存取。例如，您可以撰寫政策條件，指定必須使用 SSL 傳送所有請求。如果透過 AWS CloudFormation 等特定服務使用 AWS 服務動作，您也可以使用條件來授予服務動作的存取權。如需詳細資訊，請參閱《IAM 使用者指南》中的 IAM JSON 政策元素：條件。
- 使用 IAM Access Analyzer 驗證 IAM 政策，確保許可安全且可正常運作 – IAM Access Analyzer 驗證新政策和現有政策，確保這些政策遵從 IAM 政策語言 (JSON) 和 IAM 最佳實務。IAM Access Analyzer 提供 100 多項政策檢查及切實可行的建議，可協助您撰寫安全且實用的政策。如需更多資訊，請參閱 IAM 使用者指南中的 IAM Access Analyzer 政策驗證。
- 需要多重要素驗證 (MFA) – 如果您的案例需要 IAM 使用者或 AWS 帳戶中的根使用者，請開啟 MFA 以提高安全性。如需在呼叫 API 操作時請求 MFA，請將 MFA 條件新增至您的政策。如需更多資訊，請參閱 IAM 使用者指南中的設定 MFA 保護的 API 存取。

如需 IAM 中最佳實務的相關資訊，請參閱 IAM 使用者指南中的 IAM 安全最佳實務。

使用 Amazon EKS 主控台

若要存取 Amazon EKS 主控台，IAM 主體必須擁有最基本的一組許可。這些許可允許委託人列出和檢視您 AWS 帳戶中 Amazon EKS 資源的詳細資訊。如果您建立比最低必要許可更嚴格的身分型政策，則對於具有該政策的主體而言，主控台將無法如預期運作。

為了確保您的 IAM 主體仍然可以使用 Amazon EKS 主控台，建立擁有唯一名稱的政策，例如 `AmazonEKSAdminPolicy`。將政策連接至主體。如需詳細資訊，請參閱 IAM 使用者指南中的新增和移除 IAM 身分許可。

> **Important**
>
> 下列範例政策可讓主體檢視主控台內的組態標籤上的資訊。若要檢視 中概觀和資源索引標籤的相關資訊 AWS 管理主控台，委託人也需要 Kubernetes 許可。如需詳細資訊，請參閱the section called “所需的許可”。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "eks:*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "eks.amazonaws.com"
                }
            }
        }
    ]
}
```

對於僅呼叫 CLI 或 AWS API AWS 的主體，您不需要允許最低主控台許可。反之，只需允許存取符合您嘗試執行之 API 作業的動作就可以了。

允許 IAM 使用者檢視他們自己的許可

此範例會示範如何建立政策，允許 IAM 使用者檢視連接到他們使用者身分的內嵌及受管政策。此政策包含在主控台或使用 CLI 或 AWS API AWS 以程式設計方式完成此動作的許可。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
```

```
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

在 AWS 雲端上建立 Kubernetes 叢集

此範例政策包含用於在 *us-west-2* AWS 區域中建立名為 *my-cluster* 的 Amazon EKS 叢集所需的最低許可。您可以將 AWS 區域取代為您要建立叢集 AWS 的區域。如果您看到警告，指出政策中的動作不支援資源層級許可，並要求您在 **All resources**中選擇 AWS 管理主控台，則可以安全地忽略它。如果您的帳戶已具有 *AWSServiceRoleForAmazonEKS* 角色，您可以移除來自政策的 `iam:CreateServiceLinkedRole` 動作。如果您曾在帳戶中建立 Amazon EKS 叢集，則此角色已存在，除非您已將其刪除。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "eks:CreateCluster",
            "Resource": "arn:aws:eks:us-west-2:111122223333:cluster/my-cluster"
        },
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
```

```
            "Resource": "arn:aws:iam::111122223333:role/aws-service-role/
eks.amazonaws.com/AWSServiceRoleForAmazonEKS",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "iam:AWSServiceName": "eks"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::111122223333:role/cluster-role-name"
        }
    ]
}
```

在 Outpost 上建立本機 Kubernetes 叢集

此範例政策包含用於在 *us-west-2* AWS 區域的 Outpost 上建立名為 *my-cluster* 的 Amazon EKS 本機叢集所需的最低許可。您可以將 AWS 區域取代為您要建立叢集 AWS 的區域。如果您看到警告，指出政策中的動作不支援資源層級許可，並要求您在 **All resources**中選擇 AWS 管理主控台，則可以安全地忽略它。如果您的帳戶已具有 AWSServiceRoleForAmazonEKSLocalOutpost 角色，您可以移除來自政策的 iam:CreateServiceLinkedRole 動作。如果您曾在帳戶中建立位於 Outpost 上的 Amazon EKS 本機叢集，則此角色已存在，除非您已將其刪除。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "eks:CreateCluster",
            "Resource": "arn:aws:eks:us-west-2:111122223333:cluster/my-cluster"
        },
        {
            "Action": [
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcs",
                "iam:GetRole"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
```

```
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::111122223333:role/aws-service-role/outposts.eks-
local.amazonaws.com/AWSServiceRoleForAmazonEKSLocalOutpost"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole",
                "iam:ListAttachedRolePolicies"
            ],
            "Resource": "arn:aws:iam::111122223333:role/cluster-role-name"
        },
        {
            "Action": [
                "iam:CreateInstanceProfile",
                "iam:TagInstanceProfile",
                "iam:AddRoleToInstanceProfile",
                "iam:GetInstanceProfile",
                "iam:DeleteInstanceProfile",
                "iam:RemoveRoleFromInstanceProfile"
            ],
            "Resource": "arn:aws:iam::*:instance-profile/eks-local-*",
            "Effect": "Allow"
        }
    ]
}
```

更新 Kubernetes 叢集

此範例政策包含更新 us*-west-2Region ### my-cluster* 的叢集所需的最低許可。 AWS

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "eks:UpdateClusterVersion",
            "Resource": "arn:aws:eks:us-west-2:111122223333:cluster/my-cluster"
        }
    ]
}
```

所有叢集的清單或描述

此範例政策包含列出及說明帳戶中所有叢集所需的最低許可。IAM 主體必須能夠列出和描述叢集，才能使用 AWS CLI `update-kubeconfig` 命令。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "eks:DescribeCluster",
                "eks:ListClusters"
            ],
            "Resource": "*"
        }
    ]
}
```

取消叢集更新 （轉返）

若要允許使用者取消進行中的回復操作，請授予 `eks:CancelUpdate`動作。您可以使用資源 ARN 或以標籤為基礎的條件來限定此範圍。

以資源 ARN 為基礎的：

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": "eks:CancelUpdate",
    "Resource": "arn:aws:eks:us-west-2:111122223333:cluster/my-cluster"
  }]
}
```

標籤型：

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
```

```
    "Action": "eks:CancelUpdate",
    "Resource": "*",
    "Condition": {
      "StringEquals": {
        "aws:ResourceTag/environment": "development"
      }
    }
  }]
}
```

如需完整叢集轉返程序的詳細資訊，請參閱 the section called “復原 Kubernetes 版本”。

使用 Amazon EKS 的服務連結角色

Amazon Elastic Kubernetes Service 可使用 AWS Identity and Access Management (IAM) 服務連結角色。服務連結角色是直接連結至 Amazon EKS 的一種特殊 IAM 角色類型。服務連結角色由 Amazon EKS 預先定義，且內含該服務代您呼叫其他 AWS 服務所需的所有許可。

主題

- 使用 Amazon EKS 叢集的角色
- 使用 Amazon EKS 節點群組的角色
- 使用 Amazon EKS Fargate 描述檔的角色
- 使用角色將 Kubernetes 叢集連線至 Amazon EKS
- 使用位於 Outpost 上的 Amazon EKS 本機叢集的角色
- 使用 Amazon EKS 儀表板的角色

使用 Amazon EKS 叢集的角色

Amazon Elastic Kubernetes Service 使用 AWS Identity and Access Management (IAM) 服務連結角色。服務連結角色是直接連結至 Amazon EKS 的一種特殊 IAM 角色類型。服務連結角色是由 Amazon EKS 預先定義，並包含該服務代表您呼叫其他 AWS 服務所需的所有許可。

服務連結角色可讓設定 Amazon EKS 更為簡單，因為您不必手動新增必要的許可。Amazon EKS 定義其服務連結角色的許可，除非另有定義，否則僅有 Amazon EKS 可以擔任其角色。定義的許可包括信任政策和許可政策，且該許可政策無法附加至其他 IAM 實體。

您必須先刪除角色的相關資源，才能刪除服務連結角色。如此可保護您的 Amazon EKS 資源，避免您不小心移除資源的存取許可。

如需關於支援服務連結角色的其他服務的資訊，請參閱可搭配 IAM 運作的AWS 服務，尋找 Service-Linked Role (服務連結角色) 欄中顯示為 Yes (是) 的服務。選擇具有連結的是，以檢視該服務的服務連結角色文件。

Amazon EKS 的服務連結角色許可

Amazon EKS 會使用名為 AWSServiceRoleForAmazonEKS 的服務連結角色。該角色可讓 Amazon EKS 管理您帳戶中的叢集。連接的政策允許角色管理下列資源：網路介面、安全群組、記錄和 VPC。

> **Note**
>
> AWSServiceRoleForAmazonEKS 服務連結角色並非叢集建立所需的角色。如需詳細資訊，請參閱the section called “叢集 IAM 角色”。

AWSServiceRoleForAmazonEKS 服務連結角色信任下列服務擔任角色：

- eks.amazonaws.com

此角色許可政策允許 Amazon EKS 對指定資源完成下列動作：

- AmazonEKSServiceRolePolicy

您必須設定許可，IAM 實體 (如使用者、群組或角色) 才可建立、編輯或刪除服務連結角色。如需詳細資訊，請參閱《IAM 使用者指南》中的服務連結角色許可。

建立 Amazon EKS 的服務連結角色

您不需要手動建立一個服務連結角色。當您在 AWS 管理主控台、 CLI 或 AWS API AWS 中建立叢集時，Amazon EKS 會為您建立服務連結角色。

若您刪除此服務連結角色，之後需要再次建立，您可以在帳戶中使用相同程序重新建立角色。當您建立叢集時，Amazon EKS 會再次為您建立服務連結角色。

編輯 Amazon EKS 的服務連結角色

Amazon EKS 不允許您編輯 AWSServiceRoleForAmazonEKS 服務連結角色。因為有各種實體可能會參考服務連結角色，所以您無法在建立角色之後變更角色名稱。然而，您可使用 IAM 來編輯角色描述。如需詳細資訊，請參閱《IAM 使用者指南》中的編輯服務連結角色。

刪除 Amazon EKS 的服務連結角色

若您不再使用需要服務連結角色的功能或服務，我們建議您刪除該角色。如此一來，您就沒有未主動監控或維護的未使用實體。然而，務必清除您的服務連結角色，之後才能以手動方式將其刪除。

清除服務連結角色

在您使用 IAM 刪除服務連結角色之前，您必須先刪除該角色所使用的任何資源。

> **Note**
>
> 若 Amazon EKS 服務在您試圖刪除資源時正在使用該角色，刪除可能會失敗。若此情況發生，請等待數分鐘後並再次嘗試操作。

1. 開啟 Amazon EKS 主控台。
2. 在左側導覽窗格中選擇叢集。
3. 如果您的叢集具有任何節點群組或 Fargate 描述檔，則必須先將其刪除，才能刪除叢集。如需詳細資訊，請參閱 the section called “刪除” 及 the section called “刪除設定檔”。
4. 在 Clusters (叢集) 頁面上，選擇您要刪除的叢集，然後選擇 Delete (刪除)。
5. 在刪除確認視窗中輸入叢集的名稱，然後選擇 Delete (刪除)。
6. 重複對您帳戶中的任何其他叢集執行此程序。等候所有刪除作業完成。

手動刪除服務連結角色

使用 IAM 主控台、CLI AWS 或 AWS API 來刪除`AWSServiceRoleForAmazonEKS`服務連結角色。如需詳細資訊，請參閱《IAM 使用者指南》中的刪除服務連結角色。

Amazon EKS 服務連結角色的支援區域

Amazon EKS 在所有提供服務的區域中支援使用服務連結角色。如需詳細資訊，請參閱 Amazon EKS endpoints and quotas (Amazon EKS 端點和配額)。

使用 Amazon EKS 節點群組的角色

Amazon EKS 使用 AWS Identity and Access Management (IAM) 服務連結角色。服務連結角色是直接連結至 Amazon EKS 的一種特殊 IAM 角色類型。服務連結角色是由 Amazon EKS 預先定義，並包含該服務代表您呼叫其他 AWS 服務所需的所有許可。

服務連結角色可讓設定 Amazon EKS 更為簡單，因為您不必手動新增必要的許可。Amazon EKS 定義其服務連結角色的許可，除非另有定義，否則僅有 Amazon EKS 可以擔任其角色。定義的許可包括信任政策和許可政策，且該許可政策無法附加至其他 IAM 實體。

您必須先刪除角色的相關資源，才能刪除服務連結角色。如此可保護您的 Amazon EKS 資源，避免您不小心移除資源的存取許可。

如需關於支援服務連結角色的其他服務的資訊，請參閱可搭配 IAM 運作的AWS 服務，尋找 Service-Linked Role (服務連結角色) 欄中顯示為 Yes (是) 的服務。選擇具有連結的是，以檢視該服務的服務連結角色文件。

Amazon EKS 的服務連結角色許可

Amazon EKS 會使用名為 AWSServiceRoleForAmazonEKSNodegroup 的服務連結角色。該角色可讓 Amazon EKS 管理您帳戶中的節點群組。連接的 AWSServiceRoleForAmazonEKSNodegroup 政策允許角色管理下列資源：Auto Scaling 群組、安全群組、啟動範本和 IAM 執行個體設定檔。如需詳細資訊，請參閱the section called “AWS 受管政策：AWSServiceRoleForAmazonEKSNodegroup”。

AWSServiceRoleForAmazonEKSNodegroup 服務連結角色信任下列服務擔任角色：

- eks-nodegroup.amazonaws.com

此角色許可政策允許 Amazon EKS 對指定資源完成下列動作：

- AWSServiceRoleForAmazonEKSNodegroup

您必須設定許可，IAM 實體 (如使用者、群組或角色) 才可建立、編輯或刪除服務連結角色。如需詳細資訊，請參閱《IAM 使用者指南》中的服務連結角色許可。

建立 Amazon EKS 的服務連結角色

您不需要手動建立一個服務連結角色。當您在 AWS 管理主控台、 CLI 或 AWS API AWS 中建立受管節點群組時，Amazon EKS 會為您建立服務連結角色。

> **Important**
>
> 此服務連結角色可以顯示在您的帳戶，如果您於其他服務中完成一項動作時，可以使用支援此角色的功能。Amazon EKS 自 2017 年 1 月 1 日開始支援服務連結角色，若您在這之前就有使用此服務，則 Amazon EKS 會在帳戶中建立 AWSServiceRoleForAmazonEKSNodegroup 角色。若要進一步了解，請參閱顯示在我的 IAM 帳戶中的新角色。

在 Amazon EKS (AWS API) 中建立服務連結角色

您不需要手動建立一個服務連結角色。當您在 AWS 管理主控台、 CLI 或 AWS API AWS 中建立受管節點群組時，Amazon EKS 會為您建立服務連結角色。

若您刪除此服務連結角色，之後需要再次建立，您可以在帳戶中使用相同程序重新建立角色。當您建立另一個受管節點群組時，Amazon EKS 會再次為您建立服務連結角色。

編輯 Amazon EKS 的服務連結角色

Amazon EKS 不允許您編輯 AWSServiceRoleForAmazonEKSNodegroup 服務連結角色。因為有各種實體可能會參考服務連結角色，所以您無法在建立角色之後變更角色名稱。然而，您可使用 IAM 來編輯角色描述。如需詳細資訊，請參閱《IAM 使用者指南》中的編輯服務連結角色。

刪除 Amazon EKS 的服務連結角色

若您不再使用需要服務連結角色的功能或服務，我們建議您刪除該角色。如此一來，您就沒有未主動監控或維護的未使用實體。然而，務必清除您的服務連結角色，之後才能以手動方式將其刪除。

清除服務連結角色

在您使用 IAM 刪除服務連結角色之前，您必須先刪除該角色所使用的任何資源。

> **Note**
>
> 若 Amazon EKS 服務在您試圖刪除資源時正在使用該角色，刪除可能會失敗。若此情況發生，請等待數分鐘後並再次嘗試操作。

1. 開啟 Amazon EKS 主控台。
2. 在左側導覽窗格中選擇叢集。
3. 選取 Compute (運算) 標籤。
4. 在 Node Groups (節點群組) 區段中，選擇要刪除的節點群組。
5. 在刪除確認視窗中，輸入節點群組的名稱，然後選擇 Delete (刪除)。
6. 重複對叢集中的任何其他節點群組執行此程序。等候所有刪除作業完成。

手動刪除服務連結角色

使用 IAM 主控台、CLI AWS 或 AWS API 來刪除AWSServiceRoleForAmazonEKSNodegroup服務連結角色。如需詳細資訊，請參閱《IAM 使用者指南》中的刪除服務連結角色。

Amazon EKS 服務連結角色的支援區域

Amazon EKS 在所有提供服務的區域中支援使用服務連結角色。如需詳細資訊，請參閱 Amazon EKS endpoints and quotas (Amazon EKS 端點和配額)。

使用 Amazon EKS Fargate 描述檔的角色

Amazon EKS 使用 AWS Identity and Access Management (IAM) 服務連結角色。服務連結角色是直接連結至 Amazon EKS 的一種特殊 IAM 角色類型。服務連結角色是由 Amazon EKS 預先定義，並包含該服務代表您呼叫其他 AWS 服務所需的所有許可。

服務連結角色可讓設定 Amazon EKS 更為簡單，因為您不必手動新增必要的許可。Amazon EKS 定義其服務連結角色的許可，除非另有定義，否則僅有 Amazon EKS 可以擔任其角色。定義的許可包括信任政策和許可政策，且該許可政策無法附加至其他 IAM 實體。

您必須先刪除角色的相關資源，才能刪除服務連結角色。如此可保護您的 Amazon EKS 資源，避免您不小心移除資源的存取許可。

如需關於支援服務連結角色的其他服務的資訊，請參閱可搭配 IAM 運作的AWS 服務，尋找 Service-Linked Role (服務連結角色) 欄中顯示為 Yes (是) 的服務。選擇具有連結的是，以檢視該服務的服務連結角色文件。

Amazon EKS 的服務連結角色許可

Amazon EKS 會使用名為 `AWSServiceRoleForAmazonEKSForFargate` 的服務連結角色。該角色允許 Amazon EKS Fargate 設定 Fargate Pod 所需的 VPC 聯網。連接的政策允許角色建立和刪除彈性網路介面，並描述彈性網路介面和資源。

`AWSServiceRoleForAmazonEKSForFargate` 服務連結角色信任下列服務以擔任角色：

- `eks-fargate.amazonaws.com`

此角色許可政策允許 Amazon EKS 對指定資源完成下列動作：

- AmazonEKSForFargateServiceRolePolicy

您必須設定許可，IAM 實體 (如使用者、群組或角色) 才可建立、編輯或刪除服務連結角色。如需詳細資訊，請參閱《IAM 使用者指南》中的服務連結角色許可。

建立 Amazon EKS 的服務連結角色

您不需要手動建立一個服務連結角色。當您在 AWS 管理主控台、 CLI 或 AWS API AWS 中建立 Fargate 設定檔時，Amazon EKS 會為您建立服務連結角色。

> **Important**
>
> 此服務連結角色可以顯示在您的帳戶，如果您於其他服務中完成一項動作時，可以使用支援此角色的功能。Amazon EKS 自 2019 年 12 月 13 日開始支援服務連結角色，若您在這之前就有使用此服務，則 Amazon EKS 會在帳戶中建立 AWSServiceRoleForAmazonEKSForFargate 角色。若要進一步了解，請參閱顯示在我的 IAM 帳戶中的新角色。

在 Amazon EKS (AWS API) 中建立服務連結角色

您不需要手動建立一個服務連結角色。當您在 AWS 管理主控台、 CLI 或 AWS API AWS 中建立 Fargate 設定檔時，Amazon EKS 會為您建立服務連結角色。

若您刪除此服務連結角色，之後需要再次建立，您可以在帳戶中使用相同程序重新建立角色。當您建立另一個 Fargate 設定檔時，Amazon EKS 會再次為您建立服務連結角色。

編輯 Amazon EKS 的服務連結角色

Amazon EKS 不允許您編輯 `AWSServiceRoleForAmazonEKSForFargate` 服務連結角色。因為有各種實體可能會參考服務連結角色，所以您無法在建立角色之後變更角色名稱。然而，您可使用 IAM 來編輯角色描述。如需詳細資訊，請參閱《IAM 使用者指南》中的編輯服務連結角色。

刪除 Amazon EKS 的服務連結角色

若您不再使用需要服務連結角色的功能或服務，我們建議您刪除該角色。如此一來，您就沒有未主動監控或維護的未使用實體。然而，務必清除您的服務連結角色，之後才能以手動方式將其刪除。

清除服務連結角色

在您使用 IAM 刪除服務連結角色之前，您必須先刪除該角色所使用的任何資源。

> **Note**
>
> 若 Amazon EKS 服務在您試圖刪除資源時正在使用該角色，刪除可能會失敗。若此情況發生，請等待數分鐘後並再次嘗試操作。

1. 開啟 Amazon EKS 主控台。
2. 在左側導覽窗格中選擇叢集。
3. 在 Clusters (叢集) 頁面上，選取您的叢集。
4. 選取 Compute (運算) 標籤。
5. 如果 Fargate Profiles (Fargate 描述檔) 區段中有任何 Fargate 描述檔，請個別選取每個描述檔，然後選擇 Delete (刪除)。
6. 在刪除確認視窗中輸入描述檔的名稱，然後選擇 Delete (刪除)。
7. 針對叢集中的任何其他 Fargate 描述檔，以及您帳戶中的任何其他叢集，重複此程序。

手動刪除服務連結角色

使用 IAM 主控台、CLI AWS 或 AWS API 來刪除 AWSServiceRoleForAmazonEKSForFargate 服務連結角色。如需詳細資訊，請參閱《IAM 使用者指南》中的刪除服務連結角色。

Amazon EKS 服務連結角色的支援區域

Amazon EKS 在所有提供服務的區域中支援使用服務連結角色。如需詳細資訊，請參閱 Amazon EKS endpoints and quotas (Amazon EKS 端點和配額)。

使用角色將 Kubernetes 叢集連線至 Amazon EKS

Amazon EKS 使用 AWS Identity and Access Management (IAM) 服務連結角色。服務連結角色是直接連結至 Amazon EKS 的一種特殊 IAM 角色類型。服務連結角色是由 Amazon EKS 預先定義，並包含該服務代表您呼叫其他 AWS 服務所需的所有許可。

服務連結角色可讓設定 Amazon EKS 更為簡單，因為您不必手動新增必要的許可。Amazon EKS 定義其服務連結角色的許可，除非另有定義，否則僅有 Amazon EKS 可以擔任其角色。定義的許可包括信任政策和許可政策，且該許可政策無法附加至其他 IAM 實體。

您必須先刪除角色的相關資源，才能刪除服務連結角色。如此可保護您的 Amazon EKS 資源，避免您不小心移除資源的存取許可。

如需關於支援服務連結角色的其他服務的資訊，請參閱可搭配 IAM 運作的AWS 服務，尋找 Service-Linked Role (服務連結角色) 欄中顯示為 Yes (是) 的服務。選擇具有連結的是，以檢視該服務的服務連結角色文件。

Amazon EKS 的服務連結角色許可

Amazon EKS 會使用名為 AWSServiceRoleForAmazonEKSConnector 的服務連結角色。該角色可讓 Amazon EKS 連接 Kubernetes 叢集。連接的政策可讓角色管理必要的資源，以連接到已註冊的 Kubernetes 叢集。

AWSServiceRoleForAmazonEKSConnector 服務連結角色信任下列服務以擔任角色：

- eks-connector.amazonaws.com

此角色許可政策允許 Amazon EKS 對指定資源完成下列動作：

- AmazonEKSConnectorServiceRolePolicy

您必須設定許可，IAM 實體 (如使用者、群組或角色) 才可建立、編輯或刪除服務連結角色。如需詳細資訊，請參閱 IAM 使用者指南中的服務連結角色許可。

此角色使用 SSM (Systems Manager) 許可來建立安全連線和管理連線的 Kubernetes 叢集。

建立 Amazon EKS 的服務連結角色

您不需要手動建立一個服務連結角色來連線叢集。當您在 AWS 管理主控台、 CLI、 eksctl或 AWS API AWS 中連接叢集時，Amazon EKS 會為您建立服務連結角色。

若您刪除此服務連結角色，之後需要再次建立，您可以在帳戶中使用相同程序重新建立角色。在連線叢集時，Amazon EKS 會再次為您建立服務連結角色。

編輯 Amazon EKS 的服務連結角色

Amazon EKS 不允許您編輯 AWSServiceRoleForAmazonEKSConnector 服務連結角色。因為有各種實體可能會參考服務連結角色，所以您無法在建立角色之後變更角色名稱。然而，您可使用 IAM 來編輯角色描述。如需詳細資訊，請參閱《IAM 使用者指南》中的編輯服務連結角色。

刪除 Amazon EKS 的服務連結角色

若您不再使用需要服務連結角色的功能或服務，我們建議您刪除該角色。如此一來，您就沒有未主動監控或維護的未使用實體。然而，務必清除您的服務連結角色，之後才能以手動方式將其刪除。

清除服務連結角色

在您使用 IAM 刪除服務連結角色之前，您必須先刪除該角色所使用的任何資源。

> **Note**
>
> 若 Amazon EKS 服務在您試圖刪除資源時正在使用該角色，刪除可能會失敗。若此情況發生，請等待數分鐘後並再次嘗試操作。

1. 開啟 Amazon EKS 主控台。
2. 在左側導覽窗格中選擇叢集。
3. 在 Clusters (叢集) 頁面上，選取您的叢集。
4. 選擇取消註冊，然後選擇確定。

手動刪除服務連結角色

使用 IAM 主控台、CLI AWS 或 AWS API 來刪除 AWSServiceRoleForAmazonEKSConnector 服務連結角色。如需詳細資訊，請參閱《IAM 使用者指南》中的刪除服務連結角色。

使用位於 Outpost 上的 Amazon EKS 本機叢集的角色

Amazon Elastic Kubernetes Service 使用 AWS Identity and Access Management (IAM) 服務連結角色。服務連結角色是直接連結至 Amazon EKS 的一種特殊 IAM 角色類型。服務連結角色是由 Amazon EKS 預先定義，並包含該服務代表您呼叫其他 AWS 服務所需的所有許可。

服務連結角色可讓設定 Amazon EKS 更為簡單，因為您不必手動新增必要的許可。Amazon EKS 定義其服務連結角色的許可，除非另有定義，否則僅有 Amazon EKS 可以擔任其角色。定義的許可包括信任政策和許可政策，且該許可政策無法附加至其他 IAM 實體。

您必須先刪除角色的相關資源，才能刪除服務連結角色。如此可保護您的 Amazon EKS 資源，避免您不小心移除資源的存取許可。

如需關於支援服務連結角色的其他服務的資訊，請參閱可搭配 IAM 運作的AWS 服務，尋找 Service-Linked Role (服務連結角色) 欄中顯示為 Yes (是) 的服務。選擇具有連結的是，以檢視該服務的服務連結角色文件。

Amazon EKS 的服務連結角色許可

Amazon EKS 會使用名為 `AWSServiceRoleForAmazonEKSLocalOutpost` 的服務連結角色。該角色可讓 Amazon EKS 管理您帳戶中的本機叢集。已連結的政策允許角色管理下列資源：網路介面、安全群組、日誌和 Amazon EC2 執行個體。

> **Note**
>
> AWSServiceRoleForAmazonEKSLocalOutpost 服務連結角色並非叢集建立所需的角色。如需詳細資訊，請參閱the section called “叢集 IAM 角色”。

AWSServiceRoleForAmazonEKSLocalOutpost 服務連結角色信任下列服務擔任角色：

- outposts.eks-local.amazonaws.com

此角色許可政策允許 Amazon EKS 對指定資源完成下列動作：

- AmazonEKSLocalOutpostServiceRolePolicy

您必須設定許可，IAM 實體 (如使用者、群組或角色) 才可建立、編輯或刪除服務連結角色。如需詳細資訊，請參閱《IAM 使用者指南》中的服務連結角色許可。

建立 Amazon EKS 的服務連結角色

您不需要手動建立一個服務連結角色。當您在 AWS 管理主控台、 CLI 或 AWS API AWS 中建立叢集時，Amazon EKS 會為您建立服務連結角色。

若您刪除此服務連結角色，之後需要再次建立，您可以在帳戶中使用相同程序重新建立角色。當您建立叢集時，Amazon EKS 會再次為您建立服務連結角色。

編輯 Amazon EKS 的服務連結角色

Amazon EKS 不允許您編輯 AWSServiceRoleForAmazonEKSLocalOutpost 服務連結角色。因為可能有各種實體會參考服務連結角色，所以您無法在建立角色之後變更其名稱。然而，您可使用 IAM 來編輯角色描述。如需詳細資訊，請參閱《IAM 使用者指南》中的編輯服務連結角色。

刪除 Amazon EKS 的服務連結角色

若您不再使用需要服務連結角色的功能或服務，我們建議您刪除該角色。如此一來，您就沒有未主動監控或維護的未使用實體。然而，務必清除您的服務連結角色，之後才能以手動方式將其刪除。

清除服務連結角色

在您使用 IAM 刪除服務連結角色之前，您必須先刪除該角色所使用的任何資源。

> **Note**
>
> 若 Amazon EKS 服務在您試圖刪除資源時正在使用該角色，刪除可能會失敗。若此情況發生，請等待數分鐘後並再次嘗試操作。

1. 開啟 Amazon EKS 主控台。
2. 在左側導覽窗格中選擇 Amazon EKS Clusters (叢集)。
3. 如果您的叢集具有任何節點群組或 Fargate 描述檔，則必須先將其刪除，才能刪除叢集。如需詳細資訊，請參閱 the section called “刪除” 及 the section called “刪除設定檔”。
4. 在 Clusters (叢集) 頁面上，選擇您要刪除的叢集，然後選擇 Delete (刪除)。
5. 在刪除確認視窗中輸入叢集的名稱，然後選擇 Delete (刪除)。
6. 重複對您帳戶中的任何其他叢集執行此程序。等候所有刪除作業完成。

手動刪除服務連結角色

使用 IAM 主控台、CLI AWS 或 AWS API 來刪除`AWSServiceRoleForAmazonEKSLocalOutpost`服務連結角色。如需詳細資訊，請參閱《IAM 使用者指南》中的刪除服務連結角色。

Amazon EKS 服務連結角色的支援區域

Amazon EKS 在所有提供服務的區域中支援使用服務連結角色。如需詳細資訊，請參閱 Amazon EKS endpoints and quotas (Amazon EKS 端點和配額)。

使用 Amazon EKS 儀表板的角色

Amazon EKS 儀表板使用此服務連結角色，來彙整多個區域及帳戶的叢集資源相關資訊。儀表板會與 AWS Organizations 整合，以安全地讀取組織中多個帳戶的相關資訊。

若要深入了解 Amazon EKS 儀表板，請參閱：the section called “Amazon EKS 儀表板”。

背景介紹

Amazon EKS 使用 AWS Identity and Access Management (IAM) 服務連結角色。服務連結角色是直接連結至 Amazon EKS 的一種特殊 IAM 角色類型。服務連結角色是由 Amazon EKS 預先定義，並包含該服務代表您呼叫其他 AWS 服務所需的所有許可。

服務連結角色可讓設定 Amazon EKS 更為簡單，因為您不必手動新增必要的許可。Amazon EKS 定義其服務連結角色的許可，除非另有定義，否則僅有 Amazon EKS 可以擔任其角色。定義的許可包括信任政策和許可政策，且該許可政策無法附加至其他 IAM 實體。

您必須先刪除角色的相關資源，才能刪除服務連結角色。如此可保護您的 Amazon EKS 資源，避免您不小心移除資源的存取許可。

如需關於支援服務連結角色的其他服務的資訊，請參閱可搭配 IAM 運作的AWS 服務，尋找 Service-Linked Role (服務連結角色) 欄中顯示為 Yes (是) 的服務。選擇具有連結的是，以檢視該服務的服務連結角色文件。

Amazon EKS 的服務連結角色許可

Amazon EKS 會使用名為 `AWSServiceRoleForAmazonEKSDashboard` 的服務連結角色。該角色可讓 Amazon EKS 生成並顯示 EKS 儀表板，包括叢集資源的彙整資訊。連接的 `AmazonEKSDashboardServiceRolePolicy` 政策允許角色管理下列資源：Auto Scaling 群組、安全群組、啟動範本和 IAM 執行個體設定檔。如需詳細資訊，請參閱the section called “AWS 受管政策：AmazonEKSDashboardServiceRolePolicy”。

`AWSServiceRoleForAmazonEKSDashboard` 服務連結角色信任下列服務擔任角色：

- `dashboard.eks.amazonaws.com`

若要檢視最新版本的 JSON 政策文件，請參閱《 AWS 受管政策參考指南》中的 AmazonEKSDashboardServiceRolePolicy。

您必須設定許可，IAM 實體 (如使用者、群組或角色) 才可建立、編輯或刪除服務連結角色。如需詳細資訊，請參閱《IAM 使用者指南》中的服務連結角色許可。

建立 Amazon EKS 的服務連結角色

您不需要手動建立一個服務連結角色。當您在 AWS 主控台中啟用儀表板時，Amazon EKS 會為您建立服務連結角色。

若要了解啟用 EKS 儀表板的相關詳細資訊，請參閱 the section called “設定組織”。

> **Important**
>
> 此服務連結角色可以顯示在您的帳戶，如果您於其他服務中完成一項動作時，可以使用支援此角色的功能。

編輯 Amazon EKS 的服務連結角色

Amazon EKS 不允許您編輯 AWSServiceRoleForAmazonEKSDashboard 服務連結角色。因為有各種實體可能會參考服務連結角色，所以您無法在建立角色之後變更角色名稱。然而，您可使用 IAM 來編輯角色描述。如需詳細資訊，請參閱《IAM 使用者指南》中的編輯服務連結角色。

刪除 Amazon EKS 的服務連結角色

若您不再使用需要服務連結角色的功能或服務，我們建議您刪除該角色。如此一來，您就沒有未主動監控或維護的未使用實體。然而，務必清除您的服務連結角色，之後才能以手動方式將其刪除。

清除服務連結角色

在您使用 IAM 刪除服務連結角色之前，您必須先刪除該角色所使用的任何資源。

> **Note**
>
> 若 Amazon EKS 服務在您試圖刪除資源時正在使用該角色，刪除可能會失敗。若此情況發生，請等待數分鐘後並再次嘗試操作。

若要了解停用 EKS 儀表板的相關詳細資訊，請參閱 the section called “設定組織”。

手動刪除服務連結角色

使用 IAM 主控台、CLI AWS 或 AWS API 來刪除AWSServiceRoleForAmazonEKSDashboard服務連結角色。如需詳細資訊，請參閱《IAM 使用者指南》中的刪除服務連結角色。

Amazon EKS 服務連結角色的支援區域

Amazon EKS 在所有提供服務的區域中支援使用服務連結角色。如需詳細資訊，請參閱 Amazon EKS endpoints and quotas (Amazon EKS 端點和配額)。

Amazon EKS Pod 執行 IAM 角色

在 AWS Fargate 基礎設施上執行 Pod 需要 Amazon EKS Pod 執行角色。

當您的叢集在 AWS Fargate 基礎設施上建立 Pod 時，在 Fargate 基礎設施上執行的元件必須代表您呼叫 AWS APIs。如此一來，他們可以執行動作，例如從 Amazon ECR 提取容器映像，或將日誌路由到其他 AWS 服務。Amazon EKS Pod 執行角色提供進行此類工作的 IAM 許可。

建立 Fargate 描述檔時，您必須使用描述檔為在 Fargate 基礎結構上執行的 Amazon EKS 元件指定 Pod 執行角色。此角色會新增至叢集的 Kubernetes 角色型存取控制 (RBAC)，以進行身分授權。這可

讓在 Fargate 基礎結構上執行的 `kubelet` 向您的 Amazon EKS 叢集註冊，以便它可以在該叢集中作為節點出現。

> **Note**
>
> Fargate 設定檔的 IAM 角色必須與 Amazon EC2 節點群組不同。

> **Important**
>
> 在 Fargate Pod 中執行的容器，無法採用與 Pod 執行角色相關聯的 IAM 許可。若要授予 Fargate Pod 中的容器存取其他服務的許可 AWS ，您必須為服務帳戶使用 IAM 角色。

建立 Fargate 設定檔之前，您必須建立具有 AmazonEKSFargatePodExecutionRolePolicy 的 IAM 角色。

檢查是否有正確設定的現有 Pod 執行角色

您可使用以下程序，檢查您的帳戶是否已有正確設定的 Amazon EKS Pod 執行角色。為了避免出現混淆代理人安全問題，根據 `SourceArn` 的角色限制存取是非常重要的一環。您可以視需要修改執行角色，以納入對其他叢集上的 Fargate 描述檔的支援。

1. 前往 https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。
2. 在左側導覽窗格中，選擇 Roles (角色)。
3. 在 Roles (角色) 頁面上，搜尋 Amazon EKS Fargate Pod Execution Role (Amazon EKS Fargate Pod 執行角色) 的角色清單。如果該角色不存在，請參閱 the section called “建立 Amazon EKS Pod 執行角色” 以建立角色。如果該角色存在，請選取角色。
4. 在 Amazon EKS Fargate Pod Execution Role (Amazon EKS Fargate Pod 執行角色) 頁面上，執行下列動作：
   a. 選擇許可。
   b. 確定 AmazonEKSFargatePodExecutionRolePolicy Amazon 受管政策已連接至角色。
   c. 選擇 Trust relationships (信任關係)。
   d. 選擇 Edit trust policy (編輯信任政策)。
5. 在 Edit trust policy (編輯信任政策)頁面上，確認信任關係包含以下政策，並且在叢集上具有 Fargate 描述檔的行。若是如此，請選擇 Cancel (取消)。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Condition": {
         "ArnLike": {
            "aws:SourceArn": "arn:aws:eks:us-east-1:111122223333:fargateprofile/my-
cluster/*"
         }
      },
      "Principal": {
        "Service": "eks-fargate-pods.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

如果政策相符，但沒有指定叢集上的 Fargate 描述檔的行，您可以在 `ArnLike` 物件頂端新增以下行。將 *region-code* 取代為您的叢集所在的 AWS 區域，將 *111122223333* 取代為您的 帳戶 ID，將 *my-cluster* 取代為您的叢集名稱。

```
"aws:SourceArn": "arn:aws: eks:region-code:111122223333:fargateprofile/my-cluster/*",
```

如果政策不相符，請將完整前一個政策複製到表單中，然後選擇 Update policy (更新政策)。將 *region-code* 取代為您的叢集所在的 AWS 區域。如果您想要在帳戶的所有 AWS 區域中使用相同的角色，請將 *region-code* 取代為 *。使用帳戶 ID 取代 *111122223333*，然後使用叢集名稱取代 *my-cluster*。若要在您的帳戶中的所有叢集使用相同角色，請使用 * 取代 *my-cluster*。

建立 Amazon EKS Pod 執行角色

如果您還沒有叢集的 Amazon EKS Pod 執行角色，您可以使用 AWS 管理主控台 或 AWS CLI 來建立它。

AWS 管理主控台

a. 前往 https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。

b. 在左側導覽窗格中，選擇 Roles (角色)。

c. 在 Roles (角色) 頁面上，選擇 Create role (建立角色)。

d. 在 Select trusted entity (選取信任的實體) 頁面上，執行以下作業：

   i. 在信任的實體類型區段中，選擇 AWS 服務。

   ii. 從 AWS 其他服務的使用案例下拉式清單中，選擇 EKS。

   iii. 選擇 EKS - Fargate Pod。

   iv. 選擇下一步。

e. 在 Add permissions (新增許可) 頁面上，選擇 Next (下一步)。

f. 在 Name, review, and create (命名、檢閱和建立) 頁面上，執行以下作業：

   i. 針對 Role name (角色名稱)，為您的角色輸入唯一名稱 (例如 `AmazonEKSFargatePodExecutionRole`)。

   ii. 藉由連接標籤做為鍵值對，在新增標籤 (選用) 下將中繼資料新增至角色。如需有關在 IAM 中使用標籤的詳細資訊，請參閱《IAM 使用者指南》中的標記 IAM 資源。

   iii. 選擇建立角色。

g. 在 Roles (角色) 頁面上，搜尋 Amazon EKS Fargate Pod Execution Role (Amazon EKS Fargate Pod 執行角色) 的角色清單。選擇角色。

h. 在 Amazon EKS Fargate Pod Execution Role (Amazon EKS Fargate Pod 執行角色) 頁面上，執行下列動作：

   i. 選擇 Trust relationships (信任關係)。

   ii. 選擇 Edit trust policy (編輯信任政策)。

i. 在 Edit trust policy (編輯信任政策) 頁面上，執行下列動作：

   i. 請複製以下內容，並在 Edit trust policy (編輯信任政策) 表單貼上。將 *region-code* 取代為您的叢集所在的 AWS 區域。如果您想要在帳戶的所有 AWS 區域中使用相同的角色，請將 *region-code* 取代為 *。使用帳戶 ID 取代 *111122223333*，然後使用叢集名稱取代 *my-cluster*。若要在您的帳戶中的所有叢集使用相同角色，請使用 * 取代 *my-cluster*。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Condition": {
         "ArnLike": {
            "aws:SourceArn": "arn:aws:eks:us-
east-1:111122223333:fargateprofile/my-cluster/*"
         }
```

```
      },
      "Principal": {
        "Service": "eks-fargate-pods.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

ii. 選擇更新政策。

AWS CLI

a. 複製以下內容並在名為 pod-execution-role-trust-policy.json 的檔案貼上。將 *region-code* 取代為您的叢集所在的 AWS 區域。如果您想要在帳戶的所有 AWS 區域中使用相同的角色，請將 *region-code* 取代為 *。使用帳戶 ID 取代 *111122223333*，然後使用叢集名稱取代 *my-cluster*。若要在您的帳戶中的所有叢集使用相同角色，請使用 * 取代 *my-cluster*。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Condition": {
         "ArnLike": {
            "aws:SourceArn": "arn:aws:eks:us-east-1:111122223333:fargateprofile/
my-cluster/*"
         }
      },
      "Principal": {
        "Service": "eks-fargate-pods.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

b. 建立 Pod 執行 IAM 角色。

```
aws iam create-role \
  --role-name AmazonEKSFargatePodExecutionRole \
  --assume-role-policy-document file://"pod-execution-role-trust-policy.json"
```

c. 將必要的 Amazon EKS 受管 IAM 政策連接到角色。

```
aws iam attach-role-policy \
  --policy-arn arn:aws: iam::aws:policy/AmazonEKSFargatePodExecutionRolePolicy \
  --role-name AmazonEKSFargatePodExecutionRole
```

Amazon EKS 連接器 IAM 角色

您可以連接 Kubernetes 叢集，以在 中檢視它們 AWS 管理主控台。若要連接至 Kubernetes 叢集，請建立 IAM 角色。

檢查是否有現有的 EKS 連接器角色

您可使用以下程序，檢查帳戶是否已有 Amazon EKS 連接器角色。

1. 前往 https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。
2. 在左側導覽窗格中，選擇 Roles (角色)。
3. 搜尋 `AmazonEKSConnectorAgentRole` 的角色清單。如果包含 `AmazonEKSConnectorAgentRole` 的角色不存在，請參閱 the section called “建立 Amazon EKS 連接器代理程式角色” 以建立角色。如果包含 `AmazonEKSConnectorAgentRole` 的角色存在，請選取角色以檢視連接的政策。
4. 選擇許可。
5. 確定 AmazonEKSConnectorAgentPolicy 受管政策已附加到該角色。如果已連接政策，則您的 Amazon EKS 連接器角色應已設定妥當。
6. 選擇 Trust Relationships (信任關係)，然後選擇 Edit trust policy (編輯信任政策)。
7. 確認信任關係包含下列政策。如果信任關係符合下列政策，請選擇 Cancel (取消)。如果信任關係不符合，請將政策複製到編輯信任政策視窗中，然後選擇更新政策。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "ssm.amazonaws.com"
                ]
            },
```

```
            "Action": "sts:AssumeRole"
        }
    ]
}
```

建立 Amazon EKS 連接器代理程式角色

您可以使用 AWS 管理主控台 or AWS CloudFormation 來建立連接器代理程式角色。

AWS CLI

a. 建立名為 eks-connector-agent-trust-policy.json 的檔案，其中包含用於 IAM 角色的下列 JSON。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "ssm.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

b. 建立名為 eks-connector-agent-policy.json 的檔案，其中包含用於 IAM 角色的下列 JSON。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "SsmControlChannel",
            "Effect": "Allow",
            "Action": [
                "ssmmessages:CreateControlChannel"
            ],
            "Resource": "arn:aws:eks:*:*:cluster/*"
```

```
        },
        {
            "Sid": "ssmDataplaneOperations",
            "Effect": "Allow",
            "Action": [
                "ssmmessages:CreateDataChannel",
                "ssmmessages:OpenDataChannel",
                "ssmmessages:OpenControlChannel"
            ],
            "Resource": "*"
        }
    ]
}
```

c. 使用信任政策和您在先前清單項目中建立的政策，建立 Amazon EKS 連接器代理程式角色。

```
aws iam create-role \
     --role-name AmazonEKSConnectorAgentRole \
     --assume-role-policy-document file://eks-connector-agent-trust-policy.json
```

d. 將政策連接到 Amazon EKS 連接器代理程式角色。

```
aws iam put-role-policy \
     --role-name AmazonEKSConnectorAgentRole \
     --policy-name AmazonEKSConnectorAgentPolicy \
     --policy-document file://eks-connector-agent-policy.json
```

AWS CloudFormation

a. 將下列 AWS CloudFormation 範本儲存至本機系統上的文字檔案。

> **Note**
>
> 此範本也會建立服務連結角色，否則會在呼叫 `registerCluster` API 時建立。如需詳細資訊，請參閱 the section called “叢集連接器角色”。

```
---
AWSTemplateFormatVersion: '2010-09-09'
Description: 'Provisions necessary resources needed to register clusters in EKS'
Parameters: {}
Resources:
```

```
  EKSConnectorSLR:
    Type: AWS::IAM::ServiceLinkedRole
    Properties:
      AWSServiceName: eks-connector.amazonaws.com

  EKSConnectorAgentRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Action: [ 'sts:AssumeRole' ]
            Principal:
              Service: 'ssm.amazonaws.com'

  EKSConnectorAgentPolicy:
    Type: AWS::IAM::Policy
    Properties:
      PolicyName: EKSConnectorAgentPolicy
      Roles:
        - {Ref: 'EKSConnectorAgentRole'}
      PolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: 'Allow'
            Action: [ 'ssmmessages:CreateControlChannel' ]
            Resource:
            - Fn::Sub: 'arn:${AWS::Partition}:eks:*:*:cluster/*'
          - Effect: 'Allow'
            Action: [ 'ssmmessages:CreateDataChannel',
 'ssmmessages:OpenDataChannel', 'ssmmessages:OpenControlChannel' ]
            Resource: "*"
Outputs:
  EKSConnectorAgentRoleArn:
    Description: The agent role that EKS connector uses to communicate with AWS
 services.
    Value: !GetAtt EKSConnectorAgentRole.Arn
```

b. 開啟 AWS CloudFormation 主控台。

c. 選擇 Create stack (建立堆疊) > 使用新資源 (標準)。

d. 針對 Specify template (指定範本)，選擇 Upload a template file (上傳範本檔案)，然後選擇 Choose file (選擇檔案)。

e. 選擇您剛建立的檔案，然後選擇 Next (下一步)。

f. 針對 Stack name (堆疊名稱)，輸入您角色的名稱 (例如 `eksConnectorAgentRole`)，然後選擇 Next (下一步)。

g. 在 Configure stack options (設定堆疊選項) 頁面，選擇 Next (下一步)。

h. 在 Review (檢閱) 頁面上，檢閱您的資訊，確認該堆疊可能會建立 IAM 資源，並選擇 Create stack (建立堆疊)。

AWS Amazon Elastic Kubernetes Service 的 受管政策

AWS 受管政策是由 AWS AWS 受管政策建立和管理的獨立政策旨在為許多常用案例提供許可，以便您可以開始將許可指派給使用者、群組和角色。

請記住， AWS 受管政策可能不會授予特定使用案例的最低權限許可，因為這些許可可供所有 AWS 客戶使用。我們建議您定義特定於使用案例的客戶管理政策，以便進一步減少許可。

您無法變更 AWS 受管政策中定義的許可。如果 AWS 更新 AWS 受管政策中定義的許可，則更新會影響政策連接的所有委託人身分 （使用者、群組和角色）。當新 AWS 服務啟動或新 API 操作可用於現有服務時， AWS 最有可能更新 AWS 受管政策。

如需詳細資訊，請參閱 IAM 使用者指南中的 AWS 受管政策。

AWS 受管政策：AmazonEKS_CNI_Policy

您可以將 `AmazonEKS_CNI_Policy` 連接到 IAM 實體。在建立 Amazon EC2 節點群組之前，必須將此政策連接到節點 IAM 角色，或連接到由適用於 Kubernetes 的 Amazon VPC CNI 外掛程式專門使用的 IAM 角色。這樣一來，可以代表您執行動作。我們建議您將政策連接至僅供外掛程式使用的角色。如需詳細資訊，請參閱the section called “Amazon VPC CNI”及the section called “為 IRSA 設定”。

許可詳細資訊

此政策包含下列許可，這些許可能讓 Amazon EKS 完成下列任務：

- **`ec2:*NetworkInterface`** 和 **`ec2:*PrivateIpAddresses`** – 允許 Amazon VPC CNI 外掛程式執行動作，例如佈建彈性網路介面和 Pod 的 IP 位址，以便為在 Amazon EKS 中執行的應用程式提供聯網。
- **`ec2`** 讀取動作 – 允許 Amazon VPC CNI 外掛程式執行動作，例如對執行個體與子網路進行描述，以查看 Amazon VPC 子網路中可用 IP 位址的數量。VPC CNI 可使用每個子網路中空閒的 IP 位址，選擇建立彈性網路介面時，使用最空閒 IP 位址的子網路。

若要檢視 JSON 政策文件的最新版本，請參閱《 受管政策參考指南》中的 AmazonEKS_CNI_Policy。AWS

AWS 受管政策：AmazonEKSClusterPolicy

您可以將 `AmazonEKSClusterPolicy` 連接到 IAM 實體。建立叢集之前，您必須具有連接了此政策的叢集 IAM 角色。由 Amazon EKS 管理的 Kubernetes 叢集會代表您呼叫其他 AWS 服務。他們會執行此作業，以管理您搭配本服務使用的資源。

此政策包含下列許可，這些許可能讓 Amazon EKS 完成下列任務：

- **`autoscaling`** – 讀取和更新 Auto Scaling 群組的組態。Amazon EKS 不會使用這些許可，但會保留在政策中以確保回溯相容性。
- **`ec2`** – 使用與 Amazon EC2 節點相關聯的磁碟區和網路資源。這是必要項目，以便 Kubernetes 控制平面可以將執行個體加入叢集，並動態佈建和管理 Kubernetes 持久性磁碟區請求的 Amazon EBS 磁碟區。
- **`ec2`** - 刪除由 VPC CNI 建立的彈性網路介面。此動作很必要，這樣一來 EKS 就能在 VPC CNI 非預期退出的情況下，清理留下的彈性網路介面。
- **`elasticloadbalancing`** – 使用 Elastic Load Balancer，並將節點作為目標新增至 Elastic Load Balancers。這是必要項目，以便 Kubernetes 控制平台可以動態佈建 Kubernetes 服務要求的 Elastic Load Balancer。
- **`iam`** – 建立服務連結角色。這是必要項目，以便 Kubernetes 控制平面可以動態佈建 Kubernetes 服務要求的 Elastic Load Balancer。
- **`kms`** – 從 AWS KMS 讀取金鑰。這是 Kubernetes 控制平面支援在 `etcd` 中存放的 Kubernetes 秘密之秘密加密的必要條件。

若要檢視最新版本的 JSON 政策文件，請參閱《 AWS 受管政策參考指南》中的 AmazonEKSClusterPolicy。

AWS 受管政策：AmazonEKSDashboardConsoleReadOnly

您可以將 `AmazonEKSDashboardConsoleReadOnly` 連接到 IAM 實體。

此政策包含下列許可，這些許可能讓 Amazon EKS 完成下列任務：

- **`eks`** - 以唯讀方式存取 EKS 儀表板資料、資源及叢集版本資訊。這允許檢視 EKS 相關的指標及叢集組態詳細資訊。
- **`organizations`** - AWS Organizations 資訊的唯讀存取權，包括：

- 檢視組織詳細資訊與服務存取權
- 列示組織根目錄、帳戶與組織單位
- 檢視組織結構

若要檢視最新版本的 JSON 政策文件，請參閱《 AWS 受管政策參考指南》中的 AmazonEKSDashboardConsoleReadOnly。

AWS 受管政策：AmazonEKSFargatePodExecutionRolePolicy

您可以將 `AmazonEKSFargatePodExecutionRolePolicy` 連接到 IAM 實體。在建立 Fargate 描述檔之前，您必須先建立 Fargate Pod 執行角色並將此政策連接至該角色。如需詳細資訊，請參閱the section called “步驟 2：建立 Fargate Pod 執行角色”及the section called “定義設定檔”。

此政策授予該角色許可，該許可提供在 Fargate 上執行 Amazon EKS Pod 所需的其他 AWS 服務資源的存取權。

許可詳細資訊

此政策包含下列許可，這些許可能讓 Amazon EKS 完成下列任務：

- **`ecr`** – 允許在 Fargate 上執行的 Pod 提取存放在 Amazon ECR 中的容器映像。

若要檢視 JSON 政策文件的最新版本，請參閱《 AWS 受管政策參考指南》中的 AmazonEKSFargatePodExecutionRolePolicy。

AWS 受管政策：AmazonEKSConnectorServiceRolePolicy

您不得將 `AmazonEKSConnectorServiceRolePolicy` 連接到 IAM 實體。此政策會連接到服務連結角色，而此角色可讓 Amazon EKS 代表您執行動作。如需詳細資訊，請參閱the section called “叢集連接器角色”。

該角色可讓 Amazon EKS 連接 Kubernetes 叢集。連接的政策可讓角色管理必要的資源，以連接到已註冊的 Kubernetes 叢集。

許可詳細資訊

此政策包含下列允許 Amazon EKS 完成下列任務的許可。

- **`SSM Management`** – 建立、描述和刪除 SSM 啟用，以及取消註冊受管執行個體。這允許基本 Systems Manager 操作。

- **Session Management** – 啟動專門針對 EKS 叢集的 SSM 工作階段，並使用 AmazonEKS 文件執行非互動式命令。
- **IAM Role Passing** – 將 IAM 角色特別傳遞至 SSM 服務，由將傳遞角色限制為 的條件控制`ssm.amazonaws.com`。
- **EventBridge Rules** – 建立 EventBridge 規則和目標，但僅限於由 管理時`eks-connector.amazonaws.com`。規則特別限於 AWS SSM 做為事件來源。

若要檢視最新版本的 JSON 政策文件，請參閱《 AWS 受管政策參考指南》中的 AmazonEKSConnectorServiceRolePolicy。

AWS 受管政策：AmazonEKSForFargateServiceRolePolicy

您不得將 `AmazonEKSForFargateServiceRolePolicy` 連接到 IAM 實體。此政策會連接到服務連結角色，而此角色可讓 Amazon EKS 代表您執行動作。如需詳細資訊，請參閱`AWSServiceRoleForAmazonEKSForFargate`。

此政策授予 Amazon EKS 執行 Fargate 任務的必要許可。只有在您擁有 Fargate 節點時才會使用此政策。

許可詳細資訊

此政策包含下列允許 Amazon EKS 完成下列任務的許可。

- **ec2** – 建立和刪除彈性網路介面，並描述彈性網路介面和資源。這是必要的，因此 Amazon EKS Fargate 服務可以設定 Fargate Pod 所需的 VPC 聯網。

若要檢視 JSON 政策文件的最新版本，請參閱《 AWS 受管政策參考指南》中的 AmazonEKSForFargateServiceRolePolicy。

AWS 受管政策：AmazonEKSComputePolicy

您可以將 `AmazonEKSComputePolicy` 連接到 IAM 實體。您可連接此政策至叢集 IAM 角色，以便擴充 EKS 可在帳戶中管理的資源。

此政策授予 Amazon EKS 針對 EKS 叢集建立及管理 EC2 執行個體的必要許可，以及設定 EC2 的必要 IAM 許可。此外，此政策還授予 Amazon EKS 代您建立 EC2 Spot 服務連結角色的許可。

許可詳細資訊

此政策包含下列許可，這些許可能讓 Amazon EKS 完成下列任務：

- **ec2** 許可：
  - `ec2:CreateFleet` 和 `ec2:RunInstances` - 允許建立 EC2 執行個體，並為 EKS 叢集節點使用特定 EC2 資源（映像、安全群組、子網路、隨需容量保留、置放群組）。
  - `ec2:CreateLaunchTemplate` - 允許針對 EKS 叢集節點建立 EC2 啟動範本。
  - 此外，對於透過 EKS 叢集名稱及其他相關標籤標記的資源，此政策還包括限制使用這些 EC2 許可的條件。
  - `ec2:CreateTags` - 允許新增標籤新增至透過 `CreateFleet`、`RunInstances` 與 `CreateLaunchTemplate` 動作建立的 EC2 資源。
  - `ec2:DescribeCapacityReservations` - 允許從可中斷的容量保留建立 EC2 執行個體
- **iam** 許可：
  - `iam:AddRoleToInstanceProfile` - 允許新增 IAM 角色至 EKS 運算執行個體設定檔。
  - `iam:PassRole` - 允許傳遞必要的 IAM 角色至 EC2 服務。

若要檢視最新版本的 JSON 政策文件，請參閱《AWS 受管政策參考指南》中的 AmazonEKSComputePolicy。

AWS 受管政策：AmazonEKSNetworkingPolicy

您可以將 `AmazonEKSNetworkingPolicy` 連接到 IAM 實體。您可連接此政策至叢集 IAM 角色，以便擴充 EKS 可在帳戶中管理的資源。

制定此政策旨在授予 Amazon EKS 建立及管理 EKS 叢集網路介面的必要許可，以便控制平面與工作節點能夠正常通訊及運作。

許可詳細資訊

此政策還授予以下許可，讓 Amazon EKS 能夠管理叢集的網路介面：

- **ec2** 網路介面許可：
  - `ec2:CreateNetworkInterface` - 允許建立 EC2 網路介面。
  - 此政策包括各種條件，旨在將此許可的使用限制為標記有 EKS 叢集名稱與 Kubernetes CNI 節點名稱的網路介面。
  - `ec2:CreateTags` - 允許新增標籤至透過 `CreateNetworkInterface` 動作建立的網路介面。
- **ec2** 網路介面管理許可：

- `ec2:AttachNetworkInterface`、`ec2:ModifyNetworkInterfaceAttribute`、`ec2:DetachNetworkInterface` - 允許連接、修改網路介面屬性，以及將網路介面分離到 EC2 執行個體。
- `ec2:UnassignPrivateIpAddresses`、`ec2:UnassignIpv6Addresses`、`ec2:AssignPrivateIp` - 允許管理網路介面的 IP 位址指派。
- 這些許可的使用限制為標記有 EKS 叢集名稱的網路介面。

若要檢視最新版本的 JSON 政策文件，請參閱《 AWS 受管政策參考指南》中的 AmazonEKSNetworkingPolicy。

AWS 受管政策：AmazonEKSBlockStoragePolicyV2

您可以將 `AmazonEKSBlockStoragePolicyV2` 連接到 IAM 實體。您可以將此政策連接至叢集 IAM 角色，以擴展 EKS 可在帳戶中管理的資源。

此政策可授予 Amazon EKS 建立、管理及維護 EKS 叢集 EC2 磁碟區與快照的必要許可，從而支援控制平面與工作節點依據 Kubernetes 工作負載需求，來佈建及使用持續性儲存。

許可詳細資訊

此 IAM 政策可授予以下許可，以便 Amazon EKS 能夠管理 EC2 磁碟區及快照：

- **ec2** 磁碟區管理許可：
  - `ec2:AttachVolume`、`ec2:DetachVolume`、`ec2:ModifyVolume`、`ec2:EnableFastSnapshotRe` - 允許連接、分離、修改及啟用 EC2 磁碟區的快速快照還原。
  - 這些許可的使用限制為標記有 EKS 叢集名稱的磁碟區。
  - `ec2:CreateTags` - 允許新增標籤至透過 `CreateVolume` 與 `CreateSnapshot` 動作建立的 EC2 磁碟區及快照。
- **ec2** 磁碟區建立許可：
  - `ec2:CreateVolume` - 允許建立全新的 EC2 磁碟區。
  - 對於透過 EKS 叢集名稱及其他相關標籤標記的磁碟區，此政策包括限制使用此許可的條件。
  - `ec2:CreateSnapshot` - 允許建立全新的 EC2 磁碟區快照。
  - 對於透過 EKS 叢集名稱及其他相關標籤標記的快照，此政策包括限制使用此許可的條件。

若要檢視最新版本的 JSON 政策文件，請參閱《 AWS 受管政策參考指南》中的 AmazonEKSBlockStoragePolicyV2。

AWS 受管政策：AmazonEKSLoadBalancingPolicy

您可以將 AmazonEKSLoadBalancingPolicy 連接到 IAM 實體。您可連接此政策至叢集 IAM 角色，以便擴充 EKS 可在帳戶中管理的資源。

此 IAM 政策授予 Amazon EKS 使用各種 AWS 服務的必要許可，以管理 Elastic Load Balancer (ELBs) 和相關資源。

許可詳細資訊

此政策授予的主要許可包括：

- **elasticloadbalancing** ：允許建立、修改與管理 Elastic Load Balancer 及目標群組。這包括建立、更新與刪除負載平衡器、目標群組、接聽程式及規則的許可。
- **ec2** ：允許建立及管理 Kubernetes 控制平面聯結執行個體至叢集，以及管理 Amazon EBS 磁碟區所需的安全群組。此外還允許描述及列示 EC2 資源，例如執行個體、VPC、子網路、安全群組，以及其他聯網資源。
- **iam** ：允許針對 Elastic Load Balancing 建立服務連結角色，Kubernetes 控制平面動態佈建 ELB 必須使用該角色。
- **kms** ：允許從 AWS KMS 讀取金鑰，這是 Kubernetes 控制平面支援加密存放在 等項目中的 Kubernetes 秘密所需的金鑰。
- **wafv2** 和 **shield** ：允許列出、關聯和取消關聯 Web ACLs，以及建立/刪除 Elastic Load Balancer AWS 的 Shield 保護。
- **cognito-idp** 、 **acm** 及 **elasticloadbalancing** ：授予描述使用者集區用戶端、列示並描述憑證，以及描述目標群組的許可，Kubernetes 控制平面管理 Elastic Load Balancer 必須使用這些許可。

此政策還包括多項條件檢查，以確保使用 eks:eks-cluster-name 標籤，將許可範圍限定為正在管理的特定 EKS 叢集。

若要檢視最新版本的 JSON 政策文件，請參閱《 AWS 受管政策參考指南》中的 AmazonEKSLoadBalancingPolicy。

AWS 受管政策：AmazonEKSMCPReadOnlyAccess

您可以將 AmazonEKSMCPReadOnlyAccess 連接到 IAM 實體。此政策提供 Amazon EKS 資源和相關 AWS 服務的唯讀存取權，讓 Amazon EKS 模型內容通訊協定 (MCP) 伺服器能夠執行可觀測性和故障診斷操作，而無需對您的基礎設施進行任何修改。

許可詳細資訊

此政策包含下列許可，允許主體完成下列任務：

- **eks** – 允許主體描述和列出 EKS 叢集、節點群組、附加元件、存取項目、洞見，以及存取 Kubernetes API 以進行唯讀操作。
- **iam** – 允許主體擷取 IAM 角色、政策及其附件的相關資訊，以了解與 EKS 資源相關聯的許可。
- **ec2** – 允許主體描述 VPCs、子網路和路由表，以了解 EKS 叢集的網路組態。
- **sts** ：允許主體擷取呼叫者身分資訊以進行身分驗證和授權。
- **logs** – 允許主體啟動查詢並從 CloudWatch Logs 擷取查詢結果，以進行疑難排解和監控。
- **cloudwatch** – 允許主體擷取指標資料，以監控叢集和工作負載效能。
- **eks-mcp** – 允許主體在 Amazon EKS MCP Server 內叫用 MCP 操作和呼叫唯讀工具。

若要檢視 JSON 政策文件的最新版本，請參閱《 AWS 受管政策參考指南》中的 AmazonEKSMCPReadOnlyAccess。

AWS 受管政策：AmazonEKSServicePolicy

您可以將 AmazonEKSServicePolicy 連接到 IAM 實體。在 2020 年 4 月 16 日之前建立的叢集，需要您建立 IAM 角色並將此政策連接到該角色。在 2020 年 4 月 16 日或之後建立的叢集不需要您建立角色，也不需要您指派此政策。當您使用具有 iam:CreateServiceLinkedRole許可的 IAM 主體建立叢集時，會自動為您建立 AWSServiceRoleForAmazonEKS 服務連結角色。服務連結角色已連接受管政策：AmazonEKSServiceRolePolicy。

此政策允許 Amazon EKS 建立和管理操作 Amazon EKS 叢集所需的資源。

許可詳細資訊

此政策包含下列允許 Amazon EKS 完成下列任務的許可。

- **eks** – 在您開始更新之後，更新叢集的 Kubernetes 版本。Amazon EKS 不會使用這些許可，但會保留在政策中以確保回溯相容性。
- **ec2** – 使用彈性網路介面及其他網路資源和標籤。Amazon EKS 要求這樣做，以設定可促進節點與 Kubernetes 控制平面之間的通訊的網路。讀取安全群組相關資訊。更新安全群組標籤。
- **route53** – 將 VPC 與託管區域建立關聯。Amazon EKS 必須執行這項操作，才能為您的 Kubernetes 叢集 API 伺服器啟用私有端點聯網。
- **logs** – 日誌事件。這是必要項目，以便 Amazon EKS 可以將 Kubernetes 控制平面日誌運送到 CloudWatch。

- **iam** – 建立服務連結角色。這是必需的，以便 Amazon EKScan 代表您建立 the section called "Amazon EKS 的服務連結角色許可" 服務連結角色。

若要檢視最新版本的 JSON 政策文件，請參閱《 AWS 受管政策參考指南》中的 AmazonEKSServicePolicy。

AWS 受管政策：AmazonEKSServiceRolePolicy

您不得將 AmazonEKSServiceRolePolicy 連接到 IAM 實體。此政策會連接到服務連結角色，而此角色可讓 Amazon EKS 代表您執行動作。如需詳細資訊，請參閱the section called "Amazon EKS 的服務連結角色許可"。當您使用具有 iam:CreateServiceLinkedRole許可的 IAM 主體建立叢集時，系統會自動為您建立 AWSServiceRoleForAmazonEKS 服務連結角色，並附加此政策。

此政策允許服務連結角色代表您呼叫 AWS 服務。

許可詳細資訊

此政策包含下列允許 Amazon EKS 完成下列任務的許可。

- **ec2** – 建立和描述建立叢集所需的彈性網路介面、Amazon EC2 執行個體及其狀態、叢集安全群組和 VPC。如需詳細資訊，請參閱the section called "安全群組要求"。讀取安全群組相關資訊。更新安全群組標籤。閱讀隨需容量保留的相關資訊。閱讀置放群組的相關資訊。讀取 VPC 組態，包括路由表和網路 ACLs，以在叢集洞察中偵測組態問題。
- **ec2** 自動模式 – 終止 EKS 自動模式建立的 EC2 執行個體。如需詳細資訊，請參閱EKS 自動模式。
- **iam** – 列出連接至 IAM 角色的所有受管政策。這是必要項目，以便 Amazon EKS 可以列出和驗證建立叢集所需的所有受管政策和許可。
- 將 VPC 與託管區域建立關聯 – Amazon EKS 必須執行這項操作，才能為您的 Kubernetes 叢集 API 伺服器啟用私有端點聯網。
- Log event (日誌事件) – 這是必要項目，以便 Amazon EKS 可以將 Kubernetes 控制平面日誌運送到 CloudWatch。
- 放置指標 – 這是必要項目，以便 Amazon EKS 可以將 Kubernetes 控制平面日誌運送到 CloudWatch。
- **eks** - 管理叢集對項目及政策的存取權，以便精細控制誰能存取 EKS 資源，及其可執行的動作。這包括在運算、聯網、負載平衡及儲存操作方面的關聯標準存取政策。
- **elasticloadbalancing** - 建立、管理及刪除與 EKS 叢集關聯的負載平衡器及其元件 (接聽程式、目標群組、憑證)。檢視負載平衡器的屬性與運作狀態。

- **`events`** - 建立和管理 EventBridge 規則，以監控與 EKS 叢集相關的 EC2 和 AWS 運作狀態事件，進而自動回應基礎設施變更和運作狀態提醒。
- **`iam`** - 使用「eks」字首管理 EC2 執行個體描述檔，包括列出、建立、刪除和角色關聯，這是 EKS 節點管理所需的。允許描述任何執行個體描述檔，讓使用者能夠為其工作者節點定義要使用的自訂執行個體描述檔。
- **`pricing & shield`** - 存取 AWS 定價資訊和 Shield 保護狀態，為 EKS 資源啟用成本管理和進階安全功能。
- 資源清理 - 在叢集清理操作期間，安全地刪除 EKS 標記的資源，包括磁碟區、快照、啟動範本，以及網路介面。

若要檢視最新版本的 JSON 政策文件，請參閱《 AWS 受管政策參考指南》中的 AmazonEKSServiceRolePolicy。

AWS 受管政策：AmazonEKSVPCResourceController

您可將 `AmazonEKSVPCResourceController` 政策連接到 IAM 身分。如果您使用的是適用於 Pod 的安全群組，您必須將此政策連接至 Amazon EKS 叢集 IAM 角色，以代表您執行動作。

此政策會授予叢集角色許可，以管理節點的彈性網路介面和 IP 地址。

許可詳細資訊

此政策包含下列許可，這些許可能讓 Amazon EKS 完成下列任務：

- **`ec2`** – 管理彈性網路介面和 IP 位址，以支援 Pod 安全群組和 Windows 節點。

若要檢視最新版本的 JSON 政策文件，請參閱《 AWS 受管政策參考指南》中的 AmazonEKSVPCResourceController。

AWS 受管政策：AmazonEKSWorkerNodePolicy

您可以將 `AmazonEKSWorkerNodePolicy` 連接到 IAM 實體。您必須將此政策連接至您建立 Amazon EC2 節點時指定的 IAM 角色，從而可讓 Amazon EKS 代表您執行動作。如果您使用 `eksctl` 建立節點群組，則其會建立節點 IAM 角色並自動將此政策連接至角色。

此政策授予 Amazon EKS Amazon EC2 節點許可，以連接到 Amazon EKS 叢集。

許可詳細資訊

此政策包含下列許可，這些許可能讓 Amazon EKS 完成下列任務：

- **ec2** – 讀取執行個體磁碟區和網路資訊。如此一來，Kubernetes 節點才能描述節點加入 Amazon EKS 叢集所需的 Amazon EC2 資源相關資訊，這是必要的。
- **eks** – 選擇性地將叢集描述為節點引導的一部分。
- **eks-auth:AssumeRoleForPodIdentity** – 允許擷取節點上 EKS 工作負載的憑證。必須要有這項，EKS Pod 身分識別才能正常運作。

若要檢視最新版本的 JSON 政策文件，請參閱《 AWS 受管政策參考指南》中的 AmazonEKSWorkerNodePolicy。

AWS 受管政策：AmazonEKSWorkerNodeMinimalPolicy

您無法將 AmazonEKSWorkerNodeMinimalPolicy 附加至您的 IAM 實體。您必須將此政策連接至您建立 Amazon EC2 節點時指定的 IAM 角色，從而可讓 Amazon EKS 代表您執行動作。

此政策授予 Amazon EKS Amazon EC2 節點許可，以連接到 Amazon EKS 叢集。此政策的許可比 AmazonEKSWorkerNodePolicy 少。

許可詳細資訊

此政策包含下列許可，這些許可能讓 Amazon EKS 完成下列任務：

- `eks-auth:AssumeRoleForPodIdentity` – 允許擷取節點上 EKS 工作負載的憑證。必須要有這項，EKS Pod 身分識別才能正常運作。

若要檢視最新版本的 JSON 政策文件，請參閱《 AWS 受管政策參考指南》中的 AmazonEKSWorkerNodeMinimalPolicy。

AWS 受管政策：AWSServiceRoleForAmazonEKSNodegroup

您不得將 `AWSServiceRoleForAmazonEKSNodegroup` 連接到 IAM 實體。此政策會連接到服務連結角色，而此角色可讓 Amazon EKS 代表您執行動作。如需詳細資訊，請參閱the section called “Amazon EKS 的服務連結角色許可”。

此政策可授予 `AWSServiceRoleForAmazonEKSNodegroup` 角色許可，允許它在您的帳戶中建立和管理 Amazon EC2 節點群組。

許可詳細資訊

此政策包含下列許可，這些許可能讓 Amazon EKS 完成下列任務：

- **ec2** – 使用安全群組、標籤、容量保留和啟動範本。Amazon EKS 受管節點群組必須使用該項來啟用遠端存取組態，以及描述可在受管節點群組使用的容量保留。此外，Amazon EKS 受管節點群組代表您建立啟動範本。這是為了設定支援每個受管節點群組的 Amazon EC2 Auto Scaling 群組。
- **iam** – 建立服務連結角色並傳遞角色。Amazon EKS 受管節點群組必須執行這項操作，才能管理建立受管節點群組時所傳遞之角色的執行個體描述檔。此執行個體描述檔適用於作為受管節點群組一部分的 Amazon EC2 執行個體。Amazon EKS 需要為其他服務 (例如 Amazon EC2 Auto Scaling 群組) 建立服務連結角色。這些許可會用於建立受管節點群組。
- **autoscaling** – 使用安全的 Auto Scaling 群組。Amazon EKS 受管節點群組必須執行這項操作，才能管理支援每個受管節點群組的 Amazon EC2 Auto Scaling 群組。它也用於支援功能，例如在節點群組更新期間終止或回收節點時移出 Pod，以及管理在受管節點群組上設定的暖集區。

若要檢視最新版本的 JSON 政策文件，請參閱《 AWS 受管政策參考指南》中的 AWSServiceRoleForAmazonEKSNodegroup。

AWS 受管政策：AmazonEKSDashboardServiceRolePolicy

您不得將 AmazonEKSDashboardServiceRolePolicy 連接到 IAM 實體。此政策會連接到服務連結角色，而此角色可讓 Amazon EKS 代表您執行動作。如需詳細資訊，請參閱the section called “Amazon EKS 的服務連結角色許可”。

此政策會授予AWSServiceRoleForAmazonEKSDashboard角色許可，允許其檢視 Organizations AWS 結構和帳戶的相關資訊。

許可詳細資訊

此政策包括下列許可，用於提供完成下列任務的存取權：

- **organizations** – 檢視 Organizations AWS 結構和帳戶的相關資訊。這包括下列各種許可：列示組織中的帳戶、檢視組織單位與根目錄、列示委派管理員、檢視可存取組織的服務，以及擷取組織與帳戶的詳細資訊。

若要檢視最新版本的 JSON 政策文件，請參閱《 AWS 受管政策參考指南》中的 AmazonEKSDashboardServiceRolePolicy。

AWS 受管政策：AmazonEBSCSIDriverPolicy

> **Important**
>
> 提供兩種新的 AWS受管 IAM 政策，可提供更嚴格的許可範圍：適用於標籤型範圍的 AmazonEBSCSIDriverPolicyV2，或適用於叢集範圍隔離的 AmazonEBSCSIDriverEKSClusterScopedPolicy。如果有興趣遷移，請參閱 EBS CSI 驅動程式政策遷移。

此AmazonEBSCSIDriverPolicy政策允許 Amazon EBS 容器儲存介面 (CSI) 驅動程式代表您建立、修改、複製、連接、分離和刪除磁碟區。這包括修改現有磁碟區的標籤，以及在 EBS 磁碟區上啟用快速快照還原 (FSR)。它還授予 EBS CSI 驅動程式建立、鎖定、還原和刪除快照，以及列出執行個體、磁碟區和快照的許可。

若要檢視最新版本的 JSON 政策文件，請參閱《 AWS 受管政策參考指南》中的 AmazonEBSCSIDriverServiceRolePolicy。

AWS 受管政策：AmazonEBSCSIDriverPolicyV2

AmazonEBSCSIDriverPolicyV2 政策是 更嚴格的替代方案AmazonEBSCSIDriverPolicy。它將 Amazon EBS CSI 驅動程式限制為僅管理以金鑰ebs.csi.aws.com/cluster設定為 標記的 EBS 磁碟區和快照true。資源kubernetes.io/created-for/pvc/name標籤也支援樹狀內 Kubernetes 磁碟區外掛程式 (CSI 遷移磁碟區 ） 佈建的磁碟區。

如果從 遷移AmazonEBSCSIDriverPolicy，請參閱 EBS CSI 驅動程式政策遷移。

若要檢視 JSON 政策文件的最新版本，請參閱《 AWS 受管政策參考指南》中的 AmazonEBSCSIDriverPolicyV2。

AWS 受管政策：AmazonEBSCSIDriverEKSClusterScopedPolicy

此AmazonEBSCSIDriverEKSClusterScopedPolicy政策限制 Amazon EBS CSI 驅動程式僅管理屬於特定 EKS 叢集的 EBS 磁碟區和快照。它需要資源標籤ebs.csi.aws.com/cluster-name才能符合 IAM 主體上的eks-cluster-name標籤，防止多個叢集共用相同 AWS 帳戶時跨叢集存取。執行個體上的連接和分離操作僅限於以標籤 eks:cluster-name （ 由受管節點群組上的 EKS 自動設定 ）、eks:eks-cluster-name標籤 （ 由遵循 EKS 控制器標記標準的控制器設定，包括開放原始碼 Karpenter 和 Amazon EKS Auto Mode) 或ebs.csi.aws.com/cluster-name標籤 （ 適用於手動標記的執行個體 ） 標記的執行個體。

如果從 遷移AmazonEBSCSIDriverPolicy，請參閱 EBS CSI 驅動程式政策遷移。

若要檢視 JSON 政策文件的最新版本，請參閱《 AWS 受管政策參考指南》中的 AmazonEBSCSIDriverEKSClusterScopedPolicy。

AWS 受管政策：AmazonEFSCSIDriverPolicy

AmazonEFSCSIDriverPolicy 政策可讓 Amazon EFS 容器儲存介面 (CSI) 代表您建立和刪除存取點。它還授予 Amazon EFS CSI 驅動程式許可，以列出您的存取點、檔案系統、掛載目標和 Amazon EC2 可用區域。

若要檢視最新版本的 JSON 政策文件，請參閱《 AWS 受管政策參考指南》中的 AmazonEFSCSIDriverPolicy。

AWS 受管政策：AmazonS3FilesCSIDriverPolicy

此AmazonS3FilesCSIDriverPolicy政策允許 Amazon EFS 容器儲存界面 (CSI) 代表您建立和刪除 Amazon S3 檔案存取點。它還授予 Amazon EFS CSI 驅動程式許可，以列出您的 Amazon S3 檔案存取點和檔案系統。

若要檢視最新版本的 JSON 政策文件，請參閱《 AWS 受管政策參考指南》中的 AmazonS3FilesCSIDriverPolicy。

AWS 受管政策：AmazonEKSLocalOutpostClusterPolicy

您可將此政策連接至 IAM 實體。建立本機叢集之前，您必須將此政策連接至您的叢集角色。由 Amazon EKS 管理的 Kubernetes 叢集會代表您呼叫其他 AWS 服務。他們會執行此作業，以管理您搭配本服務使用的資源。

AmazonEKSLocalOutpostClusterPolicy 包含以下許可：

- **ec2** 讀取動作 – 允許控制平面執行個體描述可用區域、路由表、執行個體，以及網路介面屬性。Amazon EC2 執行個體作為控制平面執行個體成功加入叢集所需的許可。
- **ssm** – 允許 Amazon EC2 Systems Manager 連線至控制平面執行個體，Amazon EKS 會使用該連線進行通訊並管理帳戶中的本機叢集。
- **logs** – 允許執行個體將日誌推送至 Amazon CloudWatch。
- **secretsmanager** – 允許執行個體從 AWS Secrets Manager 安全地取得和刪除控制平面執行個體的引導資料。
- **ecr** – 允許在控制平面執行個體上執行的 Pod 和容器提取存放在 Amazon Elastic Container Registry 中的容器映像。

若要檢視 JSON 政策文件的最新版本，請參閱《 AWS 受管政策參考指南》中的 AmazonEKSLocalOutpostClusterPolicy。

AWS 受管政策：AmazonEKSLocalOutpostServiceRolePolicy

您無法將此政策連接至 IAM 實體。當您使用具有 `iam:CreateServiceLinkedRole`許可的 IAM 主體建立叢集時，Amazon EKS 會自動為您建立 AWSServiceRoleForAmazonEKSLocalOutpost 服務連結角色，並將此政策連接到該角色。此政策允許服務連結角色代表您為本機叢集呼叫 AWS 服務。

`AmazonEKSLocalOutpostServiceRolePolicy` 包含以下許可：

- **`ec2`** – 允許 Amazon EKS 使用安全功能、網路和其他資源，成功啟動並管理帳戶中的控制平面執行個體。
- **`ssm`、`ssmmessages`** – 允許 Amazon EC2 Systems Manager 連線至控制平面執行個體，Amazon EKS 會使用該連線進行通訊並管理帳戶中的本機叢集。
- **`iam`** – 允許 Amazon EKS 管理與控制平面執行個體相關聯的執行個體設定檔。
- **`secretsmanager`** - 允許 Amazon EKS 將控制平面執行個體的引導資料放入 AWS Secrets Manager，以便在執行個體引導期間安全地參考。
- **`outposts`** – 允許 Amazon EKS 從您的帳戶取得 Outpost 資訊，以在 Outpost 中成功啟動本機叢集。

若要檢視 JSON 政策文件的最新版本，請參閱《 AWS 受管政策參考指南》中的 AmazonEKSLocalOutpostServiceRolePolicy。

AWS 受管政策的 Amazon EKS 更新

檢視自此服務開始追蹤 Amazon EKS AWS 受管政策更新以來的詳細資訊。

如需接收有關此特定文件頁面所有來源檔案變更的通知，您可透過 RSS 閱讀器來訂閱以下 URL：

```
https://github.com/awsdocs/amazon-eks-user-guide/commits/mainline/latest/ug/security/iam-reference/security-iam-awsmanpol.adoc.atom
```

變更	描述	日期
已更新許可至 the section called “AWS 受管政策：	已將 `gateway.eks.amazonaws.com.alb/stack`、`gateway.e`	2026 年 9 月 2 日

變更	描述	日期
AmazonEKSLoadBalancingPolicy”。	ks.amazonaws.com.nlb/stack 、 gateway.eks.amazonaws.com.alb/resource 和 gateway.eks.amazonaws.com.nlb/resource 標籤金鑰新增至 中的資源標記允許清單AmazonEKSLoadBalancingPolicy 。這可讓 Amazon EKS Auto Mode Controller 管理 Kubernetes Gateway API (ALB 和 NLB) 工作負載 AWS 的資源。	
新增了許可至 the section called “AWS 受管政策：AmazonEKSLoadBalancingPolicy”。	在 中新增wafv2:ListWebACLs 許可AmazonEKSLoadBalancingPolicy 。這可讓 Amazon EKS 自動模式控制器在協調 Application Load Balancer 關聯時列出 AWS WAF Web ACLs。	2026 年 9 月 2 日
新增了許可至 AmazonEKSServiceRolePolicy。	AmazonEKSPodIdentityPolicy 已新增至服務連結角色可以與叢集存取項目建立關聯和取消關聯的存取政策清單。此外，Amazon EKS 可以管理在 EKS 叢集上執行之工作負載的 Pod 身分憑證。	2026 年 6 月 26 日

變更	描述	日期
已更新許可至 the section called “AWS 受管政策：AmazonEKSLoadBalancingPolicy”。	在 中新增 `elasticloadbalancing:SetRulePriorities` 、`ec2:DescribeIpamPools` 許可AmazonEKSLoadBalancingPolicy 。這可讓 Amazon EKS 自動模式控制器分別設定 ALB 指定接聽程式規則的優先順序，並描述 IPAM 集區。	2026 年 6 月 15 日
新增 AmazonEBSCSIDriverEKSClusterScopedPolicy 的許可。	在以 `eks:eks-cluster-name` 標籤標記的執行個體 `ec2:DetachVolume` 上授權 `ec2:AttachVolume` 和 ，新增與遵循 EKS 控制器標記標準的控制器的相容性，包括開放原始碼 Karpenter 和 Amazon EKS Auto 模式。	2026 年 5 月 28 日
新增了許可至 the section called “AWS 受管政策：AmazonEKSComputePolicy”。	新增資源許可以包含 `ec2:DescribeCapacityReservations` 動作。這可讓 Amazon EKS Auto 模式從您帳戶中的可中斷容量保留啟動執行個體。	2026 年 5 月 18 日
推出了 the section called “AWS 受管政策：AmazonEKSBlockStoragePolicyV2”。	AWS 推出 `AmazonEKSBlockStoragePolicyV2` 。	2026 年 5 月 12 日

變更	描述	日期
新增了許可至 AmazonEBSCSIDriverPolicy。	新增 API 動作ec2:DescribeInstanceTypes ，讓 EBS CSI 驅動程式在執行時間動態擷取磁碟區限制/卡片資訊。	2026 年 5 月 12 日
新增了許可至 the section called “AWS 受管政策：AmazonEKSLoadBalancingPolicy”	在 中新增elasticloadbalancing:DescribeLoadBalancers 許可AmazonEKSLoadBalancingPolicy ，以允許 Amazon EKS 自動模式控制器描述負載平衡器。	2026 年 4 月 27 日
新增了許可至 the section called “AWS 受管政策：AmazonEKSLoadBalancingPolicy”	在 中新增elasticloadbalancing:DescribeCapacityReservation 許可AmazonEKSLoadBalancingPolicy ，以允許 Amazon EKS 自動模式控制器描述 LoadBalancer 容量單位	2026 年 4 月 20 日
新增了許可至 the section called “AWS 受管政策：AmazonEKSServiceRolePolicy”	在 中新增 ec2:DescribeInstanceStatus 和 iam:ListInstanceProfiles 許可AmazonEKSServiceRolePolicy 。這些許可允許 Amazon EKS Auto Mode 探索受管執行個體的 EC2 運作狀態事件，並清除受管執行個體設定檔	2026 年 4 月 20 日

變更	描述	日期
新增了許可至 the section called “AWS 受管政策：AmazonEKSComputePolicy”。	更新 `ec2:RunInstances` 和 `ec2:CreateFleet` 動作的資源許可，以包含置放群組 `arn:aws: ec2:*:*:placement-group/*` 。這可讓 Amazon EKS Auto 模式在帳戶中的 EC2 置放群組中啟動執行個體。	2026 年 4 月 17 日
推出 AmazonEBSCSIDriverPolicyV2 和 AmazonEBSCSIDriverEKSClusterScopedPolicy。	引進 `AmazonEBSCSIDriverPolicyV2` 和 `AmazonEBSCSIDriverEKSClusterScopedPolicy` 作為更嚴格的替代方案`AmazonEBSCSIDriverPolicy` 。會將 EBS CSI 驅動程式`AmazonEBSCSIDriverPolicyV2` 範圍限定為標記為 `ebs.csi.aws.com/cluster` 的磁碟區和快照`true`。會使用 `ebs.csi.aws.com/cluster-name` 標籤將驅動程式`AmazonEBSCSIDriverEKSClusterScopedPolicy` 範圍限定為屬於特定 EKS 叢集的磁碟區和快照。	2026 年 4 月 16 日
新增了許可至 AmazonEKSServiceRolePolicy。	新增`ec2:DescribePlacementGroups` 許可，允許 Amazon EKS Auto Mode 在帳戶中的 EC2 置放群組中啟動執行個體。	2026 年 4 月 15 日

變更	描述	日期
已更新許可至 the section called “AWS 受管政策：AmazonEKSLoadBalancingPolicy”。	已更新 `shield:CreateProtection` ，中的`shield:DeleteProtection` 許可`AmazonEKSLoadBalancingPolicy` 。這可讓 Amazon EKS 自動模式控制器新增防護。	2026 年 4 月 14 日
新增了許可至 the section called “AWS 受管政策：AmazonEKSLoadBalancingPolicy”。	更新 中的`elasticloadbalancing:RegisterTargets` 許可`AmazonEKSLoadBalancingPolicy` ，以新增對多叢集和自訂目標群組繫結關係的支援	2026 年 3 月 20 日
新增了許可至 the section called “AWS 受管政策：AmazonEKSLoadBalancingPolicy”。	在 中新增`elasticloadbalancing:ModifyIpPools` 許可`AmazonEKSLoadBalancingPolicy` ，以新增設定 ALB IPAM 集區的支援	2026 年 3 月 20 日
新增了許可至 the section called “AWS 受管政策：AmazonEKSNetworkingPolicy”。	在 中新增`ec2:ModifyNetworkInterfaceAttribute` 許可`AmazonEKSNetworkingPolicy` 。這可讓 Amazon EKS 自動模式控制器修改與 EC2 執行個體相關的網路介面屬性	2026 年 2 月 3 日

變更	描述	日期
新增了許可至 the section called “AWS 受管政策：AWSServiceRoleForAmazonEKSNodegroup”。	已將 `autoscaling:PutWarmPool` 、 `autoscaling:DeleteWarmPool` 和 `autoscaling:DescribeWarmPool` 許可新增至 `AWSServiceRoleForAmazonEKSNodegroup` 。這可讓 Amazon EKS 受管節點群組在整個節點群組生命週期中管理基礎 ASG 暖集區資源。	2026 年 2 月 17 日
新增了許可至 the section called “AWS 受管政策：AmazonEKSServiceRolePolicy”。	針對 中的`iam:GetInstanceProfile` 許可，移除目標執行個體描述檔名稱中的「eks」字首需求 `AmazonEKSServiceRolePolicy` 。這可讓 Amazon EKS Auto Mode 驗證和使用 NodeClasses 中的自訂執行個體描述檔，而不需要「eks」命名字首。	2026 年 2 月 2 日
新增了許可至 AmazonEBSCSIDriverPolicy。	新增允許 EBS CSI 驅動程式直接鎖定 EBS 快照的`ec2:LockSnapshot` 許可。	2026 年 1 月 15 日
推出了 the section called “AWS 受管政策：AmazonEKSMCPReadOnlyAccess”。	Amazon EKS 推出了新的受管政策`AmazonEKSMCPReadOnlyAccess` ，以在 Amazon EKS MCP Server 中啟用唯讀工具，以實現可觀測性和故障診斷。	2025 年 11 月 21 日

變更	描述	日期
新增了許可至 AmazonEBSCSIDriverPolicy。	新增允許 EBS CSI 驅動程式直接複製 EBS 磁碟區的ec2:CopyVolumes 許可。	2025 年 11 月 17 日
新增了許可至 the section called “AWS 受管政策：AmazonEKSServiceRolePolicy”。	新增 ec2:DescribeRouteTables 和 ec2:DescribeNetworkAcls 許可至 AmazonEKSServiceRolePolicy 。這可讓 Amazon EKS 在叢集洞察中偵測混合節點的 VPC 路由表和網路 ACLs 的組態問題。	2025 年 10 月 22 日
新增對 AWSServiceRoleForAmazonEKSConnector 的許可	新增ssmmessages:OpenDataChannel 許可至 AmazonEKSConnectorServiceRolePolicy	2025 年 10 月 15 日
新增了許可至 the section called “AWS 受管政策：AmazonEKSServiceRolePolicy”	此角色可連接全新存取政策 AmazonEKSEventPolicy 。ec2:DeleteLaunchTemplate 與 ec2:TerminateInstances 的限制許可。	2025 年 8 月 26 日
新增了許可至 the section called “AWS 受管政策：AmazonEKSLocalOutpostServiceRolePolicy”	新增了 ssmmessages:OpenDataChannel 許可至 AmazonEKSLocalOutpostServiceRolePolicy 。	2025 年 6 月 26 日

變更	描述	日期
新增了許可至 the section called “AWS 受管政策：AmazonEKSComputePolicy”。	更新了 ec2:RunInstances 與 ec2:CreateFleet 動作的資源許可，以包括容量保留 arn:aws: ec2:*:*:capacity-reservation/* 。這樣一來，Amazon EKS 自動模式可使用帳戶中的 EC2 隨需容量保留來啟動執行個體。新增了 iam:CreateServiceLinkedRole ，以允許 Amazon EKS 自動模式代您建立 EC2 Spot 服務連結角色 AWSServiceRoleForEC2Spot 。	2025 年 6 月 20 日
新增了許可至 AmazonEKSServiceRolePolicy。	新增了 ec2:DescribeCapacityReservations 許可，以允許 Amazon EKS 自動模式使用帳戶中的 EC2 隨需容量保留來啟動執行個體。	2025 年 6 月 20 日
推出了 the section called “AWS 受管政策：AmazonEKSDashboardConsoleReadOnly”。	推出了全新 AmazonEKSDashboardConsoleReadOnly 政策。	2025 年 6 月 19 日
推出了 the section called “AWS 受管政策：AmazonEKSDashboardServiceRolePolicy”。	推出了全新 AmazonEKSDashboardServiceRolePolicy 政策。	2025 年 5 月 21 日

變更	描述	日期
已新增許可至 AmazonEKS ClusterPolicy。	新增了 `ec2:DeleteNetworkInterfaces` 許可，以允許 Amazon EKS 在 VPC CNI 在非預期退出的情況下，刪除留下的彈性網路介面。	2025 年 4 月 16 日
新增了許可至 AmazonEKS ServiceRolePolicy。	作為 EKS 1.33 發行版本的一部分，新增了 `ec2:RevokeSecurityGroupEgress` 與 `ec2:AuthorizeSecurityGroupEgress` 許可，以允許 EKS AI/ML 客戶新增安全群組輸出規則至與 EFA 相容的預設 EKS 叢集 SG。	2025 年 4 月 14 日
新增了許可至 AmazonEKS ServiceRolePolicy。	新增了許可，以終止 EKS 自動模式建立的 EC2 執行個體。	2025 年 2 月 28 日

變更	描述	日期
新增了許可至 AmazonEBSCSIDriverPolicy。	新增了新的陳述式，來授權 EBS CSI 驅動程式還原所有快照。現有的政策之前允許此操作，但由於 `CreateVolume` 對 IAM 的處理有所變更，因此，需要新的明確的陳述式。 新增了 EBS CSI 驅動程式在現有磁碟區上修改標籤的功能。EBS CSI 驅動程式可透過 Kubernetes VolumeAttributesClasses 中的參數，來修改現有磁碟區的標籤。 新增了 EBS CSI 驅動程式在 EBS 磁碟區上啟用快速快照還原 (FSR) 的功能。藉助 EBS CSI 驅動程式，可透過 Kubernetes 儲存類別中的參數在新的磁碟區上啟用 FSR。	2025 年 1 月 13 日
新增了許可至 the section called “AWS 受管政策：AmazonEKSLoadBalancingPolicy”。	更新了 `AmazonEKSLoadBalancingPolicy` ，以允許列示與描述聯網及 IP 位址資源。	2024 年 12 月 26 日
新增了許可至 the section called “AWS 受管政策：AWSServiceRoleForAmazonEKSNodegroup”。	更新了 `AWSServiceRoleForAmazonEKSNodegroup` ，以適用於中國區域。	2024 年 11 月 22 日

變更	描述	日期
新增了許可至 the section called “AWS 受管政策：AmazonEKSLocalOutpostClusterPolicy”	新增 的ec2:DescribeAvailabilityZones 許可，AmazonEKSLocalOutpostClusterPolicy 讓叢集控制平面上的 AWS Cloud Controller Manager 可以識別每個節點所在的可用區域。	2024 年 11 月 21 日
新增了許可至 the section called “AWS 受管政策：AWSServiceRoleForAmazonEKSNodegroup”。	更新了 AWSServiceRoleForAmazonEKSNodegroup 政策，以允許透過 Amazon EKS 受管節點群組建立的執行個體的 ec2:RebootInstances 。限制了針對 Amazon EC2 資源的 ec2:CreateTags 許可。	2024 年 11 月 20 日
新增了許可至 the section called “AWS 受管政策：AmazonEKSServiceRolePolicy”。	EKS 已更新 AWS 受管政策 AmazonEKSServiceRolePolicy 。新增了針對下列各項的許可：EKS 存取政策、負載平衡器管理，以及自動化叢集資源清理。	2024 年 11 月 16 日
推出了 the section called “AWS 受管政策：AmazonEKSComputePolicy”。	EKS 已更新 AWS 受管政策 AmazonEKSComputePolicy 。更新了針對 iam:AddRoleToInstanceProfile 動作的資源許可。	2024 年 11 月 7 日
推出了 the section called “AWS 受管政策：AmazonEKSComputePolicy”。	AWS 推出 AmazonEKSComputePolicy 。	2024 年 11 月 1 日

變更	描述	日期
新增了許可至 AmazonEKSClusterPolicy	新增了 ec2:DescribeInstanceTopology 許可，以允許 Amazon EKS 將拓撲資訊作為標籤連接至節點。	2024 年 11 月 1 日
推出 AmazonEKSBlockStoragePolicy。	AWS 推出 AmazonEKSBlockStoragePolicy 。	2024 年 10 月 30 日
推出了 the section called “AWS 受管政策：AmazonEKSLoadBalancingPolicy”。	AWS 推出 AmazonEKSLoadBalancingPolicy 。	2024 年 10 月 30 日
新增了許可至 AmazonEKSServiceRolePolicy。	新增了 cloudwatch:PutMetricData 許可，以允許 Amazon EKS 發布指標至 Amazon CloudWatch。	2024 年 10 月 29 日
推出了 the section called “AWS 受管政策：AmazonEKSNetworkingPolicy”。	AWS 推出 AmazonEKSNetworkingPolicy 。	2024 年 10 月 28 日
新增了許可至 AmazonEKSServicePolicy 和 AmazonEKSServiceRolePolicy	新增了 ec2:GetSecurityGroupsForVpc 及關聯的標籤許可，以允許 EKS 讀取安全群組資訊及更新相關標籤。	2024 年 10 月 10 日
推出了 AmazonEKSWorkerNodeMinimalPolicy。	AWS 推出 AmazonEKSWorkerNodeMinimalPolicy 。	2024 年 10 月 3 日

變更	描述	日期
已新增許可至 AWSServiceRoleForAmazonEKSNodegroup。	新增了 `autoscaling:ResumeProcesses` 及 `autoscaling:SuspendProcesses` 許可，以允許 Amazon EKS 在 Amazon EKS 受管 Auto Scaling 群組中暫停及恢復 `AZRebalance` 。	2024 年 8 月 21 日
已新增許可至 AWSServiceRoleForAmazonEKSNodegroup。	新增了 `ec2:DescribeCapacityReservations` 許可，以允許 Amazon EKS 描述使用者帳戶中容量保留。新增了 `autoscaling:PutScheduledUpdateGroupAction` 許可，以支援在 `CAPACITY_BLOCK` 節點群組設定排程擴展。	2024 年 6 月 27 日
AmazonEKS_CNI_Policy – 更新至現有的政策	Amazon EKS 新增了全新的 `ec2:DescribeSubnets` 許可，以允許適用於 Kubernetes 的 Amazon VPC CNI 外掛程式查看 Amazon VPC 子網路中的空閒 IP 位址數量。VPC CNI 可使用每個子網路中空閒的 IP 位址，選擇建立彈性網路介面時，使用最空閒 IP 位址的子網路。	2024 年 3 月 4 日
AmazonEKSWorkerNodePolicy – 更新至現有政策	Amazon EKS 新增了新的許可來允許 EKS Pod 身分識別。Amazon EKS Pod 身分識別代理程式使用節點角色。	2023 年 11 月 26 日

變更	描述	日期
引進 AmazonEFSCSIDriverPolicy。	AWS 推出 AmazonEFSCSIDriverPolicy 。	2023 年 7 月 26 日
已新增許可至 AmazonEKSClusterPolicy。	在建立負載平衡器時，新增允許 Amazon EKS 在子網路自動探索期間取得 AZ 詳細資訊的 ec2:DescribeAvailabilityZones 許可。	2023 年 2 月 7 日
已更新 AmazonEBSCSIDriverPolicy 中的政策條件。	已移除在 StringLike 金鑰欄位中使用萬用字元的無效政策條件。也已將新條件 ec2:ResourceTag/kubernetes.io/created-for/pvc/name: "*" 新增至 ec2:DeleteVolume ，允許 EBS CSI 驅動程式刪除由樹狀內外掛程式建立的磁碟區。	2022 年 11 月 17 日
已新增許可至 AmazonEKSLocalOutpostServiceRolePolicy。	已新增 ec2:DescribeVPCAttribute 、ec2:GetConsoleOutput 和 ec2:DescribeSecret ，以允許更好的先決條件驗證和受管生命週期控制。還將 ec2:DescribePlacementGroups 和 "arn:aws: ec2:*:*:placement-group/*" 新增至 ec2:RunInstances ，以支援 Outposts 上的控制平面 Amazon EC2 執行個體的置放控制。	2022 年 10 月 24 日

變更	描述	日期
更新 AmazonEKSLocalOutpostClusterPolicy 中的 Amazon Elastic Container Registry 許可。	已將動作 `ecr:GetDownloadUrlForLayer` 從所有資源區段移至限定範圍區段。已新增資源 `arn:aws:ecr:*:*:repository/eks/` 。已移除資源 `arn:aws: ecr:`。新增的 `arn:aws: ecr:*:*:repository/eks/*` 資源涵蓋此資源。	2022 年 10 月 20 日
已新增許可至 AmazonEKSLocalOutpostClusterPolicy。	已新增 `arn:aws:ecr:*:*:repository/kubelet-config-updater` Amazon Elastic Container Registry 儲存庫，以便叢集控制平面執行個體可以更新部分 `kubelet` 引數。	2022 年 8 月 31 日
已引進 AmazonEKSLocalOutpostClusterPolicy。	AWS 推出 `AmazonEKSLocalOutpostClusterPolicy` 。	2022 年 8 月 24 日
已引進 AmazonEKSLocalOutpostServiceRolePolicy.	AWS 推出 `AmazonEKSLocalOutpostServiceRolePolicy` 。	2022 年 8 月 23 日
引進 Amazon EBS CSI Driver Policy (Amazon EBS CSI 驅動程式政策)。	AWS 推出 `AmazonEBSCSIDriverPolicy` 。	2022 年 4 月 4 日

變更	描述	日期
已新增許可至 AmazonEKSWorkerNodePolicy。	已新增 `ec2:DescribeInstanceTypes` 以啟用可自動發現發現執行個體層級屬性的 Amazon EKS 最佳化 AMI。	2022 年 3 月 21 日
已新增許可至 AWSServiceRoleForAmazonEKSNodegroup。	已新增 `autoscaling:EnableMetricsCollection` 許可以允許 Amazon EKS 啟用指標收集。	2021 年 12 月 13 日
已新增許可至 AmazonEKSClusterPolicy。	已新增 `ec2:DescribeAccountAttributes` 、`ec2:DescribeAddresses` 以及 `ec2:DescribeInternetGateways` 許可，以允許 Amazon EKS 為 Network Load Balancer 建立服務連結角色。	2021 年 6 月 17 日
Amazon EKS 已開始追蹤變更。	Amazon EKS 開始追蹤其 AWS 受管政策的變更。	2021 年 6 月 17 日

疑難排解 IAM

本主題涵蓋一些搭配 Amazon EKS 使用 IAM 時可能遇到的常見錯誤，並提供解決方法。

AccessDeniedException

如果您在呼叫 AWS API 操作`AccessDeniedException`時收到 ，則您使用的 IAM 主體憑證沒有發出該呼叫所需的許可。

```
An error occurred (AccessDeniedException) when calling the DescribeCluster operation:
User: arn:aws: iam::111122223333:user/user_name is not authorized to perform:
eks:DescribeCluster on resource: arn:aws: eks:region:111122223333:cluster/my-cluster
```

在先前的範例訊息中，使用者不具備呼叫 Amazon EKS `DescribeCluster` API 操作的許可。若要將 Amazon EKS 管理員許可提供給 IAM 主體，請參閱 the section called “身分型政策”。

如需關於 IAM 的一般資訊，請參閱《IAM 使用者指南》的使用政策控制存取。

在運算索引標籤或資源索引標籤上看不到節點，且您在 中收到錯誤 AWS 管理主控台

您可能會看到內容為 `Your current user or role does not have access to Kubernetes objects on this EKS cluster` 的主控台錯誤訊息。請確定您 AWS 管理主控台搭配 使用 的 IAM 主體使用者具有必要的許可。如需詳細資訊，請參閱the section called “所需的許可”。

aws-auth **`ConfigMap`** 不會授予對叢集的存取權限

AWS IAM Authenticator 不允許在 `ConfigMap` 中使用角色 ARN 的路徑。
因此，在您指定 `rolearn` 之前，請移除該路徑。例如，請將 `arn:aws:iam::111122223333:role/team/developers/eks-admin` 變更為 `arn:aws:iam::111122223333:role/eks-admin` 。

我未獲得執行 iam:PassRole 的授權

如果您收到錯誤，告知您無權執行 `iam:PassRole` 動作，您的政策必須更新，允許您將角色傳遞給 Amazon EKS。

有些 AWS 服務可讓您將現有角色傳遞給該服務，而不是建立新的服務角色或服務連結角色。如需執行此作業，您必須擁有將角色傳遞至該服務的許可。

當名為 `marymajor` 的 IAM 使用者嘗試使用主控台在 Amazon EKS 中執行動作時，發生下列範例錯誤。但是，動作請求服務具備服務角色授予的許可。Mary 沒有將角色傳遞給服務的許可。

```
User: {arn-aws}iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

在這種情況下，Mary 的政策必須更新，允許她執行 `iam:PassRole` 動作。

如果您需要協助，請聯絡您的 AWS 管理員。您的管理員提供您的簽署憑證。

我想要允許 AWS 帳戶外的人員存取我的 Amazon EKS 資源

您可以建立一個角色，讓其他帳戶中的使用者或您組織外部的人員存取您的資源。您可以指定要允許哪些信任物件取得該角色。針對支援基於資源的政策或存取控制清單 (ACL) 的服務，您可以使用那些政策來授予人員存取您的資源的許可。

若要進一步了解，請參閱以下內容：

- 若要了解 Amazon EKS 是否支援這些功能，請參閱 the section called “Amazon EKS 和 IAM”。
- 若要了解如何在您擁有的 AWS 帳戶中提供資源的存取權，請參閱《IAM 使用者指南》中的為您擁有的另一個 AWS 帳戶中的 IAM 使用者提供存取權。
- 若要了解如何將資源的存取權提供給第三方 AWS 帳戶，請參閱《IAM 使用者指南》中的將存取權提供給第三方擁有 AWS 的帳戶。
- 如需了解如何透過聯合身分提供存取權，請參閱《IAM 使用者指南》中的將存取權提供給在外部進行身分驗證的使用者 (聯合身分)。
- 如需了解使用角色和資源型政策進行跨帳戶存取之間的差異，請參閱《IAM 使用者指南》中的 IAM 中的跨帳戶資源存取。

Pod 容器會接收到下列錯誤：`An error occurred (SignatureDoesNotMatch) when calling the GetCallerIdentity operation: Credential should be scoped to a valid region`

如果您的應用程式明確向 AWS STS 全域端點 (`https://sts.amazonaws.com`) 提出請求，且您的 Kubernetes 服務帳戶設定為使用區域端點，則您的容器會收到此錯誤。您可以使用下列其中一種選項來解決此問題：

- 更新您的應用程式程式碼，以移除對 AWS STS 全域端點的明確呼叫。
- 更新應用程式的程式碼以明確呼叫區域端點，例如 `https://sts.us-west-2.amazonaws.com`。您的應用程式應內建備援，以便在 AWS 區域中的服務故障時挑選不同的 AWS 區域。如需詳細資訊，請參閱《IAM 使用者指南》中的在 AWS 區域中管理 AWS STS。
- 將服務帳戶設定為使用全域端點。叢集預設會使用區域端點。如需詳細資訊，請參閱the section called “STS 端點”。

Amazon EKS 叢集 IAM 角色

每個叢集都需要一個 Amazon EKS 叢集 IAM 角色。受 Amazon EKS 管理的 Kubernetes 叢集使用此角色來管理節點，而舊式雲端供應商則使用此角色為服務建立具有 Elastic Load Balancing 的負載平衡器。

建立 Amazon EKS 叢集之前，您必須先建立具有以下任一 IAM 政策的 IAM 角色：

- AmazonEKSClusterPolicy

- 自訂 IAM 政策。以下的最低許可允許 Kubernetes 叢集管理節點，但不允許舊版雲端提供者使用 Elastic Load Balancing 建立負載平衡器。您的自訂 IAM 政策必須至少擁有下列許可：

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:CreateTags"
      ],
      "Resource": "arn:aws:ec2:*:*:instance/*",
      "Condition": {
        "ForAnyValue:StringLike": {
          "aws:TagKeys": "kubernetes.io/cluster/*"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeInstances",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DescribeVpcs",
        "ec2:DescribeDhcpOptions",
        "ec2:DescribeAvailabilityZones",
        "ec2:DescribeInstanceTopology",
        "kms:DescribeKey"
      ],
      "Resource": "*"
    }
  ]
}
```

> **Note**
>
> 在 2023 年 10 月 3 日之前，每個叢集的 IAM 角色都必須使用 AmazonEKSClusterPolicy。在 2020 年 4 月 16 日之前，需要 AmazonEKSServicePolicy 和 AmazonEKSClusterPolicy，且該角色的建議名稱為 `eksServiceRole`。使用 `AWSServiceRoleForAmazonEKS` 服務連

結角色時，AmazonEKSServicePolicy 政策不再需要針對 2020 年 4 月 16 日或之後建立的叢集。

檢查現有的叢集角色

您可使用以下程序，檢查您的帳戶是否已有 Amazon EKS 叢集角色。

1. 前往 https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。
2. 在左側導覽窗格中，選擇 Roles (角色)。
3. 搜尋 `eksClusterRole` 的角色清單。如果包含 `eksClusterRole` 的角色不存在，請參閱 the section called “建立 Amazon EKS 叢集角色” 以建立角色。如果包含 `eksClusterRole` 的角色存在，請選取角色以檢視連接的政策。
4. 選擇許可。
5. 確定 AmazonEKSClusterPolicy 受管政策已附加到該角色。如果已連接政策，則您的 Amazon EKS 叢集角色應已設定妥當。
6. 選擇 Trust Relationships (信任關係)，然後選擇 Edit trust policy (編輯信任政策)。
7. 確認信任關係包含下列政策。如果信任關係符合下列政策，請選擇 Cancel (取消)。如果信任關係不符合，請將政策複製到編輯信任政策視窗中，然後選擇更新政策。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

建立 Amazon EKS 叢集角色

您可以使用 AWS 管理主控台 或 AWS CLI 來建立叢集角色。

AWS 管理主控台

a. 前往 https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。

b. 選擇 Roles (角色)，然後選擇 Create role (建立角色)。

c. 在信任的實體類型下，選取 AWS 服務。

d. 從 AWS 其他服務的使用案例下拉式清單中，選擇 EKS。

e. 針對您的使用案例選擇 EKS - Cluster (EKS - 叢集)，然後選擇 Next (下一步)。

f. 在 Add permissions (新增許可) 標籤上，選擇 Next (下一步)。

g. 針對 Role name (角色名稱)，為您的角色輸入唯一名稱 (例如 `eksClusterRole`)。

h. 針對 Description (描述)，輸入描述性文字，如 `Amazon EKS - Cluster role`。

i. 選擇建立角色。

AWS CLI

a. 將下列內容複製到名為 *`cluster-trust-policy.json`* 的檔案。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

b. 建立角色。您可以將 *`eksClusterRole`* 取代為您選擇的任何名稱。

```
aws iam create-role \
  --role-name eksClusterRole \
  --assume-role-policy-document file://"cluster-trust-policy.json"
```

c. 將所需的 IAM 政策連接至角色。

```
aws iam attach-role-policy \
  --policy-arn arn:aws: iam::aws:policy/AmazonEKSClusterPolicy \
  --role-name eksClusterRole
```

Amazon EKS 節點 IAM 角色

Amazon EKS `kubelet` 節點協助程式會代表您呼叫 AWS APIs。節點透過 IAM 執行個體描述檔和關聯的政策，取得這些 API 呼叫的許可。啟動節點並在叢集中註冊之前，您必須先為那些節點建立啟動時要使用的 IAM 角色。這項要求會套用到運用 Amazon 提供的 Amazon EKS 最佳化 AMI，或是任何其他您打算使用的節點 AMI 來啟動的節點。此外，此要求皆適用於受管節點群組和自我管理節點。

> **Note**
>
> 您無法使用用於建立任何叢集的相同角色。

建立節點之前，您必須先建立具有以下許可的 IAM 角色：

- `kubelet` 描述 VPC 中之 Amazon EC2 資源的許可，例如 AmazonEKSWorkerNodePolicy 政策提供的許可。此政策也為 Amazon EKS Pod 身分識別代理程式提供許可。
- `kubelet` 使用 Amazon Elastic Container Registry (Amazon ECR) 中之容器映像的許可，例如 AmazonEC2ContainerRegistryPullOnly 政策提供的許可。由於用於聯網的內建附加元件會執行使用 Amazon ECR 容器映像的 Pod，因此需要 Amazon Elastic Container Registry (Amazon ECR) 容器映像的使用許可。
- (選用) Amazon EKS Pod 身分識別代理程式使用 `eks-auth:AssumeRoleForPodIdentity` 動作擷取 Pod 憑證的許可。如果您不使用 AmazonEKSWorkerNodePolicy，則除了 EC2 許可之外，您還必須提供此許可才能使用 EKS Pod 身分識別。
- (選用) 如果您不使用 IRSA 或 EKS Pod 身分識別向 VPC CNI Pod 授予許可，則必須透過執行個體角色為 VPC CNI 提供許可。您可以使用 `AmazonEKS_CNI_Policy` 受管政策 (如果您使用 `IPv4` 系列建立叢集) 或您建立的 IPv6 政策 (如果您使用 `IPv6` 系列建立叢集)。但是，我們建議您將政策連接至專門用於 Amazon VPC CNI 附加元件的單獨角色，而不是將政策連接至此角色。如需為 Amazon VPC CNI 附加元件建立單獨角色的更多資訊，請參閱 the section called “為 IRSA 設定”。
- （選用） `kubelet`使用來自 Amazon ECR Public (public.ecr.aws) 之容器映像的許可，例如 AmazonElasticContainerRegistryPublicReadOnly 政策所提供的許可。如果您從 ECR Public 提取容器映像，建議使用這些許可。如果沒有它們，提取將會成功，但可能會受到速率限制。

> **Note**
>
> Amazon EC2 節點群組的 IAM 角色必須與 Fargate 設定檔不同。如需詳細資訊，請參閱the section called “Pod 執行 IAM 角色”。

檢查現有的節點角色

您可使用以下程序，檢查您的帳戶是否已有 Amazon EKS 節點角色。

1. 前往 https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。
2. 在左側導覽窗格中，選擇 Roles (角色)。
3. 搜尋 `eksNodeRole`、`AmazonEKSNodeRole` 或 `NodeInstanceRole` 的角色清單。如果具有這些名稱之一的角色不存在，請參閱 the section called “建立 Amazon EKS 節點 IAM 角色” 以建立角色。如果包含 `eksNodeRole`、`AmazonEKSNodeRole` 或 `NodeInstanceRole` 的角色存在，請選取角色以檢視連接的政策。
4. 選擇許可。
5. 請確定 AmazonEKSWorkerNodePolicy 和 AmazonEC2ContainerRegistryPullOnly 受管政策已連接到該角色，或具有最低許可的自訂政策已連接至該角色。

> **Note**
>
> 如果 AmazonEKS_CNI_Policy 政策已連接至角色，我們建議將其移除，並轉而將其連接到映射至 `aws-node` Kubernetes 服務帳戶的 IAM 角色。如需詳細資訊，請參閱the section called “為 IRSA 設定”。

6. 選擇 Trust Relationships (信任關係)，然後選擇 Edit trust policy (編輯信任政策)。
7. 確認信任關係包含下列政策。如果信任關係符合下列政策，請選擇 Cancel (取消)。如果信任關係不符合，請將政策複製到編輯信任政策視窗中，然後選擇更新政策。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sts:AssumeRole"
            ],
            "Principal": {
                "Service": [
                    "ec2.amazonaws.com"
                ]
            }
        }
    ]
```

```
}
```

建立 Amazon EKS 節點 IAM 角色

您可以使用 AWS 管理主控台 或 CLI 建立節點 IAM AWS 角色。

AWS 管理主控台

a. 前往 https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。
b. 在左側導覽窗格中，選擇 Roles (角色)。
c. 在 Roles (角色) 頁面上，選擇 Create role (建立角色)。
d. 在 Select trusted entity (選取信任的實體) 頁面上，執行以下作業：
   i. 在信任的實體類型區段中，選擇 AWS 服務。
   ii. 在 Use case (使用案例) 下，選擇 EC2。
   iii. 選擇下一步。
e. 在新增許可頁面上，連接自訂政策或執行下列動作：
   i. 在 Filter policies (篩選政策) 方塊中，輸入 `AmazonEKSWorkerNodePolicy`。
   ii. 勾選搜尋結果中 AmazonEKSWorkerNodePolicy 左側的核取方塊。
   iii. 選擇 Clear filters (清除篩選條件)。
   iv. 在 Filter policies (篩選政策) 方塊中，輸入 `AmazonEC2ContainerRegistryPullOnly`。
   v. 勾選搜尋結果中 AmazonEC2ContainerRegistryPullOnly 左側的核取方塊。

      AmazonEKS_CNI_Policy 受管政策或您建立的 IPv6 政策也必須連接至此角色或映射至 `aws-node` Kubernetes 服務帳戶的其他角色。建議您將政策指派給與 Kubernetes 服務帳戶相關聯的角色，而不要將政策指派給這個角色。如需詳細資訊，請參閱the section called “為 IRSA 設定”。
   vi. 選擇下一步。
f. 在 Name, review, and create (命名、檢閱和建立) 頁面上，執行以下作業：
   i. 針對 Role name (角色名稱)，為您的角色輸入唯一名稱 (例如 `AmazonEKSNodeRole`)。
   ii. 針對 Description (描述)，請以描述性文字 (例如 `Amazon EKS - Node role`) 取代目前的文字。
   iii. 藉由連接標籤做為鍵值對，在新增標籤 (選用) 下將中繼資料新增至角色。如需有關在 IAM 中使用標籤的詳細資訊，請參閱《IAM 使用者指南》中的標記 IAM 資源。
   iv. 選擇建立角色。

AWS CLI

a. 執行下列命令以建立 node-role-trust-relationship.json 檔案。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sts:AssumeRole"
            ],
            "Principal": {
                "Service": [
                    "ec2.amazonaws.com"
                ]
            }
        }
    ]
}
```

b. 建立 IAM 角色。

```
aws iam create-role \
  --role-name AmazonEKSNodeRole \
  --assume-role-policy-document file://"node-role-trust-relationship.json"
```

c. 將兩個所需的受管 IAM 政策連接到角色。

```
aws iam attach-role-policy \
  --policy-arn arn:aws: iam::aws:policy/AmazonEKSWorkerNodePolicy \
  --role-name AmazonEKSNodeRole
aws iam attach-role-policy \
  --policy-arn arn:aws: iam::aws:policy/AmazonEC2ContainerRegistryPullOnly \
  --role-name AmazonEKSNodeRole
```

d. 根據您建立叢集所使用的 IP 系列，將以下 IAM 政策之一連接至 IAM 角色。政策必須連接到此角色或與 Kubernetes aws-node 服務帳戶相關聯的角色，且該帳戶可用於 Kubernetes 專用 Amazon VPC CNI 外掛程式。建議您將政策指派給與 Kubernetes 服務帳戶相關聯的角色。要將政策指派給與 Kubernetes 服務帳戶相關聯的角色，請參閱 the section called “為 IRSA 設定”。

- IPv4

```
aws iam attach-role-policy \
  --policy-arn arn:aws: iam::aws:policy/AmazonEKS_CNI_Policy \
  --role-name AmazonEKSNodeRole
```

- IPv6

  A. 複製下列文字並將它儲存至名為 vpc-cni-ipv6-policy.json 的檔案。

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:AssignIpv6Addresses",
                "ec2:DescribeInstances",
                "ec2:DescribeTags",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeInstanceTypes"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateTags"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:network-interface/*"
            ]
        }
    ]
}
```

  B. 建立 IAM 政策。

```
aws iam create-policy --policy-name AmazonEKS_CNI_IPv6_Policy --policy-
document file://vpc-cni-ipv6-policy.json
```

  C. 將 IAM 政策連接至 IAM 角色。使用您的帳戶 ID 取代 *111122223333*。

```
aws iam attach-role-policy \
  --policy-arn arn:aws: iam::111122223333:policy/AmazonEKS_CNI_IPv6_Policy \
```

```
    --role-name AmazonEKSNodeRole
```

Amazon EKS 自動模式叢集 IAM 角色

每個叢集都需要一個 Amazon EKS 叢集 IAM 角色。由 Amazon EKS 管理的 Kubernetes 叢集會使用此角色，自動化儲存、聯網及運算自動擴展的常規任務。

建立 Amazon EKS 叢集之前，您必須先建立具有 EKS 自動模式所需 IAM 政策的 IAM 角色。您可以連接建議的 AWS IAM 受管政策，或建立具有同等許可的自訂政策。

- AmazonEKSComputePolicy
- AmazonEKSBlockStoragePolicyV2
- AmazonEKSLoadBalancingPolicy
- AmazonEKSNetworkingPolicy
- AmazonEKSClusterPolicy

EKS Auto 資源的自訂 AWS 標籤

根據預設，與 EKS Auto Mode 相關的受管政策不允許將使用者定義的標籤套用至 Auto Mode 佈建 AWS 的資源。如果您想要將使用者定義的標籤套用至 AWS 資源，您必須將其他許可連接到叢集 IAM 角色，並具有足夠的許可來建立和修改 AWS 資源上的標籤。以下是允許無限制標籤存取的政策範例：

檢視自訂標籤政策範例

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "Compute",
            "Effect": "Allow",
            "Action": [
                "ec2:CreateFleet",
                "ec2:RunInstances",
                "ec2:CreateLaunchTemplate"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
```

```
                    "aws:RequestTag/eks:eks-cluster-name": "${aws:PrincipalTag/eks:eks-
cluster-name}"
                },
                "StringLike": {
                    "aws:RequestTag/eks:kubernetes-node-class-name": "*",
                    "aws:RequestTag/eks:kubernetes-node-pool-name": "*"
                }
            }
        },
        {
            "Sid": "Storage",
            "Effect": "Allow",
            "Action": [
                "ec2:CreateVolume",
                "ec2:CreateSnapshot"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:volume/*",
                "arn:aws:ec2:*:*:snapshot/*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/eks:eks-cluster-name": "${aws:PrincipalTag/eks:eks-
cluster-name}"
                }
            }
        },
        {
            "Sid": "Networking",
            "Effect": "Allow",
            "Action": "ec2:CreateNetworkInterface",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/eks:eks-cluster-name": "${aws:PrincipalTag/eks:eks-
cluster-name}"
                },
                "StringLike": {
                    "aws:RequestTag/eks:kubernetes-cni-node-name": "*"
                }
            }
        },
        {
            "Sid": "LoadBalancer",
```

```
            "Effect": "Allow",
            "Action": [
                "elasticloadbalancing:CreateLoadBalancer",
                "elasticloadbalancing:CreateTargetGroup",
                "elasticloadbalancing:CreateListener",
                "elasticloadbalancing:CreateRule",
                "ec2:CreateSecurityGroup"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/eks:eks-cluster-name": "${aws:PrincipalTag/eks:eks-
cluster-name}"
                }
            }
        },
        {
            "Sid": "ShieldProtection",
            "Effect": "Allow",
            "Action": [
                "shield:CreateProtection"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/eks:eks-cluster-name": "${aws:PrincipalTag/eks:eks-
cluster-name}"
                }
            }
        },
        {
            "Sid": "ShieldTagResource",
            "Effect": "Allow",
            "Action": [
                "shield:TagResource"
            ],
            "Resource": "arn:aws:shield::*:protection/*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/eks:eks-cluster-name": "${aws:PrincipalTag/eks:eks-
cluster-name}"
                }
            }
        }
```

```
    ]
}
```

檢查現有的叢集角色

您可使用以下程序，檢查您的帳戶是否已有 Amazon EKS 叢集角色。

1. 前往 https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。
2. 在左側導覽窗格中，選擇 Roles (角色)。
3. 搜尋 `AmazonEKSAutoClusterRole` 的角色清單。如果包含 `AmazonEKSAutoClusterRole` 的角色不存在，請參閱下一節中的指示以建立角色。如果包含 `AmazonEKSAutoClusterRole` 的角色存在，請選取角色以檢視連接的政策。
4. 選擇許可。
5. 確定 AmazonEKSClusterPolicy 受管政策已附加到該角色。如果已連接政策，則您的 Amazon EKS 叢集角色應已設定妥當。
6. 選擇 Trust Relationships (信任關係)，然後選擇 Edit trust policy (編輯信任政策)。
7. 確認信任關係包含下列政策。如果信任關係符合下列政策，請選擇 Cancel (取消)。如果信任關係不符合，請將政策複製到編輯信任政策視窗中，然後選擇更新政策。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ]
    }
  ]
}
```

> **Note**
>
> AWS 不需要`AmazonEKSAutoClusterRole`此角色的名稱。

建立 Amazon EKS 叢集角色

您可以使用 AWS 管理主控台 或 AWS CLI 來建立叢集角色。

AWS 管理主控台

1. 前往 https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。
2. 選擇 Roles (角色)，然後選擇 Create role (建立角色)。
3. 在信任的實體類型下，選取 AWS 服務。
4. 從 AWS 其他服務的使用案例下拉式清單中，選擇 EKS。
5. 針對您的使用案例選擇 EKS - Cluster (EKS - 叢集)，然後選擇 Next (下一步)。
6. 在新增許可索引標籤上，選取政策，然後選擇下一步。
   - AmazonEKSComputePolicy
   - AmazonEKSBlockStoragePolicyV2
   - AmazonEKSLoadBalancingPolicy
   - AmazonEKSNetworkingPolicy
   - AmazonEKSClusterPolicy
7. 針對 Role name (角色名稱)，為您的角色輸入唯一名稱 (例如 `AmazonEKSAutoClusterRole`)。
8. 針對 Description (描述)，輸入描述性文字，如 `Amazon EKS - Cluster role`。
9. 選擇建立角色。

AWS CLI

1. 將下列內容複製到名為 *`cluster-trust-policy.json`* 的檔案。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
```

```
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ]
    }
  ]
}
```

2. 建立角色。您可以將 *AmazonEKSAutoClusterRole* 取代為選擇的任何名稱。

```
aws iam create-role \
  --role-name AmazonEKSAutoClusterRole \
  --assume-role-policy-document file://"cluster-trust-policy.json"
```

3. 將所需的 IAM 政策連接至角色：

AmazonEKSClusterPolicy：

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws: iam::aws:policy/AmazonEKSClusterPolicy
```

AmazonEKSComputePolicy：

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws: iam::aws:policy/AmazonEKSComputePolicy
```

AmazonEKSBlockStoragePolicyV2：

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws: iam::aws:policy/AmazonEKSBlockStoragePolicyV2
```

AmazonEKSLoadBalancingPolicy：

```
aws iam attach-role-policy \
```

```
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws: iam::aws:policy/AmazonEKSLoadBalancingPolicy
```

AmazonEKSNetworkingPolicy：

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoClusterRole \
    --policy-arn arn:aws: iam::aws:policy/AmazonEKSNetworkingPolicy
```

Amazon EKS 自動模式節點 IAM 角色

> **Note**
>
> 您無法使用用於建立任何叢集的相同角色。

建立節點之前，您必須先建立具有以下政策或同等許可的 IAM 角色：

- AmazonEKSWorkerNodeMinimalPolicy
- AmazonEC2ContainerRegistryPullOnly

檢查現有的節點角色

您可使用以下程序，檢查您的帳戶是否已有 Amazon EKS 節點角色。

1. 前往 https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。
2. 在左側導覽窗格中，選擇 Roles (角色)。
3. 搜尋 `AmazonEKSAutoNodeRole` 的角色清單。如果不存在具有該名稱的角色，請參閱下一節中的說明來建立角色。如果包含 `AmazonEKSAutoNodeRole` 的角色存在，請選取角色以檢視連接的政策。
4. 選擇許可。
5. 確保已附加上述所需的政策或同等的自訂政策。
6. 選擇 Trust Relationships (信任關係)，然後選擇 Edit trust policy (編輯信任政策)。
7. 確認信任關係包含下列政策。如果信任關係符合下列政策，請選擇 Cancel (取消)。如果信任關係不符合，請將政策複製到編輯信任政策視窗中，然後選擇更新政策。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

建立 Amazon EKS 節點 IAM 角色

您可以使用 AWS 管理主控台 或 CLI 建立節點 IAM AWS 角色。

AWS 管理主控台

1. 前往 https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。
2. 在左側導覽窗格中，選擇 Roles (角色)。
3. 在 Roles (角色) 頁面上，選擇 Create role (建立角色)。
4. 在 Select trusted entity (選取信任的實體) 頁面上，執行以下作業：
   a. 在信任的實體類型區段中，選擇 AWS 服務。
   b. 在 Use case (使用案例) 下，選擇 EC2。
   c. 選擇下一步。
5. 在新增許可頁面上，選擇以下政策：
   - AmazonEKSWorkerNodeMinimalPolicy
   - AmazonEC2ContainerRegistryPullOnly
6. 在 Name, review, and create (命名、檢閱和建立) 頁面上，執行以下作業：
   a. 針對 Role name (角色名稱)，為您的角色輸入唯一名稱 (例如 `AmazonEKSAutoNodeRole`)。
   b. 針對 Description (描述)，請以描述性文字 (例如 `Amazon EKS - Node role`) 取代目前的文字。
   c. 藉由連接標籤做為鍵值對，在新增標籤 (選用) 下將中繼資料新增至角色。如需有關在 IAM 中使用標籤的詳細資訊，請參閱《IAM 使用者指南》中的標記 IAM 資源。

d. 選擇建立角色。

AWS CLI

建立節點 IAM 角色

使用上一步中的 node-trust-policy.json 檔案來定義哪些實體可擔任該角色。執行下列命令以建立節點 IAM 角色：

```
aws iam create-role \
    --role-name AmazonEKSAutoNodeRole \
    --assume-role-policy-document file://node-trust-policy.json
```

請記下角色 ARN

建立角色後，擷取並儲存節點 IAM 角色的 ARN。您在後續步驟中將需要此 ARN。使用以下命令來取得 ARN：

```
aws iam get-role --role-name AmazonEKSAutoNodeRole --query "Role.Arn" --output text
```

連接必要的政策

將下列 AWS 受管政策連接至節點 IAM 角色，以提供必要的許可：

要連接 AmazonEKSWorkerNodeMinimalPolicy：

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoNodeRole \
    --policy-arn arn:aws: iam::aws:policy/AmazonEKSWorkerNodeMinimalPolicy
```

要連接 AmazonEC2ContainerRegistryPullOnly：

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoNodeRole \
    --policy-arn arn:aws: iam::aws:policy/AmazonEC2ContainerRegistryPullOnly
```

連接選用政策

如果您要從公有 ECR 提取容器映像，您應該將下列 AWS 受管政策連接至節點 IAM 角色，以確保您的節點可以向公有 ECR 進行身分驗證，並提取映像而不限流。

若要連接 AmazonElasticContainerRegistryPublicReadOnly：

```
aws iam attach-role-policy \
    --role-name AmazonEKSAutoNodeRole \
    --policy-arn arn:aws: iam::aws:policy/AmazonElasticContainerRegistryPublicReadOnly
```

Amazon EKS 功能 IAM 角色

EKS 功能需要設定功能 IAM 角色（或功能角色）。功能會使用此角色在 AWS 服務上執行動作，並透過自動建立的存取項目存取叢集中的 Kubernetes 資源。

在建立功能期間指定功能角色之前，您必須使用適當的信任政策和功能類型的許可來建立 IAM 角色。建立此 IAM 角色後，它可以重複使用於任意數量的功能資源。

功能角色需求

功能角色必須符合下列要求：

- 角色必須與叢集和功能資源位於相同的 AWS 帳戶中
- 該角色必須具有信任政策，允許 EKS 功能服務擔任該角色
- 角色必須具有適合 功能類型和使用案例需求的許可（請參閱 the section called “依功能類型列出的許可”）

功能角色的信任政策

所有功能角色都必須包含下列信任政策：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "capabilities.eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ]
    }
  ]
```

```
}
```

此信任政策允許 EKS：

- 擔任角色以執行 AWS API 操作
- 標記工作階段以進行稽核和追蹤

依功能類型列出的許可

所需的 IAM 許可取決於您使用的功能和部署模型。

> **Note**
>
> 對於搭配 ACK 使用 IAM 角色選取器的生產部署，或在不使用 AWS 服務整合的情況下使用 kro 或 Argo CD 時，功能角色可能不需要信任政策以外的任何 IAM 許可。

kro (Kube Resource Orchestrator)

不需要 IAM 許可。您可以建立沒有連接政策的功能角色。 kro 只需要 Kubernetes RBAC 許可來建立和管理 Kubernetes 資源。

AWS Kubernetes (ACK) 的控制器

ACK 支援兩種許可模型：

- 簡單設定（開發/測試）：將 AWS 服務許可直接新增至功能角色。這適用於入門、單一帳戶部署，或當所有使用者都需要相同的許可時。
- 生產最佳實務：使用 IAM 角色選取器實作最低權限存取。透過此方法，能力角色只需要擔任服務特定角色的`sts:AssumeRole`許可。您不會將 AWS 服務許可（例如 S3 或 RDS) 新增至功能角色本身，這些許可會授予對應至特定命名空間的個別 IAM 角色。

  IAM 角色選取器啟用：

  - 命名空間層級許可隔離
  - 跨帳戶資源管理
  - 團隊特定的 IAM 角色
  - 最低權限安全模型

    IAM 角色選取器方法的功能角色政策範例：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
      "Resource": [
        "arn:aws:iam::111122223333:role/ACK-S3-Role",
        "arn:aws:iam::111122223333:role/ACK-RDS-Role",
        "arn:aws:iam::444455556666:role/ACKCrossAccountRole"
      ]
    }
  ]
}
```

如需包含 IAM 角色選取器的詳細 ACK 許可組態，請參閱 the section called “設定 許可”。

Argo CD

預設不需要 IAM 許可。可能需要下列選用許可：

- AWS Secrets Manager：如果使用 Secrets Manager 存放 Git 儲存庫登入資料
- AWS CodeConnections：如果使用 CodeConnections for Git 儲存庫身分驗證

Secrets Manager 和 CodeConnections 的政策範例：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:GetSecretValue",
        "secretsmanager:DescribeSecret"
      ],
      "Resource": "arn:aws:secretsmanager:region:account-id:secret:argocd/*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "codeconnections:UseConnection",
        "codeconnections:GetConnection"
      ],
```

```
      "Resource": "arn:aws:codeconnections:region:account-id:connection/*"
    }
  ]
}
```

如需詳細的 Argo CD 許可需求，請參閱 the section called “Argo CD 考量事項”。

檢查現有功能角色

您可以使用下列程序來檢查您的帳戶是否已有適合您使用案例的功能 IAM 角色。

1. 前往 https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。
2. 在左側導覽窗格中，選擇 Roles (角色)。
3. 搜尋您職能角色名稱的角色清單（例如 `ACKCapabilityRole`或 `ArgoCDCapabilityRole`)。
4. 如果角色存在，請選取該角色以檢視連接的政策和信任關係。
5. 選擇 Trust Relationships (信任關係)，然後選擇 Edit trust policy (編輯信任政策)。
6. 確認信任關係符合功能信任政策。如果不相符，請更新信任政策。
7. 選擇許可，並確認角色具有適合您功能類型和使用案例的適當許可。

建立功能 IAM 角色

您可以使用 AWS 管理主控台 或 AWS CLI 來建立功能角色。

AWS 管理主控台

a. 前往 https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。
b. 選擇 Roles (角色)，然後選擇 Create role (建立角色)。
c. 在信任的實體類型下，選取自訂信任政策。
d. 將功能信任政策複製並貼到信任政策編輯器中。
e. 選擇下一步。
f. 在新增許可索引標籤上，選取或建立適合您功能類型的政策（請參閱 the section called “依功能類型列出的許可”)。對於 kro，您可以略過此步驟。
g. 選擇下一步。
h. 針對角色名稱，輸入角色的唯一名稱，例如 `ACKCapabilityRole`、`ArgoCDCapabilityRole`或 `kroCapabilityRole`。
i. 針對 Description (描述)，輸入描述性文字，如 `Amazon EKS - ACK capability role`。

j. 選擇建立角色。

AWS CLI

a. 將功能信任政策複製到名為 的檔案`capability-trust-policy.json`。

b. 建立角色。將 取代`ACKCapabilityRole`為您所需的角色名稱。

```
aws iam create-role \
  --role-name ACKCapabilityRole \
  --assume-role-policy-document file://capability-trust-policy.json
```

c. 將必要的 IAM 政策連接至角色。針對 ACK，為您要管理 AWS 的服務連接政策。針對 Argo CD，視需要連接 Secrets Manager 或 CodeConnections 的政策。對於 kro，您可以略過此步驟。

具有 S3 許可的 ACK 範例：

```
aws iam put-role-policy \
  --role-name ACKCapabilityRole \
  --policy-name S3Management \
  --policy-document file://s3-policy.json
```

疑難排解功能角色問題

功能建立失敗，並顯示「無效的 IAM 角色」

請驗證：

- 角色與叢集位於相同的帳戶中
- 信任政策符合功能信任政策
- 您有 角色的`iam:PassRole`許可

功能顯示許可錯誤

請驗證：

- 角色具有 功能類型的必要 IAM 許可
- 角色的 叢集上存在存取項目
- 視需要設定其他 Kubernetes 許可 （ 請參閱 the section called “其他 Kubernetes 許可”)

ACK 資源失敗，並出現「許可遭拒」錯誤

請驗證：

- 角色具有使用案例所需的 IAM 許可
- 對於參考秘密的 ACK 控制器，請確定您已將範圍為適當命名空間的`AmazonEKSSecretReaderPolicy`存取項目政策建立關聯。

如需更多故障診斷指引，請參閱 the section called “EKS 功能的考量事項”。

監控叢集效能與檢視日誌

您可以使用許多可用的監控或日誌記錄工具在 Amazon EKS 中觀察您的資料。您的 Amazon EKS 日誌資料可以串流到 AWS 服務或合作夥伴工具以進行資料分析。中有許多 服務可提供資料 AWS 管理主控台 ，以針對 Amazon EKS 問題進行故障診斷。您也可以使用 AWS支援的開放原始碼解決方案來監控 Amazon EKS 基礎設施。

在 Amazon EKS 主控台左側導覽窗格中選取叢集後，您就可以透過選擇叢集名稱，以及可觀測性索引標籤，以檢視叢集執行狀況和詳細資訊。若要檢視有關部署到叢集的任何現有 Kubernetes 資源的詳細資訊，請參閱 the section called “存取叢集資源”。

監控是維護 Amazon EKS 和 AWS 解決方案可靠性、可用性和效能的重要部分。我們建議您從 AWS 解決方案的所有部分收集監控資料。這樣，如果出現多點故障，您可以更輕鬆地進行偵錯。開始監控 Amazon EKS 前，請確保您的監控計畫可以解決下列問題。

- 您的目標是什麼？ 如果叢集大幅擴展，您是否需要即時通知？
- 需要觀察哪些資源？
- 您需要多長時間觀察這些資源？ 貴公司是否希望快速應對風險？
- 您要使用哪些工具？ 如果您已在啟動時執行 AWS Fargate，則可以使用內建日誌路由器。
- 您要由誰來執行監控任務？
- 當出現問題時，您希望向誰傳送通知？

在 Amazon EKS 上記錄和監控

Amazon EKS 提供了用於日誌監控和記錄的內建工具。對於支援的版本，可觀測性儀表板能讓您了解叢集的效能。它有助於您快速進行偵測、故障診斷及修復問題。除了監控功能之外，它還包含基於控制平面稽核日誌的清單。Kubernetes 控制平面公開了許多也可以在主控台外部抓取的指標。

控制平面日誌記錄工具記錄對叢集的所有 API 呼叫、稽核資訊 (擷取哪些使用者對叢集執行哪些操作)，以及以角色為基礎的資訊。如需詳細資訊，請參閱 AWS 規範指引中的在 Amazon EKS 上記錄和監控。

Amazon EKS 控制平面記錄從 Amazon EKS 控制平面將稽核和診斷日誌直接提供至您帳戶中的 CloudWatch Logs。這些日誌可讓您輕鬆執行叢集並確保叢集的安全。您可以選取所需的確切日誌類型，且日誌將以日誌串流傳送至 CloudWatch 中各個 Amazon EKS 叢集的群組中。如需詳細資訊，請參閱the section called “控制平面日誌”。

> **Note**
>
> 當您檢查 Amazon CloudWatch 中的 Amazon EKS 驗證器日誌時，會顯示包含類似下列範例文字的項目。
>
> ```
> level=info msg="mapping IAM role" groups="[]" role="arn:aws:
>  iam::111122223333:role/XXXXXXXXXXXXXXXXXX-NodeManagerRole-XXXXXXXX"
>  username="eks:node-manager"
> ```
>
> 預期應包含此文字的項目。`username` 是 Amazon EKS 內部服務角色，可對受管節點群組和 Fargate 執行特定操作。
> 對於低層級、可自訂的日誌記錄，可以使用 Kubernetes 日誌記錄。

Amazon EKS 已與 AWS CloudTrail 整合，CloudTrail 是一種服務，可提供 Amazon EKS 中使用者、角色或 AWS 服務所採取動作的記錄。CloudTrail 會將 Amazon EKS 的所有 API 呼叫擷取為事件。擷取的呼叫包括從 Amazon EKS 主控台執行的呼叫，以及對 Amazon EKS API 作業發出的程式碼呼叫。如需詳細資訊，請參閱the section called “ AWS CloudTrail”。

Kubernetes API 伺服器公開多個可用於監控和分析的指標。如需詳細資訊，請參閱the section called “Prometheus 指標”。

如要為自訂 Amazon CloudWatch logs 設定 Fluent Bit，請參閱《Amazon CloudWatch 使用者指南》中的設定 Fluent Bit。

在 Amazon EKS 中監控和記錄工具

Amazon Web Services 提供各種工具讓您可用於監控 Amazon EKS。您可以設定某些工具來設定自動監控，但有些工具則需要手動呼叫。建議您在您的環境和現有工具集允許的範圍內自動執行監控任務。

下表描述了各種監控工具選項。

區域	工具	說明	設定
控制平面	可觀測性儀表板	對於支援的版本，可觀測性儀表板能讓您了解叢集的效能。它有助於您快速進行偵	設定程序

區域	工具	說明	設定
		測、故障診斷及修復問題。	
應用程式/控制平面	Prometheus	Prometheus 可用於監控應用程式與控制平面的指標和提醒。	設定程序
應用程式	CloudWatch Container Insights	CloudWatch Container Insights 會從您的容器化應用程式和微型服務收集、彙總及總結指標和日誌。	設定程序
應用程式	AWS Distro for OpenTelemetry (ADOT)	ADOT 可以收集相關指標、追蹤資料和中繼資料，並將其傳送至 AWS 監控服務或合作夥伴。可以透過 CloudWatch Container Insights 進行設定。	設定程序
應用程式	Amazon DevOps Guru	Amazon DevOps Guru 可偵測節點級的執行性能和可用性。	設定程序
應用程式	AWS X-Ray	AWS X-Ray 會接收有關應用程式的追蹤資料。此追蹤資料包括傳入和傳出請求以及有關請求的中繼資料。對於 Amazon EKS，實作需要 OpenTelemetry 附加元件。	設定程序

區域	工具	說明	設定
應用程式	Amazon CloudWatch	CloudWatch 在支援的版本上可免費提供一些基本的 Amazon EKS 指標。您可使用 CloudWatch 可觀測性運算子擴展此功能，以處理收集指標、日誌和追蹤資料。	設定程序

下表描述了各種記錄工具選項。

區域	工具	說明	設定
控制平面	可觀測性儀表板	對於支援的版本，可觀測性儀表板會顯示控制平面稽核日誌的清單。它還包含 Amazon CloudWatch 中控制平面日誌的連結。	設定程序
應用程式	Amazon CloudWatch Container Insights	Amazon CloudWatch Container Insights 會從您的容器化應用程式和微型服務收集、彙總及總結指標和日誌。	設定程序
控制平面	Amazon CloudWatch Logs	您可以從 Amazon EKS 控制平面將稽核和診斷日誌直接傳送至您帳戶中的 CloudWatch Logs。	設定程序

區域	工具	說明	設定
控制平面	AWS CloudTrail	它記錄由使用者、角色或服務所進行的 API 呼叫。	設定程序
AWS Fargate 執行個體的多個區域	AWS Fargate 日誌路由器	對於 AWS Fargate 執行個體，日誌路由器會將日誌串流到 AWS 服務或合作夥伴工具。它使用 AWS for Fluent Bit。日誌可以串流到其他 AWS 服務或合作夥伴工具。	設定程序

使用可觀測性儀表板監控您的叢集

Amazon EKS 主控台包括可觀測性儀表板，您可在此深入洞察叢集的效能。藉助其提供的資訊，您可進行快速偵測、故障診斷及修復問題。您可藉由在運作狀態與效能摘要中選擇一個項目，開啟可觀測性儀表板的適用區段。在多個位置均包括此摘要，包括可觀測性索引標籤。

可觀測性儀表板將索引標籤分成多個部分。

摘要

運作狀態與效能摘要會列出不同類別的項目數。每個數字都是一個指向可觀測性儀表板中某個位置的超連結，並且有該類別的清單。

叢集運作狀態

叢集運作狀態提供需要注意的重要通知，其中有些通知可能需要您盡快採取行動。您可在此清單中查看說明及受影響的資源。叢集運作狀態包括兩個資料表：運作狀態問題與組態洞察。如需重新整理運作狀態問題的狀態，可選擇重新整理按鈕 (↻)。組態洞察每 24 小時會自動更新一次，且不可手動重新整理。

若要了解運作狀態問題的相關詳細資訊，請參閱 the section called “叢集運作狀態常見問答集與包含解析路徑的錯誤代碼”。若要了解組態洞察的相關資訊，請參閱 the section called “叢集洞察”。

控制平面監控

控制平面監控索引標籤分成三個部分，每個部分都可協助您監控叢集的控制平面和進行故障診斷。

指標

若是 Kubernetes 版本 1.28 及更高版本的叢集，指標區段可顯示針對各種控制平面元件收集的多個指標圖。

您可在區段頂端做出選擇，藉此來設定每一個指標圖 X 軸所用的時段。您可透過重新整理按鈕 (↻) 來重新整理資料。針對每個單獨的指標圖，垂直三點按鈕 (⋮) 會開啟功能表，其中包含 CloudWatch 的多個選項。

這些指標及更多選項在 CloudWatch 的 AWS/EKS 命名空間項下，可自動作為基本監控指標使用。如需詳細資訊，請參閱 Amazon CloudWatch 使用者指南中的基本監控與詳細監控。如需了解更多詳細的指標、視覺化與洞察，請參閱 Amazon CloudWatch 使用者指南中的 Container Insights。或者，若您更偏好 Prometheus 型監控，請參閱 the section called “Prometheus 指標”。

下表說明了可用的指標。

指標	說明
APIServer 請求	向 API 伺服器每分鐘發出的請求。
APIServer 請求總計 4XX	每分鐘 API 伺服器請求計數 (包含 HTTP 4XX 回應碼 (用戶端側錯誤))。
APIServer 請求總計 5XX	每分鐘 API 伺服器請求計數 (包含 HTTP 5XX 回應碼 (伺服器側錯誤))。
APIServer 請求總計 429	每分鐘 API 伺服器請求計數 (包含 HTTP 429 回應碼 (太多請求))。
儲存大小	儲存資料庫 (etcd) 大小。
排程器嘗試	依結果 "unschedulable"、"error" 和 "scheduled" 排程 Pod 的嘗試次數。
待處理 Pod	依「作用中」、「退避」、「無法排程」及「閘道控制」佇列類型的待處理 Pod 數目。

指標	說明
API 伺服器請求延遲	API 伺服器請求發生延遲。
API 伺服器目前的傳輸中請求	API 伺服器目前發生的傳輸中請求。
Webhook 請求	每分鐘 Webhook 請求。
Webhook 請求拒絕	遭到拒絕的 Webhook 請求計數。
Webhook 請求延遲 P99	外部、第三方 Webhook 請求的第 99 百分位延遲。

CloudWatch Log Insights

CloudWatch Log Insights 區段依據控制平面稽核日誌來顯示各種清單。需要開啟 Amazon EKS 控制平面日誌，才可使用此功能，您可透過檢視 CloudWatch 中的控制平面日誌區段來執行此動作。

若收集資料的時間已足夠多，您可執行全部查詢，或者一次僅針對單一清單選擇執行查詢。每當您執行查詢時，CloudWatch 都會產生額外費用。在區段頂端選擇您想要檢視的結果時段。如果您想要對任何查詢進行更進階的控制，您可以選擇在 CloudWatch 中檢視。這樣一來，您可在 CloudWatch 中更新查詢，來滿足您的需求。

如需詳細資訊，請參閱《Amazon CloudWatch Logs 使用者指南》中的藉助 CloudWatch Logs Insights 分析日誌資料。

在 CloudWatch 中檢視控制平面日誌

若要更新可用的日誌類型，請選擇管理日誌記錄。您啟用日誌記錄之後，日誌顯示於 CloudWatch Logs 中需要幾分鐘時間。若過去的時間已足夠多，在此區段中選擇任何檢視連結，可導覽至適用的日誌。

如需詳細資訊，請參閱the section called “控制平面日誌”。

叢集洞察

升級洞察資料表能夠顯露問題並建議修正措施，從而加速升級至新 Kubernetes 版本的驗證程序。Amazon EKS 會針對影響問題的潛在 Kubernetes 版本升級清單來自動掃描叢集。升級洞察資料表列示了 Amazon EKS 針對此叢集執行的洞察檢查，及其關聯的狀態。

Amazon EKS 可根據 Kubernetes 專案中的變更的評估，以及根據與新版本關聯的 Amazon EKS 服務變更，執行洞察檢查清單的維護和定期重新整理。Amazon EKS 主控台可自動重新整理每個洞察的狀態，在上次重新整理資料欄中可看到該資訊。

如需詳細資訊，請參閱the section called “叢集洞察”。

節點運作狀態問題

Amazon EKS 節點監控代理程式會自動讀取節點日誌以偵測運作狀態問題。無論自動修復設定為何，系統都會回報所有節點運作狀態問題，以便您根據需要進行調查。如果列出的問題類型未附帶說明，您可以在其彈出元素中閱讀說明。

當您重新整理頁面時，所有已解決的問題都將從清單中消失。如果啟用自動修復，您可能會暫時看到一些運作狀態問題，這些問題將會自行解決，您無須採取任何行動。根據類型，自動維修不支援的問題可能需要您手動操作。

如需報告節點運作狀態問題，叢集必須使用 Amazon EKS 自動模式，或者擁有節點監控代理程式附加元件。如需詳細資訊，請參閱the section called “節點運作狀態和修復”。

EKS 功能

功能區段顯示叢集中 EKS 功能資源的狀態和運作狀態。您可以在這裡監控叢集中功能及其受管 Kubernetes 資源的運作狀態通知。當您重新整理頁面時，所有已解決的問題都將從清單中消失。

如需詳細資訊，請參閱the section called “使用 功能”。

使用容器網路可觀測性監控 Kubernetes 工作負載流量

Amazon EKS 提供增強的網路可觀測性功能，可讓您更深入了解容器聯網環境。這些功能可協助您更好地了解、監控和疑難排解 Kubernetes 網路環境 AWS。透過增強的容器網路可觀測性，您可以利用精細的網路相關指標，跨叢集流量、跨可用區域流程 AWS 和服務進行更主動的異常偵測。您可以使用這些指標來測量系統效能，並使用您偏好的可觀測性堆疊視覺化基礎指標。

此外，Amazon EKS 現在在 AWS 主控台中提供網路監控視覺化，可加速和增強精確的故障診斷，以加快根本原因分析速度。您也可以利用這些視覺化功能，精確找出導致重新傳輸和重新傳輸逾時的頂端追蹤器和網路流程，在事件期間消除盲點。

這些功能由 Amazon CloudWatch Network Flow Monitor 啟用。

使用案例

測量網路效能以偵測異常

數個團隊會在可觀測性堆疊上標準化，讓他們能夠測量系統效能、視覺化系統指標，並在超過特定閾值時發出警示。EKS 中的容器網路可觀測性透過公開關鍵系統指標來保持一致，您可以在 Pod 和工作者節點層級擴展系統網路效能的可觀測性。

利用主控台視覺化進行更精確的故障診斷

如果監控系統發出警示，您可能想要在問題起源的叢集和工作負載上進行 。若要支援此功能，您可以在 EKS 主控台中利用視覺化，以縮小叢集層級的調查範圍，並加速揭露負責最多重新傳輸、重新傳輸逾時和傳輸資料量的網路流程。

在您的 Amazon EKS 環境中追蹤最高手數

許多團隊會執行 EKS 作為其平台的基礎，使其成為應用程式環境網路活動的焦點。使用此功能中的網路監控功能，您可以追蹤哪些工作負載負責叢集內、跨 AZs 的最多流量（以資料磁碟區測量），以及通往 AWS (DynamoDB 和 S3) 內外 AWS 雲端（網際網路或內部部署）外部目的地的流量。此外，您可以根據重新傳輸、重新傳輸逾時和傳輸的資料，監控每個流程的效能。

功能

1. 效能指標 - 此功能可讓您直接從 EKS 叢集中執行的網路流量監控 (NFM) 代理程式，抓取 Pod 和工作者節點的網路相關系統指標。
2. 服務映射 - 此功能可動態視覺化叢集中工作負載之間的通訊，讓您快速揭露與通訊 Pod 之間的網路流程相關聯的關鍵指標（重新傳輸 - RT、重新傳輸逾時 - RTO 和資料傳輸 - DT)。
3. 流程表 - 透過此表，您可以從三個不同的角度監控叢集中 Kubernetes 工作負載中的熱門發言者：AWS 服務檢視、叢集檢視和外部檢視。對於每個檢視，您可以看到重新傳輸、重新傳輸逾時，以及在來源 Pod 及其目的地之間傳輸的資料。
   - AWS 服務檢視：顯示 AWS 服務的最佳發言者 (DynamoDB 和 S3)
   - 叢集檢視：顯示叢集中的熱門發言者（從 ← 到 → 西部）
   - 外部檢視：顯示外部叢集外部目的地的熱門發言者 AWS

開始使用

若要開始使用，請在 EKS 主控台中為新的或現有的叢集啟用容器網路可觀測性。這將自動化建立網路流量監控 (NFM) 相依性 (範圍和監控資源）。此外，您必須安裝網路流量監控代理程式附加元件。或

者，您可以使用 AWS CLI、EKS APIs（適用於附加元件）、NFM APIs或基礎設施即程式碼（例如 Terraform) 安裝這些相依性。一旦這些相依性就緒，您就可以設定偏好的監控工具，從 NFM 代理程式抓取 Pod 和工作者節點的網路效能指標。若要視覺化工作負載的網路活動和效能，您可以導覽至叢集可觀測性儀表板的「網路」索引標籤下的 EKS 主控台。

在 EKS 中使用網路流量監視器時，您可以維護現有的可觀測性工作流程和技術堆疊，同時利用一組額外的功能，進一步讓您了解和最佳化 EKS 環境的網路層。您可以在此處進一步了解 Network Flow Monitor 定價。

先決條件和重要備註

1. 如上所述，如果您從 EKS 主控台啟用容器網路可觀測性，則會代表您自動建立基礎 NFM 資源相依性（範圍和監控），並引導您完成 NFM 的 EKS 附加元件安裝程序。
2. 如果您想要使用基礎設施即程式碼 (IaC) 如 Terraform 啟用此功能，您必須在 IaC 中定義下列相依性：NFM 範圍、NFM 監視器、適用於 NFM 的 EKS 附加元件。此外，您必須使用服務帳戶 (IRSA) 的 Pod Identity 或 IAM 角色，將相關許可授予 EKS 附加元件。 https://docs.aws.amazon.com/eks/latest/userguide/iam-roles-for-service-accounts.html
3. 您必須為 NFM 代理程式的 EKS 附加元件執行最低版本 1.1.0。
4. 您必須使用 Terraform AWS 提供者的 v6.21.0 或更高版本，才能支援網路流量監視器資源。

所需的 IAM 許可

NFM 代理程式的 EKS 附加元件

您可以使用 CloudWatchNetworkFlowMonitorAgentPublishPolicyAWS 受管政策搭配 Pod Identity。此政策包含 NFM 代理程式將遙測報告（指標）傳送至網路流量監控端點的許可。

```
{
  "Version" : "2012-10-17",
  "Statement" : [
    {
      "Effect" : "Allow",
      "Action" : [
        "networkflowmonitor:Publish"
      ],
      "Resource" : "*"
    }
  ]
}
```

EKS 主控台中的容器網路可觀測性

需要下列許可，才能在 主控台中啟用此功能並視覺化服務地圖和流程圖。

```
{
  "Version" : "2012-10-17",
  "Statement" : [
    {
      "Effect": "Allow",
      "Action": [
        "networkflowmonitor:ListScopes",
        "networkflowmonitor:ListMonitors",
        "networkflowmonitor:GetScope",
        "networkflowmonitor:GetMonitor",
        "networkflowmonitor:CreateScope",
        "networkflowmonitor:CreateMonitor",
        "networkflowmonitor:TagResource",
        "networkflowmonitor:StartQueryMonitorTopContributors",
        "networkflowmonitor:StopQueryMonitorTopContributors",
        "networkflowmonitor:GetQueryStatusMonitorTopContributors",
        "networkflowmonitor:GetQueryResultsMonitorTopContributors"
      ],
      "Resource": "*"
    }
  ]
}
```

使用 AWS CLI、EKS API 和 NFM API

```
#!/bin/bash

# Script to create required Network Flow Monitor resources
set -e

CLUSTER_NAME="my-eks-cluster"
CLUSTER_ARN="arn:aws:eks:{Region}:{Account}:cluster/{ClusterName}"
REGION="us-west-2"
AGENT_NAMESPACE="amazon-network-flow-monitor"

echo "Creating Network Flow Monitor resources..."

# Check if Network Flow Monitor agent is running in the cluster
echo "Checking for Network Flow Monitor agent in cluster..."
```

```
if kubectl get pods -n "$AGENT_NAMESPACE" --no-headers 2>/dev/null | grep -q "Running";
 then
    echo "Network Flow Monitor agent exists and is running in the cluster"
else
    echo "Network Flow Monitor agent not found. Installing as EKS addon..."
    aws eks create-addon \
        --cluster-name "$CLUSTER_NAME" \
        --addon-name "$AGENT_NAMESPACE" \
        --region "$REGION"
    echo "Network Flow Monitor addon installation initiated"
fi

# Get Account ID
ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)

echo "Cluster ARN: $CLUSTER_ARN"
echo "Account ID: $ACCOUNT_ID"

# Check for existing scope
echo "Checking for existing Network Flow Monitor Scope..."
EXISTING_SCOPE=$(aws networkflowmonitor list-scopes --region $REGION --query
 'scopes[0].scopeArn' --output text 2>/dev/null || echo "None")

if [ "$EXISTING_SCOPE" != "None" ] && [ "$EXISTING_SCOPE" != "null" ]; then
    echo "Using existing scope: $EXISTING_SCOPE"
    SCOPE_ARN=$EXISTING_SCOPE
else
    echo "Creating new Network Flow Monitor Scope..."
    SCOPE_RESPONSE=$(aws networkflowmonitor create-scope \
        --targets "[{\"targetIdentifier\":{\"targetId\":{\"accountId\":
\"${ACCOUNT_ID}\"},\"targetType\":\"ACCOUNT\"},\"region\":\"${REGION}\"}]" \
        --region $REGION \
        --output json)

    SCOPE_ARN=$(echo $SCOPE_RESPONSE | jq -r '.scopeArn')
    echo "Scope created: $SCOPE_ARN"
fi

# Create Network Flow Monitor with EKS Cluster as local resource
echo "Creating Network Flow Monitor..."
MONITOR_RESPONSE=$(aws networkflowmonitor create-monitor \
    --monitor-name "${CLUSTER_NAME}-monitor" \
    --local-resources "type=AWS::EKS::Cluster,identifier=${CLUSTER_ARN}" \
    --scope-arn "$SCOPE_ARN" \
```

```
    --region $REGION \
    --output json)

MONITOR_ARN=$(echo $MONITOR_RESPONSE | jq -r '.monitorArn')

echo "Monitor created: $MONITOR_ARN"

echo "Network Flow Monitor setup complete!"
echo "Monitor ARN: $MONITOR_ARN"
echo "Scope ARN: $SCOPE_ARN"
echo "Local Resource: AWS::EKS::Cluster (${CLUSTER_ARN})"
```

使用基礎設施做為程式碼 (IaC)

Terraform

如果您使用 Terraform 來管理 AWS 雲端基礎設施，您可以包含下列資源組態，為您的叢集啟用容器網路可觀測性。

NFM 範圍

```
data "aws_caller_identity" "current" {}

resource "aws_networkflowmonitor_scope" "example" {
  target {
    region = "us-east-1"
    target_identifier {
      target_type = "ACCOUNT"
      target_id {
        account_id = data.aws_caller_identity.current.account_id
      }
    }
  }

  tags = {
    Name = "example"
  }
}
```

NFM 監視器

```
resource "aws_networkflowmonitor_monitor" "example" {
```

```
  monitor_name = "eks-cluster-name-monitor"
  scope_arn    = aws_networkflowmonitor_scope.example.scope_arn

  local_resource {
    type       = "AWS::EKS::Cluster"
    identifier = aws_eks_cluster.example.arn
  }

  remote_resource {
    type       = "AWS::Region"
    identifier = "us-east-1" # this must be the same region that the cluster is in
  }

  tags = {
    Name = "example"
  }
}
```

NFM 的 EKS 附加元件

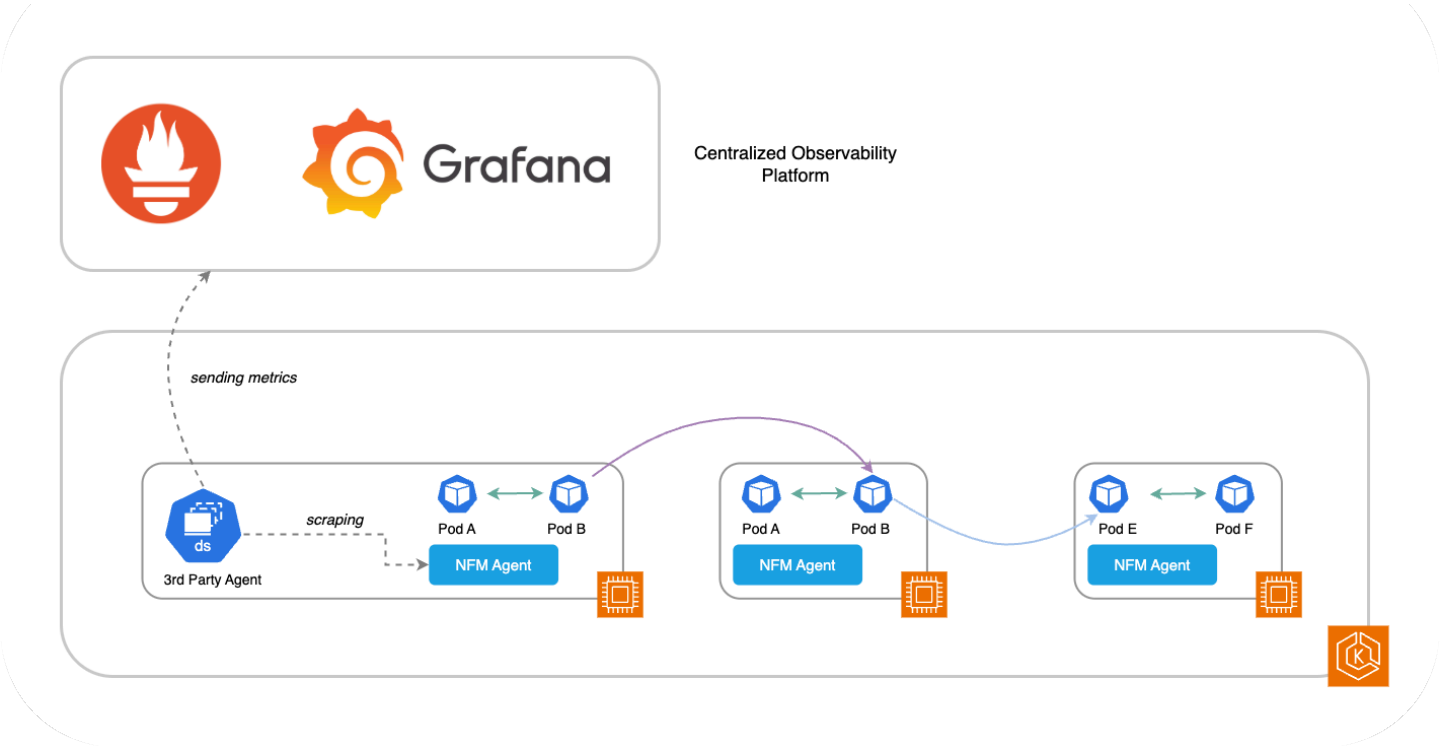
```
resource "aws_eks_addon" "example" {
  cluster_name                = aws_eks_cluster.example.name
  addon_name                  = "aws-network-flow-monitoring-agent"
}
```

其運作方式？

效能指標

系統指標

如果您執行第三方 (3P) 工具來監控 EKS 環境 （ 例如 Prometheus 和 Grafana)，則可以直接從網路流量監控代理程式抓取支援的系統指標。這些指標可以傳送到您的監控堆疊，以在 Pod 和工作者節點層級擴展系統網路效能的測量。可用的指標會列在支援的系統指標下的表格中。



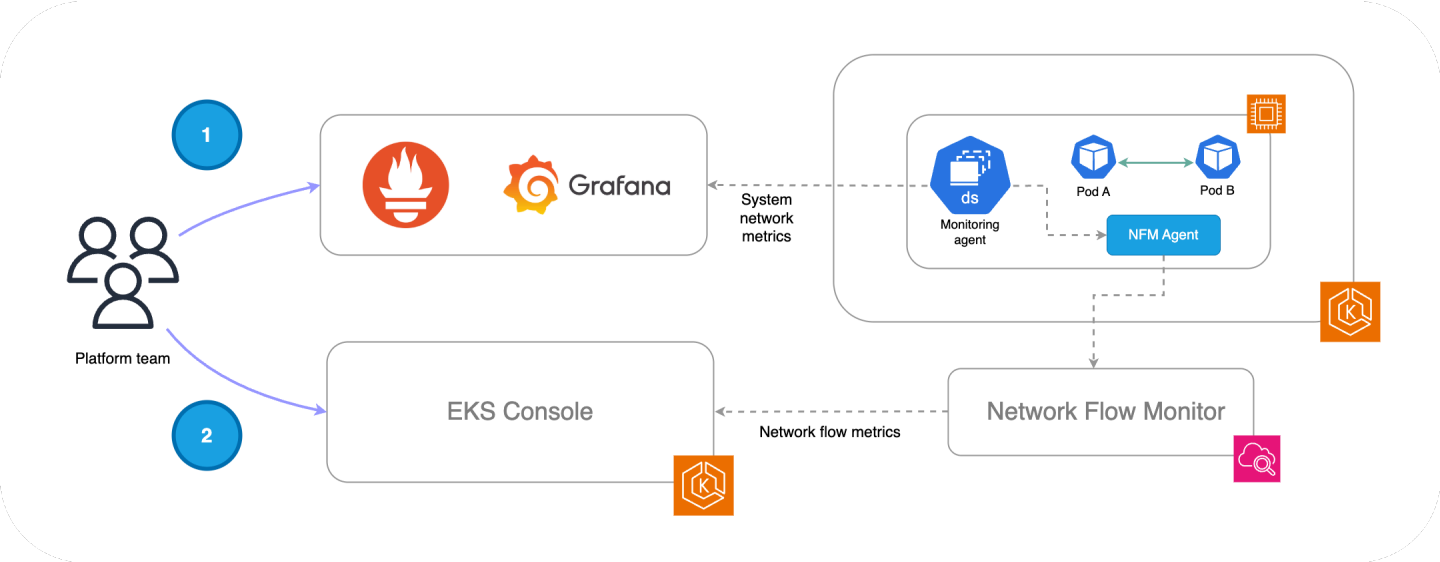
若要啟用這些指標，請在安裝程序期間使用組態變數覆寫下列環境變數（請參閱：https：//https://aws.amazon.com/blogs/containers/amazon-eks-add-ons-advanced-configuration/）：

```
OPEN_METRICS:
    Enable or disable open metrics. Disabled if not supplied
    Type: String
    Values: [“on”, “off”]
OPEN_METRICS_ADDRESS:
    Listening IP address for open metrics endpoint. Defaults to 127.0.0.1 if not
 supplied
    Type: String
OPEN_METRICS_PORT:
    Listening port for open metrics endpoint. Defaults to 80 if not supplied
    Type: Integer
    Range: [0..65535]
```

流程層級指標

此外，Network Flow Monitor 會擷取網路流程資料以及流程層級指標：重新傳輸、重新傳輸逾時和傳輸的資料。此資料由 Network Flow Monitor 處理，並在 EKS 主控台中視覺化，以呈現叢集環境中的流量，以及它如何根據這些流程層級指標執行。

下圖描述了工作流程，其中可以利用這兩種類型的指標（系統和流程層級）來獲得更多操作智慧。



1. 平台團隊可以在其監控堆疊中收集和視覺化系統指標。設定提醒後，他們可以使用 NFM 代理程式的系統指標，偵測網路異常或影響 Pod 或工作者節點的問題。
2. 下一步，平台團隊可以利用 EKS 主控台中的原生視覺化，進一步縮小調查範圍，並根據流程呈現及其相關指標加速故障診斷。

重要注意事項：從 NFM 代理程式抓取系統指標，以及將流程層級指標推送至 NFM 後端的 NFM 代理程式程序，都是獨立的程序。

支援的系統指標

重要注意事項：系統指標會以 OpenMetrics 格式匯出。

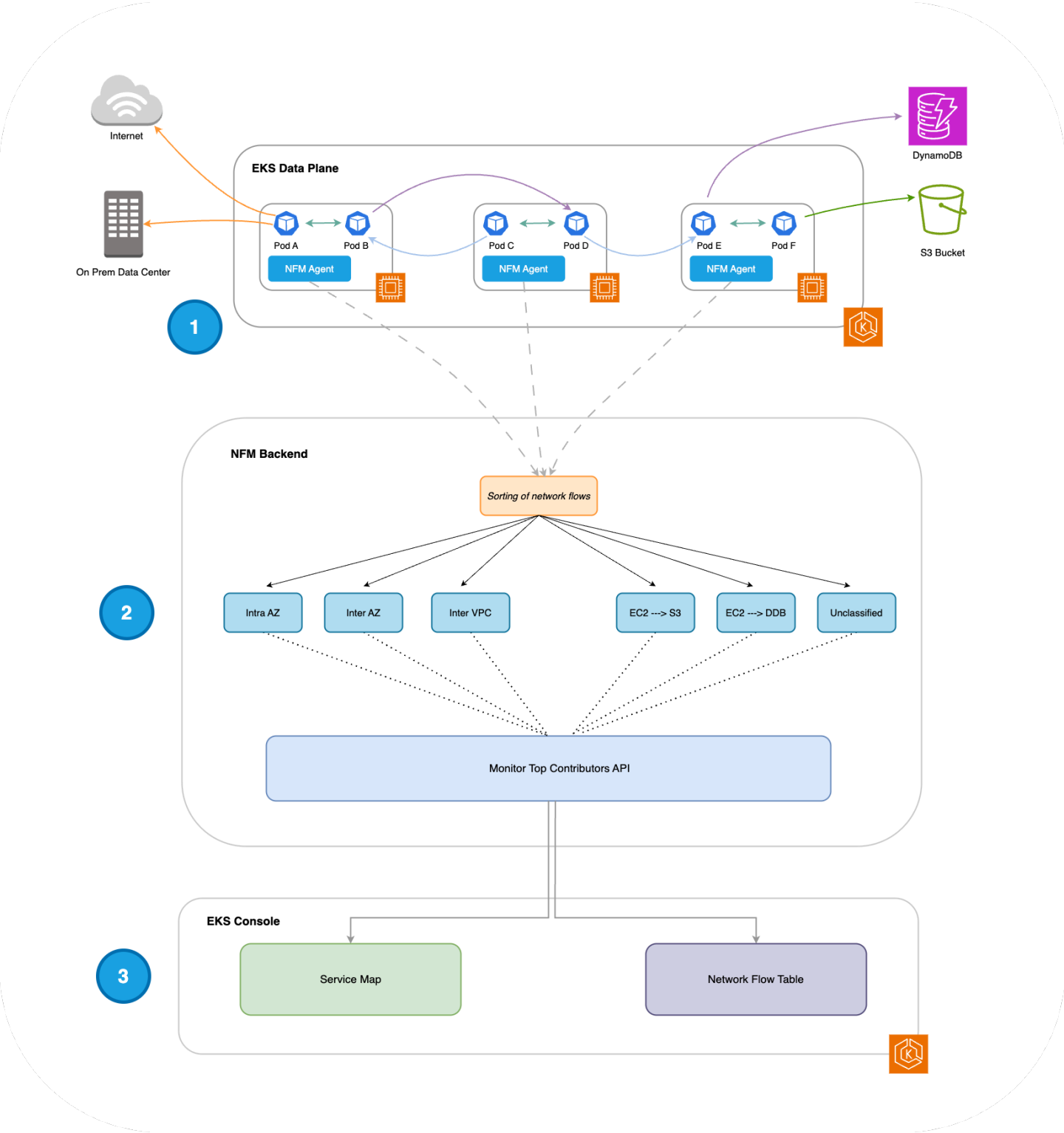
指標名稱	Type	維度	Description
輸入_流程	量測計	instance_id、iface、Pod、命名空間、節點	傳入 TCP 流量計數 (TcpPassiveOpens)
egress_flow	量測計	instance_id、iface、Pod、命名空間、節點	輸出 TCP 流量計數 (TcpActiveOpens)
inress_packets	量測計	instance_id、iface、Pod、命名空間、節點	傳入封包計數（差異）
egress_packets	量測計	instance_id、iface、Pod、命名空間、節點	輸出封包計數（差異）

指標名稱	Type	維度	Description
inress_bytes	量測計	instance_id、iface、Pod、命名空間、節點	輸入位元組計數（差異）
egress_bytes	量測計	instance_id、iface、Pod、命名空間、節點	輸出位元組計數（差異）
bw_in_allowance_exceeded	量測計	instance_id、eni、節點	由於傳入頻寬限制而排入佇列/捨棄的封包
bw_out_allowance_exceeded	量測計	instance_id、eni、節點	由於傳出頻寬限制而佇列/捨棄的封包
pps_allowance_exceeded	量測計	instance_id、eni、節點	由於雙向 PPS 限制而排入佇列/捨棄的封包
conntrack_allowance_exceeded	量測計	instance_id、eni、節點	由於連線追蹤限制而捨棄的封包
linklocal_allowance_exceeded	量測計	instance_id、eni、節點	由於本機代理服務 PPS 限制而捨棄的封包

支援的流程層級指標

指標名稱	Type	說明
TCP 重新傳輸	計數器	傳送者在傳輸期間重新傳送封包遺失或損毀的次數。
TCP 重新傳輸逾時	計數器	寄件者啟動等待期間的次數，以判斷封包是否在傳輸中遺失。
傳輸的資料（位元組）	計數器	在指定流程的來源和目的地之間傳輸的資料量。

服務映射和流程圖



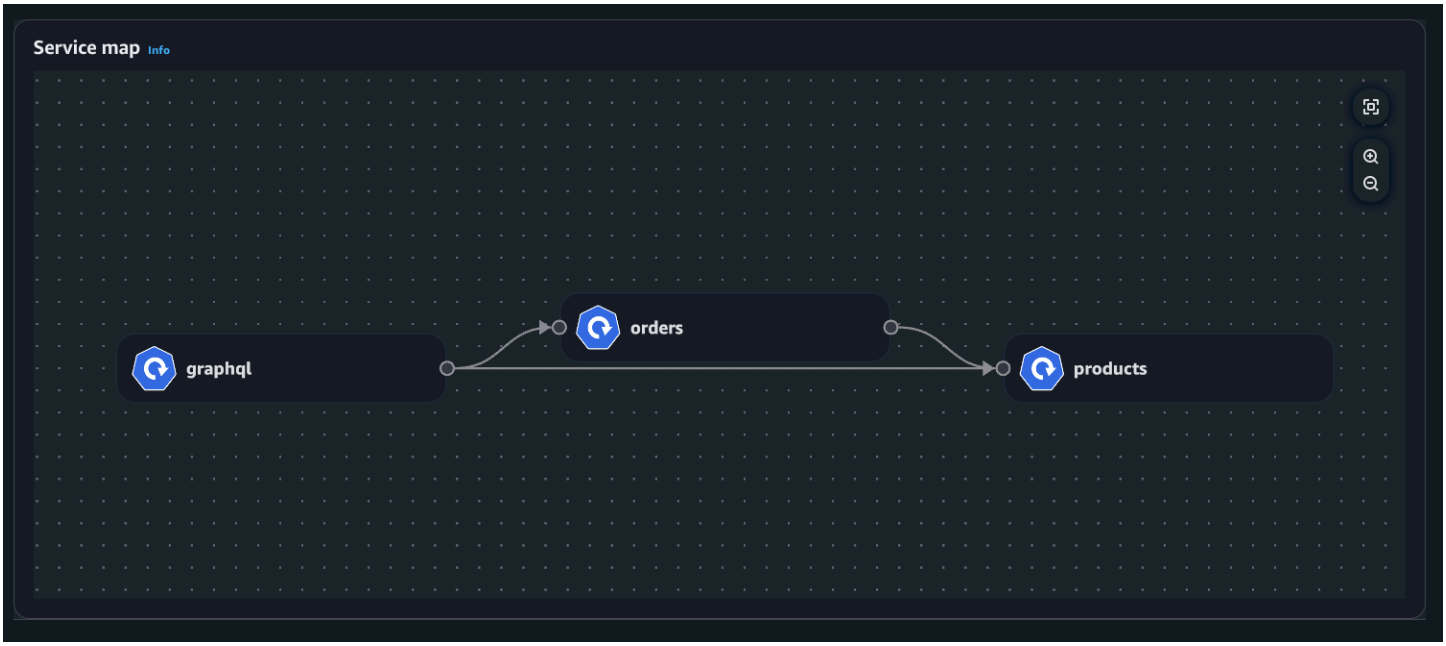
1. 安裝後，Network Flow Monitor 代理程式會在每個工作者節點上以 DaemonSet 身分執行，並每 30 秒收集前 500 個網路流程 （根據傳輸的資料量）。

2. 這些網路流程會依下列類別排序：AZ 內部、AZ 間、EC2 → S3、EC2 → DynamoDB (DDB) 和未分類。每個流程都有 3 個與其相關聯的指標：重新傳輸、重新傳輸逾時和傳輸的資料 （以位元組為單位）。
   - AZ 內部 - 相同 AZ 中 Pod 之間的網路流程
   - AZ 間 - 不同 AZs 中 Pod 之間的網路流程
   - EC2 → S3 - 從 Pod 到 S3 的網路流程
   - EC2 → DDB - 從 Pod 到 DDB 的網路流程
   - 未分類 - 從 Pod 到網際網路或內部部署的網路流程
3. 來自 Network Flow Monitor Top Contributors API 的網路流程用於為 EKS 主控台中的下列體驗提供支援：
   - 服務映射：叢集內網路流程的視覺化 （可用區域內部和可用區域間）。
   - 流程表：叢集內網路流程的表格呈現 （內部可用區和內部可用區）、從 Pod 到 AWS 服務 (EC2 → S3 和 EC2 → DDB)，以及從 Pod 到外部目的地 （未分類）。

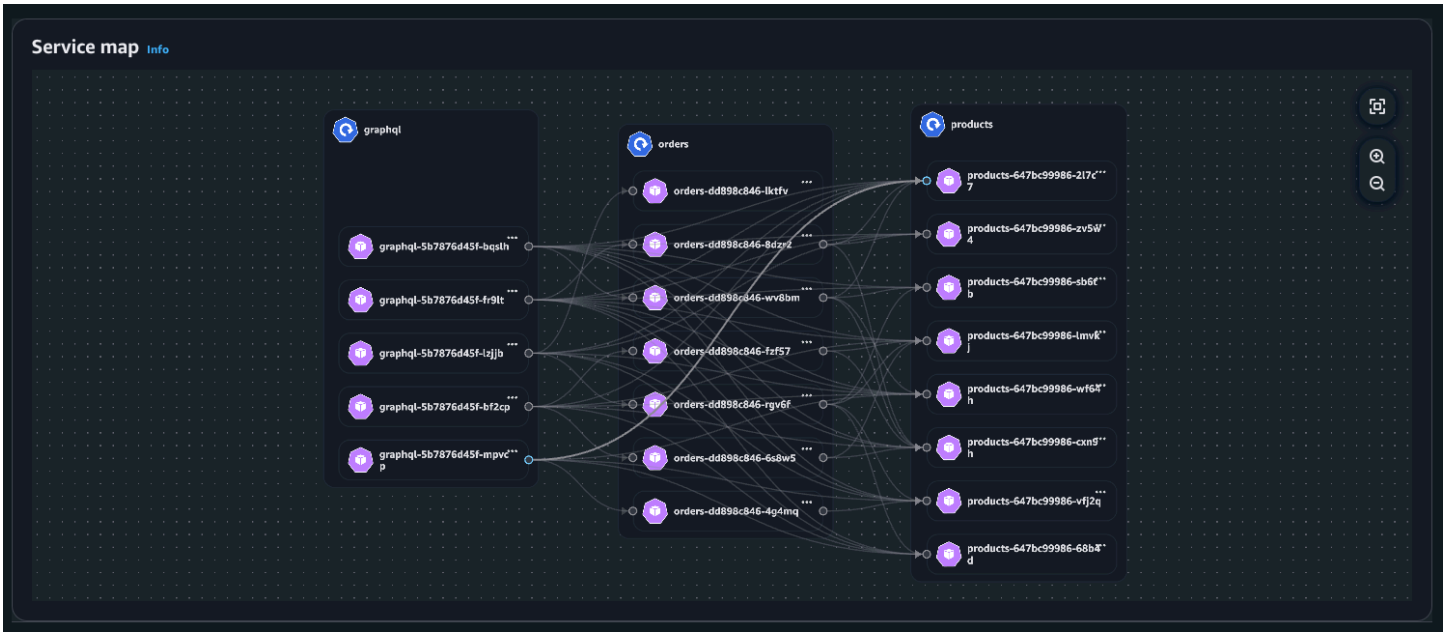
從 Top Contributors API 提取的網路流程範圍為 1 小時時間範圍，且每個類別最多可包含 500 個流程。對於服務映射，這表示在 1 小時的時間範圍內，最多可從 AZ 內部和 AZ 間流程類別取得和呈現 1000 個流程。對於流程表，這表示在 2 小時的時間範圍內，最多可以從所有 5 個網路流程類別中取得和呈現 2500 個網路流程。

範例：服務映射

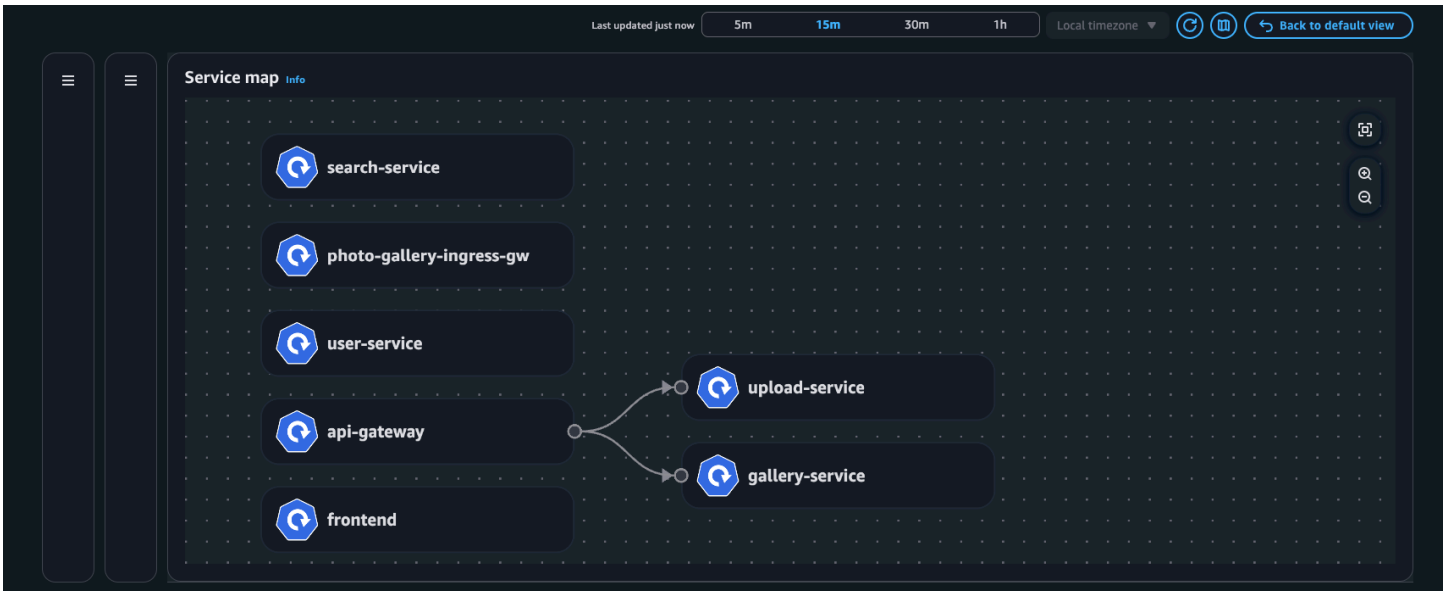
部署檢視



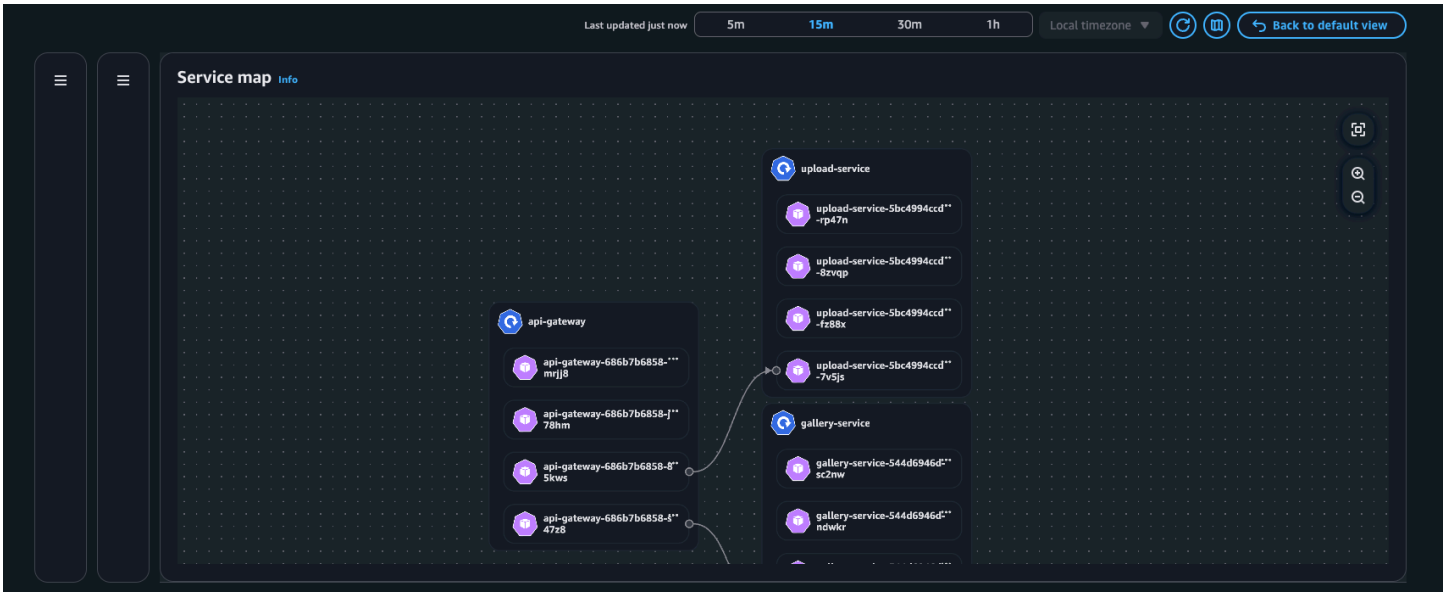
Pod 檢視



部署檢視

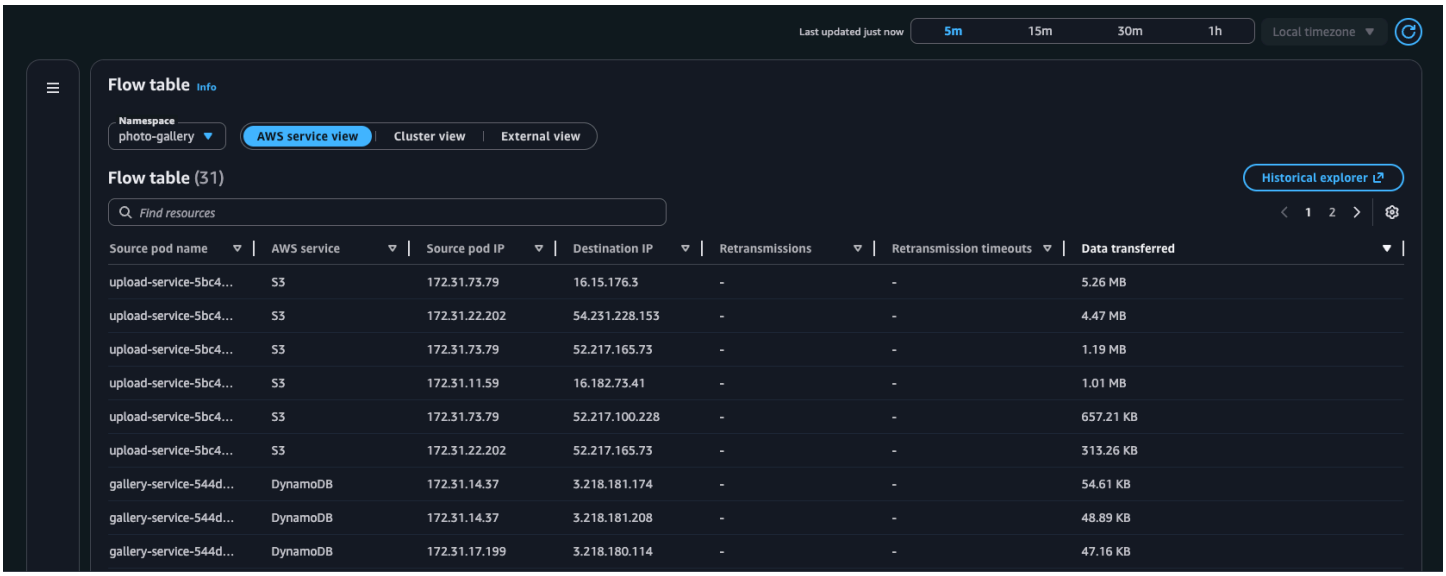


Pod 檢視

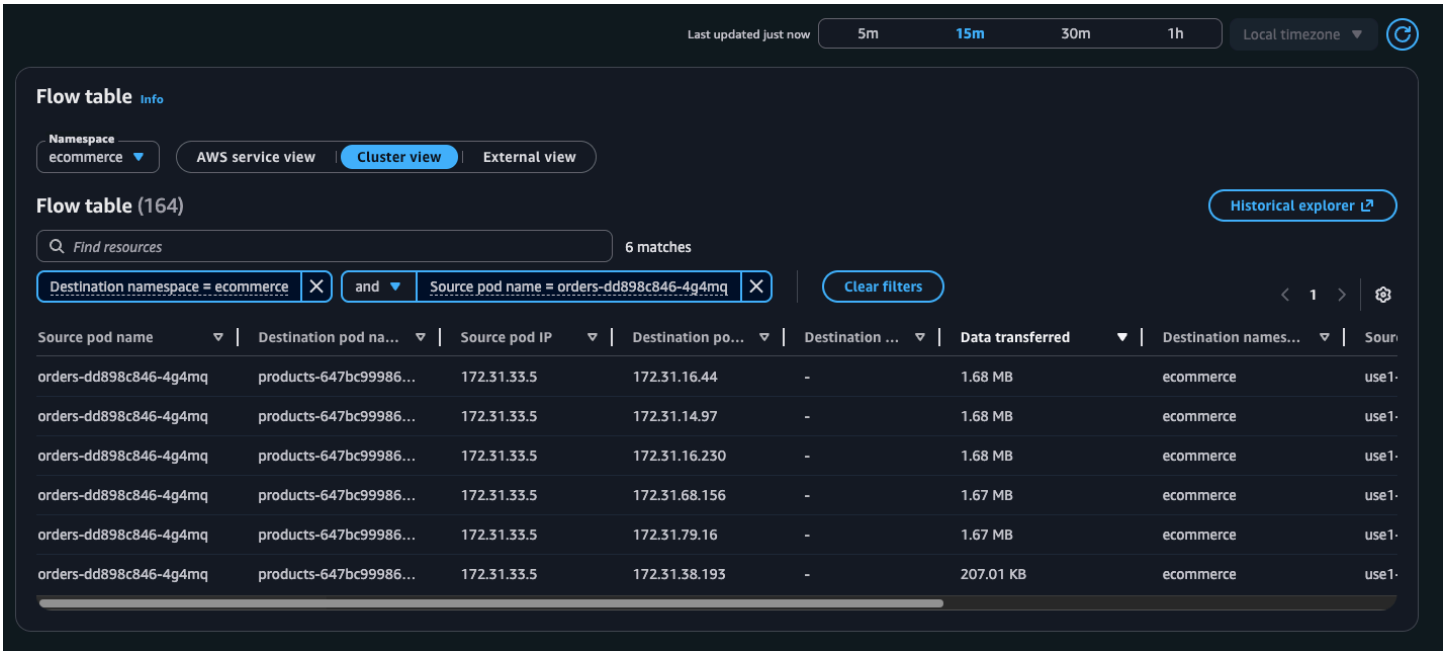


範例：流程資料表

AWS 服務檢視



叢集檢視



考量和限制

- EKS 中的容器網路可觀測性僅適用於支援網路流量監控的區域。
- 支援的系統指標採用 OpenMetrics 格式，並且可以直接從網路流量監控 (NFM) 代理程式中抓取。
- 若要使用基礎設施即程式碼 (IaC) 如 Terraform 在 EKS 中啟用容器網路可觀測性，您需要在組態中定義和建立這些相依性：NFM 範圍、NFM 監控和 NFM 代理程式。
- Network Flow Monitor 每分鐘最多支援約 500 萬個流程。這是已安裝 Network Flow Monitor 代理程式的大約 5，000 個 EC2 執行個體 (EKS 工作者節點）。在超過 5000 個執行個體上安裝代理程式可能會影響監控效能，直到有額外的容量可用為止。
- 您必須為 NFM 代理程式的 EKS 附加元件執行最低版本 1.1.0。
- 您必須使用 Terraform AWS 提供者的 v6.21.0 或更高版本，才能支援網路流量監視器資源。
- 若要使用 Pod 中繼資料擴充網路流程，您的 Pod 應該在自己的隔離網路命名空間中執行，而不是在主機網路命名空間中執行。

藉助 Prometheus 監控叢集指標

Prometheus 是湊集端點的監控和時間序列資料庫。它提供查詢、彙總和儲存收集之資料的能力。您也可以將其用於警示和警示彙總。本主題說明如何將 Prometheus 設定為受管或開放原始碼選項。監控 Amazon EKS 控制平面指標是常見的使用案例。

Amazon Managed Service for Prometheus 是與 Prometheus 相容的監控和提示服務，可讓您輕鬆地大規模監控容器化應用程式和基礎設施。這是一項全受管服務，既可自動擴展指標的擷取、儲存、查詢和提醒，也可以與 AWS 安全服務整合，以便快速且安全地存取資料。您可以使用開放原始碼 PromQL 查詢語言來查詢指標並根據指標發出提醒。此外，您還可藉助 Amazon Managed Service for Prometheus 中的提醒管理員，為關鍵提醒設定提醒規則。您隨後可將這些關鍵提醒作為通知傳送至 Amazon SNS 主題。

提供了多種不同的選項來將 Prometheus 與 Amazon EKS 配合使用：

- 您可在首次建立 Amazon EKS 叢集時，開啟 Prometheus 指標，或者針對現有的叢集建立自己的 Prometheus 湊集器。本主題將介紹這兩個選項。
- 您可使用 Helm 來部署 Prometheus。如需詳細資訊，請參閱 the section called “使用 Helm 部署”。
- 您能夠以 Prometheus 格式來檢視控制平面原始指標。如需詳細資訊，請參閱 the section called “控制平面”。

步驟 1：開啟 Prometheus 指標

> **Important**
>
> 叢集生命週期不包括適用於 Prometheus 的 Amazon Managed Service 資源，需要獨立於叢集進行維護。刪除叢集後，確認亦同時刪除任何適用的湊集器，以免產生適用的費用。如需詳細資訊，請參閱 Amazon Managed Service for Prometheus 使用者指南中的尋找及刪除湊集器。

Prometheus 透過稱為湊集的提取型模式，從您的叢集中探索並收集指標。湊集器的設定是為了從您的叢集基礎設施和容器化應用程式收集資料。當您開啟傳送 Prometheus 指標的選項時，Amazon Managed Service for Prometheus 會提供完全受管的無代理程式湊集器。

若尚未建立叢集，您可在首次建立叢集時，開啟將指標傳送至 Prometheus 的選項。在 Amazon EKS 主控台中，此選項位於建立新叢集的設定可觀測性步驟中。如需詳細資訊，請參閱 the section called “建立 叢集”。

若目前已有叢集，您可建立自己的 Prometheus 湊集器。如需在 Amazon EKS 主控台執行此動作，導覽至叢集的可觀測性索引標籤，然後選擇新增湊集器按鈕。若您想透過 AWS API 或 AWS CLI 執行此動作，請參閱適用於 Prometheus 的 Amazon Managed Service 使用者指南中的建立湊集器。

透過 Amazon EKS 主控台建立湊集器時，以下選項可供使用。

湊集器別名

(選用) 輸入湊集器的唯一別名。

目的地

選擇 Amazon Managed Service for Prometheus 工作區。工作區是專門用於儲存和查詢 Prometheus 指標的邏輯空間。使用此工作區，您能夠跨具有該工作區存取權的帳戶檢視 Prometheus 指標。建立新工作區選項會讓 Amazon EKS 知道可以使用您提供的工作區別名來代表您建立工作區。使用選取現有工作區選項，您可以從下拉式清單中選取現有的工作區。如需工作區的詳細資訊，請參閱《Amazon Managed Service for Prometheus 使用者指南》中的管理工作區。

服務存取

本節總結您在傳送 Prometheus 指標時授予的許可：

- 允許 Amazon Managed Service for Prometheus 描述湊集的 Amazon EKS 叢集
- 允許遠端寫入 Amazon 受管 Prometheus 工作區

如果 `AmazonManagedScraperRole` 已經存在，則湊集器會使用它。選擇 `AmazonManagedScraperRole` 連結以查看許可詳細資訊。如果 `AmazonManagedScraperRole` 尚未存在，請選擇檢視許可詳細資訊連結，以查看您透過傳送 Prometheus 指標來授予的特定許可。

子網路

修改湊集器會視需繼承的子網路。若您需要新增呈現灰色的子網路選項，請返回建立叢集指定聯網步驟。

湊集器組態

視需要修改 YAML 格式的湊集器組態。若要執行這項操作，請使用表單或上傳取代 YAML 檔案。如需詳細資訊，請參閱《Amazon Managed Service for Prometheus 使用者指南》中的湊集器組態。

Amazon Managed Service for Prometheus 是指作為 AWS 受管收集器與叢集一起建立的無代理程式湊集器。若要了解 AWS 受管收集器的相關詳細資訊，請參閱適用於 Prometheus 的 Amazon Managed Service 使用者指南中的透過 AWS 受管收集器來擷取指標。

> **Important**
>
> - 若使用 AWS CLI 或 AWS API 建立 Prometheus 湊集器，需要調整其組態，以授予湊集器叢集內許可。如需詳細資訊，請參閱《Amazon Managed Service for Prometheus 使用者指南》中的設定 Amazon EKS 叢集。
> - 若在 2024 年 11 月 11 日之前建立 Prometheus 湊集器，且使用 `aws-auth ConfigMap` 而非存取項目，您需要進行更新，才可透過 Amazon EKS 叢集控制平面來存取額外的指標。若要了解更新的組態，請參閱適用於 Prometheus 的 Amazon Managed Service 使用者指南中的手動設定 Amazon EKS 的湊集器存取權。

步驟 2：使用 Prometheus 指標

若要了解如何在為叢集開啟 Prometheus 指標後使用該指標的相關詳細資訊，請參閱適用於 Prometheus 的 Amazon Managed Service 使用者指南。

步驟 3：管理 Prometheus 湊集器

如需管理湊集器，在 Amazon EKS 主控台中，選擇可觀測性索引標籤。表格會顯示叢集的湊集器清單，包含湊集器 ID、別名、狀態和建立日期等資訊。您可新增更多湊集器、編輯湊集器、刪除湊集器，或者檢視目前湊集器的相關詳細資訊。

如需檢視湊集器的相關詳細資訊，請選擇湊集器 ID 連結。舉例來說，您可檢視 ARN、環境、工作區 ID、IAM 角色、組態，以及聯網資訊。您可使用湊集器 ID 作為輸入到 Amazon Managed Service for Prometheus API 操作 (例如 `DescribeScraper`、`UpdateScraper` 及 `DeleteScraper`)。若要了解使用 Prometheus API 的相關詳細資訊，請參閱適用於 Prometheus 的 Amazon Managed Service API 參考。

使用 Helm 部署 Prometheus

作為使用 Amazon Managed Service for Prometheus 的替代方案，您可以使用 Helm 將 Prometheus 部署到您的叢集中。如果您已經安裝了 Helm，可以使用 `helm version` 命令檢查您的版本。Helm 是 Kubernetes 叢集的套件管理工具。如需有關 Helm 及如何安裝它的詳細資訊，請參閱 the section called “使用 Helm 部署應用程式”。

在您為 Amazon EKS 叢集設定 Helm 之後，即可用它來依照下列步驟部署 Prometheus。

1. 建立 Prometheus 命名空間。

```
kubectl create namespace prometheus
```

2. 新增 prometheus-community 圖表儲存庫。

```
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
```

3. 部署 Prometheus。

```
helm upgrade -i prometheus prometheus-community/prometheus \
    --namespace prometheus \
    --set alertmanager.persistence.storageClass="gp2" \
    --set server.persistentVolume.storageClass="gp2"
```

> **Note**
>
> 如果您在執行此命令Error: failed to download "stable/prometheus" (hint: running helm repo update may help)時收到錯誤，請執行 helm repo update prometheus-community，然後再次嘗試執行步驟 3 命令。

如果收到錯誤 Error: rendered manifests contain a resource that already exists，請執行 helm uninstall *your-release-name* -n *namespace* ，然後嘗試再次執行步驟 3 命令。

4. 確認 prometheus 命名空間中的所有裝置皆處於 READY 狀態。

```
kubectl get pods -n prometheus
```

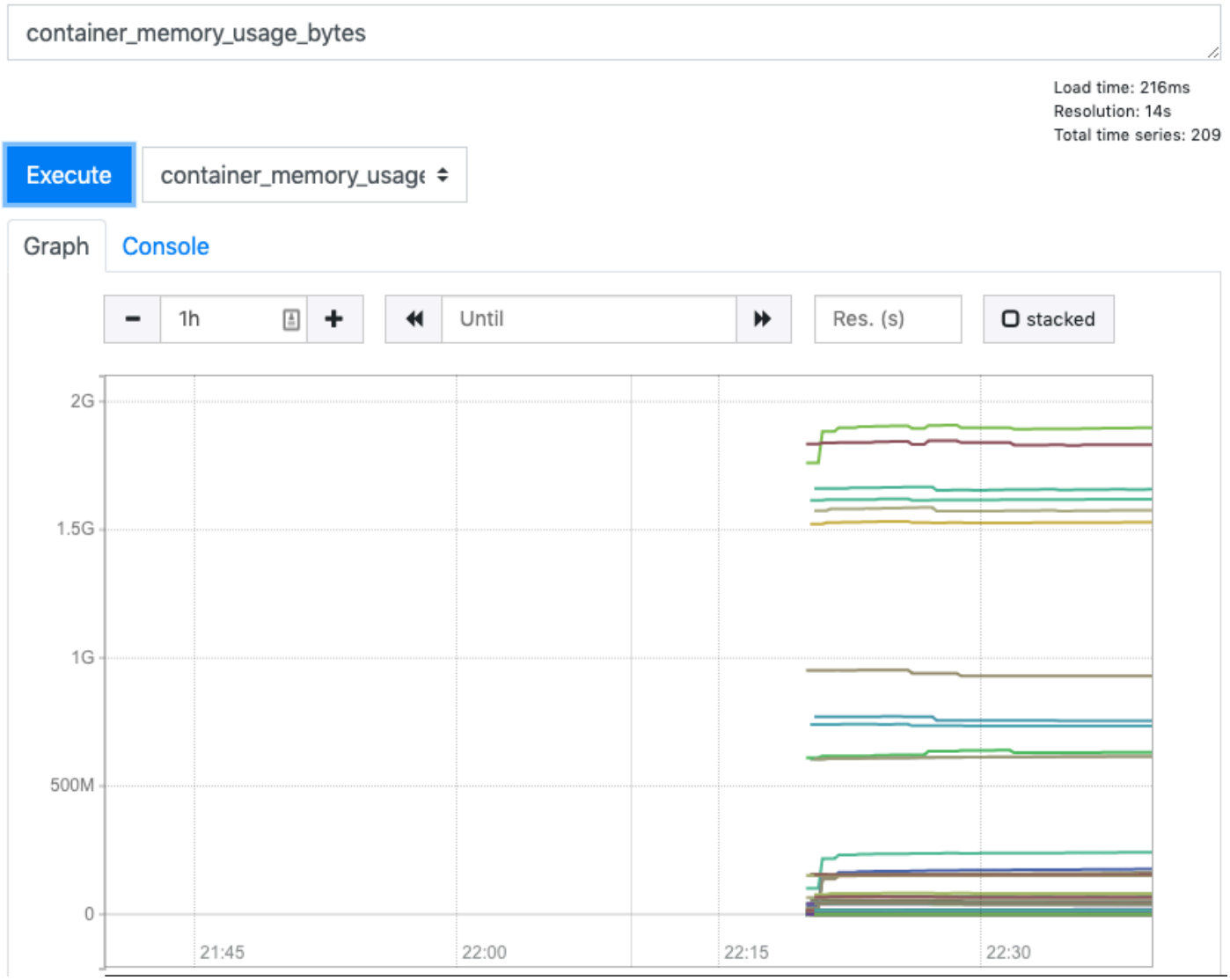
範例輸出如下。

```
NAME                                             READY   STATUS    RESTARTS   AGE
prometheus-alertmanager-59b4c8c744-r7bgp         1/2     Running   0          48s
prometheus-kube-state-metrics-7cfd87cf99-jkz2f   1/1     Running   0          48s
prometheus-node-exporter-jcjqz                   1/1     Running   0          48s
prometheus-node-exporter-jxv2h                   1/1     Running   0          48s
prometheus-node-exporter-vbdks                   1/1     Running   0          48s
prometheus-pushgateway-76c444b68c-82tnw          1/1     Running   0          48s
prometheus-server-775957f748-mmht9               1/2     Running   0          48s
```

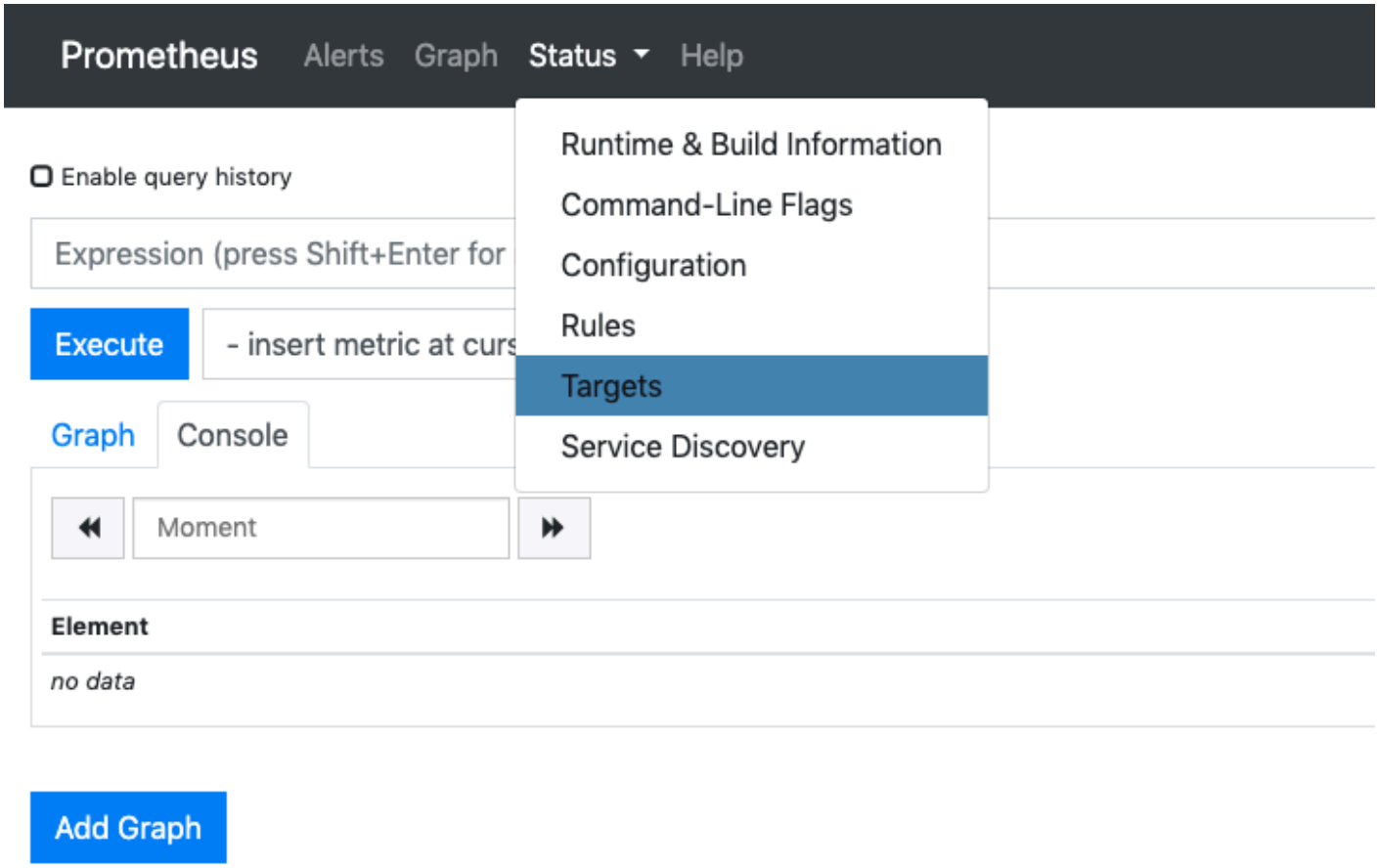
5. 使用 `kubectl` 將 Prometheus 主控台的連接埠轉送到本機機器。

```
kubectl --namespace=prometheus port-forward deploy/prometheus-server 9090
```

6. 將 Web 瀏覽器連接至 `http://localhost:9090` 以檢視 Prometheus 主控台。
7. 從 – insert metric at cursor (- 在游標處插入指標) 功能表中選擇一個指標，然後選擇 Execute (執行)。選擇 Graph (圖表) 標籤以顯示一段時間內的指標。下圖顯示一段時間內的 `container_memory_usage_bytes`。



8. 選擇頂部導覽列中的 Status (狀態)，然後選擇 Targets (目標)。



將會顯示使用服務探索連接至 Prometheus 的所有 Kubernetes 端點。

擷取採用 Prometheus 格式的控制平面原始指標

Kubernetes 控制平面公開眾多採用 Prometheus 格式的指標。這些指標對於監視和分析非常有用。透過指標端點在內部公開這些指標，並且可在完全不部署 Prometheus 的情況下存取。然而，部署 Prometheus 可更輕鬆地分析一段時間內的指標。

如需檢視原始指標輸出，可取代 `endpoint`，然後執行以下命令。

```
kubectl get --raw endpoint
```

此命令可讓您傳遞任何端點路徑並傳回原始回應。輸出會逐行列示不同的指標，每一行都包括指標名稱、標籤和值。

```
metric_name{tag="value"[,...]} value
```

透過 API 伺服器擷取指標

一般 API 伺服器端點在 Amazon EKS 控制平面公開。此端點主要在查看特定指標時有用。

```
kubectl get --raw /metrics
```

範例輸出如下。

```
[...]
# HELP rest_client_requests_total Number of HTTP requests, partitioned by status code,
 method, and host.
# TYPE rest_client_requests_total counter
rest_client_requests_total{code="200",host="127.0.0.1:21362",method="POST"} 4994
rest_client_requests_total{code="200",host="127.0.0.1:443",method="DELETE"} 1
rest_client_requests_total{code="200",host="127.0.0.1:443",method="GET"} 1.326086e+06
rest_client_requests_total{code="200",host="127.0.0.1:443",method="PUT"} 862173
rest_client_requests_total{code="404",host="127.0.0.1:443",method="GET"} 2
rest_client_requests_total{code="409",host="127.0.0.1:443",method="POST"} 3
rest_client_requests_total{code="409",host="127.0.0.1:443",method="PUT"} 8
# HELP ssh_tunnel_open_count Counter of ssh tunnel total open attempts
# TYPE ssh_tunnel_open_count counter
ssh_tunnel_open_count 0
# HELP ssh_tunnel_open_fail_count Counter of ssh tunnel failed open attempts
# TYPE ssh_tunnel_open_fail_count counter
ssh_tunnel_open_fail_count 0
```

此原始輸出會逐字傳回 API 伺服器公開的內容。

透過 `metrics.eks.amazonaws.com` 擷取控制平面指標

若是 Kubernetes `1.28` 版及更新版本的叢集，Amazon EKS 亦會在 API 群組 `metrics.eks.amazonaws.com` 項下公開指標。這些指標包括 `kube-scheduler` 及 `kube-controller-manager` 等控制平面元件。

> **Note**
>
> 若您的 Webhook 組態可能會阻止在叢集上建立新的 `APIService` 資源 `v1.metrics.eks.amazonaws.com`，則可能無法使用指標端點功能。您可藉由搜尋 `v1.metrics.eks.amazonaws.com` 關鍵字，在 `kube-apiserver` 稽核日誌中進行確認。

擷取 kube-scheduler 指標

如需擷取 kube-scheduler 指標，請使用以下命令。

```
kubectl get --raw "/apis/metrics.eks.amazonaws.com/v1/ksh/container/metrics"
```

範例輸出如下。

```
# TYPE scheduler_pending_pods gauge
scheduler_pending_pods{queue="active"} 0
scheduler_pending_pods{queue="backoff"} 0
scheduler_pending_pods{queue="gated"} 0
scheduler_pending_pods{queue="unschedulable"} 18
# HELP scheduler_pod_scheduling_attempts [STABLE] Number of attempts to successfully
 schedule a pod.
# TYPE scheduler_pod_scheduling_attempts histogram
scheduler_pod_scheduling_attempts_bucket{le="1"} 79
scheduler_pod_scheduling_attempts_bucket{le="2"} 79
scheduler_pod_scheduling_attempts_bucket{le="4"} 79
scheduler_pod_scheduling_attempts_bucket{le="8"} 79
scheduler_pod_scheduling_attempts_bucket{le="16"} 79
scheduler_pod_scheduling_attempts_bucket{le="+Inf"} 81
[...]
```

由於傳回的資料量可能很大，因此有額外的kube-scheduler指標端點可供擷取 kube_pod_resource_request和 kube_pod_resource_limit指標。若要擷取這些指標，請使用下列命令。

```
kubectl get --raw "/apis/metrics.eks.amazonaws.com/v1/ksh/container/resourcemetrics"
```

範例輸出如下。

```
# HELP kube_pod_resource_limit [STABLE] Resources limit for workloads on the cluster,
 broken down by pod. This shows the resource usage the scheduler and kubelet expect per
 pod for resources along with the unit for the resource if any.
# TYPE kube_pod_resource_limit gauge
kube_pod_resource_limit{namespace="kube-system",node="ip-192-168-87-3.us-
west-2.compute.internal",pod="coredns-7bf648ff5d-
dj4ss",priority="2000000000",resource="memory",scheduler="default-
scheduler",unit="bytes"} 1.7825792e+08
```

```
# HELP kube_pod_resource_request [STABLE] Resources requested by workloads on the
 cluster, broken down by pod. This shows the resource usage the scheduler and kubelet
 expect per pod for resources along with the unit for the resource if any.
# TYPE kube_pod_resource_request gauge
kube_pod_resource_request{namespace="kube-system",node="ip-192-168-87-3.us-
west-2.compute.internal",pod="aws-node-
x7znh",priority="2000001000",resource="cpu",scheduler="default-scheduler",unit="cores"}
 0.05
kube_pod_resource_request{namespace="kube-system",node="ip-192-168-87-3.us-
west-2.compute.internal",pod="coredns-7bf648ff5d-
dj4ss",priority="2000000000",resource="cpu",scheduler="default-scheduler",unit="cores"}
 0.1
[...]
```

擷取 **kube-controller-manager** 指標

如需擷取 kube-controller-manager 指標，請使用以下命令。

```
kubectl get --raw "/apis/metrics.eks.amazonaws.com/v1/kcm/container/metrics"
```

範例輸出如下。

```
[...]
workqueue_work_duration_seconds_sum{name="pvprotection"} 0
workqueue_work_duration_seconds_count{name="pvprotection"} 0
workqueue_work_duration_seconds_bucket{name="replicaset",le="1e-08"} 0
workqueue_work_duration_seconds_bucket{name="replicaset",le="1e-07"} 0
workqueue_work_duration_seconds_bucket{name="replicaset",le="1e-06"} 0
workqueue_work_duration_seconds_bucket{name="replicaset",le="9.999999999999999e-06"} 0
workqueue_work_duration_seconds_bucket{name="replicaset",le="9.999999999999999e-05"} 19
workqueue_work_duration_seconds_bucket{name="replicaset",le="0.001"} 109
workqueue_work_duration_seconds_bucket{name="replicaset",le="0.01"} 139
workqueue_work_duration_seconds_bucket{name="replicaset",le="0.1"} 181
workqueue_work_duration_seconds_bucket{name="replicaset",le="1"} 191
workqueue_work_duration_seconds_bucket{name="replicaset",le="10"} 191
workqueue_work_duration_seconds_bucket{name="replicaset",le="+Inf"} 191
workqueue_work_duration_seconds_sum{name="replicaset"} 4.265655885000002
[...]
```

了解排程器與控制器管理員指標

下面的資料表列示了可用於 Prometheus 樣式湊集的排程器與控制器管理員指標。若要了解這些指標的相關詳細資訊，請參閱 Kubernetes 文件中的 Kubernetes 指標參考。

指標	控制平面元件	Description
scheduler_pending_pods	排程器	等候排程到節點上執行的 Pod 數目。
scheduler_schedule_attempts_total	排程器	排程 Pod 的嘗試次數。
scheduler_preemption_attempts_total	排程器	排程器藉由移出較低優先級 Pod 以排程較高優先級 Pod 的嘗試次數。
scheduler_preemption_victims	排程器	已經選取要移出，以便為較高優先級 Pod 騰出空間的 Pod 數目。
scheduler_pod_scheduling_attempts	排程器	已成功排程 Pod 的嘗試次數。
scheduler_scheduling_attempt_duration_seconds	排程器	指示排程器能夠依據資源可用性及排程規則等各種因素，尋找執行 Pod 的適當位置的速度快慢程度。
scheduler_pod_scheduling_sli_duration_seconds	排程器	自 Pod 進入排程佇列開始，正在排程的 Pod 的端對端延遲。這可能會涉及多次排程嘗試。
kube_pod_resource_limit	排程器	叢集上工作負載的資源限制，依 Pod 細分。這會顯示排程器和 kubelet 針對資源每個 Pod 預期的資源用量，以及如果有的話的資源單位。
kube_pod_resource_request	排程器	叢集上工作負載請求的資源，依 Pod 細分。這會顯示排程器和 kubelet 針對資源每個 Pod

指標	控制平面元件	Description
		預期的資源用量，以及如果有的話的資源單位。
cronjob_controller_job_creation_skew_duration_seconds	控制器管理員	排程執行 cronjob 的時間，及建立相應任務的時間。
workqueue_depth	控制器管理員	佇列目前的深度。
workqueue_adds_total	控制器管理員	工作佇列處理的新增項目總數。
workqueue_queue_duration_seconds	控制器管理員	發出請求之前，項目保持在工作佇列的時間 (秒)。
workqueue_work_duration_seconds	控制器管理員	處理工作佇列中的項目所需的時間 (秒)。

部署 Prometheus 湊集器以持續湊集指標

若要部署 Prometheus 湊集器以持續湊集指標，請使用下列組態：

```
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-conf
data:
  prometheus.yml: |-
    global:
      scrape_interval: 30s
    scrape_configs:
    # apiserver metrics
    - job_name: apiserver-metrics
      kubernetes_sd_configs:
      - role: endpoints
      scheme: https
      tls_config:
        ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
        insecure_skip_verify: true
```

```
      bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
      relabel_configs:
      - source_labels:
          [
            __meta_kubernetes_namespace,
            __meta_kubernetes_service_name,
            __meta_kubernetes_endpoint_port_name,
          ]
        action: keep
        regex: default;kubernetes;https
    # Scheduler metrics
    - job_name: 'ksh-metrics'
      kubernetes_sd_configs:
      - role: endpoints
      metrics_path: /apis/metrics.eks.amazonaws.com/v1/ksh/container/metrics
      scheme: https
      tls_config:
        ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
        insecure_skip_verify: true
      bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
      relabel_configs:
      - source_labels:
          [
            __meta_kubernetes_namespace,
            __meta_kubernetes_service_name,
            __meta_kubernetes_endpoint_port_name,
          ]
        action: keep
        regex: default;kubernetes;https
    # Controller Manager metrics
    - job_name: 'kcm-metrics'
      kubernetes_sd_configs:
      - role: endpoints
      metrics_path: /apis/metrics.eks.amazonaws.com/v1/kcm/container/metrics
      scheme: https
      tls_config:
        ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
        insecure_skip_verify: true
      bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
      relabel_configs:
      - source_labels:
          [
            __meta_kubernetes_namespace,
            __meta_kubernetes_service_name,
```

```
            __meta_kubernetes_endpoint_port_name,
          ]
        action: keep
        regex: default;kubernetes;https
---
apiVersion: v1
kind: Pod
metadata:
  name: prom-pod
spec:
  containers:
  - name: prom-container
    image: prom/prometheus
    ports:
    - containerPort: 9090
    volumeMounts:
    - name: config-volume
      mountPath: /etc/prometheus/
  volumes:
  - name: config-volume
    configMap:
      name: prometheus-conf
```

Pod 存取全新的指標端點需要下列許可。

```
{
  "effect": "allow",
  "apiGroups": [
    "metrics.eks.amazonaws.com"
  ],
  "resources": [
    "kcm/metrics",
    "ksh/metrics"
  ],
  "verbs": [
    "get"
  ] },
```

若要修補使用的角色，您可使用以下命令。

```
kubectl patch clusterrole <role-name> --type=json -p='[
  {
    "op": "add",
```

```
    "path": "/rules/-",
    "value": {
      "verbs": ["get"],
      "apiGroups": ["metrics.eks.amazonaws.com"],
      "resources": ["kcm/metrics", "ksh/metrics"]
    }
  }
]'
```

然後，您可用 Prometheus 湊集器的連接埠代理本機連接埠，以檢視 Prometheus 儀表板。

```
kubectl port-forward pods/prom-pod 9090:9090
```

若是 Amazon EKS 叢集，核心 Kubernetes 控制平面指標亦可擷取到 AWS/EKS 命名空間項下的 Amazon CloudWatch 指標。如需檢視這些指標，開啟 CloudWatch 主控台，然後從左側導覽窗格中選取全部指標。在指標選取頁面上，選擇叢集的 AWS/EKS 命名空間與指標維度。

使用 Amazon CloudWatch 監測叢集資料

Amazon CloudWatch 是一項監控服務，可從您的雲端資源收集指標和日誌。使用版本 1.28 及更高版本的新叢集時，CloudWatch 免費提供一些基本的 Amazon EKS 指標。然而，當使用 CloudWatch 可觀測性運算子作為 Amazon EKS 附加元件時，您可以獲得增強的可觀測性功能。

Amazon CloudWatch 中的基本指標

對於 Kubernetes 版本 1.28 及更高版本的叢集，您可以在 AWS/EKS 命名空間中免費獲得 CloudWatch 代售指標。下表列出了支援版本可用的基本指標清單。列出的每個指標的頻率均為一分鐘。

指標名稱	Description
apiserver_flowcontrol_current_executing_seats	目前用於執行 API 請求的座位數量。座位配置取決於 Kubernetes API Priority and Fairness 功能中的 priority_level 和 flow_schema 組態。 單位：計數 有效的統計資訊：總和

指標名稱	Description
scheduler_schedule_attempts_total	排程器在給定期間內嘗試在叢集中排程 Pod 的總次數。此指標有助於監控排程器的工作負載，並可指示排程壓力或 Pod 放置的潛在問題。 單位：計數 有效的統計資訊：總和
scheduler_schedule_attempts_SCHEDULED	排程器在給定期間內成功將 Pod 排程到叢集中節點的次數。 單位：計數 有效的統計資訊：總和
scheduler_schedule_attempts_UNSCHEDULABLE	在給定期間內，由於有效限制 (例如節點上的 CPU 或記憶體不足) 而導致 Pod 無法排程的嘗試次數。 單位：計數 有效的統計資訊：總和
scheduler_schedule_attempts_ERROR	在給定期間內，由於排程器本身的內部問題 (例如 API 伺服器連線問題) 而導致 Pod 排程失敗的嘗試次數。 單位：計數 有效的統計資訊：總和
scheduler_pending_pods	在給定期間內，排程器需要在叢集中排程的待處理 Pod 總數。 單位：計數 有效的統計資訊：總和

指標名稱	Description
scheduler_pending_pods_ACTIVEQ	在給定期間內，位於 activeQ 中等待在叢集內排程的待處理 Pod 數量。 單位：計數 有效的統計資訊：總和
scheduler_pending_pods_UNSCHEDULABLE	排程器嘗試排程但失敗，並保持在不可排程狀態以進行重試的待處理 Pod 數量。 單位：計數 有效的統計資訊：總和
scheduler_pending_pods_BACKOFF	處於退避狀態、位於 backoffQ 中等待其退避期到期的待處理 Pod 數量。 單位：計數 有效的統計資訊：總和
scheduler_pending_pods_GATED	當前處於門控狀態等待的待處理 Pod 數量，這些 Pod 在滿足所需條件之前無法被排程。 單位：計數 有效的統計資訊：總和
apiserver_request_total	叢集中所有 API 伺服器發出的 HTTP 請求數量。 單位：計數 有效的統計資訊：總和

指標名稱	Description
apiserver_request_total_4XX	對叢集中所有 API 伺服器發出的、導致 4XX (用戶端錯誤) 狀態碼的 HTTP 請求數量。 單位：計數 有效的統計資訊：總和
apiserver_request_total_429	對叢集中所有 API 伺服器發出的、導致 429 狀態碼的 HTTP 請求數量，該狀態碼在用戶端超過速率限制閾值時發生。 單位：計數 有效的統計資訊：總和
apiserver_request_total_5XX	對叢集中所有 API 伺服器發出的、導致 5XX (伺服器錯誤) 狀態碼的 HTTP 請求數量。 單位：計數 有效的統計資訊：總和
apiserver_request_total_LIST_PODS	對叢集中所有 API 伺服器發出的 LIST Pod 請求數量。 單位：計數 有效的統計資訊：總和
apiserver_request_duration_seconds_PUT_P99	根據叢集中所有 API 伺服器的所有請求，計算得出的 PUT 請求延遲的第 99 個百分位數。表示 99% 的 PUT 請求在此響應時間內完成。 單位：秒 有效的統計資訊：平均

指標名稱	Description
apiserver_request_duration_seconds_PATCH_P99	根據叢集中所有 API 伺服器的所有請求，計算得出的 PATCH 請求延遲的第 99 個百分位數。表示 99% 的 PATCH 請求在此響應時間內完成。 單位：秒 有效的統計資訊：平均
apiserver_request_duration_seconds_POST_P99	根據叢集中所有 API 伺服器的所有請求，計算得出的 POST 請求延遲的第 99 個百分位數。表示 99% 的 POST 請求在此響應時間內完成。 單位：秒 有效的統計資訊：平均
apiserver_request_duration_seconds_GET_P99	根據叢集中所有 API 伺服器的所有請求，計算得出的 GET 請求延遲的第 99 個百分位數。表示 99% 的 GET 請求在此響應時間內完成。 單位：秒 有效的統計資訊：平均
apiserver_request_duration_seconds_LIST_P99	根據叢集中所有 API 伺服器的所有請求，計算得出的 LIST 請求延遲的第 99 個百分位數。表示 99% 的 LIST 請求在此響應時間內完成。 單位：秒 有效的統計資訊：平均
apiserver_request_duration_seconds_DELETE_P99	根據叢集中所有 API 伺服器的所有請求，計算得出的 DELETE 請求延遲的第 99 個百分位數。表示 99% 的 DELETE 請求在此響應時間內完成。 單位：秒 有效的統計資訊：平均

指標名稱	Description
apiserver_current_inflight_requests_MUTATING	叢集中所有 API 伺服器目前正在處理的變更請求 (POST、PUT、DELETE、PATCH) 數量。此指標表示正在進行且尚未完成處理的請求。 單位：計數 有效的統計資訊：總和
apiserver_current_inflight_requests_READONLY	叢集中所有 API 伺服器目前正在處理的唯讀請求 (GET、LIST) 數量。此指標表示正在進行且尚未完成處理的請求。 單位：計數 有效的統計資訊：總和
apiserver_admission_webhook_request_total	叢集中所有 API 伺服器發出的准入 Webhook 請求數量。 單位：計數 有效的統計資訊：總和
apiserver_admission_webhook_request_total_ADMIT	叢集中所有 API 伺服器發出的變更准入 Webhook 請求數量。 單位：計數 有效的統計資訊：總和
apiserver_admission_webhook_request_total_VALIDATING	叢集中所有 API 伺服器發出的驗證准入 Webhook 請求數量。 單位：計數 有效的統計資訊：總和

指標名稱	Description
apiserver_admission_webhook_rejection_count	叢集中所有 API 伺服器發出的被拒絕的准入 Webhook 請求數量。 單位：計數 有效的統計資訊：總和
apiserver_admission_webhook_rejection_count_ADMIT	叢集中所有 API 伺服器發出的被拒絕的變更准入 Webhook 請求數量。 單位：計數 有效的統計資訊：總和
apiserver_admission_webhook_rejection_count_VALIDATING	叢集中所有 API 伺服器發出的被拒絕的驗證准入 Webhook 請求數量。 單位：計數 有效的統計資訊：總和
apiserver_admission_webhook_admission_duration_seconds	根據叢集中所有 API 伺服器的所有請求，計算得出的第三方准入 Webhook 請求延遲的第 99 個百分位數。表示 99% 的第三方准入 Webhook 請求在此響應時間內完成。 單位：秒 有效的統計資訊：平均
apiserver_admission_webhook_admission_duration_seconds_ADMIT_P99	根據叢集中所有 API 伺服器的所有請求，計算得出的第三方變更准入 Webhook 請求延遲的第 99 個百分位數。表示 99% 的第三方變更准入 Webhook 請求在此響應時間內完成。 單位：秒 有效的統計資訊：平均

指標名稱	Description
apiserver_admission_webhook_admission_duration_seconds_VALIDATING_P99	根據叢集中所有 API 伺服器的所有請求，計算得出的第三方驗證准入 Webhook 請求延遲的第 99 個百分位數。表示 99% 的第三方驗證准入 Webhook 請求在此響應時間內完成。 單位：秒 有效的統計資訊：平均
etcd_mvcc_db_total_size_in_bytes	以位元組為單位實際配置的基礎資料庫總大小。請注意，此指標會測量實體磁碟配置，且不用於配額強制執行。（也稱為 apiserver_storage_size_bytes) 單位：位元組 有效的統計資訊：上限
etcd_mvcc_db_total_size_in_use_in_bytes	刻印資料庫的實際資料大小，不包括等待重組的可用空間。此指標指出目前的資料庫用量，並判斷叢集是否超過資料庫大小配額，並進入唯讀模式。 單位：位元組 有效的統計資訊：上限

Amazon CloudWatch 可觀測性運算子

Amazon CloudWatch 可觀測性運算子會收集即時日誌、指標和追蹤資料。它會將那些資料發送到 Amazon CloudWatch 和 AWS X-Ray。您可以安裝此附加元件來啟用 CloudWatch Application Signals 和 CloudWatch Container Insights，搭配 Amazon EKS 的增強可觀測性。這有助於監控基礎設施和容器化應用程式的運作狀態與效能。Amazon CloudWatch 可觀測性運算子旨在安裝和設定必要的元件。

Amazon EKS 支援將 CloudWatch 可觀測性運算子作為 Amazon EKS 附加元件。該附加元件允許在叢集中 Linux 和 Windows 工作節點上使用容器洞見。要在 Windows 上啟用容器洞見，Amazon EKS

附加元件版本必須為 1.5.0 或更新版本。目前，CloudWatch Application Signals 在 Amazon EKS Windows 上不受支援。

以下主題描述如何開始使用 Amazon EKS 叢集的 CloudWatch 可觀測性運算子。

- 如需安裝此附加元件的說明，請參閱《Amazon CloudWatch 使用者指南》中的使用 Amazon CloudWatch 可觀測性 EKS 附加元件或 Helm Chart 安裝 CloudWatch 代理程式。
- 如需 CloudWatch Application Signals 的詳細資訊，請參閱《Amazon CloudWatch 使用者指南》中的 Application Insights。
- 如需有關 Container Insights 的詳細資訊，請參閱《Amazon CloudWatch 使用者指南》中的使用 Container Insights。

將控制平面日誌傳送至 CloudWatch Logs

Amazon EKS 控制平面記錄從 Amazon EKS 控制平面將稽核和診斷日誌直接提供至您帳戶中的 CloudWatch Logs。這些日誌可讓您輕鬆執行叢集並確保叢集的安全。您可以選取所需的確切日誌類型，且日誌將以日誌串流傳送至 CloudWatch 中各個 Amazon EKS 叢集的群組中。您可使用 CloudWatch 訂閱篩選條件對日誌進行即時分析，或將其轉送至其他服務 (日誌將以 Base64 編碼並以 gzip 格式壓縮)。如需詳細資訊，請參閱 Amazon CloudWatch logging (Amazon CloudWatch 日誌記錄)。

> **Note**
>
> Amazon EKS 控制平面日誌會在幾分鐘內交付至 CloudWatch Logs。不過，日誌交付是最好的方式。

您可以選擇要為每個新的或現有的 Amazon EKS 叢集啟用哪些日誌類型，以開始使用 Amazon EKS 控制平面記錄。您可以使用 AWS 管理主控台、 AWS CLI （版本 1.16.139或更高版本） 或透過 Amazon EKS API，以每個叢集為基礎啟用或停用每個日誌類型。啟用之後，日誌會自動從 Amazon EKS 叢集傳送至相同帳戶中的 CloudWatch Logs。

當您使用 Amazon EKS 控制平面記錄時，必須為您執行的每個叢集支付標準 Amazon EKS 定價。您必須為從叢集傳送至 CloudWatch Logs 的任何日誌，支付標準 CloudWatch Logs 資料擷取和儲存成本。您也需要為作為叢集一部分佈建的任何 AWS 資源付費，例如 Amazon EC2 執行個體或 Amazon EBS 磁碟區。

以下叢集控制平面日誌類型可供使用。每個日誌類型對應至 Kubernetes 控制平面的元件。如需進一步了解這些元件的詳細資訊，請參閱 Kubernetes 文件中的 Kubernetes 元件。

API 伺服器**api** ()

叢集的 API 伺服器是公開 Kubernetes API 的控制平面元件。如果您在啟動叢集時或之後隨即啟用 API 伺服器日誌，則日誌中會包含用於啟動 API 伺服器的 API 伺服器旗標。如需詳細資訊，請參閱 Kubernetes 文件中的 kube-apiserver 和稽核政策。

稽核**audit** ()

Kubernetes 稽核日誌可提供影響叢集之個別使用者、系統管理員或系統元件的記錄。如需詳細資訊，請參閱 Kubernetes 文件中的稽核。

驗證器**authenticator** ()

驗證器日誌是 Amazon EKS 獨有的。這些日誌代表 Amazon EKS 使用 IAM 憑證進行 Kubernetes 角色型存取控制 (RBAC) 身分驗證的控制平面元件。如需詳細資訊，請參閱叢集管理。

控制器管理員**controllerManager** ()

控制器管理員負責管理 Kubernetes 隨附的核心控制迴圈。如需詳細資訊，請參閱 Kubernetes 文件中的 kube-controller-manager。

排程器**scheduler** ()

排程器元件會管理在叢集中執行 Pod 的時間和位置。如需詳細資訊，請參閱 Kubernetes 文件中的 kube-scheduler。

啟用或停用控制平面日誌

根據預設，叢集控制平面記錄不會傳送至 CloudWatch Logs。您必須個別啟用各個日誌類型，才能傳送您叢集的日誌。CloudWatch Logs 擷取、封存儲存和資料掃描速率會套用啟用的控制平面記錄。如需詳細資訊，請參閱 CloudWatch 定價。

若要更新控制平面日誌記錄組態，Amazon EKS 需要每個子網路中最多五個可用的 IP 地址。當您啟用日誌類型時，將會以日誌詳細資訊等級 2 傳送日誌。

您可使用 AWS 管理主控台 或 AWS CLI 啟用或停用控制平面日誌。

AWS 管理主控台

1. 開啟 Amazon EKS 主控台。

2. 選擇叢集名稱以顯示您叢集的資訊。
3. 選擇可觀測性索引標籤。
4. 在控制平面日誌記錄區段中，選擇管理日誌記錄。
5. 對於每個日誌類型，選擇其日誌類型為開啟或關閉。根據預設，系統會關閉每個日誌類型。
6. 選擇 Save changes (儲存變更) 以完成操作。

AWS CLI

1. 使用以下命令檢查您的 AWS CLI 版本。

```
aws --version
```

如果您的 AWS CLI 版本早於 `1.16.139`，您必須先更新至最新版本。若要安裝或升級 AWS CLI，請參閱《AWS 命令列界面使用者指南》中的安裝 AWS 命令列界面。

2. 使用以下 CLI AWS 命令更新叢集的控制平面日誌匯出組態。使用叢集名稱取代 *my-cluster* 並指定所需的端點存取值。

> **Note**
>
> 以下命令會將所有可用的日誌類型傳送至 CloudWatch Logs。

```
aws eks update-cluster-config \
    --region region-code \
    --name my-cluster \
    --logging '{"clusterLogging":[{"types":
["api","audit","authenticator","controllerManager","scheduler"],"enabled":true}]}'
```

範例輸出如下。

```
{
    "update": {
        "id": "883405c8-65c6-4758-8cee-2a7c1340a6d9",
        "status": "InProgress",
        "type": "LoggingUpdate",
        "params": [
            {
```

```
                "type": "ClusterLogging",
                "value": "{\"clusterLogging\":[{\"types\":[\"api\",\"audit\",
\"authenticator\",\"controllerManager\",\"scheduler\"],\"enabled\":true}]}"
            }
        ],
        "createdAt": 1553271814.684,
        "errors": []
    }
}
```

3. 使用以下命令以及上個命令傳回的叢集名稱和更新 ID，監控日誌組態更新的狀態。當狀態出現 Successful 時，您的更新就完成了。

```
aws eks describe-update \
    --region region-code \
    --name my-cluster \
    --update-id 883405c8-65c6-4758-8cee-2a7c1340a6d9
```

範例輸出如下。

```
{
    "update": {
        "id": "883405c8-65c6-4758-8cee-2a7c1340a6d9",
        "status": "Successful",
        "type": "LoggingUpdate",
        "params": [
            {
                "type": "ClusterLogging",
                "value": "{\"clusterLogging\":[{\"types\":[\"api\",\"audit\",
\"authenticator\",\"controllerManager\",\"scheduler\"],\"enabled\":true}]}"
            }
        ],
        "createdAt": 1553271814.684,
        "errors": []
    }
}
```

檢視叢集控制平面日誌

當您為您的 Amazon EKS 叢集啟用任何控制平面日誌類型之後，即可在 CloudWatch 主控台上檢視它們。

若要進一步了解在 CloudWatch 中檢視、分析和管理日誌，請參閱 Amazon CloudWatch Logs 使用者指南。

1. 開啟 CloudWatch 主控台。此連結會開啟主控台及顯示目前可用的日誌群組，並以 /aws/eks 字首進行篩選。
2. 選擇您要檢視日誌的叢集。日誌群組的名稱格式是 /aws/eks/*my-cluster*/cluster。
3. 選擇要檢視的日誌串流。以下清單說明每個日誌類型的日誌串流名稱格式。

> **Note**
>
> 隨著日誌串流資料增加，日誌串流名稱將會輪換。當特定日誌類型有多個日誌串流時，您可以使用最新的 Last Event Time (上次事件時間) 來尋找日誌串流名稱，以檢視最新的日誌串流。

   - Kubernetes API 伺服器元件記錄 (**api**) – kube-apiserver-*1234567890abcdef01234567890abcde*
   - 稽核 (**audit**) – kube-apiserver-audit-*1234567890abcdef01234567890abcde*
   - 驗證器 (**authenticator**) – authenticator-*1234567890abcdef01234567890abcde*
   - 控制器管理員 (**controllerManager**) – kube-controller-manager-*1234567890abcdef01234567890abcde*
   - 排程器 (**scheduler**) – kube-scheduler-*1234567890abcdef01234567890abcde*
4. 查看日誌串流事件。

   例如，在檢視 kube-apiserver-*1234567890abcdef01234567890abcde* 的頂端時，您應該會看到叢集的初始 API 伺服器旗標。

> **Note**
>
> 如果您未在日誌串流的開頭看到 API 伺服器日誌，則 API 伺服器日誌檔案可能已在伺服器上啟用 API 伺服器記錄之前在伺服器上輪換。在啟用 API 伺服器日誌記錄之前輪換的任何日誌檔案皆無法匯出至 CloudWatch。

不過，您可以使用相同的 Kubernetes 版本建立新的叢集，並在建立叢集時啟用 API 伺服器記錄。使用相同平台版本的叢集已啟用相同的旗標，因此您的旗標應符合新叢集的旗標。當您檢視完 CloudWatch 中新叢集的旗標時，即可刪除新叢集。

將 API 呼叫記錄為 AWS CloudTrail 事件

Amazon EKS 已與 AWS CloudTrail 整合。CloudTrail 服務會提供由 Amazon EKS 中的使用者、角色或 AWS 服務所執行之動作的記錄。CloudTrail 會將 Amazon EKS 的所有 API 呼叫擷取為事件。事件包括從 Amazon EKS 主控台的呼叫，以及對 Amazon EKS API 操作的程式碼呼叫。

若您建立追蹤，便可將 CloudTrail 事件持續交付至 Amazon S3 儲存貯體，包括 的事件。其中包括 Amazon EKS 的事件。即使您未設定追蹤，依然可以透過 CloudTrail 主控台中的事件歷史記錄檢視最新事件。使用 CloudTrail 收集的資訊，您可以判斷關於請求的數個詳細資訊。例如，您可以判斷向 Amazon EKS 提出請求的時間、提出請求的 IP 地址、以及提出請求的人員。

若要進一步了解 CloudTrail，請參閱《AWS CloudTrail 使用者指南》。

主題

- 檢視適用於 AWS CloudTrail 的實用參考
- Analyze AWS CloudTrail 日誌檔案項目
- 檢視 Amazon EC2 Auto Scaling 群組的指標

檢視適用於 AWS CloudTrail 的實用參考

當您建立 AWS 帳戶時，也會在 AWS 帳戶中啟用 CloudTrail。在 Amazon EKS 中發生活動時，該活動將與 Event history (事件歷史紀錄) 中的其他 AWS 服務事件一起記錄在 CloudTrail 事件中。您可以檢視、搜尋和下載 AWS 帳戶的最新事件。如需詳細資訊，請參閱使用 CloudTrail 事件歷史記錄檢視事件。

若要持續記錄您 AWS 帳戶中正在進行的事件 (包括 Amazon EKS 的事件)，請建立追蹤。追蹤能讓 CloudTrail 將日誌檔交付至 Amazon S3 儲存貯體。根據預設，當您在主控台建立線索時，線索會套用到所有 AWS 區域。權杖會記錄來自 AWS 分割區中所有 AWS 區域的事件，然後將記錄檔案交付到您指定的 Amazon S3 儲存貯體。此外，您可以設定其他 AWS 服務，以進一步分析和處理 CloudTrail 日誌中所收集的事件資料。如需詳細資訊，請參閱下列資源。

- 建立追蹤的概觀
- CloudTrail 支援的服務和整合
- 設定 CloudTrail 的 Amazon SNS 通知

- 接收多個區域的 CloudTrail 日誌檔案和接收多個帳戶的 CloudTrail 日誌檔案

CloudTrail 會記錄所有 Amazon EKS 動作，並記錄在《Amazon EKS API 參考》中。例如，對 CreateCluster、ListClusters 和 DeleteCluster 區段的呼叫，會在 CloudTrail 日誌檔案中生成項目。

每個事件或日誌項目包含提出請求之 IAM 身分類型及所使用之憑證的資訊。如果使用暫時性憑證，此項目會說明憑證的取得方式。

如需詳細資訊，請參閱 CloudTrail userIdentity 元素。

Analyze AWS CloudTrail 日誌檔案項目

追蹤是一種組態，能讓事件以日誌檔案的形式交付到您指定的 Amazon S3 儲存貯體。CloudTrail 日誌檔案包含一或多個日誌專案。事件代表來自任何來源的單一請求，其中包含請求動作的相關資訊。這包括動作的日期和時間，以及已使用之請求參數等資訊。CloudTrail 日誌檔並非依公有 API 呼叫的堆疊追蹤排序，因此不會以任何特定順序出現。

以下範例顯示的是展示 `CreateCluster` 動作的 CloudTrail 日誌項目。

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AKIAIOSFODNN7EXAMPLE",
    "arn": "arn:aws: iam::111122223333:user/username",
    "accountId": "111122223333",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "username"
  },
  "eventTime": "2018-05-28T19:16:43Z",
  "eventSource": "eks.amazonaws.com",
  "eventName": "CreateCluster",
  "awsRegion": "region-code",
  "sourceIPAddress": "205.251.233.178",
  "userAgent": "PostmanRuntime/6.4.0",
  "requestParameters": {
    "resourcesVpcConfig": {
      "subnetIds": [
        "subnet-a670c2df",
        "subnet-4f8c5004"
      ]
    },
```

```
    "roleArn": "arn:aws: iam::111122223333:role/AWSServiceRoleForAmazonEKS-
CAC1G1VH3ZKZ",
    "clusterName": "test"
  },
  "responseElements": {
    "cluster": {
      "clusterName": "test",
      "status": "CREATING",
      "createdAt": 1527535003.208,
      "certificateAuthority": {},
      "arn": "arn:aws: eks:region-code:111122223333:cluster/test",
      "roleArn": "arn:aws: iam::111122223333:role/AWSServiceRoleForAmazonEKS-
CAC1G1VH3ZKZ",
      "version": "1.10",
      "resourcesVpcConfig": {
        "securityGroupIds": [],
        "vpcId": "vpc-21277358",
        "subnetIds": [
          "subnet-a670c2df",
          "subnet-4f8c5004"
        ]
      }
    }
  },
  "requestID": "a7a0735d-62ab-11e8-9f79-81ce5b2b7d37",
  "eventID": "eab22523-174a-499c-9dd6-91e7be3ff8e3",
  "readOnly": false,
  "eventType": "AwsApiCall",
  "recipientAccountId": "111122223333"
}
```

Amazon EKS 服務連結角色的日誌項目

Amazon EKS 服務連結角色會對 AWS 資源進行 API 呼叫。對於 Amazon EKS 服務連結角色進行的呼叫，username: AWSServiceRoleForAmazonEKSNodegroup會顯示具有 username: AWSServiceRoleForAmazonEKS和 的 CloudTrail 日誌項目。如需 Amazon EKS 和服務連結角色的詳細資訊，請參閱 the section called “服務連結角色”。

下列範例顯示 CloudTrail 日誌項目，其示範 AWSServiceRoleForAmazonEKSNodegroup 服務連結角色所做的 DeleteInstanceProfile 動作，如 sessionContext 中所示。

```
{
    "eventVersion": "1.05",
```

```
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROA3WHGPEZ7SJ2CW55C5:EKS",
        "arn": "arn:aws: sts::111122223333:assumed-role/
AWSServiceRoleForAmazonEKSNodegroup/EKS",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROA3WHGPEZ7SJ2CW55C5",
                "arn": "arn:aws: iam::111122223333:role/aws-service-role/eks-
nodegroup.amazonaws.com/AWSServiceRoleForAmazonEKSNodegroup",
                "accountId": "111122223333",
                "userName": "AWSServiceRoleForAmazonEKSNodegroup"
            },
            "webIdFederationData": {},
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2020-02-26T00:56:33Z"
            }
        },
        "invokedBy": "eks-nodegroup.amazonaws.com"
    },
    "eventTime": "2020-02-26T00:56:34Z",
    "eventSource": "iam.amazonaws.com",
    "eventName": "DeleteInstanceProfile",
    "awsRegion": "region-code",
    "sourceIPAddress": "eks-nodegroup.amazonaws.com",
    "userAgent": "eks-nodegroup.amazonaws.com",
    "requestParameters": {
        "instanceProfileName": "eks-11111111-2222-3333-4444-abcdef123456"
    },
    "responseElements": null,
    "requestID": "11111111-2222-3333-4444-abcdef123456",
    "eventID": "11111111-2222-3333-4444-abcdef123456",
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

檢視 Amazon EC2 Auto Scaling 群組的指標

Amazon EKS 受管節點群組已預設啟用 Amazon EC2 Auto Scaling 群組指標，且無需任何額外費用。Auto Scaling 群組會每分鐘傳送一次取樣資料到 Amazon CloudWatch。這些指標可依據 Auto Scaling 群組的名稱進行精簡。透過這些指標，您可以持續地了解 Auto Scaling 群組為您的受管節點群組提供支援的歷史記錄，例如群組大小隨時間的變化。Auto Scaling 群組指標可在 Amazon CloudWatch 或 Auto Scaling 主控台中使用。如需詳細資訊，請參閱監控 Auto Scaling 群組和執行個體的 CloudWatch 指標。

利用 Auto Scaling 群組指標集合，您將能夠監控受管節點群組的擴展。Auto Scaling 群組指標報告 Auto Scaling 群組最小、最大和所需規模。如果節點群組中的節點數量低於最小規模，這代表節點群組運作狀態不佳，您可以建立警示。追蹤節點群組規模在調整最大計數時也很有用，讓資料平面不會耗盡容量。

如果您不想要收集這些指標，您可以選擇停用所有或僅停用部分指標。例如，您可以執行此操作，以避免 CloudWatch 儀表板中的雜訊。如需詳細資訊，請參閱 Amazon EC2 Auto Scaling 的 Amazon CloudWatch 指標。

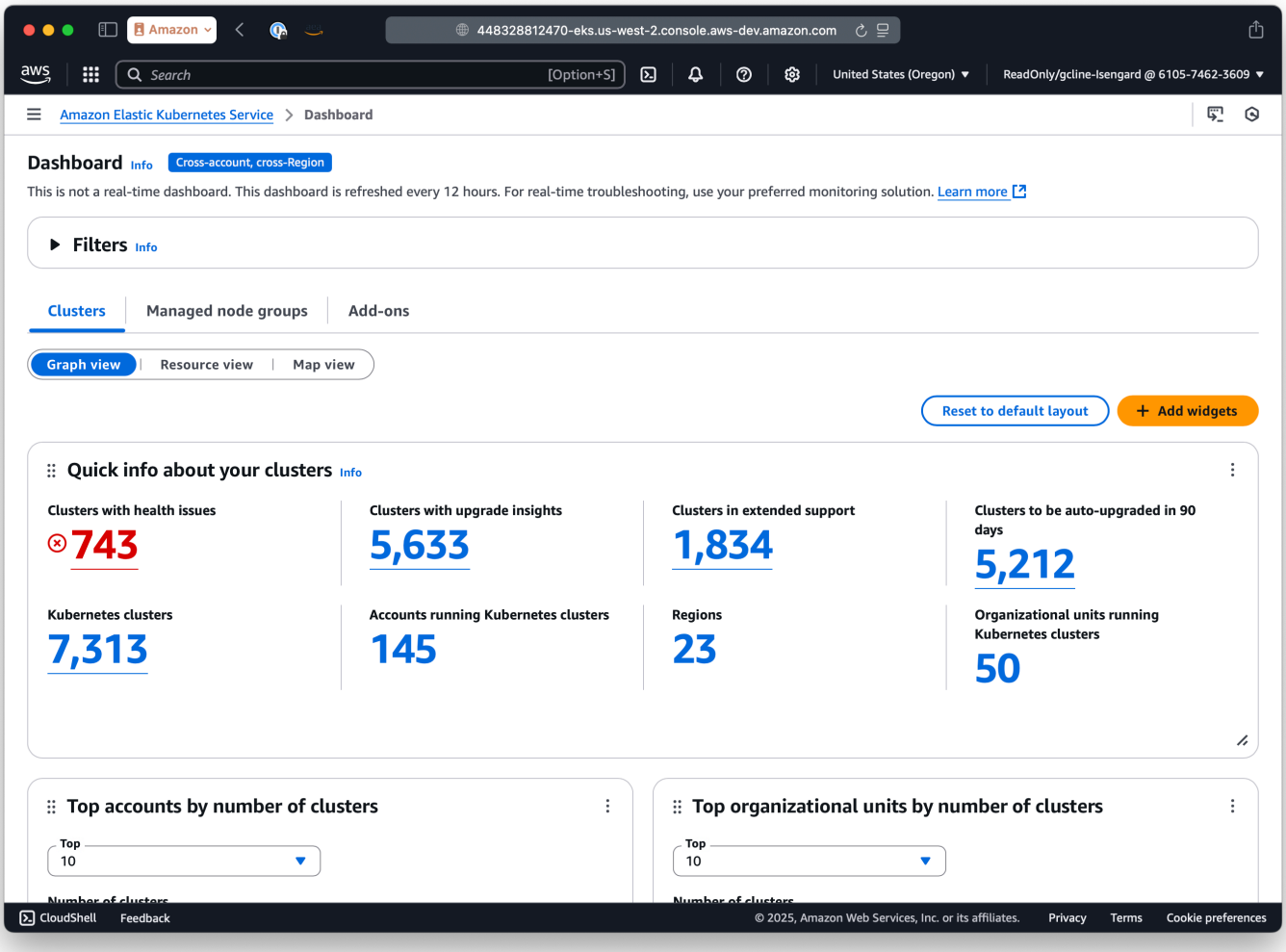
使用 ADOT 運算子傳送指標和追蹤資料

Amazon EKS 支援使用 AWS 管理主控台、AWS CLI 和 Amazon EKS API 來安裝和管理 AWS Distro for OpenTelemetry (ADOT) 運算子。這可以讓您更輕鬆地啟用在 Amazon EKS 上執行的應用程式，以便將指標和追蹤資料傳送到多個監控服務選項，例如 Amazon CloudWatch、Prometheus 及 X-Ray。

如需詳細資訊，請參閱 AWS Distro for OpenTelemetry 文件中的使用 EKS 附加元件的 AWS Distro for OpenTelemetry 入門。

使用 EKS 儀表板檢視叢集資源的彙總資料

什麼是 Amazon EKS 儀表板？



Amazon EKS 儀表板提供跨多個 AWS 區域和 AWS 帳戶的 Kubernetes 叢集整合可見性。透過此儀表板，您可以：

- 追蹤未來 90 天內計劃進行支援終止自動升級的叢集。
- 預估延伸支援叢集的 EKS 控制平面成本。
- 檢視在升級前需要注意洞見問題的叢集。
- 識別正在執行特定 AMI 版本的受管節點群組。
- 監控叢集支援類型分佈 (標準支援與延伸支援的比較)。

EKS 儀表板與 EKS 叢集洞察功能整合，以顯示叢集的問題，例如使用已棄用的 Kubernetes API。如需詳細資訊，請參閱the section called “叢集洞察”。

> **Note**
>
> EKS 儀表板並非即時更新，每 12 小時更新一次。如需即時叢集監控，請參閱 監控叢集。

儀表板如何使用 AWS Organizations？

Amazon EKS 儀表板需要 AWS Organizations 整合功能。它利用 AWS Organizations 跨帳戶安全地收集叢集資訊。隨著您的 AWS 基礎設施擴展，此整合提供集中式管理和管控。

如果您的基礎設施未啟用 AWS Organizations，請參閱 AWS Organizations 使用者指南中的設定說明。

跨帳戶和跨帳戶存取權

EKS 儀表板可以在屬於 AWS 組織成員的任何帳戶中查看叢集資源。若要產生組織中 AWS 的帳戶清單，請參閱匯出組織中所有帳戶的詳細資訊。

us-east-1 AWS region 會產生儀表板。您必須登入此區域方能查看儀表板。儀表板會跨 AWS 區域彙總資料，但這不包括 GovCloud或 中國區域。

重要用語

- AWS 組織：多個 AWS 帳戶的統一管理結構。
- 管理帳戶：控制 AWS Organization 的主要帳戶。
- 會員帳戶：組織內除管理帳戶以外的任何帳戶。
- 委派管理員：被授予特定跨帳戶管理許可的會員帳戶。在管理帳戶中，您可以為每個 AWS 服務選擇一個委派管理員帳戶。
- 可信存取：授權 EKS 儀表板跨組織帳戶存取叢集資訊。
- 服務連結角色 (SLR)：直接連結至 AWS 服務的唯一 IAM 角色類型。EKS 儀表板使用 SLR 來讀取有關您的帳戶和組織的資訊。
- 如需詳細資訊，請參閱《 AWS Organizations 使用者指南》中的術語和概念。

一般概觀

1. 存取您 AWS Organization 的管理帳戶。
   - 存取管理帳戶的步驟取決於您設定 AWS Organization 的方式。例如，您可透過 AWS Identity Center 或 Okta 存取管理帳戶。
2. 透過 EKS 主控台啟用可信存取。
3. 使用其 AWS 帳戶 ID 指派委派管理員。
4. 切換到委派管理員帳戶。
5. 存取具有組織範圍可見性的增強型 EKS 主控台。

使用 AWS 主控台啟用 EKS 儀表板

> **Important**
>
> 您必須登入您 AWS 組織的管理帳戶，才能啟用 EKS 儀表板。

存取 EKS 儀表板設定

1. 請確認以下內容：
   a. 您已啟用並設定 AWS Organizations。
   b. 您已登入組織的管理帳戶。
   c. 您正在檢視 us-east-1 區域中的 AWS 管理主控台。
2. 導覽至 EKS 主控台。
3. 在左側邊欄中開啟儀表板設定。

設定 Amazon EKS 儀表板的存取權

1. 尋找您要允許檢視 EKS 儀表板之 AWS 帳戶的帳戶 AWS ID。
   a. 此步驟為選用，但建議執行。如果您不執行此步驟，則只能從管理帳戶存取儀表板。作為最佳實務，您應限制對管理帳戶的存取。
2. 按一下啟用可信存取。
   a. 您目前可以從管理帳戶檢視儀表板。

3. 按一下註冊委派管理員，然後輸入您將用來檢視儀表板之 AWS 帳戶的帳戶 ID。
   a. 您現在可以從委派管理員帳戶或管理帳戶檢視儀表板。

有關啟用儀表板所需許可的資訊，請參閱所需的最低 IAM 政策。

檢視 EKS 儀表板

1. 登入委派管理員帳戶 (建議) 或管理帳戶。
2. 登入 us-east-1 區域。
3. 前往 EKS 服務，並從左側邊欄中選取儀表板。

> **Note**
>
> 檢閱用於檢視 EKS 儀表板所需的 IAM 許可。

設定儀表板

您可設定儀表板的檢視，並篩選資源。

可用的資源

- 叢集：檢視有關 EKS 叢集狀態和位置的彙總資訊。
  - 運作狀態有問題的叢集。
  - 處於 EKS 延伸支援的叢集。
  - 按 Kubernetes 版本劃分的叢集明細。
- 受管節點群組：檢閱受管節點群組與 EC2 執行個體。
  - 按 AMI 類型劃分的節點群組，例如 Amazon Linux 或 Bottlerocket。
  - 節點群組運作狀態問題。
  - 執行個體類型分佈。
- 附加元件：了解您安裝了哪些 Amazon EKS 附加元件及其狀態。
  - 每個附加元件的安裝數量。
  - 存在運作狀態問題的附加元件。
  - 每個附加元件的版本分佈。

可用檢視

- 圖形檢視
  - 可自訂的小工具檢視，顯示所選資源的圖形和視覺化。
  - EKS 儀表板的所有使用者均可見圖形檢視的變更，例如移除小工具。
- 資源檢視
  - 所選資源的清單檢視，支援篩選條件。
- 地圖檢視
  - 檢視所選資源的地理分佈。

篩選 EKS 儀表板

您可以透過以下方式篩選 EKS 儀表板：

- AWS 帳戶
- Organizations 定義的 AWS 組織單位
- AWS 區域

使用 AWS 主控台停用 EKS 儀表板

1. 請確認以下內容：
   a. 您已啟用並設定 AWS Organizations。
   b. 您已登入組織的管理帳戶。
   c. 您正在檢視 us-east-1 區域中的 AWS 管理主控台。
2. 導覽至 EKS 主控台。
3. 在左側邊欄中開啟儀表板設定。
4. 按一下停用可信存取。

EKS 儀表板疑難排解

啟用 EKS 儀表板時遇到問題

- 您必須登入 AWS 組織的管理帳戶。

  - 如果您沒有 AWS 組織，請建立一個組織。了解如何建立和設定組織。
  - 如果 AWS 您的帳戶已經是 AWS 組織的成員，請識別組織的管理員。
- 您必須以足夠的 IAM 許可登入 AWS 帳戶，才能建立和更新 AWS Organizations 資源。

檢視 EKS 儀表板時遇到問題

- 您必須登入下列其中一個 AWS 帳戶：
  - AWS 組織的管理帳戶
  - 在管理帳戶的 EKS 儀表板設定中識別的委派管理員帳戶。
- 如果您最近啟用了 EKS 儀表板，請注意初始資料填充可能需要長達 12 小時。
- 嘗試使用 CLI 重新啟用儀表板，包括建立服務連結角色。

儀表板小工具意外移動

- EKS 儀表板會在 AWS 帳戶層級儲存可設定的小工具檢視。如果您變更小工具檢視，使用相同 AWS 帳戶的其他人員將看到變更。

設定 EKS 儀表板與 AWS Organizations 的整合

本節提供step-by-step說明。 AWS 您將了解如何啟用與停用服務之間可信存取，以及如何註冊與取消註冊委派管理員帳戶。您可以使用 AWS 主控台或 CLI AWS 來執行每個組態任務。

啟用受信任存取

可信存取授權 EKS 儀表板安全地存取組織中所有帳戶的叢集資訊。

使用 AWS 主控台

1. 登入您 AWS Organization 的管理帳戶。
2. 導覽至 us-east-1 區域的 EKS 主控台。
3. 在左側邊欄中選取儀表板設定。
4. 按一下啟用可信存取。

> **Note**
>
> 當您透過 EKS 主控台啟用可信存取時，系統會自動建立 `AWSServiceRoleForAmazonEKSDashboard` 服務連結角色。如果您使用 CLI 或 AWS Organizations AWS 主控台啟用受信任存取，則不會發生此自動建立。

使用 AWS CLI

1. 登入您 AWS Organization 的管理帳戶。
2. 執行下列命令：

```
aws iam create-service-linked-role --aws-service-name dashboard.eks.amazonaws.com
aws organizations enable-aws-service-access --service-principal eks.amazonaws.com
```

停用受信任存取

停用可信存取會撤銷 EKS 儀表板存取您組織帳戶中叢集資訊的許可。

使用 AWS 主控台

1. 登入您 AWS Organization 的管理帳戶。
2. 導覽至 us-east-1 區域的 EKS 主控台。
3. 在左側邊欄中選取儀表板設定。
4. 按一下停用可信存取。

使用 AWS CLI

1. 登入您 AWS Organization 的管理帳戶。
2. 執行以下命令：

```
aws organizations disable-aws-service-access --service-principal eks.amazonaws.com
```

啟用委派管理員帳戶

委派管理員是被授予存取 EKS 儀表板許可的會員帳戶。

使用 AWS 主控台

1. 登入您 AWS Organization 的管理帳戶。
2. 導覽至 us-east-1 區域的 EKS 主控台。
3. 在左側邊欄中選取儀表板設定。
4. 按一下註冊委派的管理員。
5. 輸入您要選擇做為委派管理員之 AWS 帳戶的帳戶 ID。
6. 確認註冊。

使用 AWS CLI

1. 登入您 AWS Organization 的管理帳戶。
2. 執行以下命令，將 123456789012 取代為您的帳戶 ID：

```
aws organizations register-delegated-administrator --account-id 123456789012 --
service-principal eks.amazonaws.com
```

停用委派管理員帳戶

停用委派管理員會移除該帳戶存取 EKS 儀表板的許可。

使用 AWS 主控台

1. 登入您 AWS Organization 的管理帳戶。
2. 導覽至 us-east-1 區域的 EKS 主控台。
3. 在左側邊欄中選取儀表板設定。
4. 在清單中找到委派管理員。
5. 在您要以委派管理員身分移除的帳戶旁邊，按一下取消註冊。

使用 AWS CLI

1. 登入您 AWS Organization 的管理帳戶。
2. 執行以下命令，將 123456789012 取代為委派管理員的帳戶 ID：

```
aws organizations deregister-delegated-administrator --account-id 123456789012 --
service-principal eks.amazonaws.com
```

所需的最低 IAM 政策

本節概述啟用受信任存取所需的最低 IAM 政策，並委派管理員與 AWS Organizations 進行 EKS 儀表板整合。

用於啟用可信存取的政策

若要在 EKS 儀表板和 AWS Organizations 之間啟用受信任存取，您需要下列許可：

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "organizations:EnableAWSServiceAccess",
                "organizations:DescribeOrganization",
                "organizations:ListAWSServiceAccessForOrganization"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateServiceLinkedRole"
            ],
            "Resource": "arn:aws:iam::*:role/aws-service-role/
dashboard.eks.amazonaws.com/AWSServiceRoleForAmazonEKSDashboard"
        }
    ]
}
```

用於委派管理員的政策

要為 EKS 儀表板註冊或取消註冊委派管理員，您需要以下許可：

```
{
    "Version":"2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "organizations:RegisterDelegatedAdministrator",
                "organizations:DeregisterDelegatedAdministrator",
                "organizations:ListDelegatedAdministrators"
            ],
            "Resource": "*"
        }
    ]
}
```

用於檢視 EKS 儀表板的政策

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AmazonEKSDashboardReadOnly",
            "Effect": "Allow",
            "Action": [
                "eks:ListDashboardData",
                "eks:ListDashboardResources",
                "eks:DescribeClusterVersions"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AmazonOrganizationsReadOnly",
            "Effect": "Allow",
            "Action": [
                "organizations:DescribeOrganization",
                "organizations:ListAWSServiceAccessForOrganization",
                "organizations:ListRoots",
                "organizations:ListAccountsForParent",
                "organizations:ListOrganizationalUnitsForParent"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AmazonOrganizationsDelegatedAdmin",
            "Effect": "Allow",
```

```
            "Action": [
                "organizations:ListDelegatedAdministrators"
            ],
            "Resource": [
                "*"
            ],
            "Condition": {
                "StringEquals": {
                    "organizations:ServicePrincipal": "eks.amazonaws.com"
                }
            }
        }
    ]
}
```

> **Note**
>
> 這些政策必須連接到 AWS Organization 管理帳戶中的 IAM 主體（使用者或角色）。會員帳戶無法啟用可信存取或委派管理員。

使用整合式 AWS 服務增強 EKS

除了其他章節涵蓋的服務之外，Amazon EKS 還與更多 AWS 服務搭配使用，以提供其他解決方案。本主題識別使用 Amazon EKS 來新增功能的其他服務，或 Amazon EKS 用來執行任務的服務。

主題

- 使用 Backup AWS 備份 EKS 叢集
- 藉助 AWS CloudFormation 建立 Amazon EKS 資源
- 使用 AWS CodeConnections 連線至 Git 儲存庫
- 使用 Amazon Detective 分析 EKS 上的安全事件
- 使用 Amazon GuardDuty 偵測威脅
- 使用 AWS Local Zones 啟動低延遲 EKS 叢集
- 使用 AWS Resilience Hub 評估 EKS 叢集彈性
- 使用 AWS Secrets Manager 管理應用程式秘密
- 使用 Security Lake 集中和分析 EKS 安全資料
- 使用 Amazon VPC Lattice 啟用安全的跨叢集連線

使用 Backup AWS 備份 EKS 叢集

AWS 備份支援 Amazon EKS 叢集的備份，包括透過持久性磁碟區宣告 (EBS 磁碟區、EFS 檔案系統和 S3 儲存貯體） 連接到 EKS 叢集的 Kubernetes 叢集狀態和持久性儲存。Amazon EKS 備份會建立複合復原點，其中每個備份的資源都會有子復原點。

- 如何備份資源：開始使用 AWS Backup
- 依資源類型建立備份：Amazon EKS 備份
- 如何還原 Amazon EKS 叢集：Amazon EKS 備份還原

若要透過 EKS 主控台、 AWS 備份主控台或 CLI 開始使用，請確定您的 IAM 角色具有下列許可：

- 這些可在 AWS Backup 的受管政策 AWSBackupServiceRolePolicyForBackup 中找到。這包含備份 Amazon EKS 叢集和 EBS 和 EFS 持久性儲存所需的許可
- 如果您的 EKS 叢集包含 S3 儲存貯體，您將需要確保已新增和啟用 S3 儲存貯體的下列政策和先決條件，如所記載：

- AWSBackupServiceRolePolicyForS3Backup
- S3 備份的先決條件

確保您的 EKS 叢集具有下列設定：

- EKS 叢集授權模式設定為 API 或 API_AND_CONFIG_MAP for AWS Backup，以建立存取項目來存取 EKS 叢集。

藉助 AWS CloudFormation 建立 Amazon EKS 資源

Amazon EKS 已整合 AWS CloudFormation，這項服務可協助您建立 AWS 資源的模型和設定，以減少建立和管理資源和基礎結構的時間。您建立一個描述所有所需之 AWS 資源的範本 (如 Amazon EKS 叢集)，而 AWS CloudFormation 負責為您佈建與設定這些資源。

當您使用 AWS CloudFormation 時，您可以重複使用您的範本，重複、一致的設定您的 Amazon EKS 資源。您只需要描述您的資源一次，然後在多個 AWS 帳戶與區域內重複佈建相同資源即可。

Amazon EKS 與 AWS CloudFormation 範本

若要佈建和配置 Amazon EKS 資源和相關服務的資源，則必須了解 AWS CloudFormation 範本。範本是以 JSON 或 YAML 格式化的文本檔案。而您亦可以透過這些範本的說明，了解欲在 AWS CloudFormation 堆疊中佈建的資源。如果您不熟悉 JSON 或 YAML，您可以使用 AWS CloudFormation Designer 協助您開始使用 AWS CloudFormation 範本。如需詳細資訊，請參閱《AWS CloudFormation 使用者指南》中的 AWS CloudFormation Designer 是什麼？。

Amazon EKS 支援在 AWS CloudFormation 中建立叢集和節點群組。如需詳細資訊 (包括 Amazon EKS 資源的 JSON 和 YAML 範本範例)，請參閱《AWS CloudFormation 使用者指南》中的 Amazon EKS 資源類型參考。

進一步了解 AWS CloudFormation

若要進一步了解 AWS CloudFormation，請參閱下列資源：

- AWS CloudFormation
- AWS CloudFormation 使用者指南
- AWS CloudFormation 命令行介面使用者指南

使用 AWS CodeConnections 連線至 Git 儲存庫

AWS CodeConnections 提供將 AWS 服務連線至第三方原始碼儲存庫的安全方法。 AWS CodeConnections 支援 GitHub、GitLab、Bitbucket 和其他供應商。若要進一步了解並開始使用，請參閱使用連線。

Use AWS CodeConnections 與 Argo CD

使用適用於 Argo CD 的 EKS 功能時，您可以選擇使用 AWS CodeConnections 啟用 Git 儲存庫的安全身分驗證，而無需管理長期憑證或個人存取字符。 AWS CodeConnections 會處理 OAuth 身分驗證流程並管理與 Git 供應商的連線，提供安全且可管理的方法，讓您存取存放在第三方 Git 供應商中的 GitOps 儲存庫和應用程式資訊清單。

先決條件

- 建立 Argo CD 功能的 Amazon EKS 叢集
- 在 AWS CodeConnections 中建立到 Git 供應商的連線
- 為 Argo CD 設定以使用連線的 IAM 許可

設定 Argo CD 儲存庫的 CodeConnections 存取

1. 在 CodeConnections 主控台中建立連線：
   a. 開啟 CodeConnections 主控台。
   b. 選擇建立連線。
   c. 選取您的提供者 (GitHub、GitLab 或 Bitbucket)，然後遵循身分驗證流程。
   d. 請注意用於 Argo CD 組態的連線 ARN。
2. 確保 Argo CD 功能角色具有使用 連線與資源型政策的許可：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codeconnections:UseConnection",
        "codeconnections:GetConnection"
      ],
```

```
      "Resource": "arn:aws:codeconnections:region:account-id:connection/connection-
id"
    }
  ]
}
```

3. 使用 CodeConnections 資源端點做為儲存庫 URL，設定 Argo CD 在新增儲存庫時參考 CodeConnections 資源。Argo CD 功能使用 連線向您的 Git 提供者進行身分驗證，而不需要長期憑證。

搭配 Argo CD 使用 CodeConnections 的考量事項

將 AWS CodeConnections 與 Argo CD 的 EKS 功能搭配使用時，請記住下列事項：

- CodeConnections 連線必須與 EKS 叢集位於相同的 AWS 區域
- Argo CD 功能角色必須具有 `codeconnections:UseConnection`和 `codeconnections:GetConnection`許可
- CodeConnections 會自動管理 OAuth 流程和登入資料生命週期

如需使用 Argo CD 設定儲存庫存取權的詳細資訊，請參閱 the section called “設定儲存庫”。

使用 Amazon Detective 分析 EKS 上的安全事件

Amazon Detective 會協助您分析、調查並快速識別安全調查結果或可疑活動的根本原因。Detective 會自動從您的 AWS 資源收集日誌資料。Detective 接著會使用機器學習、統計分析和圖論來產生視覺化內容，協助您更快地進行有效率的安全調查。Detective 提供預先建置的資料彙總、摘要和內容，可協助您快速分析並判斷潛在安全問題的本質和範圍。如需詳細資訊，請參閱 Amazon Detective 使用者指南。

Detective 會將 Kubernetes 和 AWS 資料組織到問題清單，例如：

- Amazon EKS 叢集詳細資訊，包括建立叢集的 IAM 身分和叢集的服務角色。您可以使用 Detective 調查這些 IAM 身分的 AWS 和 Kubernetes API 活動。
- 容器詳細資料，例如映像和安全性內容。您也可檢閱已終止 Pod 的詳細資訊。
- Kubernetes API 活動，包括 API 活動的整體趨勢以及特定 API 呼叫的詳細資料。例如，您可以顯示在指定時間範圍內成功和失敗的 Kubernetes API 呼叫資料。此外，最新觀察到的 API 呼叫部分可能有助於識別可疑活動。

Amazon EKS 稽核日誌是選用的資料來源套件，可新增至您的 Detective 行為圖表。您可以在帳戶中檢視可用的選用來源套件及其狀態。如需詳細資訊，請參閱《Amazon Detective 使用者指南》中的 Detective 的 Amazon EKS 稽核日誌。

將 Amazon Detective 與 Amazon EKS 搭配使用

您必須先在叢集所在的相同 AWS 區域中啟用 Detective 至少 48 小時，才能檢閱問題清單。如需詳細資訊，請參閱《Amazon Detective 使用者指南》中的 設定 Amazon Detective。

1. 打開 Detective 主控台，網址為 https://console.aws.amazon.com/detective/。
2. 從左側導覽窗格選取搜尋。
3. 選取選擇類型，然後選取 EKS 叢集。
4. 輸入叢集名稱或 ARN，然後選擇搜尋。
5. 在搜尋結果中，選取要檢視其活動的叢集名稱。如需有關可檢視內容的詳細資訊，請參閱《Amazon Detective 使用者指南》中涉及 Amazon EKS 叢集的整體 Kubernetes API 活動。

使用 Amazon GuardDuty 偵測威脅

Amazon GuardDuty 是一種威脅偵測服務，可協助保護您的帳戶、容器、工作負載和 AWS 環境中的資料。GuardDuty 使用機器學習 (ML) 模型，以及異常和威脅偵測功能，持續監控不同的日誌來源和執行時期活動，以識別環境中的潛在安全風險和惡意活動和排定其優先順序。

除了這些功能外，GuardDuty 還提供下列兩種功能，可偵測對 EKS 叢集的潛在威脅：EKS 保護和執行時期監控。

> **Note**
>
> 新增：Amazon EKS 自動模式可與 GuardDuty 整合。

EKS 保護

此功能提供威脅偵測涵蓋範圍，可透過監控相關聯的 Kubernetes 稽核日誌來協助您保護 Amazon EKS 叢集。Kubernetes 稽核日誌會擷取叢集中的連續動作，包括來自使用者的活動、使用 Kubernetes API 的應用程式以及控制平面。例如，GuardDuty 可以識別未經身分驗證的使用者調用的可能竄改 Kubernetes 叢集中字元的 API 呼叫。

當您啟用 EKS 保護時，GuardDuty 將只能存取 Amazon EKS 稽核日誌，以實現持續威脅偵測。如果 GuardDuty 識別出叢集的潛在威脅，則其會產生特定類型的相關聯 Kubernetes 稽核日誌調查結果。如需有關 Kubernetes 稽核日誌中可用調查結果類型的詳細資訊，請參閱《Amazon GuardDuty 使用者指南》中的 Kubernetes 稽核日誌調查結果類型。

如需詳細資訊，請參閱《Amazon GuardDuty 使用者指南》中的 EKS 保護。

執行時期監控

此功能會監控和分析作業系統層級、聯網和檔案事件，以協助您偵測環境中特定 AWS 工作負載的潛在威脅。

當您在 Amazon EKS 啟用執行時期監控並安裝 GuardDuty 代理程式時，GuardDuty 會開始監控與此叢集相關聯的執行時期事件。請注意，GuardDuty 代理程式和執行時期監控不適用於 Amazon EKS 混合節點，因此執行時期監控不適用於在混合節點上發生的執行時期事件。如果 GuardDuty 識別出叢集的潛在威脅，則其會產生相關聯的執行時期監控調查結果。例如，威脅可能會透過入侵執行具漏洞 Web 應用程式的單一容器開始。此 Web 應用程式可能擁有基礎容器和工作負載的存取許可。在這種情況下，設定錯誤的憑證可能會導致更廣泛地存取帳戶及其中儲存的資料。

若要設定執行時期監控，您可將 GuardDuty 代理程式作為 Amazon EKS 附加元件安裝至您的叢集中。如需有關附加元件的詳細資訊，請參閱the section called “ AWS 附加元件”。

如需詳細資訊，請參閱《Amazon GuardDuty 使用者指南》中的執行時期監控。

使用 AWS Local Zones 啟動低延遲 EKS 叢集

Local AWS Zone 是 AWS 區域的延伸，其地理位置接近您的使用者。Local Zones 有其自己的網際網路連線，並支援 AWS Direct Connect。在本機區域中建立的資源可以為本機使用者提供低延遲的通訊服務。如需詳細資訊，請參閱《AWS Local Zones 使用者指南》和《Amazon EC2 使用者指南》中的 Local Zones。

Amazon EKS 支援本機區域中的特定資源。其中包括受管節點群組、自我管理的 Amazon EC2 節點、Amazon EBS 磁碟區和 Application Load Balancer (ALB)。在使用本機區域作為 Amazon EKS 叢集的一部分時，建議您考慮以下事項。

- 您無法在 Local Zones 中使用 Amazon EKS 建立 Fargate 節點。
- Amazon EKS 受管 Kubernetes 控制平面一律會在 區域中執行 AWS 。Amazon EKS 受管 Kubernetes 控制平面無法在 Local Zone 中執行。由於 Local Zones 在 VPC 內顯示為子網路，因此 Kubernetes 會將您的本機區域資源作為該子網路的一部分。

- Amazon EKS Kubernetes 叢集會使用 Amazon EKS 受管彈性網路介面，與您在區域或本機區域中執行的 Amazon EC2 執行個體通訊。 AWS https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-eni.html若要進一步了解 Amazon EKS 聯網架構，請參閱 設定聯網。
- 與區域子網路不同，Amazon EKS 無法將跨帳戶彈性網路介面放入您的本機區域子網路。這些界面用於控制平面到節點的通訊。這表示建立叢集時，您不得指定本機區域子網路。不過，您可以在多個本機區域中將工作者節點連接到相同的叢集。

使用 AWS Resilience Hub 評估 EKS 叢集彈性

AWS Resilience Hub 會透過分析其基礎結構以評估 Amazon EKS 叢集的彈性。AWSResilience Hub 會使用 Kubernetes 角色型存取控制 (RBAC) 組態，以評估部署至叢集的 Kubernetes 工作負載。如需詳細資訊，請參閱《AWS Resilience Hub 使用者指南》中的啟用對 Amazon EKS 叢集的 AWS Resilience Hub 存取權。

使用 AWS Secrets Manager 管理應用程式秘密

AWS Secrets Manager 可協助您在整個生命週期中管理、存取和輪換登入資料、API 金鑰和其他秘密。使用 Secrets Manager，您可以保護和管理用來存取 AWS、第三方服務和內部部署中資源的秘密。如需詳細資訊，請參閱 AWS Secrets Manager 使用者指南。

使用適用於 Argo CD 的 EKS 功能時，Secrets Manager 提供安全的方式來存放和擷取 Git 儲存庫登入資料，而無需在 Argo CD 組態和資源中硬式編碼敏感資料。此整合特別適用於管理私有儲存庫存取權杖，以及 Argo CD 用來從 Git 儲存庫同步應用程式的 SSH 金鑰。

搭配 Argo CD 使用 AWS Secrets Manager

使用適用於 Argo CD 的 EKS 功能時，您可以將 Git 儲存庫登入資料存放在 Secrets Manager 中，並設定 Argo CD 進行擷取。這種方法比直接將登入資料存放在 Argo CD 組態中或使用長期個人存取字符更安全。

先決條件

- 已啟用 Argo CD 功能的 Amazon EKS 叢集
- 存放在 AWS Secrets Manager 中的 Git 儲存庫登入資料
- 為 Argo CD 設定存取 Secrets Manager 的 IAM 許可

設定 Argo CD 以使用 Secrets Manager 儲存庫登入資料

1. 將您的 Git 登入資料儲存在 Secrets Manager 中。例如，若要存放 GitHub 個人存取字符：

```
aws secretsmanager create-secret \
  --name argocd/github-token \
  --secret-string '{"username":"git","password":"ghp_xxxxxxxxxxxx"}'
```

2. 確保 Argo CD 功能角色具有擷取秘密的許可：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:GetSecretValue",
        "secretsmanager:DescribeSecret"
      ],
      "Resource": "arn:aws:secretsmanager:region:account-id:secret:argocd/github-
token*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt"
      ],
      "Resource": "arn:aws:kms:region:account-id:key/*",
      "Condition": {
        "StringLike": {
          "kms:EncryptionContext:SecretARN": "arn:aws:secretsmanager:region:account-
id:secret:argocd/*",
          "kms:ViaService": "secretsmanager.*.amazonaws.com"
        }
      }
    }
  ]
}
```

> **Note**
>
> KMS 解密許可是必要的，因為 Secrets Manager 會使用 AWS KMS 加密所有秘密。條件會將解密限制為僅具有 `argocd/` 字首的秘密。如果您使用 Secrets Manager 的預設 AWS

受管金鑰，則此許可已足夠。對於客戶受管 KMS 金鑰，請使用您的特定金鑰 ARN 更新 `Resource` 欄位。

3. 設定 Argo CD 以使用 Secrets Manager 的登入資料。如需有關將秘密從 Secrets Manager 同步到 Argo CD 可以參考的 Kubernetes 秘密的資訊，請參閱 Argo CD 文件中的秘密管理。
4. 建立參考秘密 ARN 的 Argo CD 儲存庫組態：

```
apiVersion: v1
kind: Secret
metadata:
  name: private-repo
  namespace: argocd
  labels:
    argocd.argoproj.io/secret-type: repository
stringData:
  type: git
  url: https://github.com/org/repo
  secretArn: arn:aws:secretsmanager:region:account-id:secret:argocd/github-token
```

如需使用 Argo CD 設定儲存庫存取權的詳細資訊，請參閱 the section called “設定儲存庫”。

使用 Security Lake 集中和分析 EKS 安全資料

Amazon Security Lake 是一項全受管的安全資料湖服務，可讓您集中各種來源的安全資料，包括 Amazon EKS。透過整合 Amazon EKS 與 Security Lake，您可以更深入地了解在 Kubernetes 資源上執行的活動，並增強 Amazon EKS 叢集的安全狀態。

> **Note**
>
> 如需有關搭配 Amazon EKS 使用 Security Lake 和設定資料來源的詳細資訊，請參閱 Amazon Security Lake 文件。

搭配 Amazon EKS 使用 Security Lake 的優勢

集中式安全資料 — Security Lake 會自動從 Amazon EKS 叢集收集和集中安全資料，以及其他 AWS 服務、SaaS 提供者、內部部署來源和第三方來源的資料。這樣可供您全面檢視整個組織的安全狀態。

標準化資料格式：Security Lake 會將收集的資料轉換為 Open Cybersecurity Schema Framework (OCSF) 格式，這是標準的開放原始碼結構描述。此標準化可讓您更輕鬆地進行分析，以及與其他安全工具和服務進行整合。

改善威脅偵測：透過分析集中式安全資料，包括 Amazon EKS 控制平面日誌，您可以更有效地偵測 Amazon EKS 叢集中的潛在可疑活動。這有助於迅速識別和回應安全事件。

簡化資料管理：Security Lake 會使用可自訂保留和複寫設定來管理安全資料的生命週期。這可簡化資料管理任務，並確保您保留必要的資料，以供合規和稽核之用。

針對 Amazon EKS 啟用 Security Lake

1. 為您的 EKS 叢集啟用 Amazon EKS 控制平面記錄。如需詳細說明，請參閱啟用和停用控制平面日誌。
2. 在 Security Lake 中，將 Amazon EKS 稽核日誌新增為來源。然後，Security Lake 會開始收集在 EKS 叢集中執行的 Kubernetes 資源上，活動表現的深入資訊。
3. 根據您的需求，在 Security Lake 中為您的安全資料設定保留和複寫設定。
4. 使用存放在 Security Lake 中的標準化 OCSF 資料，進行事件回應、安全分析，以及與其他 AWS 服務或第三方工具整合。例如，您可以使用 Amazon OpenSearch Ingestion 從 Amazon Security Lake 資料產生安全洞察。

分析 Security Lake 中的 EKS 日誌

Security Lake 會將 EKS 日誌事件標準化為 OCSF 格式，從而讓您能更輕鬆地分析資料並與其他安全事件建立關聯。您可以使用各種工具和服務，例如 Amazon Athena、Amazon QuickSight 或第三方安全分析工具，來查詢標準化資料並將其視覺化。

如需 EKS 日誌事件的 OCSF 映射的詳細資訊，請參閱 OCSF GitHub 儲存庫中的映射參考。

使用 Amazon VPC Lattice 啟用安全的跨叢集連線

Amazon VPC Lattice 是直接內建到 AWS 聯網基礎設施的全受管應用程式聯網服務，可用於跨多個帳戶和虛擬私有雲端 (VPC) 連接、保護和監控服務。有了 Amazon EKS，您可以透過使用 AWS 閘道 API 控制器 (Kubernetes 閘道 API 的實作) 來利用 Amazon VPC Lattice。您可以使用 Amazon VPC Lattice，以簡單且一致的方式設定具有標準 Kubernetes 語意的跨叢集連線。若要開始將 Amazon VPC Lattice 與 Amazon EKS 搭配使用，請參閱 AWS 閘道 API 控制器使用者指南。

使用 Amazon EKS 連接器將 Kubernetes 叢集連接到 Amazon EKS 管理主控台

您可以使用 Amazon EKS 連接器來註冊任何符合標準的 Kubernetes 叢集，並將其連接到 ， AWS 並在 Amazon EKS 主控台中將其視覺化。將叢集連線後，您可以在 Amazon EKS 主控台中查看叢集的狀態、組態和工作負載。您可以使用此功能來檢視 Amazon EKS 主控台中的已連線叢集，但無法管理它們。Amazon EKS 連接器需要 GitHub 上的開放原始碼專案代理程式。有關其他技術內容，包括常見問題集和故障排除，請參閱 the section called “對 EKS 連接器進行疑難排解”。

Amazon EKS 連接器會將下列類型的 Kubernetes 叢集連線至 Amazon EKS。

- 內部部署 Kubernetes 叢集
- 在 Amazon EC2 上執行的自我管理叢集
- 其他雲端供應商的受管叢集

Amazon EKS 連接器考量事項

使用 Amazon EKS 連接器之前，請先詳讀下列注意事項：

- 您必須擁有 Kubernetes 叢集的管理權限，才能將其連線至 Amazon EKS。
- 在連線之前，Kubernetes 叢集必須已具備 Linux 64 位元 (x86) 的工作節點。不支援 ARM 工作節點。
- 您的 Kubernetes 叢集中必須具備工作節點，這些節點可向外存取 `ssm.` 和 `ssmmessages.` Systems Manager 端點。如需詳細資訊，請參閱《 AWS 一般參考》中的 Systems Manager 端點。
- 根據預設，一個區域中最多可連線 10 個叢集。您可以透過 service quota 主控台請求增加配額。如需詳細資訊，請參閱請求增加配額。
- 外部 Kubernetes 叢集僅支援 Amazon EKS `RegisterCluster`、`ListClusters`、`DescribeCluster` 以及 `DeregisterCluster` API。
- 您必須擁有以下許可才能註冊叢集：
  - eks:RegisterCluster
  - ssm:CreateActivation
  - ssm:DeleteActivation
  - iam:PassRole

- 您必須擁有以下許可才能取消註冊叢集：
  - eks:DeregisterCluster
  - ssm:DeleteActivation
  - ssm:DeregisterManagedInstance

Amazon EKS 連接器的必要 IAM 角色

使用 Amazon EKS 連接器需要以下兩種 IAM 角色：

- Amazon EKS 連接器服務連結角色是在第一次註冊叢集時建立。
- 您必須建立 Amazon EKS 連接器代理程式 IAM 角色。如需詳細資訊，請參閱 the section called “連接器 IAM 角色”。

若要為 IAM 主體啟用叢集和工作負載檢視許可，您必須將 `eks-connector` 和 Amazon EKS 連接器叢集角色套用至叢集。請依照授予在 Amazon EKS 主控台上檢視 Kubernetes 叢集資源的存取權中的步驟操作。

將外部 Kubernetes 叢集連線至 Amazon EKS 管理主控台

您可使用下列程序中的多種方法，將外部 Kubernetes 叢集連接到 Amazon EKS。此程序包含兩個步驟：向 Amazon EKS 註冊叢集，以及在叢集中安裝 `eks-connector` 代理程式。

> **Important**
>
> 您必須在完成第一個步驟後的 3 天內完成第二個步驟，以免註冊過期。

考量事項

安裝代理程式時，您可以使用 YAML 資訊清單。或者，如果您向 AWS 管理主控台 或 AWS 命令列界面註冊叢集，則可以使用 Helm。但是，如果您使用 `eksctl` 註冊叢集，則無法使用 Helm 安裝代理程式。

先決條件

- 確認已建立 Amazon EKS 連接器代理程式角色。請遵循建立 Amazon EKS 連接器代理程式角色中的步驟操作。

- 您必須擁有以下許可才能註冊叢集：
  - `eks:RegisterCluster`
  - `ssm:CreateActivation`
  - `ssm:DeleteActivation`
  - `iam:PassRole`

步驟 1：註冊叢集

若要向 Amazon EKS 連接器註冊叢集，您可以使用下列其中一個工具：

- the section called “AWS CLI”
- the section called “AWS 管理主控台”
- the section called “`eksctl`”

AWS CLI

1. AWS 必須安裝 CLI。若要安裝或升級，請參閱安裝 AWS CLI。
2. 對於連接器組態，請指定 Amazon EKS 連接器代理程式 IAM 角色。如需詳細資訊，請參閱the section called “Amazon EKS 連接器的必要 IAM 角色”。

```
aws eks register-cluster \
     --name my-first-registered-cluster \
     --connector-config roleArn=arn:aws: iam::111122223333:role/
AmazonEKSConnectorAgentRole,provider="OTHER" \
     --region aws-region
```

範例輸出如下。

```
{
    "cluster": {
        "name": "my-first-registered-cluster",
        "arn": "arn:aws: eks:region:111122223333:cluster/my-first-registered-
cluster",
        "createdAt": 1627669203.531,
        "ConnectorConfig": {
            "activationId": "xxxxxxxxACTIVATION_IDxxxxxxxx",
            "activationCode": "xxxxxxxxACTIVATION_CODExxxxxxxx",
```

```
            "activationExpiry": 1627672543.0,
            "provider": "OTHER",
            "roleArn": "arn:aws: iam::111122223333:role/AmazonEKSConnectorAgentRole"
        },
        "status": "CREATING"
    }
}
```

您會在接下來的步驟中使用 `aws-region`、`activationId` 和 `activationCode` 值。

AWS 管理主控台

1. 開啟 Amazon EKS 主控台。
2. 選擇 Add cluster (新增叢集)，然後選取 Register (註冊) 以顯示組態頁面。
3. 在 Configure cluster (設定叢集) 區段上，填寫下列欄位：
   - Name (名稱) – 叢集的唯一名稱。
   - Provider (供應商) – 選擇以顯示 Kubernetes 叢集供應商的下拉式清單。若沒有相符的特定供應商，請選取其他。
   - EKS Connector role (EKS 連接器角色)：選取用於連接叢集的角色。
4. 選取 Register cluster (註冊叢集)。
5. 系統隨即會顯示叢集概觀頁面。如果您想使用 Helm Chart，請複製 `helm install` 命令並繼續下一個步驟。如果您想要使用 YAML 清單檔案，請選擇下載 YAML 檔案，將清單檔案檔案下載至本機磁碟機。

> **Important**
>
> 這是您複製 `helm install` 命令或下載此檔案的唯一機會。請勿離開此頁面，以免連結無法存取；若已離開此頁，則必須取消註冊叢集並從頭開始執行步驟。

僅能針對已註冊叢集使用命令或清單檔案檔案一次。如果從 Kubernetes 叢集刪除資源，您必須重新註冊叢集並取得新的清單檔案檔案。

繼續執行下一步，將清單檔案檔案套用到 Kubernetes 叢集。

eksctl

1. 必須安裝 eksctl 版本 0.68 或更新版本。若要將其安裝或升級，請參閱 the section called “建立叢集 (eksctl)”。
2. 透過提供名稱、供應商和區域來註冊叢集。

```
eksctl register cluster --name my-cluster --provider my-provider --region region-code
```

輸出範例：

```
2021-08-19 13:47:26 [#]  creating IAM role "eksctl-20210819194112186040"
2021-08-19 13:47:26 [#]  registered cluster "<name>" successfully
2021-08-19 13:47:26 [#]  wrote file eks-connector.yaml to <current directory>
2021-08-19 13:47:26 [#]  wrote file eks-connector-clusterrole.yaml to <current
 directory>
2021-08-19 13:47:26 [#]  wrote file eks-connector-console-dashboard-full-access-
group.yaml to <current directory>
2021-08-19 13:47:26 [!]  note: "eks-connector-clusterrole.yaml" and "eks-connector-
console-dashboard-full-access-group.yaml" give full EKS Console access to IAM
 identity "<aws-arn>", edit if required; read https://eksctl.io/usage/eks-connector
 for more info
2021-08-19 13:47:26 [#]  run `kubectl apply -f eks-connector.yaml,eks-connector-
clusterrole.yaml,eks-connector-console-dashboard-full-access-group.yaml` before
 expiry to connect the cluster
```

這會在您的本機電腦上建立檔案。這些檔案必須在 3 天內套用至外部叢集，否則註冊將會過期。

3. 在可存取叢集的終端機中，套用產生的檔案：

```
kubectl apply -f eks-connector.yaml,eks-connector-clusterrole.yaml,eks-connector-
console-dashboard-full-access-group.yaml
```

步驟 2：安裝 eks-connector 代理程式

要安裝 eks-connector 代理程式，請使用以下工具之一：

- the section called “Helm”
- the section called “YAML”

Helm

> **Note**
>
> 如果您已向 eksctl 註冊叢集，請使用 YAML 資訊清單方法，而非 Helm Chart 方法。

1. 如果您在上一個步驟中使用 AWS CLI，請將下列命令ACTIVATION_ID中的 ACTIVATION_CODE和分別取代為 activationId、 和 activationCode值。aws-region 將 取代為您在上一個步驟中使用的 AWS 區域。然後執行 命令，在已註冊的叢集上安裝eks-connector代理程式：

   ```
   $ helm install eks-connector \
     --namespace eks-connector \
     oci://public.ecr.aws/eks-connector/eks-connector-chart \
     --set eks.activationCode=ACTIVATION_CODE \
     --set eks.activationId=ACTIVATION_ID \
     --set eks.agentRegion=aws-region
   ```

   如果您在上一個步驟 AWS 管理主控台 中使用 ，請使用您從上一個步驟複製的命令，其中已填入這些值。
2. 檢查已安裝 eks-connector 部署的健康狀態，並等待 Amazon EKS 中已註冊叢集的狀態成為 ACTIVE。

YAML

將 Amazon EKS 連接器清單檔案檔案套用至 Kubernetes 叢集，以便完成連線。若要執行此動作，您必須使用先前描述的方法。如果未於三天內套用資訊清單，Amazon EKS 連接器註冊將到期。如果叢集連線過期，則必須先取消註冊叢集，才能再次連接叢集。

1. 下載 Amazon EKS 連接器 YAML 檔案。

   ```
   curl -O https://amazon-eks.s3.us-west-2.amazonaws.com/eks-connector/manifests/eks-connector/latest/eks-connector.yaml
   ```
2. 編輯 Amazon EKS 連接器 YAML 檔案，以來自上一個步驟輸出的 aws-region、activationId 和 activationCode 來取代 %AWS_REGION%、%EKS_ACTIVATION_ID %、%EKS_ACTIVATION_CODE% 的所有參考。

   下列範例命令可以取代這些值。

```
sed -i "s~%AWS_REGION%~$aws-region~g; s~%EKS_ACTIVATION_ID%~$EKS_ACTIVATION_ID~g; s~
%EKS_ACTIVATION_CODE%~$(echo -n $EKS_ACTIVATION_CODE | base64)~g" eks-connector.yaml
```

> **Important**
>
> 確認您的啟用代碼格式為 Base64。

3. 在可存取叢集的終端機中，您可以執行以下命令來套用已更新的清單檔案檔案：

```
kubectl apply -f eks-connector.yaml
```

4. 將 Amazon EKS 連接器清單檔案，以及角色繫結 YAML 檔案套用至 Kubernetes 叢集後，請確認叢集已成功連線。

```
aws eks describe-cluster \
     --name "my-first-registered-cluster" \
     --region AWS_REGION
```

   輸出應該包含 `status=ACTIVE`。

5. (選用) 將標籤新增至您的叢集。如需詳細資訊，請參閱the section called “標記您的 資源”。

後續步驟

如果您對這些步驟有任何問題，請參閱 the section called “對 EKS 連接器進行疑難排解”。

若要授予其他 IAM 主體對 Amazon EKS 主控台的存取權，以檢視已連接叢集中的 Kubernetes 資源，請參閱 the section called “授予存取所有叢集”。

授予在 Amazon EKS 主控台上檢視 Kubernetes 叢集資源的存取權

授予 IAM 主體存取 Amazon EKS 主控台的許可，以便他們檢視已連線叢集上所執行的 Kubernete 資源等相關資訊。

先決條件

您用來存取 AWS 管理主控台 的 IAM 主體必須符合以下要求：

- 它必須具有 `eks:AccessKubernetesApi` IAM 許可。
- Amazon EKS 連接器服務帳戶可以模擬叢集中的 IAM 主體。這可讓 Amazon EKS Connector 將 IAM 主體映射至 Kubernetes 使用者。

建立和套用 Amazon EKS 叢集角色

1. 下載 `eks-connector` 叢集角色範本。

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/eks-connector/manifests/eks-connector-console-roles/eks-connector-clusterrole.yaml
```

2. 編輯叢集角色範本 YAML 檔案。將對 `%IAM_ARN%` 的參考取代為 IAM 主體的 Amazon Resource Name (ARN)。
3. 將 Amazon EKS 連接器叢集角色 YAML 套用至 Kubernetes 叢集。

```
kubectl apply -f eks-connector-clusterrole.yaml
```

若要讓 IAM 主體在 Amazon EKS 主控台中檢視 Kubernetes 資源，該主體必須將這些工作負載與 Kubernetes `role` 或 `clusterrole` 相關聯，且須擁有這些資源的必要讀取許可。如需詳細資訊，請參閱 Kubernetes 文件中的使用 RBAC 授權。

設定存取連接叢集的 IAM 主體

1. 您可以下載照下範例清單檔案來分別建立 `clusterrole` 和 `clusterrolebinding` 或 `role` 和 `rolebinding`：

   檢視所有命名空間中的 Kubernetes 資源

   - `eks-connector-console-dashboard-full-access-clusterrole` 叢集角色可讓您存取所有可在主控台中視覺化的命名空間和資源。您可以變更 `role`、`clusterrole` 及其相應繫結的名稱，然後再將其套用至叢集。使用以下命令下載範例檔案。

     ```
     curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/eks-connector/manifests/eks-connector-console-roles/eks-connector-console-dashboard-full-access-group.yaml
     ```

   檢視特定命名空間中的 Kubernetes 資源

   - 此檔案中的命名空間為 `default`，因此若要指定不同的命名空間，請在將其套用至叢集之前編輯該檔案。使用以下命令下載範例檔案。

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/eks-connector/manifests/
eks-connector-console-roles/eks-connector-console-dashboard-restricted-access-
group.yaml
```

2. 編輯 YAML 檔案的完整存取權或受限存取權，會將 %IAM_ARN% 的參考取代為 IAM 主體的 Amazon Resource Name (ARN)。
3. 將 YAML 檔案的完整存取權或受限存取權套用到 Kubernetes 叢集。以您自己的值取代 YAML 檔案值。

```
kubectl apply -f eks-connector-console-dashboard-full-access-group.yaml
```

若要檢視已連接叢集中的 Kubernetes 資源，請參閱 the section called “存取叢集資源”。資源索引標籤上的某些資源類型的資料不適用於已連接的叢集。

從 Amazon EKS 主控台取消註冊 Kubernetes 叢集

如果連接叢集已使用完畢，即可將其取消註冊。取消註冊後，Amazon EKS 主控台中便不再顯示該叢集。

您必須擁有以下許可，才能呼叫 deregisterCluster API：

- `eks:DeregisterCluster`
- `ssm:DeleteActivation`
- `ssm:DeregisterManagedInstance`

此程序包含兩個步驟：向 Amazon EKS 取消註冊叢集，以及在叢集中取消安裝 eks-connector 代理程式。

取消註冊 Kubernetes 叢集

若要從 Amazon EKS 連接器取消註冊叢集，您可以使用下列其中一個工具：

- the section called “AWS CLI”
- the section called “AWS 管理主控台”
- the section called “`eksctl`”

AWS CLI

1. AWS 必須安裝 CLI。若要安裝或升級，請參閱安裝 AWS CLI。
2. 確認已建立 Amazon EKS 連接器代理程式角色。
3. 取消註冊已連接叢集。

```
aws eks deregister-cluster \
    --name my-cluster \
    --region region-code
```

AWS 管理主控台

1. 開啟 Amazon EKS 主控台。
2. 選擇 Clusters (叢集)。
3. 在 Clusters (叢集) 頁面上，選取已連接叢集，然後選取 Deregister (取消註冊)。
4. 確認您要取消註冊叢集。

`eksctl`

1. 安裝 `eksctl` 版本 `0.68` 或更新版本。若要將其安裝或升級，請參閱 the section called “建立叢集 (eksctl)”。
2. 確認已建立 Amazon EKS 連接器代理程式角色。
3. 取消註冊已連接叢集：

```
eksctl deregister cluster --name my-cluster
```

清除 Kubernetes 叢集中的資源

要解除安裝 `eks-connector` 代理程式，請使用以下工具之一：

- the section called “Helm”
- the section called “YAML”

Helm

執行下列命令以解除安裝代理程式。

```
helm -n eks-connector uninstall eks-connector
```

YAML

1. 從 Kubernetes 叢集中刪除 Amazon EKS 連接器 YAML 檔案。

   ```
   kubectl delete -f eks-connector.yaml
   ```

2. 若您已建立 `clusterrole` 或 `clusterrolebindings` 來為其他 IAM 主體提供叢集存取權，請將其從 Kubernetes 叢集中刪除。

排解 Amazon EKS 連接器問題

本主題涵蓋您在使用 Amazon EKS 連接器時可能遇到的一些常見錯誤，包括有關如何解決錯誤和因應措施的說明。

基本疑難排解

本節介紹了診斷 Amazon EKS 連接器問題的步驟。

檢查 Amazon EKS 連接器狀態

如需檢查 Amazon EKS 連接器狀態，請輸入：

```
kubectl get pods -n eks-connector
```

檢查 Amazon EKS 連接器日誌

Amazon EKS 連接器 Pod 由三個容器組成。若要擷取所有這些容器的完整日誌，以便您可以檢查它們，請執行以下命令：

- `connector-init`

  ```
  kubectl logs eks-connector-0 --container connector-init -n eks-connector
  ```

```
kubectl logs eks-connector-1 --container connector-init -n eks-connector
```

- connector-proxy

```
kubectl logs eks-connector-0 --container connector-proxy -n eks-connector
kubectl logs eks-connector-1 --container connector-proxy -n eks-connector
```

- connector-agent

```
kubectl exec eks-connector-0 --container connector-agent -n eks-connector -- cat /
var/log/amazon/ssm/amazon-ssm-agent.log
kubectl exec eks-connector-1 --container connector-agent -n eks-connector -- cat /
var/log/amazon/ssm/amazon-ssm-agent.log
```

取得有效的叢集名稱

在clusterName單一 AWS 帳戶和 AWS 區域中，Amazon EKS 叢集由 唯一識別。如果您在 Amazon EKS 中有多個連接的叢集，您可以確認目前 Kubernetes 叢集註冊了哪個 Amazon EKS 叢集。為此，請輸入下列內容來找出目前叢集的 clusterName。

```
kubectl exec eks-connector-0 --container connector-agent -n eks-connector \
  -- cat /var/log/amazon/ssm/amazon-ssm-agent.log | grep -m1 -oE "eks_c:[a-zA-Z0-9_-]+"
 | sed -E "s/^.*eks_c:([a-zA-Z0-9_-]+)_[a-zA-Z0-9]+.*$/\1/"
kubectl exec eks-connector-1 --container connector-agent -n eks-connector \
  -- cat /var/log/amazon/ssm/amazon-ssm-agent.log | grep -m1 -oE "eks_c:[a-zA-Z0-9_-]+"
 | sed -E "s/^.*eks_c:([a-zA-Z0-9_-]+)_[a-zA-Z0-9]+.*$/\1/"
```

其他命令

擷取所需資訊以進行問題疑難排解時，以下命令很有用。

- 使用下列命令來收集 Pod 在 Amazon EKS 連接器中使用的映像。

```
kubectl get pods -n eks-connector -o jsonpath="{.items[*].spec.containers[*].image}"
 | tr -s '[[:space:]]' '\n'
```

- 使用以下命令判斷正在執行 Amazon EKS 連接器的節點名稱。

```
kubectl get pods -n eks-connector -o jsonpath="{.items[*].spec.nodeName}" | tr -s
 '[[:space:]]' '\n'
```

- 執行以下命令，以取得 Kubernetes 用戶端和伺服器版本。

```
kubectl version
```

- 執行以下命令，以取得有關節點的資訊。

```
kubectl get nodes -o wide --show-labels
```

Helm 問題：403 禁止

如果您在執行 helm 安裝命令時收到下列錯誤：

```
Error: INSTALLATION FAILED: unexpected status from HEAD request to https://
public.ecr.aws/v2/eks-connector/eks-connector-chart/manifests/0.0.6: 403 Forbidden
```

您可以執行下列行來修復錯誤：

```
docker logout public.ecr.aws
```

主控台錯誤：叢集停留在待定狀態

如果叢集在您註冊之後卡在 Amazon EKS 主控台`Pending`的狀態，可能是因為 Amazon EKS 連接器尚未成功將叢集連線至 AWS 。對於已註冊的叢集，`Pending` 狀態表示連接尚未成功建立。若要解決此問題，請確定您已將清單檔案應用於目標 Kubernetes 叢集。如果您將其套用於叢集，但叢集仍處於 `Pending` 狀態，則 `eks-connector` statefulset 很可能無法正常運作。若要疑難排解此問題，請參閱本主題the section called “Amazon EKS 連接器 Pod 正在迴圈當機”中的 。

主控台錯誤：使用者 system:serviceaccount:eks-connector:eks-connector 無法在叢集範圍內模擬 API 群組內的資源使用者

Amazon EKS 連接器使用 Kubernetes 使用者模擬，以代表來自 AWS 管理主控台的 IAM 主體採取動作。從`eks-connector`服務帳戶存取 Kubernetes API AWS 的每個委託人都必須獲得許可，才能以 IAM ARN 做為其 Kubernetes 使用者名稱來模擬對應的 Kubernetes 使用者。在以下範例中，IAM ARN 映射到 Kubernetes 使用者。

- 從 AWS 帳戶 *111122223333 ##*的 IAM 使用者會映射到 Kubernetes 使用者。IAM 最佳實務建議您將許可授予角色而非使用者。

```
arn:aws: iam::111122223333:user/john
```

- AWS 帳戶 *111122223333* 的 IAM 角色*###*會映射至 Kubernetes 使用者：

```
arn:aws: iam::111122223333:role/admin
```

結果是 IAM 角色 ARN，而不是 AWS STS 工作階段 ARN。

如需如何設定 `ClusterRole` 和 `ClusterRoleBinding` 以授予 `eks-connector` 服務帳戶權限來模擬映射使用者的說明，請參閱 the section called “授予存取所有叢集”。請確定在 範本中， `%IAM_ARN%` 已取代為 IAM 主體的 AWS 管理主控台 IAM ARN。

主控台錯誤：[...]被禁止：使用者 [...] 無法在叢集範圍中列出「在 API 群組中的 [...]」的資源

請考量下列問題。Amazon EKS 連接器已成功模擬目標 Kubernetes AWS 管理主控台 叢集中的請求 IAM 主體。但是，模擬主體沒有針對 Kubernetes API 操作的 RBAC 許可。

若要解決此問題，有兩種方法可將許可授予其他使用者。如果您先前透過 Helm Chart 安裝了 eks-connector，您可以透過執行下列命令輕鬆授予使用者存取權。以 IAM 角色的 ARN 清單取代 `userARN1` 和 `userARN2` 以提供存取權來檢視 Kubernetes 資源：

```
helm upgrade eks-connector oci://public.ecr.aws/eks-connector/eks-connector-chart \
    --reuse-values \
    --set 'authentication.allowedUserARNs={userARN1,userARN2}'
```

或者，身為叢集管理員，向個別 Kubernetes 使用者授予適當層級的 RBAC 權限。如需詳細資訊和範例，請參閱 the section called “授予存取所有叢集”。

主控台錯誤：Amazon EKS 無法與您的 Kubernetes 叢集 API 伺服器通訊。叢集必須處於作用中狀態，才能成功連接。請在幾分鐘後再試一次。

如果 Amazon EKS 服務無法與目標叢集中的 Amazon EKS 連接器通訊，可能是因為下列其中一個原因：

- 目標叢集中的 Amazon EKS 連接器無法正常運作。
- 目標叢集與 AWS 區域之間的連線能力不佳或連線中斷。

若要解決此問題，請查看 Amazon EKS 連接器日誌。如果未看到 Amazon EKS 連接器的錯誤，請在幾分鐘後重試連線。如果您經常遇到目標叢集的高延遲或間歇性連線，請考慮將叢集重新註冊到離您較近 AWS 的區域。

Amazon EKS 連接器 Pod 正在迴圈當機

有許多原因可能導致 Amazon EKS 連接器 Pod 進入 `CrashLoopBackOff` 狀態。此問題可能涉及 `connector-init` 容器。檢查 Amazon EKS 連接器 Pod 的狀態。

```
kubectl get pods -n eks-connector
```

範例輸出如下。

```
NAME               READY   STATUS                  RESTARTS   AGE
eks-connector-0    0/2     Init:CrashLoopBackOff   1          7s
```

如果您的輸出與之前的輸出相似，請參閱 the section called “檢查 Amazon EKS 連接器日誌” 以排除問題。

無法初始化 eks-connector：InvalidActivation

當您第一次啟動 Amazon EKS 連接器時，它會使用 Amazon Web Services 註冊 `activationId` 和 `activationCode`。註冊可能會失敗，這可能會導致 `connector-init` 容器損毀，產生與以下錯誤相似的錯誤。

```
F1116 20:30:47.261469       1 init.go:43] failed to initiate eks-connector:
 InvalidActivation:
```

若要對此問題進行疑難排解，請考慮以下原因和建議的修正：

- 註冊可能會失敗，因為 `activationId` 和 `activationCode` 不在清單檔案檔案中。如果是這種情況，請確保它們是從 `RegisterCluster` API 操作傳回的正確值，並且 `activationCode` 位於清單檔案檔案中。`activationCode` 新增至 Kubernetes 機密中，因此它必須採用 `base64` 編碼。如需詳細資訊，請參閱the section called “步驟 1：註冊叢集”。
- 註冊可能因啟用過期而失敗。因為出於安全原因，您必須在註冊叢集後的 3 天內啟用 Amazon EKS 連接器。若要解決此問題，請確保在過期日期和時間之前將 Amazon EKS 連接器清單檔案應用於目標 Kubernetes 叢集。若要確認您的啟用過期日期，請呼叫 `DescribeCluster` API 操作。

```
aws eks describe-cluster --name my-cluster
```

在以下範例回應中，過期日期和時間記錄為 2021-11-12T22:28:51.101000-08:00。

```
{
    "cluster": {
        "name": "my-cluster",
        "arn": "arn:aws: eks:region:111122223333:cluster/my-cluster",
        "createdAt": "2021-11-09T22:28:51.449000-08:00",
        "status": "FAILED",
        "tags": {
        },
        "connectorConfig": {
            "activationId": "00000000-0000-0000-0000-000000000000",
            "activationExpiry": "2021-11-12T22:28:51.101000-08:00",
            "provider": "OTHER",
            "roleArn": "arn:aws: iam::111122223333:role/my-connector-role"
        }
    }
}
```

如果超過 activationExpiry，請取消註冊叢集，並將其重新註冊。這樣做會產生一個新的啟用。

叢集節點遺漏對外連線

若要正常運作，Amazon EKS 連接器需要與多個 AWS 端點的傳出連線。您無法在沒有傳出連線至目標 AWS 區域的私有叢集。若要解決此問題，您必須新增必要的對外連線。如需連接器要求的相關資訊，請參閱 the section called “Amazon EKS 連接器考量事項”。

Amazon EKS 連接器 Pod 處於 **ImagePullBackOff** 狀態

如果您執行 get pods 命令，並且 Pod 處於 ImagePullBackOff 狀態，則它們無法正常運作。如果 Amazon EKS 連接器 Pod 處於 ImagePullBackOff 狀態，則它們無法正常運作。檢查 Amazon EKS 連接器 Pod 的狀態。

```
kubectl get pods -n eks-connector
```

範例輸出如下。

```
NAME              READY   STATUS                  RESTARTS   AGE
eks-connector-0   0/2     Init:ImagePullBackOff   0          4s
```

預設的 Amazon EKS 連接器清單檔案檔案引用來自 Amazon ECR Public Gallery 的映像。目標 Kubernetes 叢集可能無法從 Amazon ECR Public Gallery 提取映像。解決 Amazon ECR Public Gallery 映像提取問題，或者考慮選擇在私有容器登錄檔中鏡像映像。

Amazon EKS 連接器常見問答集

問：Amazon EKS 連接器背後的基礎技術如何運作？

答：Amazon EKS 連接器是以 AWS Systems Manager (Systems Manager) 代理程式為基礎。Amazon EKS 連接器作為您的 Kubernetes 叢集上的 `StatefulSet` 執行。它建立叢集的 API 伺服器與 Amazon Web Services 之間的連線並代理通訊。這樣做會在 Amazon EKS 主控台中顯示叢集資料，直到您中斷叢集與 的連線為止 AWS。Systems Manager 代理程式是開放原始碼專案。如需有關此專案的詳細資訊，請參閱 GitHub 專案頁面。

問：我有一個想要連接的內部部署 Kubernetes 叢集。我是否需要開啟防火牆連接埠才能連接此叢集？

答：您不需要開啟任何防火牆連接埠。Kubernetes 叢集只需要對 AWS Regions. AWS services 的傳出連線。 服務絕不會存取您內部部署網路中的資源。Amazon EKS 連接器會在您的叢集上執行，並啟動 的連線 AWS。當叢集註冊完成時， AWS 只有在您從 Amazon EKS 主控台啟動需要叢集上 Kubernetes API 伺服器資訊的動作之後， 才會向 Amazon EKS 連接器發出命令。

問：Amazon EKS 連接器會將哪些資料從我的叢集傳送至 AWS ？

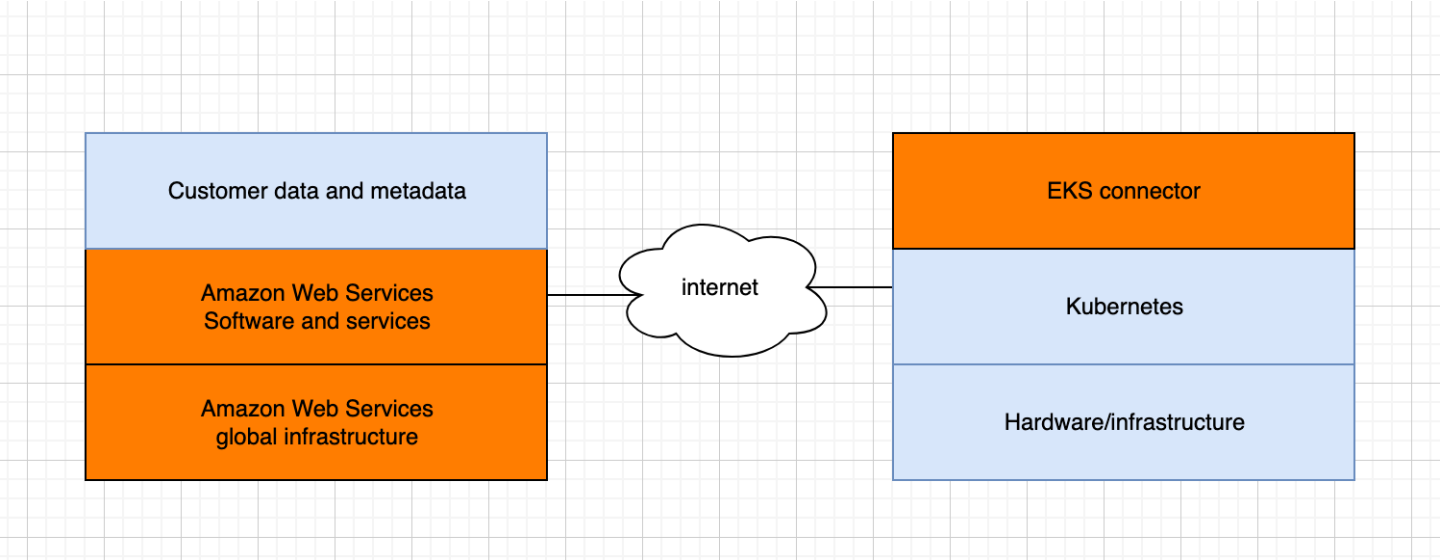
答：Amazon EKS 連接器會傳送您的叢集在 AWS上註冊所需的技術資訊。它還會針對客戶請求的 Amazon EKS 主控台功能傳送叢集和工作負載中繼資料。僅在您從必須將資料傳送至 AWS的 Amazon EKS 主控台或 Amazon EKS API 啟動動作時，Amazon EKS 連接器才會收集或傳送此資料。除了 Kubernetes 版本編號之外，預設 AWS 不會儲存任何資料。它只有在您授權時才會存放資料。

問：我可以在 AWS 區域之外連接叢集嗎？

答：您可以將來自任何位置的叢集連接到 Amazon EKS。此外，您的 Amazon EKS 服務可以位於任何公有商業 AWS 區域。這適用於從叢集到目標 AWS 區域的有效網路連線。我們建議您挑選最接近叢集位置 AWS 的區域，以進行 UI 效能最佳化。例如，如果您的叢集在東京執行，請將叢集連線到東京的 AWS 區域 （ 也就是 `ap-northeast-1` AWS 區域 ） 以獲得低延遲。您可以將叢集從任何位置連線到任何公有商業 AWS 區域中的 Amazon EKS，但中國或 GovCloud AWS 區域除外。

了解 Amazon EKS 連接器安全性

Amazon EKS 連接器是一個在您的 Kubernetes 叢集上執行的開放原始碼元件。此叢集可以位於 AWS 環境之外。這對安全責任產生了其他考量。可透過下圖說明此設定。橘色代表 AWS 責任，藍色則代表客戶責任：



本主題說明了如果連接的叢集位於 AWS 之外的責任模型內的差異。

AWS 責任

- 維護、建置和交付 Amazon EKS 連接器，該連接器是在客戶的 Kubernetes 叢集上執行的開放原始碼元件，並與 AWS 通訊。
- 維護連接的 Kubernetes 叢集和 AWS 服務之間的傳輸與應用程式層級通訊安全。

客戶責任

- Kubernetes 叢集的特定安全，尤其是沿著以下幾行：
  - Kubernetes 祕密必須得到適當的加密和保護。
  - 封鎖對 `eks-connector` 命名空間的存取。
- 設定角色型存取控制 (RBAC) 許可，以管理從 AWS 的 IAM 主體存取。如需說明，請參閱the section called “授予存取所有叢集”。
- 安裝和升級 Amazon EKS 連接器。
- 維護支援連接的 Kubernetes 叢集的硬體、軟體和基礎架構。

- 保護他們的 AWS 帳戶 (例如，透過保護您的根使用者憑證)。

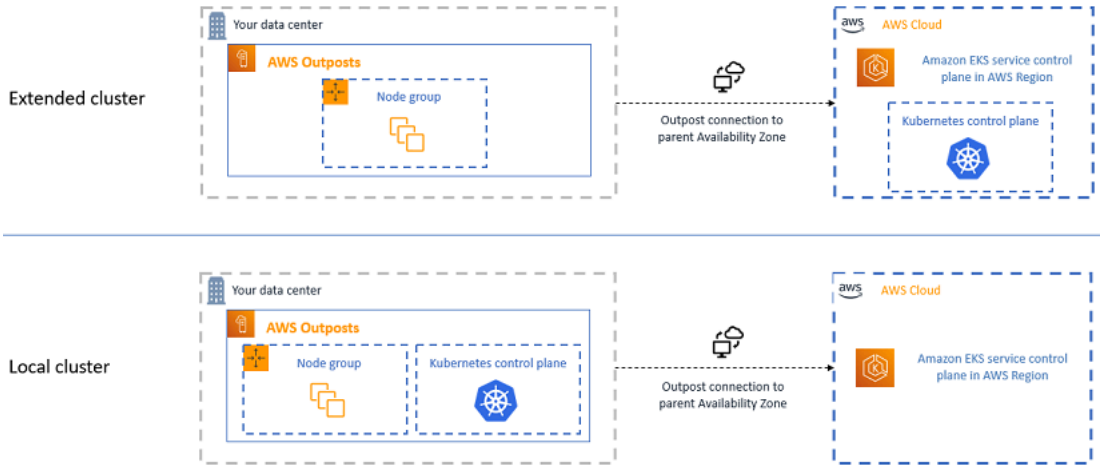
使用 AWS Outposts 部署 Amazon EKS 內部部署

您可以使用 Amazon EKS 在 AWS Outposts 上執行內部部署 Kubernetes 應用程式。您可使用以下方式在 Outpost 上部署 Amazon EKS：

- 擴充叢集 – 在 Outpost 的 AWS 區域和節點中執行 Kubernetes 控制平面。
- 本機叢集 – 執行 Outpost 上的 Kubernetes 控制平面和節點。

針對兩個部署選項，AWS控制平面會受 Kubernetes 完全管理。您可以使用在雲端中使用的相同 Amazon EKS API、工具和主控台，在 Outpost 上建立和執行 Amazon EKS。

下圖顯示這些部署選項。



使用每個部署選項的時機

本機叢集和擴充叢集皆為一般用途的部署選項，可用於多種應用程式。

透過擴充叢集，您可以節省 Outpost 的容量，因為 Kubernetes 控制平面在父 AWS 區域中執行。如果您擁有從 Outpost 到 AWS 區域的可靠網路連線能力，則此選項適用。建議您透過從 AWS 區域中斷網路連線，以設計靜態穩定性的應用程式。Kubernetes 處理 Kubernetes 控制平面和節點之間的網路連線中斷的方式可能會導致應用程式停機。如需有關 Kubernetes 行為的詳細資訊，請參閱 Kubernetes 文件中的排程、先佔和移出。

使用本機叢集，您可以在 Outposts 上本機執行整個 Amazon EKS 叢集。此選項可降低因網路暫時與雲端中斷連線而可能導致的應用程式停機風險。這些網路連線中斷可能是因光纖切斷或天氣事件引起的。由於整個 Amazon EKS 叢集會在 Outposts 本機執行，因此仍可使用應用程式。您可在網路與雲

端中斷連線期間執行叢集操作。如果您需要透過網路中斷連線持續執行 Kubernetes 操作，或者需要資料駐留命令，請選擇本機叢集。

兩個本機叢集實作

Amazon EKS 在 Outposts 機架上支援兩個本機叢集實作。Outpost 設定為使用 Amazon EBS 或 Amazon EC2 執行個體存放區作為其上執行之 EC2 執行個體的根磁碟區類型。Amazon EKS 用於本機叢集控制平面的實作取決於此組態：

- AWS 使用 Amazon EBS 設定的 Outpost。Kubernetes 控制平面在您的 Outpost AWS 帳戶中執行，並堆疊在三個由 Amazon EBS 支援的 Amazon EC2 執行個體`etcd`上。如需詳細資訊，請參閱the section called “執行本機叢集”。
- AWS 使用 EC2 執行個體存放區設定的 Outpost。Amazon EKS 部署更新的本機叢集架構。Kubernetes 控制平面在 Outpost 上受 AWS管服務帳戶中的 6 個 Amazon EC2 執行個體上執行，其中 3 個執行個體專用於 `etcd` ，而 3 個執行個體用於其他控制平面元件。實作與雲端中的 Amazon EKS 具有更大的同位，遵循 Amazon EKS Kubernetes 版本生命週期，並支援其他功能，例如 Amazon EKS 附加元件和 Bottlerocket 型工作者節點（除了 AL2023 之外）。它可用於提供 AWS Outposts 的所有標準 AWS 區域。如需詳細資訊，請參閱the section called “執行本機叢集 (EC2 執行個體存放區）”。

下表概述了每個實作可用的功能。

比較部署選項

下表比較延伸叢集與兩個本機叢集實作之間的差異。

功能	擴充叢集	使用 EBS 的 Outposts 上的本機叢集	Outposts 上的本機叢集搭配 EC2 執行個體存放區
Kubernetes 控制平面位置	AWS 區域	Outpost	Outpost
Kubernetes 控制平面帳戶	AWS 帳戶	AWS 您的帳戶	AWS受管帳戶

功能	擴充叢集	使用 EBS 的 Outposts 上的本機叢集	Outposts 上的本機叢集搭配 EC2 執行個體存放區
Kubernetes 控制平面拓撲	雲端受管	3 個 Amazon EC2 執行個體，堆疊式等	6 個 Amazon EC2 執行個體 (3 `etcd` + 3 個 API 伺服器)
控制平面儲存	雲端受管	Amazon EBS	Amazon EC2 執行個體儲存體
區域可用性	請參閱 Service endpoints (服務端點)。	美國東部 (俄亥俄)、美國東部 (維吉尼亞北部)、美國西部 (加利佛尼亞北部)、美國西部 (奧勒岡)、亞太區域 (首爾)、亞太區域 (新加坡)、亞太區域 (雪梨)、亞太區域 (東京)、加拿大 (中部)、歐洲 (法蘭克福)、歐洲 (愛爾蘭)、歐洲 (倫敦)、中東 (巴林) 和南美洲 (聖保羅)	提供 AWS Outposts 的所有商業 AWS 區域
Kubernetes 次要版本	支援的 Amazon EKS 版本	請參閱 the section called “EKS 平台版本”	支援的 Amazon EKS 版本
平台版本	請參閱 Amazon EKS 平台版本	請參閱 the section called “EKS 平台版本”	請參閱 Amazon EKS 平台版本
Outpost 外形規格	Outpost 機架	Outpost 機架	Outpost 機架
使用者介面	AWS 管理主控台、AWS CLI、Amazon EKS API、、`eksctl` AWS CloudFormation 和 Terraform	AWS 管理主控台、AWS CLI、Amazon EKS API、、`eksctl` AWS CloudFormation 和 Terraform	AWS 管理主控台、AWS CLI、Amazon EKS API、 AWS CloudFormation

功能	擴充叢集	使用 EBS 的 Outposts 上的本機叢集	Outposts 上的本機叢集搭配 EC2 執行個體存放區
受管政策	AmazonEKS ClusterPolicy 與 the section called “AWS 受管政策：AmazonEKSServiceRolePolicy”	AmazonEKS LocalOutpostClusterPolicy 與 the section called “AWS 受管政策：AmazonEKSLocalOutpostServiceRolePolicy”	AmazonEKS ClusterPolicy 與 the section called “AWS 受管政策：AmazonEKSServiceRolePolicy”
叢集 VPC 和子網路	請參閱 the section called “VPC 和子網要求”	請參閱 the section called “建立 VPC 和子網路”	請參閱 the section called “建立 VPC 和子網路 (EC2 執行個體存放區) ”
叢集端點存取	公有或私有，或兩者兼具	僅限私有	私有或公有和私有
Kubernetes API 伺服器身分驗證	AWS Identity and Access Management (IAM)、OIDC 和存取項目	IAM 和 x.509 憑證	IAM、OIDC、存取項目、ConfigMap aws-auth 和x.509憑證（用於網路連線中斷）
節點類型	僅限自我管理	僅限自我管理	僅限自我管理
節點運算類型	Amazon EC2 隨需	Amazon EC2 隨需	Amazon EC2 隨需
節點儲存類型	Amazon EBS gp2 和本機 NVMe SSD	Amazon EBS gp2 和本機 NVMe SSD	本機 NVMe SSD
Amazon EKS 最佳化 AMI	Amazon Linux、Windows 和 Bottlerocket	Amazon Linux	Amazon Linux 和 Bottlerocket
IP 版本	僅限 IPv4	僅限 IPv4	僅限 IPv4

功能	擴充叢集	使用 EBS 的 Outposts 上的本機叢集	Outposts 上的本機叢集搭配 EC2 執行個體存放區
附加元件	Amazon EKS 附加元件，或自我管理附加元件	自我管理的附加元件	Amazon EKS 附加元件 (已驗證清單) 或自我管理附加元件
預設容器網路介面	Kubernetes 專用 Amazon VPC CNI 外掛程式	Kubernetes 專用 Amazon VPC CNI 外掛程式	Kubernetes 專用 Amazon VPC CNI 外掛程式
Kubernetes 控制平面日誌	Amazon CloudWatch Logs	Amazon CloudWatch Logs	Amazon CloudWatch Logs
`etcd` 備份	雲端受管	客戶受管 (`etcdctl` 或 Amazon EBS 磁碟區備份）	由 Amazon EKS 管理
Load balancing	僅限使用 AWS Load Balancer 控制器佈建 Application Load Balancer (無 Network Load Balancer)	僅限使用 AWS Load Balancer 控制器佈建 Application Load Balancer (無 Network Load Balancer)	僅限使用 AWS Load Balancer 控制器佈建 Application Load Balancer (無 Network Load Balancer)
秘密封套加密	支援，請參閱 the section called “啟用秘密加密”	不支援	不支援
服務帳戶的 IAM 角色 (IRSA)	支援，請參閱 the section called “具有 IRSA 的憑證”	不支援	支援（需要連線至 AWS 區域；在網路連線中斷期間無法使用）
EKS Pod 身分識別	支援，請參閱 the section called “Pod 身分識別”	不支援	支援（需要連線至 AWS 區域；在網路連線中斷期間無法使用）

功能	擴充叢集	使用 EBS 的 Outposts 上的本機叢集	Outposts 上的本機叢集搭配 EC2 執行個體存放區
疑難排解	請參閱 疑難排解	請參閱 the section called “對叢集進行疑難排解”	請參閱 the section called “叢集疑難排解 (EC2 執行個體存放區) ”

Important

- 在 RAM 共用 Outpost 上建立本機叢集會將叢集的控制平面置於 Outpost 擁有者的實體存取下方
- 在 EC2 執行個體存放區 Outposts 上，IRSA 和 EKS Pod Identity 取決於 AWS 區域中的 AWS STS。在網路連線中斷期間，使用 IRSA 或 Pod Identity 的工作負載無法取得新的登入資料。如需詳細資訊，請參閱the section called “準備中斷連線 (EC2 執行個體存放區) ”。

在 AWS Outposts 上建立本機 Amazon EKS 叢集以獲得高可用性

您可以使用本機叢集在 AWS Outposts 本機執行整個 Amazon EKS 叢集。這有助於降低因網路暫時與雲端中斷連線而可能導致的應用程式停機風險。這些連線中斷可能是因光纖切斷或天氣事件引起的。由於整個 Kubernetes 叢集會在 Outpost 本機執行，因此仍可使用應用程式。您可在網路與雲端中斷連線期間執行叢集操作。如需詳細資訊，請參閱the section called “準備中斷連線”。下圖顯示本機叢集部署。

本機叢集通常可與 Outpost 機架搭配使用。

支援 AWS 的區域

您可以在下列 AWS 區域中建立本機叢集：美國東部 （俄亥俄）、美國東部 （維吉尼亞北部）、美國西部 （加利佛尼亞北部）、美國西部 （奧勒岡）、亞太區域 （首爾）、亞太區域 （新加坡）、亞太區域 （雪梨）、亞太區域 （東京）、加拿大 （中部）、歐洲 （法蘭克福）、歐洲 （愛爾蘭）、歐洲 （倫敦）、中東 （巴林） 和南美洲 （聖保羅）。如需有關支援功能的詳細資訊，請參閱the section called “比較部署選項”。

在 AWS Outposts 上部署 Amazon EKS 叢集

本主題概述了在 Outpost 上執行本機叢集時要考量的事項。本主題也提供如何在 Outpost 上部署本機叢集的指示。

> **Important**
>
> - 這些考量事項並不是相關 Amazon EKS 文件的重複內容。如果其他 Amazon EKS 文件主題與此處的考量衝突，則請遵循此處的考量事項。
> - 這些考量事項隨時會變更，而且可能會經常變更。因此，我們建議您定期檢閱此主題。
> - 許多考量事項與在 AWS 雲端上建立叢集的考量事項不同。

- 本機叢集僅支援 Outpost 機架。單一本機叢集可以在包含單一邏輯 Outpost 的多個實體 Outpost 機架上執行。單一本機叢集無法跨多個邏輯 Outposts 執行。每個邏輯 Outpost 都有一個單一的 Outpost ARN。
- 本機叢集在 Outpost 上執行和管理您帳戶中的 Kubernetes 控制平面。您無法在 Kubernetes 控制平面執行個體上執行工作負載，或修改 Kubernetes 控制平面元件。這些節點由 Amazon EKS 服務管理。執行自動 Amazon EKS 管理動作 (例如修補) 後，對 Kubernetes 控制平面的變更不會持續存在。
- 本機叢集支援自我管理的附加元件和自我管理的 Amazon Linux 節點群組。Kubernetes 專用 Amazon VPC CNI 外掛程式、kube-proxy 和 CoreDNS 附加元件會自動安裝在本機叢集上。
- 本機叢集需要在 Outpost 上使用 Amazon EBS。您的 Outpost 必須具有可用於 Kubernetes 控制平面儲存的 Amazon EBS。Outpost 僅支援 Amazon EBS gp2 磁碟區。
- 如果使用 Amazon EBS CSI 驅動程式，可支援 Amazon EBS 所支援的 Kubernetes `PersistentVolumes`。
- 本機叢集的控制平面執行個體可在堆疊的高度可用拓撲中進行設定。三分之二的控制平面執行個體必須始終保持運作狀態良好，這樣才能維持規定人數。如果規定人數遺失，請聯絡 AWS 支援，因為需要一些服務端動作才能啟用新的受管執行個體。

先決條件

- 熟悉 Outposts 部署選項、根據容量考量以及 VPC 需求和考量，為 AWS Outposts 上的 Amazon EKS 叢集選取執行個體類型和置放群組。 ???
- 現有的 Outpost。如需詳細資訊，請參閱什麼是 AWS Outpost。

- `kubectl` 命令列工具安裝在您的電腦或 AWS CloudShell 上。版本可以與您的叢集 Kubernetes 版本相同，或是為最多比該版本更舊一版或更新一版的次要版本。例如，如果您的叢集版本為 `1.29`，則可以搭配使用 `kubectl` `1.28`、`1.29` 或 `1.30` 版。若要安裝或升級 `kubectl`，請參閱 the section called “設定 kubectl 和 eksctl”。
- 在您的裝置或 AWS CloudShell 上安裝和設定的 AWS 命令列界面 (AWS CLI) 版本 `1.27.160` `2.12.3`或更新版本。若要檢查您目前的版本，請使用 `aws --version | cut -d / -f2 | cut -d ' ' -f1`。適用於 macOS 的 `yum`、 `apt-get`或 Homebrew 等套件管理員通常是最新版本 CLI AWS 後面的幾個版本。若要安裝最新版本，請參閱《 AWS 命令列界面使用者指南》中的使用 aws 設定安裝 和 快速組態。 https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-quickstart.html#cli-configure-quickstart-config安裝在 AWS CloudShell 中的 AWS CLI 版本也可能是最新版本後面的幾個版本。若要更新它，請參閱《CloudShell AWS 使用者指南》中的將 CLI 安裝到您的主目錄。 AWS CloudShell
- 具有 `create` 和 `describe` Amazon EKS 叢集之許可的 IAM 主體 (使用者或角色)。如需詳細資訊，請參閱the section called “在 Outpost 上建立本機 Kubernetes 叢集”及the section called “所有叢集的清單或描述”。

建立本機 Amazon EKS 叢集後，系統會永久新增建立叢集的 IAM 主體。主體會專門新增至 Kubernetes RBAC 授權資料表作為管理員。此實體具有 `system:masters` 許可。此實體的標識在您的叢集組態中不可見。因此，請務必注意建立叢集的實體，並確保永遠不會將其刪除。一開始，只有建立伺服器的主體可以使用 `kubectl` 對 Kubernetes API 伺服器進行呼叫。如果您使用 主控台建立叢集，請確定當您在叢集上執行`kubectl`命令時，相同的 IAM 登入資料位於 AWS SDK 登入資料鏈中。建立叢集之後，您可以授予其他 IAM 主體存取您的叢集。

建立 Amazon EKS 本機叢集

您可以使用此頁面中所述的下列工具，建立本機叢集：

- the section called “`eksctl`”
- the section called “AWS 管理主控台”

您也可以使用 AWS CLI、Amazon EKS API、AWS SDK、AWS CloudFormation 或 Terraform 在 Outposts 上建立叢集。

`eksctl`

使用 **`eksctl`** 建立本機叢集

1. 在您的裝置0.215.0或 AWS CloudShell 上安裝版本 或更新版本的eksctl命令列工具。如需有關安裝或更新 eksctl 的指示，請參閱 eksctl 文件中的安裝一節。
2. 將隨後的內容複製到您的裝置。取代以下值，然後執行修改後的命令來建立 outpost-control-plane.yaml 檔案：
   - 將 *region-code* 取代為您要在其中建立叢集的支援 AWS 區域。
   - 使用您的叢集名稱取代 *my-cluster*。此名稱僅能使用英數字元 (區分大小寫) 和連字號。必須以英數字元開頭，且長度不可超過 100 個字元。名稱在您要建立叢集 AWS 的區域和 AWS 帳戶中必須是唯一的。
   - 使用現有 VPC 和子網路的 ID 取代 *vpc-ExampleID1* 和 *subnet-ExampleID1*。VPC 和子網路必須符合在 AWS Outposts 上為 Amazon EKS 叢集建立 VPC 和子網路的要求。
   - 使用您的 Outpost 的 ID 取代 *uniqueid*。
   - 使用您 Outpost 上可用的執行個體類型取代 *m5.large*。選擇執行個體類型之前，請先參閱the section called “容量考量事項”。部署了三個控制平面執行個體。您無法變更此數字。

```
cat >outpost-control-plane.yaml <<EOF
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: my-cluster
  region: region-code
  version: "1.36"

vpc:
  clusterEndpoints:
    privateAccess: true
  id: "vpc-vpc-ExampleID1"
  subnets:
    private:
      outpost-subnet-1:
        id: "subnet-subnet-ExampleID1"

outpost:
  controlPlaneOutpostARN: arn:aws: outposts:region-code:111122223333:outpost/op-
uniqueid
  controlPlaneInstanceType: m5.large
EOF
```

如需所有可用選項和預設值的完整清單，請參閱 eksctl 文件中的 AWS Outposts 支援和組態檔案結構描述。

3. 使用您在上一步驟中建立的組態檔案建立叢集。eksctl 會在 Outpost 上建立 VPC 和一個子網路，以便在其中部署叢集。

```
eksctl create cluster -f outpost-control-plane.yaml
```

叢集佈建需要幾分鐘的時間。建立叢集時，會出現幾行輸出。輸出的最後一行類似於下面的範例行。

```
[#]  EKS cluster "my-cluster" in "region-code" region is ready
```

> **Tip**
>
> 若要查看使用 eksctl 建立叢集時可指定的大部分選項，請使用 `eksctl create cluster --help` 命令。若要查看所有可用的選項，您可使用 config 檔案。如需詳細資訊，請參閱 eksctl 文件中的使用組態檔與組態檔結構描述。您可以在 GitHub 上找到組態檔範例。

eksctl 命令會自動為建立叢集的 IAM 主體 (使用者或角色) 建立存取項目，並向該 IAM 主體授予對叢集上的 Kubernetes 物件的管理員許可。如果您不希望叢集建立者對叢集上的 Kubernetes 物件具有管理員存取權，請將下列文字新增至先前的組態檔：`bootstrapClusterCreatorAdminPermissions: false` (與 metadata、vpc 和 outpost 處於同一層級)。如果您新增了該選項，則在建立叢集後，您需要為至少一個 IAM 主體建立存取項目，否則任何 IAM 主體均將無法存取叢集上的 Kubernetes 物件。

AWS 管理主控台

使用 AWS 管理主控台 建立叢集

1. 您需要的現有 VPC 和子網路應符合 Amazon EKS 要求。如需詳細資訊，請參閱the section called “建立 VPC 和子網路”。
2. 如果您已擁有本機叢集 IAM 角色，或者您要使用 eksctl 來建立叢集，則可以略過此步驟。根據預設，eksctl 會為您建立角色。

   a. 執行下列命令以建立 IAM 信任政策 JSON 檔案。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

b. 建立 Amazon EKS 叢集 IAM 角色。若要建立 IAM 角色，必須為建立角色的 IAM 主體指派以下 `iam:CreateRole` 動作 (許可)。

```
aws iam create-role --role-name myAmazonEKSLocalClusterRole --assume-role-policy-
document file://"eks-local-cluster-role-trust-policy.json"
```

c. 將名為 AmazonEKSLocalOutpostClusterPolicy 的 Amazon EKS 受管政策連接至角色。若要將 IAM 政策連接至 IAM 主體，必須為連接政策的 IAM 實體指派以下 IAM 動作之一 (許可)：`iam:AttachUserPolicy` 或 `iam:AttachRolePolicy`。

```
aws iam attach-role-policy --policy-arn arn:aws: iam::aws:policy/
AmazonEKSLocalOutpostClusterPolicy --role-name myAmazonEKSLocalClusterRole
```

3. 開啟 Amazon EKS 主控台。
4. 在主控台畫面頂端，請確定您已選取支援 AWS 的區域。
5. 選取 Add cluster (新增叢集)，然後選取 Create (建立)。
6. 在 Configure cluster (設定叢集) 頁面上，輸入或選取下列欄位的值：
   - Kubernetes 控制平面位置 – 選擇 AWS Outpost。
   - Outpost ID：選擇要在其上建立控制平面的 Outpost 的 ID。
   - Instance type (執行個體類型)：選取執行個體類型。只會顯示 Outpost 中可用的執行個體類型。在下拉式清單中，每個執行個體類型都會說明為其建議的節點數量。選擇執行個體類型之前，請先參閱the section called “容量考量事項”。所有複本均使用相同的執行個體類型進行部署。叢集建立後，就無法再變更執行個體類型。部署了三個控制平面執行個體。您無法變更此數字。

- Name (名稱)：叢集的名稱。它在您的帳戶中必須是唯一的 AWS 。此名稱僅能使用英數字元 (區分大小寫) 和連字號。必須以英數字元開頭，且長度不可超過 100 個字元。名稱在您要建立叢集 AWS 的區域和 AWS 帳戶中必須是唯一的。
- Kubernetes 版本：選擇您要用於您叢集的 Kubernetes 版本。我們建議選取最新版本，除非您需要使用較早版本。
- 叢集服務角色 – 選擇您在上一個步驟中建立的 Amazon EKS 叢集 IAM 角色，以允許 Kubernetes 控制平面管理 AWS 資源。
- Kubernetes 叢集管理員存取權：如果您希望建立叢集的 IAM 主體 (角色或使用者) 對叢集上的 Kubernetes 物件具有管理員存取權，請接受預設值 (允許)。Amazon EKS 會為該 IAM 主體建立存取項目，並向該存取項目授予叢集管理員許可。如需存取項目的詳細資訊，請參閱 the section called “存取項目”。

  如果您不希望建立叢集的 IAM 主體擁有對 Kubernetes 叢集物件的管理員存取權，而是希望其他主體擁有此權限，請選擇「不允許」選項。建立叢集後，具有 IAM 許可的任何 IAM 主體都可以為需要存取 Kubernetes 叢集物件的任何 IAM 主體新增存取項目。如需所需 IAM 許可的詳細資訊，請參閱《服務授權參考》中的 Amazon Elastic Kubernetes Service 定義的動作一節。如果您選擇「不允許」選項且不建立任何存取項目，則任何 IAM 主體均將無法存取叢集上的 Kubernete 物件。
- Tags (標籤) – (選用) 將任何標籤新增到您的叢集。如需詳細資訊，請參閱the section called “標記您的 資源”。完成此頁面後，請選擇下一步。

7. 在 Specify networking (指定網路) 頁面上，選取下列欄位的值：
   - VPC：選擇現有的 VPC。針對您要建立的叢集、任何節點和其他 Kubernetes 資源，VPC 必須有足夠數目的 IP 地址。您的 VPC 必須符合 VPC 要求和考量事項中的要求。
   - Subnets (子網路)：根據預設，前一個欄位指定的 VPC 中的所有可用子網路會預先選取。您選擇的子網路必須符合子網路要求和考量事項中的要求。
   - 安全群組:(選用) 指定一或多個您希望 Amazon EKS 與其建立的網路介面相關聯的安全群組。Amazon EKS 會自動建立一個安全群組，以支援您的叢集和 VPC 之間的通訊。Amazon EKS 將此安全群組及您選擇的任何群組與其建立的網路介面相關聯。如需有關 Amazon EKS 建立的叢集安全群組的詳細資訊，請參閱 the section called “安全群組要求”。您可以修改 Amazon EKS 建立的叢集安全群組中的規則。如果您選擇新增自己的安全群組，在叢集建立後，即無法變更選擇的安全群組。若要使內部部署主機與叢集端點通訊，您必須允許來自叢集安全群組的傳入流量。對於沒有輸入和輸出網際網路連線的叢集 （ 也稱為私有叢集 ），您必須執行下列其中一項：
     - 新增與所需 VPC 端點關聯的安全群組。如需必要端點的詳細資訊，請參閱子網路存取 AWS 服務the section called “使用 介面 VPC 端點”中的 。

- 修改 Amazon EKS 建立的安全群組，以允許來自與 VPC 端點關聯之安全群組的流量。完成此頁面後，請選擇下一步。

8. 在設定可觀測性頁面上，您可以選擇性地選擇要開啟的指標和控制平面日誌記錄選項。根據預設，系統會關閉每個日誌類型。
   - 如需有關 Prometheus 指標選項的詳細資訊，請參閱 the section called “步驟 1：開啟 Prometheus 指標”。
   - 如需控制平面日誌記錄選項的詳細資訊，請參閱 the section called “控制平面日誌”。完成此頁面後，請選擇下一步。
9. 在 Review and create (檢閱並建立) 頁面上，檢閱您在先前頁面上輸入或選取的資訊。如需變更，請選擇 Edit (編輯)。當您感到滿意時，請選擇建立。在佈建叢集時，Status (狀態) 欄位顯示 CREATING (正在建立)。

   叢集佈建需要幾分鐘的時間。

檢視 Amazon EKS 本機叢集

1. 叢集建立之後，您便可以檢視已建立的 Amazon EC2 控制平面執行個體。

```
aws ec2 describe-instances --query 'Reservations[*].Instances[*].{Name:Tags[?
Key==`Name`]|[0].Value}' | grep my-cluster-control-plane
```

   範例輸出如下。

```
"Name": "my-cluster-control-plane-id1"
"Name": "my-cluster-control-plane-id2"
"Name": "my-cluster-control-plane-id3"
```

   每個執行個體均受到 `node-role.eks-local.amazonaws.com/control-plane` 污染，因此沒有在控制平面執行個體上排程工作負載。如需關於污點的詳細資訊，請參閱 Kubernetes 文件中的容差和污點。Amazon EKS 會持續監控本機叢集的狀態。我們會執行自動管理動作，如安全性修補程式和修復運作狀態不良的執行個體。如果本機叢集與雲端中斷連線，我們會完成動作，以確保在重新連線時將叢集修復為健全狀態。

2. 如果您使用 `eksctl` 建立叢集，則可以略過此步驟。`eksctl` 會為您完成此步驟。透過向 `kubectl config` 檔案新增內容，使 `kubectl` 能夠與您的叢集通訊。如需如何建立和更新檔案的說明，請參閱 the section called “透過 kubectl 存取叢集”。

```
aws eks update-kubeconfig --region region-code --name my-cluster
```

範例輸出如下。

```
Added new context arn:aws: eks:region-code:111122223333:cluster/my-cluster to /home/
username/.kube/config
```

3. 若要連線至您的本機叢集的 Kubernetes API 伺服器，請存取子網路的本機閘道，或從 VPC 內進行連線。如需將 Outpost 機架連線至內部部署網路的詳細資訊，請參閱 AWS Outposts 使用者指南中的機架的本機閘道運作方式。如果使用直接 VPC 路由，且 Outpost 子網路具有通往您本機閘道的路由，則 Kubernetes 控制平面執行個體的私有 IP 位址會自動透過您的區域網路廣播。本機叢集的 Kubernetes API 伺服器端點託管在 Amazon Route 53 (Route 53) 中。API 服務端點可透過公有 DNS 伺服器將其解析為 Kubernetes API 伺服器的私有 IP 地址。

   本機叢集的 Kubernetes 控制平面執行個體設定為具有固定私有 IP 位址的靜態彈性網路介面，這些 IP 位址在叢集生命週期內不會變更。在網路連線中斷期間，與 Kubernetes API 伺服器互動的機器可能無法連線至 Route 53。如果是這種情況，建議使用靜態私有 IP 地址來設定 `/etc/hosts` 以繼續操作。我們還建議您設定本機 DNS 伺服器，並將其連接至 Outpost。如需詳細資訊，請參閱 AWS Outposts 文件。執行下列命令以確認與您的叢集建立通訊。

   ```
   kubectl get svc
   ```

   範例輸出如下。

   ```
   NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
   kubernetes   ClusterIP   10.100.0.1   <none>        443/TCP   28h
   ```

4. （選用）當本機叢集與 AWS 雲端處於中斷連線狀態時，對本機叢集測試身分驗證。如需說明，請參閱the section called “準備中斷連線”。

內部資源

Amazon EKS 會在您的叢集上建立下列資源。這些資源供 Amazon EKS 內部使用。為了讓您的叢集正常運作，請勿編輯或修改這些資源。

- 下列鏡射 Pod：
  - `aws-iam-authenticator-`*`node-hostname`*

  - eks-certificates-controller-*node-hostname*
  - etcd-*node-hostname*
  - kube-apiserver-*node-hostname*
  - kube-controller-manager-*node-hostname*
  - kube-scheduler-*node-hostname*
- 下列自我管理的附加元件：
  - kube-system/coredns
  - kube-system/ kube-proxy (在新增第一個節點之前不會建立)
  - kube-system/aws-node (在新增第一個節點之前不會建立)。本機叢集使用 Kubernetes 專用 Amazon VPC CNI 外掛程式進行叢集聯網。請勿變更控制平面執行個體 (名為 aws-node-controlplane-* 的 Pod) 的組態。外掛程式建立新網路介面時，您可以使用組態變數來變更預設值。如需詳細資訊，請參閱 GitHub 上的文件。
- 下列服務：
  - default/kubernetes
  - kube-system/kube-dns
- 名為 eks.system 的 PodSecurityPolicy
- 名為 eks:system:podsecuritypolicy 的 ClusterRole
- 名為 eks:system 的 ClusterRoleBinding
- 除了叢集安全群組之外，Amazon EKS 還會在您的 AWS 帳戶中建立名為 的安全群組eks-local-internal-do-not-use-or-edit-*cluster-name*-*uniqueid* 。此安全群組允許流量在控制平面執行個體上執行的 Kubernetes 元件之間自由流動。

建議的後續步驟：

- 授予建立叢集的 IAM 主體在 中檢視 Kubernetes 資源所需的許可 AWS 管理主控台
- 授予 IAM 實體對叢集的存取權。如果您希望實體在 Amazon EKS 主控台中檢視 Kubernetes 資源，請對實體授予必要許可。
- 設定叢集的日誌
- 熟悉網路連線中斷期間發生的情況。
- 將節點新增至叢集

- 考慮為您的 `etcd` 設定備份計劃。Amazon EKS 不支援本機叢集的 `etcd` 自動備份和還原。如需詳細資訊，請參閱 Kubernetes 文件中的備份 etcd 叢集。兩個主要選項是使用 `etcdctl` 自動拍攝快照或使用 Amazon EBS 儲存磁碟區備份。

了解 AWS Outposts 的 Kubernetes 和 Amazon EKS 平台版本

本機叢集平台版本代表 AWS Outposts 上 Amazon EKS 叢集的功能。這些版本包括在 Kubernetes 控制平面上執行的元件，已啟用 Kubernetes API 伺服器旗標。其也包含目前的 Kubernetes 修補程式版本。每個 Kubernetes 次要版本皆有一或多個相關的 平台版本。適用於不同 Kubernetes 次要版本的平台版本都是獨立的。雲端中本機叢集和 Amazon EKS 叢集的平台版本皆各自獨立。

當本機叢集推出新的 Kubernetes 次要版本 (例如 `1.31`) 時，該 Kubernetes 次要版本的初始平台版本便會從 `eks-local-outposts.1` 開始。但是，Amazon EKS 會定期發佈新平台版本，以啟用新的 Kubernetes 控制平面設定，以及提供安全性問題修正。

當有新的本機叢集平台版本可供次要版本使用時：

- 平台版本編號將會遞增 (`eks-local-outposts.n+1`)。
- Amazon EKS 會自動將所有現有的本機叢集更新到與其對應之 Kubernetes 次要版本的最新平台版本。現有平台版本的自動更新將會逐步發佈。推展程序包含取代在 Outpost 上執行的受管 Kubernetes 控制平面執行個體，一次一個，直到全部 3 個執行個體皆已取代為新的執行個體。
- 如果存在服務中斷的風險，Kubernetes 控制平面執行個體取代程序將會停止進行。只有在其他 2 個 Kubernetes 控制平面執行個體運作狀態良好，並以叢集節點的形式傳遞所有整備條件時，Amazon EKS 才會嘗試取代執行個體。
- 平台版本推展通常需要不到 30 分鐘的時間即可完成。如果叢集長時間處於 `UPDATING` 狀態，請參閱 the section called “對叢集進行疑難排解”並向 AWS Support 尋求協助。除非 AWS Support 指示，否則請勿手動終止 Kubernetes 控制平面執行個體。
- Amazon EKS 可能會發佈具有對應之修補程式版本的新節點 AMI。針對單一 Kubernetes 次要版本，所有修補程式版本在 Kubernetes 控制平面和節點 AMI 之間皆相容。

新的平台版本不會帶來重大變更或導致服務中斷。

本機叢集一律會透過指定 Kubernetes 版本的最新可用平台版本 (`eks-local-outposts.n`) 建立。

的目前及最新平台版本如下表所示。

如需接收有關此特定文件頁面所有來源檔案變更的通知，您可透過 RSS 閱讀器來訂閱以下 URL：

```
https://github.com/awsdocs/amazon-eks-user-guide/commits/mainline/latest/ug/outposts/
eks-outposts-platform-versions.adoc.atom
```

Kubernetes 版本 **1.32**

下列許可控制器已為所有 1.32 平台版本啟用：CertificateApproval、CertificateSigning、CertificateSubjectRestriction、ClusterTrustBundleAttest 以及 ValidatingAdmissionWebhook。

Kubernetes 版本	Amazon EKS 平台版本	版本備註	版本日期
1.32.12	eks-local-outposts.3	Outposts. kube-proxy .v1.32.12 AWS IAM Authenticator 上本機 Amazon EKS 叢集v1.32的 Kubernetes 版本初始版本v0.7.10。適用於 Kubernetes 的 Amazon VPC CNI 外掛程式v1.20.4。CoreDNS v1.11.4. AWS Cloud Controller Manager v1.32.5。Bottlerocket 版本 v1.57.0。	2026 年 4 月 10 日

Kubernetes 版本 **1.31**

下列許可控制器已為所有 1.31 平台版本啟用：CertificateApproval、CertificateSigning、CertificateSubjectRestriction、ClusterTrustBundleAttest 以及 ValidatingAdmissionWebhook。

Kubernetes 版本	Amazon EKS 平台版本	版本備註	版本日期
1.31.14	eks-local-outposts.8	具有安全性修正和增強功能的新平台版本。kube-proxy 已更新至 v1.31.14. AWS IAM Authenticator 已更新至 v0.7.8。Kubernetes 專用 Amazon VPC CNI 外掛程式已更新至 v1.20.4。Bottlerocket 版本已更新至 v1.52.0。	2025 年 12 月 23 日
1.31.12	eks-local-outposts.5	具有安全性修正和增強功能的新平台版本。kube-proxy 已更新至 v1.31.10. AWS IAM Authenticator 已更新至 v0.7.4。Kubernetes 專用 Amazon VPC CNI 外掛程式已更新至 v1.20.2。Bottlerocket 版本已更新至 v1.47.0。	2025 年 10 月 3 日
1.31.9	eks-local-outposts.4	具有安全性修正和增強功能的新平台版本。kube-proxy 已更新至 v1.31.9. AWS IAM Authenticator 已更新至 v0.7.2。Kubernetes 專用 Amazon VPC	2025 年 8 月 9 日

Kubernetes 版本	Amazon EKS 平台版本	版本備註	版本日期
		CNI 外掛程式已更新至 v1.20.0。Bottlerocket 版本已更新至 v1.43.0。	
1.31.7	eks-local-outposts.3	具有安全性修正和增強功能的新平台版本。kube-proxy 已更新至 v1.31.9. AWS IAM Authenticator 已更新至 v0.7.1。Kubernetes 專用 Amazon VPC CNI 外掛程式已更新至 v1.19.5。Bottlerocket 版本已更新至 v1.40.0。	2025 年 6 月 19 日
1.31.6	eks-local-outposts.2	具有安全性修正和增強功能的新平台版本。Bottlerocket 版本已更新至 v1.36.0。	2025 年 4 月 24 日
1.31.6	eks-local-outposts.1	Outpost 上本機 Amazon EKS 叢集的 Kubernetes 版本 v1.31 的初始版本。	2025 年 4 月 9 日

Kubernetes 版本 **1.30**

下列許可控制器已為所有 1.30 平台版本啟用：CertificateApproval、CertificateSigning、CertificateSubjectRestriction、Cluste
以及 ValidatingAdmissionWebhook。

Kubernetes 版本	Amazon EKS 平台版本	版本備註	版本日期
1.30.14	eks-local-outposts.10	具有安全性修正和增強功能的新平台版本。 AWS IAM Authenticator 已更新為 v0.7.8。Kubernetes 專用 Amazon VPC CNI 外掛程式已更新至 v1.20.4。Bottlerocket 版本已更新至 v1.52.0。	2025 年 12 月 23 日
1.30.14	eks-local-outposts.7	具有安全性修正和增強功能的新平台版本。kube-proxy 已更新至 v1.30.14. AWS IAM Authenticator 已更新至 v0.7.4。Kubernetes 專用 Amazon VPC CNI 外掛程式已更新至 v1.20.2。Bottlerocket 版本已更新至 v1.47.0。	2025 年 10 月 3 日
1.30.13	eks-local-outposts.6	具有安全性修正和增強功能的新平台版本。kube-proxy 已更新至 v1.30.13. AWS IAM Authenticator 已更新至 v0.7.2。Kubernetes 專用 Amazon VPC CNI 外掛程式已更新至 v1.20.0。Bottlero	2025 年 8 月 9 日

Kubernetes 版本	Amazon EKS 平台版本	版本備註	版本日期
		cket 版本已更新至 v1.43.0。	
1.30.11	eks-local-outposts.5	具有安全性修正和增強功能的新平台版本。kube-proxy 已更新至 v1.30.11. AWS IAM Authenticator 已更新至 v0.7.1。Kubernetes 專用 Amazon VPC CNI 外掛程式已更新至 v1.19.5。Bottlerocket 版本已更新至 v1.40.0。	2025 年 6 月 19 日
1.30.10	eks-local-outposts.4	具有安全性修正和增強功能的新平台版本。Bottlerocket 版本已更新至 v1.36.0。	2025 年 4 月 24 日

Kubernetes 版本	Amazon EKS 平台版本	版本備註	版本日期
1.30.10	eks-local-outposts.3	具有安全性修正和增強功能的新平台版本。kube-proxy 已更新至 v1.30.10. AWS IAM Authenticator 已更新至 v0.6.29。Kubernetes 專用 Amazon VPC CNI 外掛程式已更新至 v1.19.2。CoreDNS 已更新至 v1.11.4. AWS Cloud Controller Manager 已更新至 v1.30.8。Bottlerocket 版本已更新至 v1.34.0。	2025 年 3 月 27 日
1.30.7	eks-local-outposts.2	具有安全性修正和增強功能的新平台版本。kube-proxy 已更新至 v1.30.7. AWS IAM Authenticator 已更新至 v0.6.28。Kubernetes 專用 Amazon VPC CNI 外掛程式已更新至 v1.19.0。已將 Bottlerocket 版本更新至 v1.29.0。	2025 年 1 月 10 日

Kubernetes 版本	Amazon EKS 平台版本	版本備註	版本日期
1.30.5	eks-local-outposts.1	Outpost 上本機 Amazon EKS 叢集的 Kubernetes 版本 v1.30 的初始版本。	2024 年 11 月 13 日

Kubernetes 版本 1.29

下列許可控制器已為所有 1.29 平台版本啟用：CertificateApproval、CertificateSigning、CertificateSubjectRestriction、Cluste
以及 ValidatingAdmissionWebhook。

Kubernetes 版本	Amazon EKS 平台版本	版本備註	版本日期
1.29.15	eks-local-outposts.13	具有安全性修正和增強功能的新平台版本。 AWS IAM Authenticator 已更新為 v0.7.8。Kubernetes 專用 Amazon VPC CNI 外掛程式已更新至 v1.20.4。Bottlerocket 版本已更新至 v1.52.0。	2025 年 12 月 23 日
1.29.15	eks-local-outposts.10	具有安全性修正和增強功能的新平台版本。 AWS IAM Authenticator 已更新為 v0.7.4。Kubernetes 專用 Amazon VPC CNI 外掛程式已更新至 v1.20.2。Bottlero	2025 年 10 月 3 日

Kubernetes 版本	Amazon EKS 平台版本	版本備註	版本日期
		cket 版本已更新至 v1.47.0。	
1.29.15	eks-local-outposts.9	具有安全性修正和增強功能的新平台版本。 AWS IAM Authenticator 已更新為 v0.7.2。Kubernetes 專用 Amazon VPC CNI 外掛程式已更新至 v1.20.0。Bottlerocket 版本已更新至 v1.43.0。	2025 年 8 月 9 日
1.29.15	eks-local-outposts.8	具有安全性修正和增強功能的新平台版本。kube-proxy 已更新至 v1.29.15. AWS IAM Authenticator 已更新至 v0.7.1。Kubernetes 專用 Amazon VPC CNI 外掛程式已更新至 v1.19.5。Bottlerocket 版本已更新至 v1.40.0。	2025 年 6 月 19 日
1.29.14	eks-local-outposts.7	具有安全性修正和增強功能的新平台版本。Bottlerocket 版本已更新至 v1.36.0。	2025 年 3 月 24 日

Kubernetes 版本	Amazon EKS 平台版本	版本備註	版本日期
1.29.14	eks-local-outposts.6	具有安全性修正和增強功能的新平台版本。kube-proxy 已更新至 v1.29.14。Kubernetes 專用 Amazon VPC CNI 外掛程式已更新至 v1.19.2。CoreDNS 已更新至 v1.11.4. AWS Cloud Controller Manager 已更新至 v1.29.8。Bottlerocket 版本已更新至 v1.34.0。	2025 年 3 月 27 日
1.29.11	eks-local-outposts.5	具有安全性修正和增強功能的新平台版本。kube-proxy 已更新至 v1.29.11。Kubernetes 專用 Amazon VPC CNI 外掛程式已更新至 v1.19.0。已將 CoreDNS 映像更新至 v1.11.3。已將 Bottlerocket 版本更新至 v1.29.0。	2025 年 1 月 10 日

Kubernetes 版本	Amazon EKS 平台版本	版本備註	版本日期
1.29.9	eks-local-outposts.4	具有安全性修正和增強功能的新平台版本。kube-proxy 已更新至 v1.29.9. AWS IAM Authenticator 已更新至 v0.6.26。已將 Bottlerocket 版本更新至 v1.26.0。	2024 年 11 月 8 日
1.29.6	eks-local-outposts.3	具有安全性修正和增強功能的新平台版本。已將 Bottlerocket 版本更新至 v1.22.0。	2024 年 10 月 22 日
1.29.6	eks-local-outposts.2	具有安全性修正和增強功能的新平台版本。已將 Bottlerocket 版本更新至 v1.21.0。	2024 年 8 月 27 日
1.29.6	eks-local-outposts.1	Outpost 上本機 Amazon EKS 叢集的 Kubernetes 版本 v1.29 的初始版本。	2024 年 8 月 20 日

在 AWS Outpost 上為 Amazon EKS 叢集建立 VPC 和子網路

建立本機叢集時，您需要指定 VPC 與至少一個在 Outpost 上執行的私人子網路。本主題概述了本機叢集的 VPC 和子網路要求和考量事項。

VPC 要求和注意事項

建立本機叢集時，您指定的 VPC 必須符合下列要求和考量事項：

- 請確保 VPC 具有足夠的 IP 位址用於本機叢集、任何節點以及您要建立的其他 Kubernetes 資源。如果要使用的 VPC 沒有足夠的 IP 位址，請增加可用 IP 位址的數目。您可以透過將其他無類別域間

路由 (CIDR) 區塊與 VPC 關聯來完成此操作。您可以在建立叢集的之前或之後，將私有 (RFC 1918) 和公有 (非 RFC 1918) CIDR 區塊與 VPC 關聯。叢集可能最多需要 5 小時才能辨識您與 VPC 關聯的 CIDR 區塊。

- VPC 無法指派 IP 字首或 IPv6 CIDR 區塊。由於這些限制，該資訊未涵蓋在將更多 IP 位址指派給具有字首的 Amazon EKS 節點中，並且 the section called “IPv6” 不適用於您的 VPC。
- VPC 已啟用 DNS 主機名稱和 DNS 解析。如果沒有這些功能，本機叢集無法建立，您需要啟用這些功能並重新建立叢集。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的 VPC 的 DNS 屬性。
- 若要透過區域網路存取本機叢集，VPC 必須與 Outpost 的本機閘道路由表關聯。如需詳細資訊，請參閱 AWS Outposts 使用者指南中的 VPC 關聯。

子網需求和注意事項

在建立叢集時，指定至少一個私有子網路。如果您指定多個子網路，Kubernetes 控制平面執行個體會平均分散在子網路上。如果指定多個子網路，則子網路必須存在於相同的 Outpost 上。此外，子網路還必須具有適當的路由和安全群組許可，才能互相通訊。在建立本機叢集時，您指定的子網路必須符合下列要求：

- 子網路都位於相同的邏輯 Outpost 上。
- 這些子網路一起至少有三個可用於 Kubernetes 控制平面執行個體的 IP 位址。如果指定三個子網路，每個子網路必須至少有一個可用的 IP 地址。如果指定兩個子網路，每個子網路必須至少有兩個可用的 IP 地址。如果指定一個子網路，該子網路必須至少有三個可用的 IP 地址。
- 子網路具有 Outpost 機架本機閘道的路由，以存取您本機網路上的 Kubernetes API 伺服器。如果子網路沒有 Outpost 機架本機閘道的路由，您必須在 VPC 內與 Kubernetes API 伺服器進行通訊。
- 子網必須使用 IP 地址型命名。Amazon EKS 不支援 Amazon EC2 資源型命名。

子網路存取 AWS 服務

Outposts 上的本機叢集私有子網路必須能夠與區域 AWS 服務通訊。為此，您可以使用 NAT 閘道進行網際網路對外存取，或者，如果您想要所有流量在 VPC 內保持私密，可使用介面 VPC 端點。

使用 NAT 閘道

Outposts 上的本機叢集私有子網路必須具有關聯的路由表，此路由表具有 Outpost 父可用區域公有子網路中 NAT 閘道的路由。公有子網路必須擁有到網際網路閘道的路由。此 NAT 閘道會啟用傳出網際網路存取，並防止來自網際網路未經要求的傳入連線連接到 Outpost 上的執行個體。

使用 介面 VPC 端點

如果 Outposts 上的本機叢集私有子網路沒有傳出網際網路連線，或者如果您想要所有流量在 VPC 內保持私密，則在建立叢集之前，您必須在區域子網路中建立下列介面 VPC 端點和閘道端點。

Endpoint	端點類型
com.amazonaws. *region-code* .ssm	介面
com.amazonaws. *region-code* .ssmmessages	介面
com.amazonaws. *region-code* .ec2messages	介面
com.amazonaws. *region-code* .ec2	介面
com.amazonaws. *region-code* .secretsmanager	介面
com.amazonaws. *region-code* .logs	介面
com.amazonaws. *region-code* .sts	介面
com.amazonaws. *region-code* .ecr.api	介面
com.amazonaws. *region-code* .ecr.dkr	介面
com.amazonaws. *region-code* .s3	閘道

端點必須符合下列需求：

- 在位於 Outpost 父可用區域的私有子網路中建立
- 已啟用私有 DNS 名稱
- 擁有連接的安全群組，該群組允許來自私有 Outpost 子網路之 CIDR 範圍的輸入 HTTPS 流量。

建立端點會產生費用。如需詳細資訊，請參閱 AWS PrivateLink 定價。如果您的 Pod 需要存取其他 AWS 服務，則需要建立其他端點。如需端點的完整清單，請參閱AWS 與 AWS PrivateLink 整合的 服務。

建立 VPC

您可以使用下列其中一個 AWS CloudFormation 範本，建立符合先前需求的 VPC：

- 範本 1 – 此範本會在 Outpost 上建立具有一個私有子網路的 VPC，並在 AWS 區域中建立一個公有子網路。私有子網路透過位於 AWS 區域的公有子網路中的 NAT Gateway 路由至網際網路。此範本可用於在具有輸出網際網路存取權的子網路中建立本機叢集。
- 範本 2：此範本會在 Outpost 上建立具有一個私有子網路的 VPC，以及在沒有輸入或輸出網際網路存取權 (亦稱為私有子網路) 的子網路中建立本機叢集所需的最基本的一組 VPC 端點。

在 AWS Outposts 上準備本機 Amazon EKS 叢集以進行網路連線中斷

如果您的本機網路與 AWS 雲端失去連線，您可以繼續使用 Outpost 上的本機 Amazon EKS 叢集。本主題包含如何準備本機叢集以應對網路連線中斷和相關考量事項。

- 本機叢集可在臨時、意外的網路中斷連線期間實現穩定性和持續操作。 AWS Outpost 仍是完全連線的方案，可做為資料中心中 AWS 雲端的延伸。如果您的 Outpost 和 AWS Cloud 之間發生網路中斷連線，建議您嘗試還原連線。如需說明，請參閱《AWS Outposts 使用者指南》中的 Outposts 機架網路故障診斷檢查清單。 AWS 如需有關如何針對本機叢集問題進行疑難排解的詳細資訊，請參閱 the section called “對叢集進行疑難排解”。
- Outpost 會發射 `ConnectedStatus` 指標，您可用此指標來監控 Outpost 的連線狀態。如需詳細資訊，請參閱 Outposts 使用者指南中的 Outposts 指標。 AWS
- 本機叢集會使用 IAM 作為預設身分驗證機制，而該機制使用 Kubernetes 的AWS Identity and Access Management 驗證器。IAM 在網路連線中斷期間無法使用。因此，本機叢集會使用 `x.509` 憑證支援替代身分驗證機制，您可使用這些憑證在網路連線中斷期間連線至叢集。如需有關如何取得和使用您叢集 `x.509` 憑證的資訊，請參閱 the section called “在網路連線中斷期間針對本機叢集進行身分驗證”。
- 如果您無法在網路連線中斷期間存取 Route 53，請考慮在內部部署環境中使用本機 DNS 伺服器。Kubernetes 控制平面執行個體會使用靜態 IP 位址。您可使用端點主機名稱和 IP 地址來設定用來連線至叢集的主機，以此作為使用本機 DNS 伺服器的替代方法。如需詳細資訊，請參閱 AWS Outposts 使用者指南中的 DNS。
- 如果您預期應用程式流量會在網路連線中斷期間增加，則可在連線至雲端時在叢集中佈建備用運算容量。Amazon EC2 執行個體包含在 AWS Outpost 的價格中。因此，執行備用執行個體不會影響您的 AWS 用量成本。
- 在網路連線中斷期間，若要啟用工作負載的建立、更新和擴展操作，則必須可透過區域網路存取應用程式的容器映像，且叢集須有足夠容量。本機叢集不會為您託管容器登錄檔。如果先前已在這些節點

上執行 Pod，則系統會在節點上快取容器映像。如果您通常會從雲端中的 Amazon ECR 提取應用程式的容器映像，則請考慮執行本機快取或登錄檔。如果您在網路連線中斷期間需要建立、更新和擴展工作負載資源的操作，則本機快取或登錄會很有幫助。

- 本機叢集會使用 Amazon EBS 作為持久性磁碟區的預設儲存類別，並使用 Amazon EBS CSI 驅動程式管理 Amazon EBS 持久性磁碟區的生命週期。在網路連線中斷期間，無法建立、更新或擴展由 Amazon EBS 支援的 Pod。這是因為這些操作需要呼叫雲端中的 Amazon EBS API。如果您要在本機叢集上部署具狀態的工作負載，且在網路連線中斷期間需要建立、更新或擴展操作，則請考慮使用替代儲存機制。
- 如果 AWS Outposts 無法存取相關的 AWS 區域內 APIs （例如 Amazon EBS 或 Amazon S3 APIs)，則無法建立或刪除 Amazon EBS 快照。
- 將 ALB （輸入）與 AWS Certificate Manager (ACM) 整合時，會將憑證推送並儲存在 AWS Outposts ALB 運算執行個體的記憶體中。如果 AWS 區域中斷連線，目前的 TLS 終止將繼續運作。在此內容中變動操作將會失敗 (例如新的輸入定義、新的 ACM 型憑證 API 操作、ALB 運算規模或憑證輪換)。如需詳細資訊，請參閱 AWS Certificate Manager 使用者指南中的對受管憑證續約進行故障診斷。
- Amazon EKS 控制平面日誌會在網路連線中斷期間於 Kubernetes 控制平面執行個體上進行本機快取。重新連線時，日誌會傳送至父 AWS 區域中的 CloudWatch Logs。您可以使用 Prometheus、Grafana 或 Amazon EKS 合作夥伴解決方案，以透過 Kubernetes API 伺服器的指標端點或使用日誌的 Fluent Bit 在本機上監控叢集。
- 如果您針對應用程式流量使用 Outposts 上的負載 AWS Load Balancer控制器，則負載 AWS Load Balancer控制器前方的現有 Pod 會在網路中斷期間繼續接收流量。在網路連線中斷期間建立的新 Pod 不會接收流量，直到 Outpost 重新連線至 AWS 雲端為止。連線至 AWS 雲端時，請考慮為您的應用程式設定複本計數，以在網路中斷連線期間滿足您的擴展需求。
- Kubernetes 專用 Amazon VPC CNI 外掛程式預設為次要 IP 模式。其使用 `WARM_ENI_TARGET=1` 來設定，這會允許外掛程式保留可用 IP 位址的「完整彈性網路介面」。請依據中斷連線狀態期間的擴展需求來考慮變更 `WARM_ENI_TARGET`、`WARM_IP_TARGET` 和 `MINIMUM_IP_TARGET` 值。如需詳細資訊，請參閱 GitHub 上外掛程式的 readme 檔案。如需每個執行個體類型支援的 Pod 數目上限清單，請參閱 GitHub 上的 eni-max-pods.txt 檔案。

調校 Kubernetes Pod 容錯移轉行為

在網路中斷連線期間，Kubernetes 節點生命週期控制器會將無法連線的節點標記為具有`node.kubernetes.io/unreachable`污點`NoExecute`效果的節點。根據預設，沒有相符公差的 Pod 會在 300 秒 (5 分鐘） 後移出。您可以使用 DaemonSets、在應用程式 Pod `tolerationSeconds`上設定，或實作自訂控制器來調整此行為，讓 Pod 在暫時中斷連線期間保留在

其節點上，而不會進行不必要的移出。如需詳細指引和範例，請參閱《Amazon EKS 最佳實務指南》中的 https://docs.aws.amazon.com/eks/latest/best-practices/hybrid-nodes-network-disconnection-best-practices.html#tune_kubernetes_pod_failover_behavior【Tune Kubernetes Pod 容錯移轉行為】。

在網路連線中斷期間針對本機叢集進行身分驗證

AWS 網路連線中斷期間無法使用 Identity and Access Management (IAM)。中斷連線時，您無法使用 IAM 憑證對本機叢集進行身分驗證。但是，您可以在中斷連線時使用 x509 憑證，透過區域網路連接至您的叢集。您需要下載並存放客戶端 X509 憑證，以在中斷連線期間使用。本此主題中，您將了解如何在中斷連線時建立和使用憑證，以對叢集進行身分驗證。

1. 建立憑證簽署請求。

   a. 產生憑證簽署請求。

   ```
   openssl req -new -newkey rsa:4096 -nodes -days 365 \
       -keyout admin.key -out admin.csr -subj "/CN=admin"
   ```

   b. 在 Kubernetes 中建立憑證簽署請求。

   ```
   BASE64_CSR=$(cat admin.csr | base64 -w 0)
   cat << EOF > admin-csr.yaml
   apiVersion: certificates.k8s.io/v1
   kind: CertificateSigningRequest
   metadata:
     name: admin-csr
   spec:
     signerName: kubernetes.io/kube-apiserver-client
     request: ${BASE64_CSR}
     usages:
     - client auth
   EOF
   ```

2. 使用 kubectl 建立憑證簽署請求。

   ```
   kubectl create -f admin-csr.yaml
   ```

3. 檢查憑證簽署請求的狀態。

   ```
   kubectl get csr admin-csr
   ```

範例輸出如下。

```
NAME       AGE   REQUESTOR                      CONDITION
admin-csr  11m   kubernetes-admin               Pending
```

Kubernetes 已建立憑證簽署請求。

4. 核准憑證簽署請求。

```
kubectl certificate approve admin-csr
```

5. 重新檢查核准的憑證簽署請求狀態。

```
kubectl get csr admin-csr
```

範例輸出如下。

```
NAME       AGE   REQUESTOR                      CONDITION
admin-csr  11m   kubernetes-admin               Approved
```

6. 擷取並驗證憑證。

a. 擷取憑證。

```
kubectl get csr admin-csr -o jsonpath='{.status.certificate}' | base64 --decode >
 admin.crt
```

b. 驗證憑證。

```
cat admin.crt
```

7. 建立與 admin 使用者綁定的叢集角色。

```
kubectl create clusterrolebinding admin --clusterrole=cluster-admin \
    --user=admin --group=system:masters
```

8. 產生中斷連線狀態的使用者範圍 kubeconfig。

您可以使用下載的 admin 憑證來產生 kubeconfig 檔案。取代下列命令中的 *my-cluster* 和 *apiserver-endpoint*。

```
aws eks describe-cluster --name my-cluster \
    --query "cluster.certificateAuthority" \
    --output text | base64 --decode > ca.crt
```

```
kubectl config --kubeconfig admin.kubeconfig set-cluster my-cluster \
    --certificate-authority=ca.crt --server apiserver-endpoint --embed-certs
```

```
kubectl config --kubeconfig admin.kubeconfig set-credentials admin \
    --client-certificate=admin.crt --client-key=admin.key --embed-certs
```

```
kubectl config --kubeconfig admin.kubeconfig set-context admin@my-cluster \
    --cluster my-cluster --user admin
```

```
kubectl config --kubeconfig admin.kubeconfig use-context admin@my-cluster
```

9. 檢視您的 kubeconfig 檔案。

```
kubectl get nodes --kubeconfig admin.kubeconfig
```

10. 如果您在 Outpost 上已有生產中的服務，請跳過此步驟。如果 Amazon EKS 是您 Outpost 上唯一執行的服務，且 Outpost 目前尚未處於生產中，則您可模擬網路連線中斷。在您使用本機叢集進入生產之前，模擬中斷連線以確保在中斷連線狀態下可存取叢集。

    a. 在將 Outpost 連接到 AWS 區域的聯網裝置上套用防火牆規則。這會中斷 Outpost 連結的服務。您無法建立任何新的執行個體。目前執行的執行個體失去與 AWS 區域和網際網路的連線。

    b. 您可在中斷連線時使用 x509 憑證，測試連結至您本機叢集的連線。請確保將您的 kubeconfig 變更為您在上一步中建立的 admin.kubeconfig。使用您本機叢集的名稱取代 *my-cluster*。

    ```
    kubectl config use-context admin@my-cluster --kubeconfig admin.kubeconfig
    ```

    如果您在本機叢集處於中斷連線狀態時發現任何問題，建議您開啟支援票證。

根據容量考量，為 AWS Outpost 上的 Amazon EKS 叢集選取執行個體類型和置放群組

本主題提供有關為 Outpost 上本機 Amazon EKS 叢集選取 Kubernetes 控制平面執行個體類型以及 (選用) 使用置放群組滿足該叢集的高可用性需求的指導。

為 Outposts 上的本機叢集 Kubernetes 控制平面選取要使用的執行個體類型 (例如 `m5`、`c5` 或 `r5`) 之前，先確認 Outpost 組態上可用的執行個體類型。識別可用的執行個體類型之後，根據工作負載所需的節點數量，來選取執行個體大小 (例如 `large`、`xlarge` 或 `2xlarge`)。下表提供有關選擇執行個體大小的建議。

> **Note**
>
> 執行個體大小必須在您的 Outposts 上設定好插槽。確保在本機叢集的生命週期內，您有足夠的容量可容納 Outposts 上可用大小的三個執行個體。如需可用 Amazon EC2 執行個體類型的清單，請參閱 AWS Outposts 機架功能中的「運算與儲存」章節。

節點數量。	Kubernetes 控制平面執行個體大小
1–20	`large`
21–100	`xlarge`
101–250	`2xlarge`
251–500	`4xlarge`

Kubernetes 控制平面的儲存空間需要每個本機叢集 246 GB 的 Amazon EBS 儲存體，才能滿足 `etcd` 所需 IOPS 佔用的空間。建立本機叢集時，系統會自動為您佈建 Amazon EBS 磁碟區。

控制平面置放

若您未使用該 `OutpostConfig.ControlPlanePlacement.GroupName` 屬性指定置放群組，則為您的 Kubernetes 控制平面所佈建的 Amazon EC2 執行個體不會在 Outpost 上可用的基礎容量上收到任何特定硬體置放的強制執行。

您可以使用置放群組來滿足 Outpost 上本機 Amazon EKS 叢集的高可用性需求。透過在叢集建立期間指定置放群組，可影響 Kubernetes 控制平面執行個體的置放。執行個體會分散到獨立的基礎硬體（機架或主機），在硬體故障時將相關執行個體影響降至最低。

您能夠設定的分布類型取決於部署中所擁有的 Outpost 機架數量。

- 單一邏輯 Outpost 中具有一或兩個實體機架的部署：您必須擁有至少三台主機，這些主機使用您為 Kubernetes 控制平面執行個體選擇的執行個體類型進行設定。使用主機層級分布的分散置放群組可確保所有 Kubernetes 控制平面執行個體在 Outpost 部署中可用的基礎機架內的不同主機上執行。
- 單一邏輯 Outpost 中具有三個或更多實體機架的部署：您必須擁有至少三台主機，這些主機使用您為 Kubernetes 控制平面執行個體選擇的執行個體類型進行設定。使用機架層級分布的分散置放群組可確保所有 Kubernetes 控制平面執行個體在 Outpost 部署中的不同機架上執行。或者，如前一個選項中所述，您也可以使用主機層級分布置放群組。

您負責建立所需的置放群組。您可以在呼叫 `CreateCluster` API 時指定置放群組。如需有關置放群組及其如何建立的詳細資訊，請參閱《Amazon EC2 使用者指南》中的置放群組。

- 指定放置群組後，Outpost 上必須有可用的開槽容量，才能成功建立本機 Amazon EKS 叢集。容量會根據您使用的是主機還是機架分布類型而有所不同。若容量不足，則叢集會維持在 `Creating` 狀態。您可以檢查 DescribeCluster API 回應於運作狀態欄位的 `Insufficient Capacity Error`。您必須釋放容量才能進行建立程序。
- 在 Amazon EKS 本機叢集平台和版本更新期間，系統會使用滾動式更新策略，將叢集中的 Kubernetes 控制平面執行個體取代為新的執行個體。在此取代過程中，每個控制平面執行個體皆會終止，從而釋放各自的插槽。新更新的執行個體已佈建在其位置中。更新後的執行個體可能會被置放於已釋放的插槽中。若插槽已被另一個不相關的執行個體使用，且沒有剩餘的容量符合所需的分布拓撲需求，則叢集會維持在 `Updating` 狀態。您可以在 DescribeCluster API 回應的運作狀態欄位上看到對應的 `Insufficient Capacity Error`。您必須釋放容量，才能進行更新程序並重新建立先前的高可用性層級。
- 每個 AWS 區域中每個帳戶最多可以建立 500 個置放群組。如需詳細資訊，請參閱《Amazon EC2 使用者指南》中的一般規則與限制。

Outposts 上的本機 Amazon EKS AWS 叢集疑難排解

本主題涵蓋一些使用本機叢集時可能遇到的常見錯誤與排除這些錯誤的方法。雖然本機叢集與雲端中的 Amazon EKS 叢集相似，但 Amazon EKS 管理本機叢集的方式仍有些許差異。

> **Important**
>
> 除非 AWS Support 明確指示，否則請勿終止在 Outpost 上執行的任何受管 EKS 本機叢集Kubernetes控制平面執行個體。終止這些執行個體會對本機叢集服務可用性造成風險，包括遺失本機叢集，以防同時終止多個執行個體。EC2 執行個體主控台上的 `eks-local:controlplane-name` 標籤可識別 EKS 本機叢集 Kubernetes 控制平面執行個體。

API 行為

本機叢集透過 Amazon EKS API 建立，但會以非同步方式執行。這表示本機叢集對 Amazon EKS API 的請求會立即傳回。但是，這些請求可能會成功、因輸入驗證錯誤而快速檢錯，或失敗並附上描述性驗證錯誤。此行為與 Kubernetes API 相似。

本機叢集不會變為 `FAILED` 狀態。Amazon EKS 會嘗試以連續的方式使叢集狀態與使用者請求的所需狀態一致。因此，本機叢集可能會長期保持在 `CREATING` 狀態，直到基礎問題解決為止。

描述叢集運作狀態欄位

您可以使用 describe-cluster Amazon EKS AWS CLI 命令來發現本機叢集問題。本機叢集問題可透過 `describe-cluster` 命令回應的 `cluster.health` 欄位顯露。此欄位內含的訊息包括錯誤代碼、描述性訊息和相關資源 ID。此資訊僅可透過 Amazon EKS API 和 AWS CLI 取得。在以下範例中，使用您本機叢集的名稱取代 *`my-cluster`*。

```
aws eks describe-cluster --name my-cluster --query 'cluster.health'
```

範例輸出如下。

```
{
    "issues": [
        {
            "code": "ConfigurationConflict",
            "message": "The instance type 'm5.large' is not supported in Outpost 'my-
outpost-arn'.",
            "resourceIds": [
                "my-cluster-arn"
            ]
        }
    ]
}
```

如果問題無法解決，您可能需要刪除本機叢集並建立新叢集。例如，嘗試使用在 Outpost 上未提供的執行個體類型佈建叢集。下表包含與運作狀態相關的常見錯誤。

錯誤情況	代碼	訊息	ResourceIds
找不到提供的子網路。	ResourceNotFound	The subnet ID *subnet-id* does not exist	所有提供的子網路 ID
提供的子網路不屬於同一 VPC。	ConfigurationConflict	Subnets specified must belong to the same VPC	所有提供的子網路 ID
某些提供的子網路不屬於指定的 Outpost。	ConfigurationConflict	Subnet *subnet-id* expected to be in *outpost-arn*, but is in *other-outpost-arn*	有問題的子網路 ID
某些提供的子網路不屬於任何 Outpost。	ConfigurationConflict	Subnet *subnet-id* is not part of any Outpost	有問題的子網路 ID
某些提供的子網路沒有足夠的可用地址來為控制平面執行個體建立彈性網路介面。	ResourceLimitExceeded	The specified subnet does not have enough free addresses to satisfy the request.	有問題的子網路 ID
您的 Outpost 不支援指定的控制平面執行個體類型。	ConfigurationConflict	The instance type *type* is not supported in Outpost *outpost-arn*	叢集 ARN

錯誤情況	代碼	訊息	ResourceIds
您已終止控制平面 Amazon EC2 執行個體，或 `run-instance` 已成功，但觀察到的狀態變更為 `Terminated` 。您的 Outpost 重新連線後一段時間可能會發生此情況，且 Amazon EBS 內部錯誤會導致 Amazon EC2 內部工作流程失敗。	`InternalFailure`	`EC2 instance state "Terminated" is unexpected`	叢集 ARN
您在 Outpost 上的容量不足。如果 Outpost 與 AWS 區域中斷連線，則在建立叢集時也會發生這種情況。	`ResourceLimitExceeded`	`There is not enough capacity on the Outpost to launch or start the instance.`	叢集 ARN
您的帳戶已超過您的安全群組配額。	`ResourceLimitExceeded`	Amazon EC2 API 傳回的錯誤訊息	目標 VPC ID
您的帳戶已超過您的彈性網路介面配額。	`ResourceLimitExceeded`	Amazon EC2 API 傳回的錯誤訊息	目標子網路 ID
無法透過 AWS Systems Manager 連線控制平面執行個體。如需解決方案，請參閱the section called “無法透過 AWS Systems Manager 存取控制平面執行個體”。	`ClusterUnreachable`	Amazon EKS 控制平面執行個體無法透過 SSM 連線。請確認您的 SSM 和網路組態，並參考 Outpost 上的 EKS 疑難排解文件。	Amazon EC2 執行個體 ID

錯誤情況	代碼	訊息	ResourceIds
取得受管安全群組或彈性網路介面的詳細資訊時發生錯誤。	根據 Amazon EC2 用戶端錯誤代碼。	Amazon EC2 API 傳回的錯誤訊息	所有受管的安全群組 ID
授權或撤銷安全群組輸入規則時發生錯誤。這適用於叢集和控制平面安全群組。	根據 Amazon EC2 用戶端錯誤代碼。	Amazon EC2 API 傳回的錯誤訊息	有問題的安全群組 ID
刪除控制平面執行個體的彈性網路介面時發生錯誤。	根據 Amazon EC2 用戶端錯誤代碼。	Amazon EC2 API 傳回的錯誤訊息	有問題的彈性網路介面 ID

下表列出來自其他 AWS 服務的錯誤，這些錯誤會顯示在`describe-cluster`回應的運作狀態欄位中。

Amazon EC2 錯誤代碼	叢集運作狀態問題代碼	說明
`AuthFailure`	`AccessDenied`	基於各種原因，可能會發生此錯誤。最常見的原因是您意外移除了服務用於從控制平面縮小服務連結角色政策範圍的標籤。如果發生這種情況，Amazon EKS 將無法再管理和監控這些 AWS 資源。
`UnauthorizedOperation`	`AccessDenied`	基於各種原因，可能會發生此錯誤。最常見的原因是您意外移除了服務用於從控制平面縮小服務連結角色政策範圍的標籤。如果發生這種情況，Amazon EKS 將無法再管理和監控這些 AWS 資源。

Amazon EC2 錯誤代碼	叢集運作狀態問題代碼	說明
InvalidSubnetID.NotFound	ResourceNotFound	找不到安全群組輸入規則的子網路 ID 時，會發生此錯誤。
InvalidPermission.NotFound	ResourceNotFound	安全群組輸入規則的許可不正確時，會發生此錯誤。
InvalidGroup.NotFound	ResourceNotFound	找不到安全群組輸入規則的群組時，會發生此錯誤。
InvalidNetworkInterfaceID.NotFound	ResourceNotFound	找不到安全群組輸入規則的網路介面 ID 時，會發生此錯誤。
InsufficientFreeAddressesInSubnet	ResourceLimitExceeded	超過子網路資源配額時，會發生此錯誤。
InsufficientCapacityOnOutpost	ResourceLimitExceeded	超過 Outpost 容量配額時，會發生此錯誤。
NetworkInterfaceLimitExceeded	ResourceLimitExceeded	超過彈性網路介面配額時，會發生此錯誤。
SecurityGroupLimitExceeded	ResourceLimitExceeded	超過安全群組配額時，會發生此錯誤。

Amazon EC2 錯誤代碼	叢集運作狀態問題代碼	說明
VcpuLimitExceeded	ResourceLimitExceeded	在新帳戶中建立 Amazon EC2 執行個體時，便會觀察到這種狀況。此錯誤可能類似以下內容："You have requested more vCPU capacity than your current vCPU limit of 32 allows for the instance bucket that the specified instance type belongs to. Please visit http://aws.amazon.com/contact-us/ec2-request to request an adjustment to this limit."
InvalidParameterValue	ConfigurationConflict	如果 Outpost 不支援指定的執行個體類型，Amazon EC2 會傳回此錯誤代碼。
所有其他失敗	InternalFailure	無

無法建立或修改叢集

本機叢集需要不同於雲端中託管的 Amazon EKS 叢集的許可和政策。叢集無法建立並產生 InvalidPermissions 錯誤時，再次檢查您使用的叢集角色是否擁有連接的 AmazonEKSLocalOutpostClusterPolicy 受管政策。所有其他 API 呼叫需要與雲端中 Amazon EKS 叢集相同的許可集。

叢集卡在 **CREATING** 狀態

建立本機叢集所需的時間量因數個因素而異。這些因素包括您的網路組態、Outpost 組態以及叢集的組態。一般而言，本機叢集會在 15-20 分鐘內建立並變更為 ACTIVE 狀態。如果本機叢集仍處於 CREATING 狀態，則您可以在 cluster.health 輸出欄位中呼叫 describe-cluster 以取得有關原因的資訊。

最常見的問題如下：

- 您的叢集無法從 Systems Manager 所在的 AWS 區域連線至控制平面執行個體。您可以從區域內的堡壘主機呼叫 `aws ssm start-session --target instance-id` 進行驗證。如果該命令不起作用，則請檢查 Systems Manager 是否在控制平面執行個體上執行。或者，另一個解決方法是刪除叢集，然後重新建立叢集。
- 由於 EBS 磁碟區的 KMS 金鑰許可，無法建立控制平面執行個體。使用加密 EBS 磁碟區的使用者受管 KMS 金鑰時，如果無法存取金鑰，控制平面執行個體將會終止。如果執行個體終止，請切換到 AWS 受管 KMS 金鑰，或確保您的使用者受管金鑰政策授予叢集角色必要的許可。
- Systems Manager 控制平面執行個體可能無法存取網際網路。檢查您在建立叢集時提供的子網路是否具有 NAT 閘道和具有網際網路閘道的 VPC。使用 VPC Reachability Analyzer 確認控制平面執行個體是否可連線至網際網路閘道。如需詳細資訊，請參閱 Getting started with VPC Reachability Analyzer (VPC Reachability Analyzer 入門)。
- 您提供的角色 ARN 缺少政策。檢查 AWS 受管政策：AmazonEKSLocalOutpostClusterPolicy 是否已從角色中移除。如果 An AWS CloudFormation 堆疊設定錯誤，也可能發生這種情況。
- 所有提供的子網路必須與相同的 Outpost 關聯，並且必須互相連線。如果在建立叢集時指定多個子網路，Amazon EKS 會嘗試將控制平面執行個體散佈到多個子網路。
- Amazon EKS 受管的安全群組將套用至彈性網路介面。但是，其他組態元素 (例如 NACL 防火牆規則) 可能會與彈性網路介面的規則衝突。
- VPC 和子網路 DNS 組態設定錯誤或遺失。檢閱在 AWS Outpost 上建立 Amazon EKS 叢集的 VPC 和子網路。

叢集卡在 **UPDATING** 狀態

Amazon EKS 會自動將所有現有的本機叢集更新到與其對應之 Kubernetes 次要版本的最新平台版本。如需平台版本的詳細資訊，請參閱 the section called “EKS 平台版本”。

在自動平台版本推展期間，叢集狀態會變更為 UPDATING。更新程序包含將所有 Kubernetes 控制平面執行個體取代為新的執行個體，其中包含針對個別 Kubernetes 次要版本發行的最新安全修補程式和錯誤修正。一般而言，本機叢集平台更新程序會在 30 分鐘內完成，且叢集會變更回 ACTIVE 狀態。如果本機叢集長期保持在 UPDATING 狀態，則您可以呼叫 `describe-cluster`，以在 `cluster.health` 輸出欄位中檢查有關原因的資訊。

Amazon EKS 可確保 3 個 Kubernetes 控制平面執行個體中至少有 2 個是運作狀態良好的操作叢集節點，以維持本機叢集可用性並防止服務中斷。如果本機叢集停滯在 UPDATING 狀態，通常是因為存在一些基礎結構或組態問題，導致在程序繼續進行的情況下，無法保證至少有兩個執行個體可用。因此，更新程序會停止進行，以防止本機叢集服務中斷。

請務必對停滯在 UPDATING 狀態的本機叢集進行孤故障診斷，並解決根本原因，以便更新程序可以完成，並將本機叢集還原為 ACTIVE 狀態，且達到 3 個 Kubernetes 控制平面執行個體的高可用性。

除非 AWS Support 明確指示，否則請勿終止 Outposts 上的任何受管 EKS 本機叢集Kubernetes執行個體。這對卡在 UPDATING 狀態的本機叢集尤其重要，因為另一個控制平面節點可能無法完全正常運作，且終止錯誤的執行個體可能會導致服務中斷，並造成本機叢集資料遺失的風險。

最常見的問題如下：

- 一或多個控制平面執行個體無法連線至 Systems Manager，因為自本機叢集首次建立以來，網路組態變更。您可以從區域內的堡壘主機呼叫 `aws ssm start-session --target instance-id` 進行驗證。如果該命令不起作用，則請檢查 Systems Manager 是否在控制平面執行個體上執行。
- 由於 EBS 磁碟區的 KMS 金鑰許可，無法建立新的控制平面執行個體。使用加密 EBS 磁碟區的使用者受管 KMS 金鑰時，如果無法存取金鑰，控制平面執行個體將會終止。如果執行個體終止，請切換到 AWS 受管 KMS 金鑰，或確保您的使用者受管金鑰政策授予叢集角色必要的許可。
- Systems Manager 控制平面執行個體可能已失去網際網路存取。檢查您在建立叢集時提供的子網路是否具有 NAT 閘道和具有網際網路閘道的 VPC。使用 VPC Reachability Analyzer 確認控制平面執行個體是否可連線至網際網路閘道。如需詳細資訊，請參閱 Getting started with VPC Reachability Analyzer (VPC Reachability Analyzer 入門)。如果您的私有網路沒有傳出網際網路連線，請確保叢集的區域子網路中仍存在所有必要的 VPC 端點和閘道端點 (請參閱 the section called “子網路存取 AWS 服務”)。
- 您提供的角色 ARN 缺少政策。檢查 AWS 受管政策：AmazonEKSLocalOutpostClusterPolicy 是否已從角色中移除。
- 其中一個新的 Kubernetes 控制平面執行個體可能會遇到未預期的引導失敗。如需有關此例外情況的故障診斷和日誌收集的進一步指引，請向 AWS Support Center 填寫票證。

無法將節點加入叢集

- AMI 問題：
  - 您正在使用不相容的 AMI。僅支援 Amazon EKS 最佳化 Amazon Linux 2023 AMIs。如需詳細資訊，請參閱在 AWS Outposts 上建立 Amazon Linux 節點。
  - 如果您使用 an AWS CloudFormation 範本來建立節點，請確定它不是使用不支援的 AMI。
- 遺失 AWS IAM Authenticator `ConfigMap` – 如果遺失，您必須建立它。如需詳細資訊，請參閱 the section called “將 `aws-auth ConfigMap` 套用至您的叢集”。

- 使用錯誤的安全群組 – 確保將 `eks-cluster-sg-cluster-name-uniqueid` 用於您工作節點的安全群組。 AWS CloudFormation 會變更選取的安全群組，以允許每次使用堆疊時有新的安全群組。
- 依照未預期的私人連結 VPC 步驟：傳遞了錯誤的 CA 資料 (`--b64-cluster-ca`) 或 API 端點 (`--apiserver-endpoint`)。

收集日誌

當 Outpost 與其相關聯的 AWS 區域中斷連線時，Kubernetes 叢集可能會繼續正常運作。不過，如果叢集無法正常運作，請遵循在 AWS Outposts 上準備本機 Amazon EKS 叢集以進行網路中斷連線中的疑難排解步驟。如果您遇到其他問題，請聯絡 AWS Support。 AWS Support 可以引導您下載和執行日誌收集工具。如此一來，您就可以從 Kubernetes 叢集控制平面執行個體收集日誌，並將其傳送至 AWS Support 進行進一步調查。

無法透過 AWS Systems Manager 存取控制平面執行個體

當 Amazon EKS 控制平面執行個體無法透過 AWS Systems Manager (Systems Manager) 連線時，Amazon EKS 會顯示叢集的下列錯誤。

```
Amazon EKS control plane instances are not reachable through SSM. Please verify your
 SSM and network configuration, and reference the EKS on Outposts troubleshooting
 documentation.
```

若要解決此問題，請確定您的 VPC 和子網路符合在 AWS Outposts 上為 Amazon EKS 叢集建立 VPC 和子網路的要求，而且您已完成 AWS Systems Manager 使用者指南中設定 Session Manager 的步驟。

使用 EC2 執行個體存放區設定的 AWS Outposts 上的本機 Amazon EKS 叢集概觀

您可以使用本機叢集在 AWS Outposts 本機執行整個 Amazon EKS 叢集。這有助於降低因暫時網路連線中斷至雲端而導致應用程式停機的風險，並有助於解決資料駐留需求。由於整個 Kubernetes 叢集在 Outposts 上於本機執行，因此應用程式仍然可用，您可以在網路與雲端中斷連線期間執行叢集操作。

本機叢集通常可與 Outpost 機架搭配使用。

具有 EC2 執行個體存放區 Outposts 的 Amazon EKS 本機叢集

Amazon EKS 本機叢集支援使用 EC2 執行個體存放區設定的 AWS Outpost。當您使用 EC2 執行個體存放區在 Outposts 上執行本機叢集時，Amazon EKS 會部署更新的本機叢集架構，以提供與 Amazon EKS 雲端叢集更大的同位。

與 EC2 執行個體存放區後端 Outposts 搭配使用的更新架構新增了下列項目：

- 受管控制平面。Kubernetes 控制平面會在 Outpost 上的 AWS受管服務帳戶中執行。Amazon EKS 會管理控制平面生命週期，包括 Amazon EKS 平台版本更新和etcd備份。
- 雲端中與 Amazon EKS 的功能同位更大。使用 Amazon EKS 附加元件、服務帳戶的 IAM 角色、EKS Pod 身分、OIDC 身分驗證和 the section called “存取項目”。除了以 Amazon Linux 2023 為基礎的工作者節點之外，使用 Bottlerocket 做為工作者節點作業系統。
- 符合 Amazon EKS Kubernetes 版本生命週期。新的 Kubernetes 版本和 Amazon EKS 平台版本可在與雲端中 Amazon EKS 相同的頻率上供本機叢集使用。擴充支援 Kubernetes 版本提供與雲端叢集相同的定價。
- 適用於所有支援 AWS Outposts 機架的商業 AWS 區域。

> **Note**
>
> 如果您的 Outpost 是使用 Amazon EBS 而非 EC2 執行個體存放區設定，則本主題中所述的架構不適用於您的 Outpost。使用 EBS 設定的 Outpost 將繼續使用現有的本機叢集實作。如需詳細資訊，請參閱the section called “執行本機叢集”。
> 如果您有興趣使用更新的本機叢集架構在 EBS 後端 Outpost 上建立本機叢集，請聯絡您的 AWS 客戶團隊。

如需本機叢集和延伸叢集feature-by-feature，請參閱 the section called “比較部署選項”。

開始使用

若要在以 EC2 執行個體存放區設定的 Outpost 上建立本機叢集，請參閱 the section called “部署本機叢集 (EC2 執行個體存放區) ”。

建立叢集之前，請檢閱：

- the section called “建立 VPC 和子網路 (EC2 執行個體存放區) ” — 本機叢集的 VPC 和子網路組態。

- the section called “容量考量 (EC2 執行個體存放區) ” — 控制平面和etcd執行個體類型選擇和放置。
- the section called “準備中斷連線 (EC2 執行個體存放區) ” — 中斷連線期間的身分驗證、可觀測性和操作考量。
- the section called “Amazon EKS 附加元件 (EC2 執行個體存放區) ” — 本機叢集的已驗證 Amazon EKS 附加元件集。

在使用 EC2 執行個體存放區設定的 AWS Outposts 上部署 Amazon EKS 本機叢集

本主題提供在以 EC2 執行個體存放區設定的 AWS Outpost 上部署本機 Amazon EKS 叢集的說明。

> **Note**
>
> 如果您的 Outpost 是使用 Amazon EBS 而非 EC2 執行個體存放區設定，則本主題中所述的架構不適用於您的 Outpost。使用 EBS 設定的 Outpost 將繼續使用現有的本機叢集實作。如需詳細資訊，請參閱the section called “部署本機叢集”。
> 如果您有興趣使用更新的本機叢集架構，在 EBS 支援的 Outpost 上建立本機叢集，請聯絡您的 AWS 客戶團隊。

> **Important**
>
> - 本機叢集僅支援 Outpost 機架。單一本機叢集可以在包含單一邏輯 Outpost 的多個實體 Outpost 機架上執行。單一本機叢集無法跨多個邏輯 Outposts 執行。
> - Kubernetes 控制平面會在 Outpost 上的 AWS受管服務帳戶中執行。您無法存取控制平面執行個體、在 Amazon EC2 主控台中檢視它們，或在它們上執行工作負載。
> - 控制平面在外部刻印拓撲中的 6 個 EC2 執行個體 (3 etcd + 3 個 API 伺服器) 上執行。如需容量需求，請參閱 the section called “容量考量 (EC2 執行個體存放區) ”。
> - 適用於 Kubernetes、kube-proxy 和 CoreDNS 的 Amazon VPC CNI 外掛程式會自動安裝在本機叢集上。 ??? CoreDNS 如需已驗證附加元件的完整清單，請參閱 the section called “Amazon EKS 附加元件 (EC2 執行個體存放區) ”。
> - Amazon EKS 會定期備份 etcd。如果 etcd失去規定人數，Amazon EKS 會從最新的快照復原叢集狀態。

先決條件

- 使用 EC2 執行個體存放區設定的現有 Outpost 機架，具有足夠用於控制平面執行個體的虛擬化容量。請參閱 the section called “容量考量 (EC2 執行個體存放區) ”。
- 符合 中所述需求的 VPC 和子網路the section called “建立 VPC 和子網路 (EC2 執行個體存放區) ”。
- `kubectl` 命令列工具安裝在您的電腦或 AWS CloudShell 上。版本可以與叢集的 Kubernetes 版本相同，也可以比叢集的 Kubernetes 版本更早或更晚一個次要版本。若要安裝或升級 `kubectl`，請參閱 the section called “設定 kubectl 和 eksctl”。
- 在您的裝置或 AWS CloudShell 上安裝和設定 AWS CLI 版本 `2.35.3`或更新版本。若要檢查您目前的版本，請使用 `aws --version`。若要安裝或升級，請參閱安裝 AWS CLI。
- IAM 主體 （使用者或角色），具有建立和描述 Amazon EKS 叢集的許可。如需詳細資訊，請參閱the section called “在 Outpost 上建立本機 Kubernetes 叢集”及the section called “所有叢集的清單或描述”。
- 已連接下列 AWS 受管政策的叢集 IAM 角色：AmazonEKSClusterPolicy 和 AmazonEKSServicePolicy。角色信任政策必須允許`eks.amazonaws.com`服務主體擔任該角色。

> **Note**
>
> 當您建立本機叢集時，建立叢集的 IAM 主體會以具有 `system:masters` 許可的管理員身分永久新增至 Kubernetes RBAC 授權表。請務必永遠不要刪除此委託人。

建立本機叢集

您可以使用 AWS 管理主控台、 AWS CLI、Amazon EKS API 或 AWS CloudFormation 建立本機叢集。

當您建立本機叢集時，請注意下列事項：

- 您必須傳遞 Outpost 所在可用區域中的子網路。如果您在多個可用區域中傳遞子網路，請求會失敗。
- 您必須將 `endpointPrivateAccess` 設為 `true`。您也可以選擇啟用 `endpointPublicAccess`。
- 您必須傳遞 的值`outpostConfig`，包括 `outpostArns`、 `controlPlaneInstanceType`和 `etcdInstanceType`。或者，您可以指定置放組態。請參閱 the section called “容量考量 (EC2 執行個體存放區) ”。

AWS CLI

```
aws eks create-cluster \
  --name my-cluster \
  --role-arn arn:aws: iam::111122223333:role/myEKSClusterRole \
  --kubernetes-version 1.36 \
  --resources-vpc-config subnetIds=subnet-
ExampleID1,endpointPrivateAccess=true,endpointPublicAccess=true \
  --logging '{"clusterLogging":[{"types":
["api","audit","authenticator","controllerManager","scheduler"],"enabled":true}]}' \
  --access-config
 authenticationMode=API_AND_CONFIG_MAP,bootstrapClusterCreatorAdminPermissions=true \
  --outpost-config outpostArns=arn:aws: outposts:region-code:111122223333:outpost/op-
uniqueid,controlPlaneInstanceType=m5.large,controlPlanePlacement={spreadLevel=host},etcdInstanc
```

AWS 管理主控台

1. 開啟 Amazon EKS 主控台。
2. 選擇 Create Cluster (建立叢集)。
3. 針對 Kubernetes 控制平面位置，選擇 AWS Outposts。
4. 設定下列項目：
   - 名稱 — 叢集的名稱。
   - Kubernetes 版本 — 要使用的版本。
   - 叢集服務角色 — 具有 AmazonEKSClusterPolicy 和 AmazonEKSServicePolicy 的 IAM 角色。
   - Outpost ID — 要部署的邏輯 Outpost。
   - 控制平面執行個體類型 — Kubernetes API 伺服器、排程器和控制器管理員的執行個體類型。
   - etcd 執行個體類型 — 的執行個體類型`etcd`。
   - 分散層級 — `host`或 `rack` 用於控制平面和`etcd`放置。
5. 設定聯網：VPC、子網路、安全群組。啟用私有端點存取。
6. 設定可觀測性：控制平面記錄。
7. 檢閱和建立。

`CreateCluster` API

```
{
    "name": "my-cluster",
```

```
    "roleArn": "arn:aws: iam::111122223333:role/myEKSClusterRole",
    "version": "1.36",
    "resourcesVpcConfig": {
        "subnetIds": ["subnet-ExampleID1"],
        "endpointPublicAccess": true,
        "endpointPrivateAccess": true
    },
    "logging": {
        "clusterLogging": [{
            "types": ["api", "audit", "authenticator", "controllerManager",
 "scheduler"],
            "enabled": true
        }]
    },
    "accessConfig": {
        "authenticationMode": "API_AND_CONFIG_MAP",
        "bootstrapClusterCreatorAdminPermissions": true
    },
    "outpostConfig": {
        "outpostArns": ["arn:aws: outposts:region-code:111122223333:outpost/op-
uniqueid"],
        "controlPlaneInstanceType": "m5.large",
        "controlPlanePlacement": {
            "spreadLevel": "host"
        },
        "etcdInstanceType": "m5.large",
        "etcdPlacement": {
            "spreadLevel": "host"
        }
    }
}
```

當您在以 EC2 執行個體存放區設定的 Outposts 上建立本機叢集時，需要 `controlPlaneInstanceType`和 `etcdInstanceType` 參數。`controlPlanePlacement` 和 `etcdPlacement` 是選用參數。如果您未指定，則不會套用置放分散策略。

> **Note**
>
> `outpostConfig` 是僅限建立的屬性。若要變更控制平面執行個體類型、etcd執行個體類型、置放或 Outpost，您必須建立新的叢集。

連線至您的叢集

設定 kubectl 以與您的叢集通訊：

```
aws eks update-kubeconfig --region region-code --name my-cluster
```

> **Note**
>
> 如果您使用低於 的 AWS CLI 版本建立叢集2.35.3，則必須更新 kubeconfig以在 args exec 中指定 --cluster-name (而不是 --cluster-id)

叢集的 Kubernetes API 伺服器端點託管於 Amazon Route 53。端點會解析為 Amazon EKS 在子網路中建立的跨帳戶彈性網路介面 (ENIs) 的私有 IP 地址。這些 ENIs具有固定的私有 IP 地址，在整個叢集生命週期中不會變更。

當您建立叢集時，Amazon EKS 會在您指定的子網路中建立 3 個彈性網路介面。這些網路介面啟用您的叢集和 VPC 之間的通訊。每個網路界面在其描述Amazon EKS *cluster-name* 中都有文字。

部署工作者節點

使用自我管理節點群組，將工作者節點部署到您的本機叢集。

1. 部署工作者 node AWS CloudFormation 範本，或在 Outpost 上建立個別 EC2 執行個體。

   ```
   curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2025-11-24/
   amazon-eks-outpost-nodegroup.yaml
   ```

2. 授予節點執行個體角色對叢集的存取權。為節點的 IAM 角色建立存取項目：

   ```
   aws eks create-access-entry \
     --cluster-name my-cluster \
     --principal-arn arn:aws: iam::111122223333:role/myNodeRole \
     --type EC2_LINUX
   ```

   或者，如果您使用 aws-auth ConfigMap 進行身分驗證，請將節點執行個體角色新增至 ConfigMap。請參閱 the section called “Kubernetes API 存取”。

3. 確認您的節點處於 Ready 狀態：

```
kubectl get nodes
```

內部資源

當您建立本機叢集時，Amazon EKS 會自動在`kube-system`命名空間中安裝下列附加元件的自我管理版本：`coredns`、 `kube-proxy`和 `aws-node`(VPC CNI)。您可以選擇透過 Amazon EKS 附加元件 API 安裝這些附加元件的受管版本。如需詳細資訊，請參閱the section called “Amazon EKS 附加元件 (EC2 執行個體存放區 ）”。

Amazon EKS 也會在您的子網路中建立跨帳戶彈性網路介面 (ENIs)，以進行叢集-VPC 通訊。請勿刪除這些網路介面。如果刪除跨帳戶 ENI 或其 IP 地址變更，則每個節點和每個使用靜態 IP 型存取的管理員都必須手動更新。

在使用 EC2 執行個體存放區設定的 AWS Outpost 上，為 Amazon EKS 本機叢集建立 VPC 和子網路

建立本機叢集時，您需要指定 VPC 與至少一個在 Outpost 上執行的私人子網路。本主題概述本機叢集的 VPC 和子網路需求。

> **Note**
>
> 如果您的 Outpost 是使用 Amazon EBS 而非 EC2 執行個體存放區設定，則本主題中所述的架構不適用於您的 Outpost。使用 EBS 設定的 Outpost 將繼續使用現有的本機叢集實作。如需詳細資訊，請參閱the section called “建立 VPC 和子網路”。
> 如果您有興趣使用更新的本機叢集架構在 EBS 後端 Outpost 上建立本機叢集，請聯絡您的 AWS 客戶團隊。

VPC 要求

- VPC 必須有足夠的 IP 地址，才能用於本機叢集、任何節點，以及您要建立的其他 Kubernetes 資源。您可以在建立叢集之前或之後，將其他 CIDR 區塊與 VPC 建立關聯。新關聯的 CIDR 區塊最多可能需要 1 小時才能辨識。
- VPC 必須具有 DNS 主機名稱和 DNS 解析支援。否則，節點無法註冊至您的叢集。如需詳細資訊，請參閱 Amazon VPC 使用者指南中的 VPC 的 DNS 屬性。

- 若要透過區域網路存取本機叢集，VPC 必須與 Outpost 的本機閘道路由表關聯。如需詳細資訊，請參閱 AWS Outposts 使用者指南中的 VPC 關聯。

子網路需求

當您建立本機叢集時，請在 Outpost 上指定至少一個私有子網路。Amazon EKS 會在您指定的子網路中建立三個跨帳戶彈性網路介面。這些網路介面啟用您的叢集和 VPC 之間的通訊。

您指定的子網路必須符合下列要求：

- 所有子網路都必須位於相同的邏輯 Outpost 上。
- 所有子網路都必須位於相同的 VPC 中。
- 所有子網路都必須位於 Outpost 所在的可用區域中。
- 每個子網路必須至少有 3 個可用的 Amazon EKS 跨帳戶彈性網路介面 IP 地址。
- 子網必須使用 IP 地址型命名。Amazon EKS 不支援 Amazon EC2 資源型命名。
- 子網路必須具有通往 Outpost 機架本機閘道的路由，才能透過本機網路存取 Kubernetes API 伺服器。如果子網路沒有本機閘道的路由，您必須從 VPC 內與 Kubernetes API 伺服器通訊。
- 子網路不能位於下列可用區域。

AWS 區域	區域名稱	不允許的可用區域 ID
us-east-1	美國東部 (維吉尼亞北部)	use1-az3
us-west-1	美國西部 (加利佛尼亞北部)	usw1-az2
ca-central-1	加拿大 (中部)	cac1-az3

子網路存取 AWS 服務

本機叢集需要連線至 AWS 區域，才能進行叢集管理操作、etcd備份和控制平面更新。在中斷連線狀態下，本機叢集可以繼續操作，但 Amazon EKS 可以採取的叢集管理操作會受到限制。如需詳細資訊，請參閱the section called “準備中斷連線 (EC2 執行個體存放區) ”。

在使用 EC2 執行個體存放區設定的 AWS Outpost 上，選取 Amazon EKS 本機叢集的執行個體類型和配置

本主題提供選擇 Kubernetes 控制平面執行個體類型和設定置放以符合使用 EC2 執行個體存放區設定之 AWS Outpost 上本機 Amazon EKS 叢集的高可用性需求的指引。

> **Note**
>
> 如果您的 Outpost 是使用 Amazon EBS 而非 EC2 執行個體存放區設定，則本主題中所述的架構不適用於您的 Outpost。使用 EBS 設定的 Outpost 將繼續使用現有的本機叢集實作。如需詳細資訊，請參閱the section called “容量考量事項”。
> 如果您有興趣使用更新的本機叢集架構在 EBS 後端 Outpost 上建立本機叢集，請聯絡您的 AWS 客戶團隊。

控制平面架構

本機叢集 Kubernetes 控制平面會在 Outpost 上的 6 個 EC2 執行個體上執行：

- 3 個控制平面執行個體：託管 Kubernetes 控制平面元件，包括 Kubernetes API 伺服器、排程器和控制器管理員。
- 3 個**`etcd`**執行個體 - 將 Kubernetes 叢集狀態存放在外部交換拓撲中。

這些執行個體會在 Outpost AWS上的受管服務帳戶中執行。它們不會在 AWS 您的帳戶或 Amazon EC2 主控台中顯示為 EC2 執行個體。 Amazon EC2 控制平面耗用的容量會顯示在 AWS Outposts 主控台中。

控制平面不會在負載下水平擴展。如果您需要不同的控制平面執行個體大小，則必須建立新的叢集。

執行個體類型選取

透過 AWS Outposts 上 Amazon EKS 本機叢集的更新架構，Kubernetes 控制平面會使用專用`etcd`拓撲：API 伺服器，並在個別執行個體`etcd`上執行。當您建立叢集時，請指定兩種執行個體類型：`controlPlaneInstanceType`適用於 API 伺服器，`etcdInstanceType`適用於 `etcd`。

您選擇的執行個體類型必須可在 Outpost 上使用。可用的執行個體系列取決於您的 Outpost 產生和 SKU，而不同的執行個體系列可以具有相同執行個體大小的不同 vCPU 和記憶體組態。

API 伺服器**`controlPlaneInstanceType`** ()

預期的工作者節點	vCPUs	記憶體
1–20	2	8 GiB
21–100	4	16 GiB
101–250	8	32 GiB
251–500	16	64 GiB

etcd (etcdInstanceType)

預期的工作者節點	vCPUs	記憶體
1–50	2	8 GiB
51–250	4	16 GiB
251–500	8	32 GiB

> **Important**
>
> 請確定您的 Outpost 在本機叢集`etcdInstanceType`的生命週期內具有 3 個 執行個體`controlPlaneInstanceType`和 3 個 執行個體的容量。

使用這些資料表做為指導方針，而非嚴格的要求。我們建議您選取符合您預期成長的執行個體系列和大小，因為 `controlPlaneInstanceType`和 `etcdInstanceType`無法在叢集建立後變更。若要變更兩者之一，您必須建立新的叢集。

控制平面和**etcd**置放

為了獲得高可用性，建議您將控制平面和`etcd`執行個體分散到多個硬體故障網域。您可以使用 `controlPlanePlacement`和 `etcdPlacement`中的 `spreadLevel` 屬性來控制此分散`outpostConfig`。當您設定 時`spreadLevel`，Amazon EKS 會使用 分散策略建立 EC2 置放群組。您不需要預先建立置放群組。

> **Note**
>
> 中的 groupName 參數controlPlanePlacement不會與執行執行個體存放區型 EC2 執行個體的 Outpost 搭配使用。如果您先前將置放群組與本機叢集搭配使用，您現在會spreadLevel改用 。

分散層級

spreadLevel 屬性接受兩個值：

- **host** — 將控制平面執行個體分散到不同的實體主機。至少需要 3 個使用所選執行個體類型設定的主機。
- **rack** — 將控制平面執行個體分散到不同的實體 Outpost 機架。至少需要 3 個主機設定為所選執行個體類型的機架。

容量規劃

規劃本機叢集的 Outpost 容量時，請考量下列事項：

- 控制平面和**etcd**執行個體：每個叢集 6 個執行個體 (3 個 controlPlaneInstanceType和 3 個 etcdInstanceType)。
- 工作者節點：自我管理節點群組的 EC2 執行個體。
- 虛擬化主機：控制平面必須在虛擬化主機上執行。您的 Outpost 必須具有具有足夠容量的虛擬化主機。

對置放問題進行故障診斷

如果您的叢集在指定分散層級後長時間保持在 CREATING或 UPDATING 狀態，請確認您的 Outpost 有足夠的主機或機架搭配所選的執行個體類型，以滿足分散拓撲。

如需本機叢集疑難排解的詳細資訊，請參閱 the section called “叢集疑難排解 (EC2 執行個體存放區) ”。

在使用 EC2 執行個體存放區設定的 AWS Outposts 上準備本機 Amazon EKS 叢集，以進行網路連線中斷

如果將本機網路連線至 AWS 雲端的 AWS Outposts 服務連結失去連線，您可以繼續使用 Outpost 上的本機 Amazon EKS 叢集。本主題涵蓋如何準備本機叢集以進行網路連線中斷和相關考量。

- 本機叢集可在臨時、意外的網路中斷連線期間實現穩定性和持續操作。 AWS Outposts 仍是完全連線的方案，可做為資料中心中 AWS 雲端的延伸。如果您的 Outpost 和 AWS 雲端之間發生網路中斷連線，建議您嘗試還原連線。如需說明，請參閱《AWS Outposts 使用者指南》中的 Outposts 機架網路故障診斷檢查清單。 AWS
- Outpost 會發射 `ConnectedStatus` 指標，您可用此指標來監控 Outpost 的連線狀態。如需詳細資訊，請參閱《 Outposts 使用者指南》中的 Outposts 指標。 AWS

網路連線中斷期間的身分驗證

本機叢集支援多個身分驗證機制。它們在網路連線中斷期間的可用性會有所不同：

身分驗證機制	在中斷連線期間可用？
AWS IAM （存取項目、`aws-auth`ConfigMap)	否。IAM 需要連線至 AWS 區域。
OIDC （客戶提供的供應商）	取決於提供者位置。如果可從 Outpost 的本機網路存取 OIDC 供應商，身分驗證會繼續運作。
x.509 用戶端憑證	是。憑證由 Kubernetes API 伺服器在本機進行驗證。
IRSA （服務帳戶的 IAM 角色）	否。請參閱the section called “中斷連線期間的 IRSA 和 Pod 身分”。
EKS Pod 身分識別	否。請參閱the section called “中斷連線期間的 IRSA 和 Pod 身分”。

x.509 用戶端憑證

若要在網路中斷連線期間維持`kubectl`存取權，請在中斷連線發生之前建立用戶端 x.509 憑證。

若要建立管理員憑證：

1. 產生私有金鑰和憑證簽署請求 (CSR)：

```
openssl req -new -newkey rsa:4096 -nodes \
    -keyout admin.key -out admin.csr -subj "/CN=admin"
```

2. 建立並核准 Kubernetes CertificateSigningRequest 資源：

```
cat admin.csr | base64 | tr -d '\n' > admin.csr.b64
```

```
apiVersion: certificates.k8s.io/v1
kind: CertificateSigningRequest
metadata:
  name: admin-csr
spec:
  request: <base64-encoded-csr>
  signerName: kubernetes.io/kube-apiserver-client
  usages:
    - client auth
```

```
kubectl apply -f admin-csr.yaml
kubectl certificate approve admin-csr
```

3. 擷取已簽章的憑證：

```
kubectl get csr admin-csr -o jsonpath='{.status.certificate}' | base64 --decode >
 admin.crt
```

4. 建立 ClusterRoleBinding以授予管理員存取權：

```
kubectl create clusterrolebinding admin --clusterrole=cluster-admin \
    --user=admin --group=system:masters
```

5. 建置kubeconfig使用憑證的 ：

```
kubectl config --kubeconfig admin.kubeconfig set-cluster my-cluster \
    --certificate-authority=ca.crt --server $APISERVER_ENDPOINT --embed-certs
kubectl config --kubeconfig admin.kubeconfig set-credentials admin \
    --client-certificate=admin.crt --client-key=admin.key --embed-certs
kubectl config --kubeconfig admin.kubeconfig set-context admin@my-cluster \
    --cluster my-cluster --user admin
kubectl config --kubeconfig admin.kubeconfig use-context admin@my-cluster
```

中斷連線期間的叢集端點 DNS 解析

本機叢集的 Kubernetes API 伺服器端點託管在 Amazon Route 53 中，並解析為 Amazon EKS 在您的子網路中建立的跨帳戶彈性網路介面 (ENIs) 的私有 IP 地址。這些 ENIs 具有靜態私有 IP 地址，在正常叢集操作期間不會變更。

在網路中斷連線期間，Outpost 無法到達 Route 53，因此叢集端點主機名稱無法解析，除非您準備好本機解析路徑。三種類型的用戶端需要連線到 API 伺服器：

- 執行 的叢集管理員`kubectl`。
- 工作者節點 (`kubelet`) 傳送節點活動訊號和提取規格。
- **`kube-proxy`** 在每個節點上，設定叢集服務 IPs。

選項 1：本機 DNS 解決方案 （ 建議 ）

AWS 建議部署本機 DNS 解決方案，快取叢集端點記錄，並在 Outpost 中斷連線時為其提供服務。您可以在快取叢集端點記錄的內部部署環境中執行自己的 DNS 伺服器。

如果您使用本機 DNS 解決方案，建議您將 `kubeconfig`和工作者節點 AMIs 指向叢集端點主機名稱（而不是 ENI IP 地址），以便解析與本機 DNS 解決方案一致。

選項 2：靜態 IP 型存取

如果您不想執行本機 DNS 解決方案，您可以使用靜態 IP 型存取。

- 管理員：設定 `kubeconfig`以直接指向跨帳戶 ENI 私有 IP 地址。透過`Amazon EKS` *`cluster-name`* 搜尋 AWS 帳戶中描述的網路介面來尋找 ENIs。在正常操作下，每個 ENI 的 IP 地址在叢集的生命週期內都是穩定的。
- 工作者節點 (Amazon EKS 最佳化 AMIs)：當您從 Amazon EKS 最佳化 AMI 啟動工作者節點時，引導指令碼會使用 ENI IP 地址`/etc/hosts`將叢集端點新增至 。不需要額外的組態。
- 工作者節點 （ 自訂 AMIs)：將叢集端點主機名稱和 ENI IP 地址新增至自訂引導`/etc/hosts`中的 。否則， `kubelet`和 `kube-proxy`無法在中斷連線期間連線到 API 伺服器。

> **Important**
>
> 如果刪除跨帳戶 ENI 或其 IP 地址變更，例如，如果您刪除或修改它，以防止 Amazon EKS 重新連接它，則每個節點和每個使用靜態 IP 型存取的管理員都必須手動更新。使用本機 DNS 解決方案時，不需要手動介入。

中斷連線期間的 Pod DNS 解析

若要防止中斷連線操作期間的 DNS 失敗，請將工作者節點啟動範本設定為覆寫kubelet's `resolvConf設定。在您的使用者資料中，建立只包含（沒有 VPC 搜尋網域/etc/kubernetes/resolv.conf) 的自訂resolv.conf檔案 nameserver 10.0.0.2（例如），然後在您的 spec.kubelet.config.resolvConf: /etc/kubernetes/resolv.conf中設定 NodeConfig。這會從 Pod DNS 組態中移除 *region-code*.compute.internal搜尋網域，防止查詢在中斷連線時轉送到無法連線的 VPC DNS 解析程式。

下列範例顯示工作者節點 userdata：

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="BOUNDARY"

--BOUNDARY
Content-Type: text/x-shellscript; charset="us-ascii"

#!/bin/bash
mkdir -p /etc/kubernetes
echo "nameserver 10.0.0.2" > /etc/kubernetes/resolv.conf

--BOUNDARY
Content-Type: application/node.eks.aws

---
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name: my-cluster
    ...
  kubelet:
    config:
      resolvConf: /etc/kubernetes/resolv.conf

--BOUNDARY--
```

中斷連線期間的 IRSA 和 Pod 身分

> **Important**
>
> IRSA 和 EKS Pod Identity 取決於在 AWS 區域中執行的 AWS STS。在網路連線中斷期間，使用 IRSA 或 Pod Identity 的工作負載無法取得新的登入資料。現有的登入資料會在一段時間後過期。

對於必須在網路中斷期間保持可用的工作負載，我們不建議對區域型 AWS 服務採用功能或操作相依性。

`etcd` 中斷連線期間的行為

在網路連線中斷期間，無法備份`etcd`快照。如果在中斷連線期間多個`etcd`執行個體無法使用，會`etcd`遺失規定人數，而且在 Outpost 重新連線並還原規定人數之前，無法使用 Kubernetes API `etcd` 操作。已經在執行的工作負載會繼續運作。

在中斷連線期間控制平面記錄

在網路中斷連線期間，控制平面日誌會在控制平面執行個體上本機快取。恢復連線時，日誌會傳送至父 AWS 區域中的 Amazon CloudWatch Logs。您不需要在控制平面上安裝或維護任何記錄代理程式。

本機可觀測性

您可以使用 Prometheus、Grafana 或其他第三方解決方案來抓取 Kubernetes API 伺服器指標端點，在中斷連線期間於本機監控叢集。

本機映像儲存庫

若要使用其他複本擴展部署，或在中斷連線期間從 Pod 失敗中復原，您必須有本機容器映像儲存庫（例如 Docker 登錄檔），或在中斷連線之前必須在節點上快取映像。Amazon ECR 在網路連線中斷期間無法使用。

調校 Kubernetes Pod 容錯移轉行為

在網路中斷連線期間，Kubernetes 控制平面無法與 AWS 區域通訊。如果節點變得無法連線，預設的 Kubernetes 行為是在逾時期間之後移出 Pod。您可以使用公差和 Pod 規格`tolerationSeconds`來調整此行為，以控制在分割區期間重新排程 Pod 的速度。如需詳細指引和範例，請參閱《Amazon EKS

最佳實務指南》中的 https://docs.aws.amazon.com/eks/latest/best-practices/hybrid-nodes-network-disconnection-best-practices.html#tune_kubernetes_pod_failover_behavior【Tune Kubernetes Pod 容錯移轉行為】。

模擬網路中斷連線

使用本機叢集進入生產環境之前，請模擬中斷連線，以確認您可以在叢集處於中斷連線狀態時存取叢集。

1. 在將 Outpost 連接到 AWS 區域的聯網裝置上套用防火牆規則。這會中斷 Outpost 連結的服務。
2. 使用您建立的 x.509 憑證測試與本機叢集的連線：

```
kubectl --kubeconfig admin.kubeconfig get nodes
```

> **Note**
>
> 如果您的 Outpost 已有生產中的服務，請勿模擬中斷連線。中斷連線服務連結會影響 Outpost 上執行的所有服務。

使用 EC2 執行個體存放區設定的 AWS Outposts 上本機叢集的 Amazon EKS 附加元件

使用 EC2 執行個體存放區設定的 AWS Outpost 上的本機叢集支援 Amazon EKS 附加元件。下列附加元件已經過驗證，可與本機叢集搭配使用。其他附加元件將隨著時間經過驗證。

> **Note**
>
> 如果您的 Outpost 是使用 Amazon EBS 而非 EC2 執行個體存放區設定，則本主題中所述的架構不適用於您的 Outpost。使用 EBS 設定的 Outpost 將繼續使用現有的本機叢集實作，這不支援 Amazon EKS 附加元件。
> 如果您有興趣使用更新的本機叢集架構在 EBS 後端 Outpost 上建立本機叢集，請聯絡您的 AWS 客戶團隊。

已驗證的附加元件

附加元件	說明
適用於 Kubernetes 的 Amazon VPC CNI 外掛程式	為 Pod 提供原生 VPC 聯網。
kube-proxy	維護節點的網路規則以進行 Kubernetes Service 通訊。
CoreDNS	提供 Kubernetes 服務的 DNS 解析。
EKS Pod 身分代理程式	讓 Pod 使用 EKS Pod Identity 來驗證 AWS 服務。

當您建立本機叢集時，會自動安裝 Amazon VPC CNI 外掛程式`kube-proxy`、 和 CoreDNS 的客戶受管版本。您可以選擇性地安裝這些附加元件的受管版本。

> **Important**
>
> EKS Pod Identity 和 IRSA 取決於區域中的 AWS STS AWS 。在網路連線中斷期間，使用這些機制的工作負載無法取得新的登入資料。如需詳細資訊，請參閱the section called “準備中斷連線 (EC2 執行個體存放區) ”。

附加元件尚未驗證

上面未列出的附加元件尚未經過驗證，可與本機叢集搭配使用。不建議使用未經驗證的附加元件。

當驗證其他附加元件時，此頁面將會更新。

替代 CNI 外掛程式

Amazon VPC CNI 是 Outposts 上 Amazon EKS 部署的官方支援 CNI。如果您想要使用替代的 CNI 外掛程式，請參閱 the section called “替代 CNI 外掛程式”。

對使用 EC2 執行個體存放區設定的 AWS Outpost 上的本機 Amazon EKS 叢集進行故障診斷

本主題涵蓋在以 EC2 執行個體存放區設定的 AWS Outposts 上使用本機 Amazon EKS 叢集時可能看到的常見錯誤，以及如何進行故障診斷。

kubectl 設定 後無法運作 **kubeconfig**

如果您在執行 後看到下列錯誤`aws eks update-kubeconfig`：

```
Either parameter --cluster-name or --cluster-id must be specified.
```

或者：

```
couldn't get current server API group list: getting credentials: exec: executable aws
 failed with exit code 1
```

確認您的 `exec` kubeconfig(`~/.kube/config`) 在 args `--cluster-name`中指定 ，而不是 `--cluster-id`：

```
      args:
        - --region
        - region-code
        - eks
        - get-token
        - --cluster-name
        - my-cluster
        - --output
        - json
      command: aws
```

未加入叢集的節點

如果您的節點在您執行 時未顯示`kubectl get nodes`：

1. 檢查節點上的**kubelet**日誌。使用 SSM 或 SSH 存取節點並執行：

   ```
   systemctl status kubelet -l
   ```

2. 檢查是否有未經授權的錯誤。未經授權的錯誤可能表示缺少節點角色許可或 `aws-auth` ConfigMap 問題。確認節點的執行個體角色已在 ConfigMap `aws-auth` 中正確映射。

3. 檢查 CSR 狀態。驗證節點的憑證簽署請求是否已核准：

```
kubectl get csr
```

如果 CSR 處於 Pending 狀態，請核准：

```
kubectl certificate approve node-csr-EXAMPLE
```

叢集卡在 **CREATING** 狀態

如果您的叢集進入 FAILED 狀態：

- 容量不足：確認您的 Outpost 具有足夠的虛擬化容量，可供指定執行個體類型的 6 個控制平面執行個體 (3 `etcd` + 3 個 API 伺服器） 使用。執行個體類型必須在 Outpost 上槽化。
- 無法滿足分散拓撲：如果您指定 `spreadLevel`的 `host`，請確認至少有 3 個主機已設定所選的執行個體類型。如果您指定 `rack`，請確認至少 3 個機架具有具有所選執行個體類型的主機。
- 子網路問題：確認您指定的子網路位於 Outpost 所在的可用區域中，且每個子網路至少有 3 個可用的 IP 地址。請參閱 the section called “建立 VPC 和子網路 (EC2 執行個體存放區 ）”。

叢集卡在 **UPDATING** 狀態

如果您的叢集在 Kubernetes 版本或平台版本更新期間保持 UPDATING 狀態：

- 容量不足：在更新期間，控制平面執行個體會在重新驗證之前刪除。如果在佈建替代執行個體之前，其他工作負載耗用釋放的容量，則更新會停止。確認您的 Outpost 具有控制平面執行個體類型的可用容量。

etcd 規定人數損失

如果您失去執行 Kubernetes API 操作的能力，且您的 Outpost 已中斷連線：

- `etcd` 可能已失去規定人數。如果在中斷連線期間多個`etcd`執行個體無法使用，就會發生這種情況。
- 已在執行中的工作負載會繼續運作，但無法使用 Kubernetes API 操作。
- 恢復連線時，Amazon EKS 會從最新的`etcd`快照復原叢集狀態。

請聯絡AWS 支援中心。

在 AWS Outposts 上建立 Amazon Linux 節點

Important

Outposts 上的 Amazon EKS 本機叢集僅支援從下列 Amazon EKS 最佳化 Amazon Linux 2023 AMIs 建立的節點：

- 標準 Amazon Linux 2023 (`amazon-linux-2023/x86_64/standard`)
- 加速 Nvidia Amazon Linux 2023 (`amazon-linux-2023/x86_64/nvidia`)
- 加速 Neuron Amazon Linux 2023 (`amazon-linux-2023/x86_64/neuron`)
- Bottlerocket （僅適用於 EC2 執行個體存放區支援的 Outpost)

AWS 已結束對 EKS AL2-optimized和 AL2-accelerated AMIs支援，自 2025 年 11 月 26 日開始生效。雖然您可以在終止支援 (EOS) 日期 (2025 年 11 月 26 日) 之後繼續使用 EKS AL2 AMI，但 EKS 將不再發布任何新的 Kubernetes 版本或對 AL2 AMI 的更新，包括此日期之後的次要版本、修補程式和錯誤修正。如需 AL2 棄用的詳細資訊，請參閱此處。

本主題會說明如何啟動 Outpost 上已向 Amazon EKS 叢集註冊的 Amazon Linux 節點的 Auto Scaling 群組。叢集可以在 AWS 雲端或 Outpost 上。

- 現有的 Outpost。如需詳細資訊，請參閱什麼是 AWS Outpost。
- 現有 Amazon EKS 叢集。若要在 AWS 雲端部署叢集，請參閱 the section called “建立 叢集”。若要在 Outpost 上部署叢集，請參閱the section called “執行本機叢集”。
- 假設您正在 AWS 雲端的叢集中建立節點，而且您在已啟用 AWS Outposts、 AWS Wavelength 或 AWS Local Zones 的 AWS 區域中有子網路。然後，在您建立叢集時，不得傳入這些子網路。如果您要在 Outpost 上的叢集中建立節點，則必須在建立叢集時便已傳入 Outpost 子網路。
- （建議用於 AWS 雲端上的叢集） 適用於 Kubernetes 附加元件的 Amazon VPC CNI 外掛程式，其設定為具有連接必要 IAM 政策的自有 IAM 角色。如需詳細資訊，請參閱the section called “為 IRSA 設定”。本機叢集不支援服務帳戶的 IAM 角色。

您可以使用 `eksctl`或 AWS 管理主控台 （使用 AWS CloudFormation 範本） 建立自我管理的 Amazon Linux 節點群組。您也可使用 Terraform。

您可以使用此頁面所述的下列工具，為本機叢集建立自我管理節點群組：

- the section called “eksctl”
- the section called “AWS 管理主控台”

> **Important**
>
> - 自我管理節點群組包含您帳戶的 Amazon EC2 執行個體。當您或 Amazon EKS 代您更新控制平面版本時，這些執行個體不會自動升級。自我管理節點群組在主控台中沒有任何需要更新的指示。您可以檢視安裝在節點上的 kubelet 版本，方法是選取您叢集的 Overview (概觀) 標籤上的 Nodes (節點) 清單上的節點，以判斷哪些節點需要更新。您必須手動更新節點。如需詳細資訊，請參閱the section called “更新方法”。
> - 您自我管理節點上的 kubelet 使用的憑證已核發，一年過期。預設不會啟用憑證輪換 （請參閱：https：//https://kubernetes.io/docs/reference/config-api/kubelet-config.v1beta1/#kubelet-config-k8s-io-v1beta1-KubeletConfiguration)，這表示如果您的自我管理節點執行超過一年，就無法再向 Kubernetes API 進行身分驗證。
> - 基於最佳實務，建議客戶定期更新其自我管理節點群組，以從最新的 Amazon EKS 最佳化 AMI 接收 CVE 和安全性修補程式。更新自我管理節點群組中使用的 AMI 也會觸發節點的重新建立，並確保它們不會因為過期的 kubelet 憑證而發生問題。
> - 或者，您也可以在建立自我管理節點群組時啟用用戶端憑證輪換 (請參閱：https://kubernetes.io/docs/tasks/tls/certificate-rotation/)，以確保在 kubelet 憑證在目前的憑證即將過期時及時更新。

eksctl

使用 **eksctl** 啟動自我管理的 Linux 節點

1. 在您的裝置或 AWS CloudShell 上安裝版本 0.215.0或更新版本的eksctl命令列工具。如需有關安裝或更新 eksctl 的指示，請參閱 eksctl 文件中的安裝一節。
2. 如果您的叢集位於 AWS 雲端，且 AmazonEKS_CNI_Policy 受管 IAM 政策已連接至您的 Amazon EKS 節點 IAM 角色，建議您改為將其指派給與 Kubernetes aws-node服務帳戶相關聯的 IAM 角色。如需詳細資訊，請參閱the section called “為 IRSA 設定”。如果您的叢集位於 Outpost 上，則政策必須連接到您的節點角色。

3. 以下命令會在現有的叢集建立節點群組。叢集必須使用 eksctl 來建立。將 *al-nodes* 取代為您的節點群組名稱。節點群組名稱不可超過 63 個字元。它必須以字母或數字開頭，但也可以包含其餘字元的連字號和底線。使用您叢集的名稱取代 *my-cluster*。此名稱僅能使用英數字元 (區分大小寫) 和連字號。必須以英數字元開頭，且長度不可超過 100 個字元。名稱在您要建立叢集 AWS 的區域和 AWS 帳戶中必須是唯一的。若您的叢集存在於 Outpost 上，請使用 Outpost 子網路的 ID 來取代 *id*。如果您的叢集存在於 AWS 雲端上，請將 *ID* 取代為您建立叢集時未指定的子網路 ID。使用您自己的值取代其餘範例值。依預設，系統會使用與控制平面相同的 Kubernetes 版本來建立節點。

   使用您 Outpost 上可用的執行個體類型來取代 *instance-type* 。

   使用 Amazon EC2 金鑰對或公有金鑰的名稱取代 *my-key*。此金鑰會在節點啟動後用於將 SSH 套用至節點。如果您還沒有 Amazon EC2 金鑰對，可以在 AWS 管理主控台中建立一個。如需詳細資訊，請參閱《Amazon EC2 使用者指南》中的 Amazon EC2 金鑰對。

   使用下列命令來建立您的節點群組。

   ```
   eksctl create nodegroup --cluster my-cluster --name al-nodes --node-type instance-
   type \
       --nodes 3 --nodes-min 1 --nodes-max 4 --managed=false \
       --node-volume-type gp2 --subnet-ids subnet-id \
       --node-ami-family AmazonLinux2023
   ```

   如果您的叢集部署在 AWS 雲端：

   - 您部署的節點群組可以從與執行個體不同的 CIDR 區塊將 IPv4 地址指派給 Pod。如需詳細資訊，請參閱the section called “自訂聯網”。
   - 您部署的節點群組不需要傳出網際網路存取。如需詳細資訊，請參閱the section called “私有叢集”。

   如需所有可用選項和預設值的完整清單，請參閱 eksctl 文件中的 AWS Outposts 支援。

   - 如果節點無法加入叢集，請參閱the section called “節點無法加入叢集”針對 Amazon EKS 叢集和節點問題進行故障診斷the section called “無法將節點加入叢集”，以及針對 AWS Outposts 上的本機 Amazon EKS 叢集進行故障診斷。
   - 範例輸出如下。建立節點時，會有數行輸出。輸出的最後幾行之一類似於以下的範例行。

     ```
     [#]  created 1 nodegroup(s) in cluster "my-cluster"
     ```

4. (選用) 部署範例應用程式以測試您的叢集和 Linux 節點。

AWS 管理主控台

步驟 1：使用 AWS 管理主控台 啟動自我管理的 Linux 節點

1. 下載最新版本的 AWS CloudFormation 範本。

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2025-11-24/
amazon-eks-outpost-nodegroup.yaml
```

2. 開啟 AWS CloudFormation 主控台。
3. 選擇 Create stack (建立堆疊)，然後選取 With new resources (standard) (使用新資源 (標準))。
4. 針對 Specify template (指定範本)，選取 Upload a template file (上傳範本檔案)，然後選取 Choose file (選擇檔案)。選取在上一步驟中下載的 `amazon-eks-outpost-nodegroup.yaml` 檔案，然後選取 Next (下一步)。
5. 在 Specify stack details (指定堆疊詳細資訊) 頁面上，據此填寫下列參數，然後選擇 Next (下一步)：
   - 堆疊名稱：為您的 AWS CloudFormation 堆疊選擇堆疊名稱。例如，您可以稱它為 *`al-nodes`*。此名稱僅能使用英數字元 (區分大小寫) 和連字號。必須以英數字元開頭，且長度不可超過 100 個字元。名稱在您要建立叢集 AWS 的區域和 AWS 帳戶中必須是唯一的。
   - ApiServerEndpoint：輸入 Kubernetes API 伺服器端點，在 EKS 主控台中或透過 DescribeCluster API 顯示。
   - ClusterName：請輸入您叢集的名稱。若此名稱與您叢集的名稱不相符，則您的節點便無法加入叢集。
   - ClusterId：輸入由 EKS 服務指派給叢集的 ID。透過 DescribeCluster API 可見。如果此 ID 與您的叢集 ID 不符，您的節點就無法加入叢集。
   - CertificateAuthority：輸入 Kubernetes Certificate Authority 的 base64 編碼字串。在 EKS 主控台中或透過 DescribeCluster API 可見。
   - ServiceCidr：輸入 Kubernetes Services CIDR。在 EKS 主控台中或透過 DescribeCluster API 可見。
   - ClusterControlPlaneSecurityGroup：從您在建立 VPC 時產生的 AWS CloudFormation 輸出中選擇 SecurityGroups 值。

     下列步驟顯示擷取適用群組的一種操作。

     i. 開啟 Amazon EKS 主控台。

     ii. 選擇叢集的名稱。

iii. 選擇 Networking (網路) 索引標籤。

iv. 從 ClusterControlPlaneSecurityGroup 下拉式清單中選取時，請使用 Additional Security Group (其他安全群組) 值作為參考。

- NodeGroupName：輸入節點群組的名稱。此名稱稍後可用於識別為您節點建立的 Auto Scaling 節點群組。
- NodeAutoScalingGroupMinSize：輸入節點 Auto Scaling 群組可以縮減的最低節點數。
- NodeAutoScalingGroupDesiredCapacity：堆疊建立時，輸入欲擴展的所需節點數量。
- NodeAutoScalingGroupMaxSize：輸入節點 Auto Scaling 群組可以擴增的最大節點數。
- NodeInstanceType：選擇節點的執行個體類型。如果您的叢集在 AWS 雲端上執行，如需詳細資訊，請參閱 the section called “Amazon EC2 執行個體類型”。若您的叢集正於 Outpost 上執行，則您僅可選取 Outpost 上可用的執行個體類型。
- NodeImageIdSSMParam：為變數 Kubernetes 版本預先填入了最近 Amazon EKS 最佳化 AMI 的 Amazon EC2 Systems Manager 參數。若要使用 Amazon EKS 支援不同的 Kubernetes 次要版本，請將 *1.XX* 取代為不同的支援的版本。建議指定與叢集相同的 Kubernetes 版本。

  若要使用 Amazon EKS 最佳化加速 AMI，請將 *NodeImageIdSSMParam* 值更新為所需的 SSM 參數。在此處了解如何從 SSM 擷取 EKS AMI IDshttps://docs.aws.amazon.com/eks/latest/userguide/retrieve-ami-id.html。

  > **Note**
  >
  > Amazon EKS 節點 AMI 以 Amazon Linux 為基礎。您可以透過選擇所需版本的標籤，在 Amazon Linux 安全中心追蹤 Amazon Linux 的安全或隱私權事件。您也可以訂閱適用的 RSS 摘要。安全與隱私權事件包含問題的概觀、哪些套件受到影響，以及如何更新您的執行個體以修正問題。

- NodeImageId：（選用）如果您使用自己的自訂 AMI（而非 Amazon EKS 最佳化 AMI)，請輸入您 AWS 區域的節點 AMI ID。如果您在此指定值，則會覆寫 NodeImageIdSSMParam 欄位中的任何值。
- NodeVolumeSize：為您的節點指定根磁碟區大小 (以 GiB 為單位)。
- NodeVolumeType：為您的節點指定根磁碟區類型。
- KeyName：輸入 Amazon EC2 SSH 金鑰對的名稱，您可以在節點啟動後使用該金鑰對來透過 SSH 連接至節點。如果您還沒有 Amazon EC2 金鑰對，可以在 AWS 管理主控台中建立一個。如需詳細資訊，請參閱《Amazon EC2 使用者指南》中的 Amazon EC2 金鑰對。

> **Note**
>
> 如果您未在此處提供金鑰對，AWS CloudFormation 堆疊建立會失敗。

- DisableIMDSv1：預設情況下，每個節點都支援執行個體中繼資料服務版本 1 (IMDSv1) 和 IMDSv2。您可以停用 IMDSv1。若要防止節點群組中的未來節點和 Pod 使用 IMDSv1，請將 DisableIMDSv1 設定為 true。如需 IMDS 的詳細資訊，請參閱設定執行個體中繼資料服務。如需在節點上限制存取的詳細資訊，請參閱限制存取指派給工作節點的執行個體設定檔。
- VpcId：輸入您建立的 VPC ID。在選擇子網路之前，請先檢閱 VPC 要求和考量事項。
- 子網路：如果叢集位於 Outpost 上，則請至少在 VPC 中選擇一個私有子網路。在選擇子網路之前，請先檢閱子網路需求和注意事項。您可以看到哪些子網是私有子網，方法是從叢集的 Networking (聯網) 標籤打開每一個子網連結。

6. 請在 Configure stack options (設定堆疊選項) 頁面上選取所需的選項，然後選擇 Next (下一頁)。
7. 選取我確認 AWS CloudFormation 可能會建立 IAM 資源的左側核取方塊，然後選擇建立堆疊。
8. 當當堆疊已完成建立時，從主控台將其選取，然後選擇 Outputs (輸出)。
9. 為已建立的節點群組記錄 NodeInstanceRole。當您設定 Amazon EKS 節點時會需要此值。

步驟 2：讓節點加入叢集

1. 檢查以瞭解是否有 `aws-auth ConfigMap`。

```
kubectl describe configmap -n kube-system aws-auth
```

2. 如果您看到 `aws-auth ConfigMap`，請視需要更新之。
   a. 開啟 `ConfigMap` 進行編輯。

```
kubectl edit -n kube-system configmap/aws-auth
```

   b. 視需要新增 `mapRoles` 個項目。將 `rolearn` 值設定為您在先前程序中記錄的 NodeInstanceRole 值。

```
[...]
data:
  mapRoles: |
    - rolearn: <ARN of instance role (not instance profile)>
      username: system:node:{{EC2PrivateDNSName}}
```

```
      groups:
        - system:bootstrappers
        - system:nodes
[...]
```

c. 儲存檔案並結束您的文字編輯器。

3. 如果您收到一個錯誤，說明 "Error from server (NotFound): configmaps "aws-auth" not found，那麼請套用股票 ConfigMap。

a. 下載組態對應。

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/aws-auth-cm.yaml
```

b. 在 aws-auth-cm.yaml 檔案中，將 rolearn 設定為您在先前程序中紀錄的 NodeInstanceRole 值。您可以使用文字編輯器來完成此操作，或者透過取代 *my-node-instance-role* 並執行下列命令：

```
sed -i.bak -e 's|<ARN of instance role (not instance profile)>|my-node-instance-role|' aws-auth-cm.yaml
```

c. 套用組態。此命令可能需要幾分鐘的時間來完成。

```
kubectl apply -f aws-auth-cm.yaml
```

4. 查看節點的狀態，並等待他們到達 Ready 狀態。

```
kubectl get nodes --watch
```

輸入 Ctrl+C 傳回 Shell 提示。

> **Note**
>
> 如果您收到任何授權或資源類型錯誤，請參閱故障診斷主題中的the section called “未經授權或存取遭拒 (kubectl)”。

如果節點無法加入叢集，請參閱the section called “節點無法加入叢集”故障診斷 Amazon EKS 叢集和節點的問題the section called “無法將節點加入叢集”，以及故障診斷 AWS Outposts 上的本機 Amazon EKS 叢集。

5. 安裝 Amazon EBS CSI 驅動程式。如需詳細資訊，請參閱 GitHub 上的 Installation (安裝)。在 Set up driver permission (設定驅動程式權限) 區段中，請務必依照 Using IAM instance profile (使用 IAM 執行個體設定檔) 選項指示操作。您必須使用 gp2 儲存類別。不支援 gp3 儲存類別。

   若要在叢集上建立 gp2 儲存類別，請完成以下步驟。

   a. 執行下列命令以建立 gp2-storage-class.yaml 檔案。

```
cat >gp2-storage-class.yaml <<EOF
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
  name: ebs-sc
provisioner: ebs.csi.aws.com
volumeBindingMode: WaitForFirstConsumer
parameters:
  type: gp2
  encrypted: "true"
allowVolumeExpansion: true
EOF
```

   b. 將清單檔案套用至叢集。

```
kubectl apply -f gp2-storage-class.yaml
```

6. (僅限 GPU 節點) 若您選擇了 GPU 執行個體類型和 Amazon EKS 最佳化加速 AMI，則必須套用 Kubernetes 專用 NVIDIA 裝置外掛程式作為叢集上的 DaemonSet。請先以您想要的 NVIDIA/k8s-device-plugin 版本來取代 *vX.X.X*，然後再執行下列命令。

```
kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/vX.X.X/
deployments/static/nvidia-device-plugin.yml
```

步驟 3：其他動作

1. (選用) 部署範例應用程式以測試您的叢集和 Linux 節點。
2. 若您的叢集部署在 Outpost 上，請跳過此步驟。如果您的叢集部署在 AWS 雲端上，則下列資訊為選用。如果 AmazonEKS_CNI_Policy 受管 IAM 政策已連接至您的 Amazon EKS 節點 IAM 角色，建

議您改為將其指派給您與 Kubernetes aws-node 服務帳戶相關聯的 IAM 角色。如需詳細資訊，請參閱the section called “為 IRSA 設定”。

Amazon EKS 上的人工智慧和Machine Learning概觀

> **Tip**
>
> 註冊即將舉行的 Amazon EKS AI/ML 研討會。

Amazon Elastic Kubernetes Service (Amazon EKS) 是一項受管 Kubernetes 服務，可讓組織部署、管理和擴展人工智慧 (AI) 和機器學習 (ML) 工作負載，並擁有前所未有的彈性和控制能力。由於 Amazon EKS 是以上游 Kubernetes 為基礎，因此您可以套用現有的 Kubernetes 專業知識，同時無縫整合開放原始碼工具和 AWS 服務。

無論您是訓練大規模模型、執行即時線上推論，還是部署生成式 AI 應用程式，Amazon EKS 都能提供 AI/ML 專案所需的效能、可擴展性和成本效益。

為什麼要將 Amazon EKS 用於 AI/ML

Amazon EKS 提供 AI/ML 專案所需的控制、整合、效能和可擴展性。Amazon EKS 建置在上游 Kubernetes 上並與 AWS 服務整合，可協助您使用現有的 Kubernetes 專業知識，同時協調複雜的工作負載。對於初次使用 AI/ML 部署的團隊，現有的 Kubernetes 技能會在沒有陡峭學習曲線的情況下轉移。

Amazon EKS 支援從作業系統自訂到運算擴展的所有功能，並提升技術彈性，以保留未來基礎設施決策的選擇。平台提供 AI/ML 工作負載所需的效能和調校選項，包括下列功能：

- 完整叢集控制：微調成本和組態，而不會隱藏抽象。
- 低於秒的延遲：在生產環境中執行即時推論工作負載。
- 進階自訂：設定多執行個體 GPUs、網路調校和作業系統層級調校。
- 統一協同運作：跨 AI/ML 管道和內部部署、邊緣和雲端環境進行協同運作。
- 成本最佳化：使用自動擴展、原生 GPU 排程，以及多樣化的 GPU 和加速器執行個體類型。

金鑰使用案例

Amazon EKS 支援廣泛的 AI/ML 工作負載，包括下列常見使用案例：

- 推論：Amazon EKS 上的自我託管模型適用於需要低延遲回應時間的使用案例。
- 批次推論：透過排程任務有效率地處理大型資料集。
- 模型訓練：長時間訓練大型資料集上的複雜模型。
- 模型微調：使用專有網域知識增強開放原始碼模型。
- 擷取擴增產生 (RAG) 管道：整合擷取和產生程序。
- 代理程式 AI：使用託管在 Amazon Bedrock、第三方或 Amazon EKS 上的模型部署代理程式。

案例研究

客戶因為各種原因選取 Amazon EKS，例如最佳化 GPU 用量，或以低於秒的延遲執行推論工作負載，如下列案例研究所示。如需 Amazon EKS 的所有案例研究清單，請參閱 AWS 客戶成功案例。

- BMW Group 營運全球最大的連線機群之一，擁有超過 25M00 萬輛連線車輛，使用 Ray 在 Amazon EKS 上建置其連線 AI 平台以進行分散式訓練，並使用 Karpenter 進行 GPU 自動擴展，將模型訓練時間從數小時縮短為 30 分鐘，每次訓練執行 5 歐元，同時支援超過 60 個 AI 使用案例的 550 位開發人員。
- 全球領先的旅遊平台之一 Booking.com 將其搜尋排名 ML 推論系統遷移至 Amazon EKS，以釋放可擴展性進行實驗，每秒處理高達 250K個請求，並延遲 40 毫秒 p99.9。
- 統一每天處理 2，600 萬部影片，使用 AI 進行內容管制。公司需要高輸送量、低延遲推論，並縮短容器開機時間 80%，這可確保在流量波動時快速回應擴展事件。
- Synthesia 提供生成式 AI 影片建立服務，讓客戶從文字提示建立逼真的影片。該公司在 ML 模型訓練輸送量方面達到了 30 倍的改善。
- Ada Support 是採用 AI 技術的客戶服務自動化公司，實現了 15% 的運算成本降低以及 30% 的運算效率提升。
- Snorkel AI 可讓企業建置和調整基礎模型和大型語言模型。該公司實作 GPU 資源的智慧型擴展機制，節省了 40% 以上的成本。
- Artera 使用 Amazon Elastic File System (Amazon EFS) 和 Amazon EKS 來訓練 ML 模型，這些模型使用高解析度切片影像來個人化癌症處理。
- Anthropic 在 Amazon EKS 上執行其旗艦 Claude 系列基礎模型，並在生產環境中操作一些最大的 EKS 叢集，包括 AWS Trainium (trn2) 執行個體和適用於 AI 工作負載的 NVIDIA GPUs，以及適用於 CPU 密集資料處理的 AWS Graviton 處理器。

指南結構

本指南包含一系列實作指南，您可以step-by-step在 Amazon EKS 上部署和管理 AI/ML 工作負載。每個指南都提供您可以直接在環境中實作的指示和組態。

除了說明之外，本指南還提供每個主題所需的背景和基礎概念。它還包含 AWS 相關文件和資源的連結，以取得所需的更深入技術詳細資訊。

在 Amazon EKS 上開始使用 AI/ML

若要開始規劃和使用 Amazon EKS 上的 AI/ML 平台和工作負載，請遵循 the section called “設定叢集”一節在您的帳戶中建立 Amazon EKS 叢集，包括必要的 Kubernetes 元件 AWS 。環境啟動並執行後，您可以繼續執行後續步驟：

- the section called “設定叢集”：建立要在整個區段中使用的 Amazon EKS 叢集、監控和 Amazon S3 儲存貯體基礎設施。
- the section called “Inference”：使用 Amazon EKS 部署、設定和開始使用具有大型語言模型 (LLM) 的推論應用程式。
- the section called “叢集組態”：設定針對 AI/ML 工作負載最佳化的 Amazon EKS 叢集。
- the section called “容量管理”：管理和最佳化 Amazon EKS 上機器學習工作負載的運算資源。
- the section called “裝置管理”：使用動態資源分配 (DRA) 和裝置外掛程式管理專用硬體裝置。

為 AI/ML 工作負載設定 Amazon EKS 叢集

> **Tip**
>
> 註冊即將舉行的 Amazon EKS AI/ML 研討會。

本節會引導您建立準備好執行推論工作負載的 Amazon EKS 叢集，包括使用 GPUs 的運算、監控堆疊和模型權重的 Amazon S3 儲存，以及必要的 AWS IAM 許可。

架構概觀

設定會建立下列基礎設施：

- 具有啟用 GPU 節點的 EKS 叢集 — 由 Karpenter 管理的 NodePool，使用 Spot 容量搭配隨需備用，動態佈建 G 系列的 GPU 執行個體。
- 監控堆疊 — Prometheus 會抓取叢集、節點和 GPU 指標，並將其遠端寫入 Amazon Managed Service for Prometheus (AMP)。Grafana 提供用於視覺化的儀表板。NVIDIA DCGM Exporter 新增 GPU 特定的指標，包括使用率、記憶體、溫度、耗電量、NVLink 頻寬和張量活動。
- 模型權重 S3 儲存貯體 — 用於存放模型權重的 Amazon S3 儲存貯體，具有授予工作負載 Pod 讀取/寫入存取權的 EKS Pod Identity 關聯。

叢集運算選項

本指南提供兩種設定叢集的路徑。選擇其中一個，並一致地執行所有步驟。

- EKS Auto Mode — 單一命令會佈建已啟用 EKS Auto Mode 的 EKS 叢集。所有必要的元件都是out-of-the-box提供，包括 Karpenter 型自動擴展、EKS 節點監控代理程式、使用 SOCI 的快速容器提取，以及 NVIDIA 裝置外掛程式。
- 自我管理 Karpenter：您可以明確地安裝和設定每個元件：透過 進行 Karpenter`eksctl`、透過其功能閘道進行自動節點修復、將 EKS 節點監控代理程式作為 EKS 附加元件，以及透過 Helm 進行 NVIDIA 裝置外掛程式。您也可以建立使用 EKS 最佳化 NVIDIA AL2023 AMIs 的自訂`EC2NodeClass`，並設定 SOCI。

您將設定的內容

步驟	說明
建立叢集	佈建 GPU 工作負載所需的 EKS 控制平面和叢集層級元件。
建立動態佈建的 GPU 節點	定義動態 GPU NodePool，在排程工作負載時佈建 G 系列 GPU 執行個體。
使用範例 Pod 進行測試	執行 Pod 來觸發 Karpenter `nvidia-smi` 佈建已啟用 GPU 的節點，以end-to-end驗證設定。
新增預留容量（選用）	將隨需容量保留 (ODCR) 連接至 NodeClass，以使用 Spot/隨需備用進行預留優先。

步驟	說明
安裝監控	使用遠端寫入 AMP 部署 kube-prometheus-stack (Prometheus + Grafana)，以及適用於 GPU 指標的 NVIDIA DCGM Exporter。
建立模型權重儲存貯體	建立 S3 儲存貯體並設定 EKS Pod Identity，讓工作負載 Pod 可以讀取和寫入模型權重。

開始使用

如需使用 CLI AWS 的step-by-step說明，請參閱 the section called “命令列界面 (CLI)”。如需使用 Terraform step-by-step說明，請參閱 the section called “Terraform”。

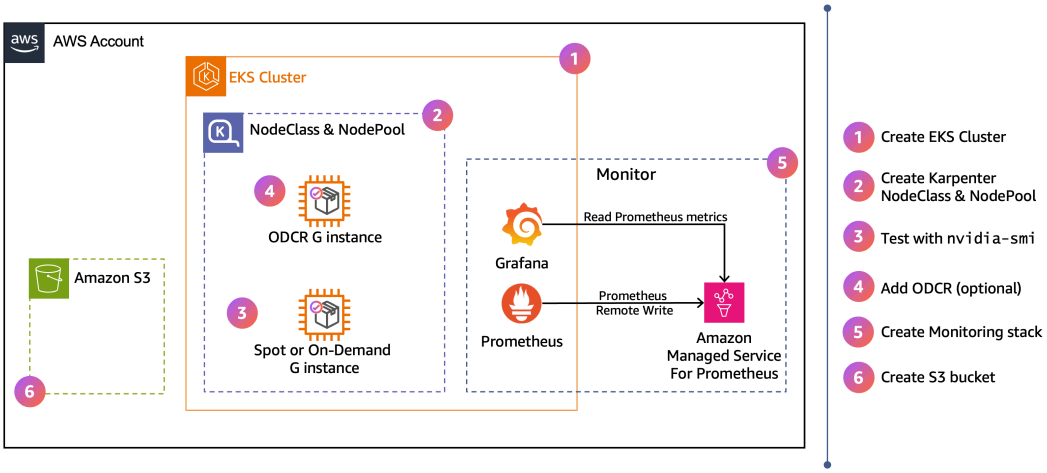
使用 CLIs 為 AI/ML 工作負載設定 Amazon EKS 叢集

> **Tip**
>
> 註冊即將舉行的 Amazon EKS AI/ML 研討會。

本節將逐步引導您建立透過 CLI 命令在 Amazon EKS 上執行訓練或推論工作負載所需的基礎設施。這些步驟包括建立 EKS 叢集、使用 EKS Auto Mode 或 Karpenter 啟用 GPU 的節點、使用 Prometheus 和 Grafana 的監控堆疊，以及用於模型權重的 Amazon S3 儲存。

如需這些功能如何在 EKS 叢集中佈建和自動擴展 EC2 執行個體的詳細資訊，請參閱 EKS Auto Mode 和 Karpenter 的文件。 EC2

高階架構和工作流程



圖表顯示本節設定的 AWS 高階架構。右側的編號步驟指出您在以下步驟中完成組態的順序。

先決條件

> **Important**
>
> 您在本教學課程中建立的資源，包括 EKS 叢集、GPU 執行個體、Application Load Balancer 和 Amazon Managed Service for Prometheus，都會產生費用。當您完成時刪除資源，以避免持續收費。

- `kubectl` >= 1.36。如需設定說明，請參閱 the section called “設定 kubectl 和 eksctl”。
- AWS CLI >= 2.27。如需設定說明，請參閱安裝。
- Helm >= 3.14。如需設定說明，請參閱設定 Helm。
- `jq`。 如需設定說明，請參閱下載 jq。
- `eksctl` >= 0.227.0。如需設定說明，請參閱 `eksctl` 文件中的安裝。

驗證您的`eksctl`版本：

```
eksctl version
```

如果您使用的版本早於 0.227.0，請遵循 eksctl 安裝指南升級至最新版本。

設定環境變數

在這些步驟中保持以下叢集名稱和 AWS 區域一致。變更它可能會導致後續命令以錯誤 EKS 叢集為目標。

```
export CLUSTER_NAME=ai-eks-docs
export AWS_REGION=us-east-2
```

使用所有可用的可用 AZs 可改善容錯能力，並增加取得 GPU 容量的機會：

```
export AZS=$(aws ec2 describe-availability-zones \
  --region ${AWS_REGION} \
  --query "AvailabilityZones[?ZoneId!='use1-az3' && ZoneId!='usw1-az2' && ZoneId!
='cac1-az3'].ZoneName" \
  --output text | tr '\t' ',')
echo $AZS
```

> **Important**
>
> `cac1-az3` 排除可用區域 `use1-az3`、 和 `usw1-az2`，因為 Amazon EKS 不支援在這些區域中放置控制平面。在任何這些區域中建立具有子網路的叢集會導致 `UnsupportedAvailabilityZoneException`。

預期的輸出結果：

```
us-east-2a,us-east-2b,us-east-2c
```

輸出中的 AZs 會因區域而異。此範例顯示 `us-east-2`區域的可用 AZs。

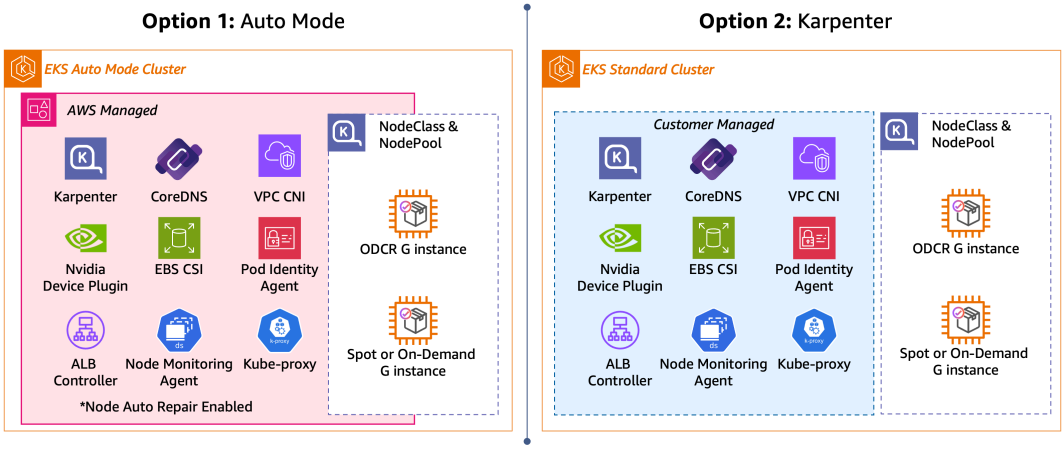
建立叢集和 GPU NodePool

本節提供兩種建立 EKS 叢集和 GPU 節點的路徑，如下圖所示。在整個指南中僅選擇一個選項。

- EKS Auto Mode — 除了核心聯網、儲存和負載平衡附加元件之外，EKS Auto Mode 還包含和管理訓練和推論工作負載的下列功能：EKS 節點監控代理程式、自動節點修復、快速容器提取的 SOCI 快照器，以及預設 NodeClass 的 GPU 準備程度。NVIDIA 裝置外掛程式包含在 Bottlerocket 加速 AMI 中，EKS Auto Mode 用於已啟用 GPU 的節點。

- 自我管理 Karpenter：在沒有 EKS Auto Mode 的 EKS 叢集上，您必須負責安裝和設定訓練和推論工作負載所需的元件。這包括聯網附加元件 (VPC CNI、CoreDNS、kube-proxy)、Karpenter、EKS 節點監控代理程式、NVIDIA 裝置外掛程式，以及用於快速容器提取的 SOCI 快照器。

EKS 叢集選項：EKS Auto Mode 和自我管理 Karpenter



在下列每個步驟中，選擇路徑 (EKS Auto Mode、Karpenter) 並完全遵循。完成所選路徑的步驟後，您將擁有具有 GPU NodePool 的 EKS 叢集，準備好排程 GPU 工作負載。

步驟 1：建立叢集

首先建立 EKS 叢集，並安裝 GPU 工作負載所需的叢集元件。

使用 EKS Auto Mode 時，單一`eksctl create cluster --enable-auto-mode`命令會佈建適用於 GPU 工作負載的 EKS 叢集。

使用自我管理的 Karpenter，`eksctl create cluster`命令會佈建核心聯網附加元件，然後需要額外的步驟，才能透過 Karpenter 功能閘道啟用自動節點修復、安裝 EKS 節點監控代理程式，以及安裝 NVIDIA 裝置外掛程式。

EKS Auto Mode

建立 EKS Auto Mode 叢集

```
eksctl create cluster \
  --name=$CLUSTER_NAME \
  --region=$AWS_REGION \
  --enable-auto-mode \
  --version=1.36 \
```

```
  --zones=$AZS
```

此命令需要幾分鐘的時間才能完成。完成後， eksctl會自動更新您的 kubeconfig 檔案，以使用新佈建的叢集。驗證叢集是否正常運作：

```
kubectl get pods --all-namespaces
```

預期的輸出結果：

```
NAMESPACE     NAME                                READY   STATUS    RESTARTS   AGE
kube-system   metrics-server-55cf976ddd-cz2mw   1/1     Running   0          3m
kube-system   metrics-server-55cf976ddd-wrjvv   1/1     Running   0          3m
```

在 EKS Auto 模式中，VPC CNI、kube-proxy 和 CoreDNS 會以受管元件的形式執行，而不會在 中顯示為 Podkube-system。

建立預設 StorageClass

GPU 工作負載通常需要持久性儲存，例如快取模型權重，因此 Kubernetes 不會在每次重新啟動 Pod 時重新下載它們。EKS Auto Mode 包含內建的區塊儲存功能，因此您不會安裝 Amazon EBS CSI 驅動程式。建立StorageClass參考自動模式佈建器的 ：

```
cat << EOF | kubectl apply -f -
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gp3
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: ebs.csi.eks.amazonaws.com
volumeBindingMode: WaitForFirstConsumer
parameters:
  type: gp3
  encrypted: "true"
EOF
```

此命令會建立 gp3StorageClass，並使用 storageclass.kubernetes.io/is-default-class註釋將其標記為叢集預設值。省略的任何 PersistentVolumeClaim storageClassName都會使用此 StorageClass。volumeBindingMode: WaitForFirstConsumer 設定會延遲磁碟區建立，直到 Kubernetes 排程 Pod，因此 EBS 磁碟區會登陸與 Pod 相同的可用區域。

Self-managed Karpenter

將 Helm 驗證為公有 ECR

eksctl 從 Amazon Public ECR 提取 Karpenter Helm Chart。在建立叢集之前進行驗證，以避免 Helm 安裝步驟發生 403 錯誤：

```
aws ecr-public get-login-password --region us-east-1 \
  | helm registry login --username AWS --password-stdin public.ecr.aws
```

Public ECR 是在 中託管的全域服務us-east-1。無論您的 EKS 叢集位於哪個區域，都可以--region us-east-1在此處使用 。

預期的輸出： Login Succeeded

使用 Karpenter 建立 EKS 叢集

將您的 Karpenter 版本存放在 環境變數中，以供日後使用。如需最新的 Karpenter 版本，請參閱 GitHub 上的 Karpenter 版本。

```
export KARPENTER_VERSION=1.14.0
```

Example叢集組態 YAML

```
cat << EOF > /tmp/cluster-karpenter.yaml
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: ${CLUSTER_NAME}
  region: ${AWS_REGION}
  version: "1.36"
  tags:
    karpenter.sh/discovery: ${CLUSTER_NAME}

availabilityZones: [$(echo $AZS | sed 's/,/, /g')]

autoModeConfig:
  enabled: false

iam:
```

```
  withOIDC: true

karpenter:
  version: "${KARPENTER_VERSION}"
  withSpotInterruptionQueue: true

managedNodeGroups:
  - name: system
    instanceTypes: ["m6i.2xlarge", "m5.2xlarge", "m6a.2xlarge", "m5a.2xlarge",
 "m7i.2xlarge"]
    desiredCapacity: 2
    minSize: 2
    maxSize: 3
    labels:
      node-role: system
    tags:
      karpenter.sh/discovery: ${CLUSTER_NAME}

addons:
  - name: eks-pod-identity-agent
  - name: eks-node-monitoring-agent
  - name: aws-ebs-csi-driver
EOF
```

system 受管節點群組會列出數個相同大小的執行個體類型 (8 個 vCPU、32 GiB)，而非單一類型。如果某個類型在可用區域中暫時無法使用，節點群組的 Auto Scaling 群組會回復為另一種類型，以避免InsufficientInstanceCapacity啟動失敗。對於隨需受管節點群組，所有列出的執行個體類型都必須具有相同的 vCPU 和記憶體，以便節點大小一致。

使用組態檔案建立叢集：

```
eksctl create cluster -f /tmp/cluster-karpenter.yaml
```

此命令大約需要 15 分鐘。它使用專用於託管附加元件和 Karpenter 控制器的受管節點群組來建立 EKS 叢集。Karpenter 安裝時已啟用 Spot 中斷佇列，因此可以處理 Spot 中斷和重新平衡建議。autoModeConfig.enabled: false 設定會明確表示此叢集不使用 EKS Auto 模式，因此在此路徑中安裝的 Karpenter 元件負責節點管理。

叢集也會取得 EKS Pod Identity Agent、EKS 節點監控代理程式，以及安裝為 EKS 附加元件的 Amazon EBS CSI 驅動程式。本指南稍後會使用 EKS Pod Identity。EKS 節點監控代理程式會在每個節點上執行，並讀取核心日誌以設定節點條件，例如 AcceleratedHardwareReady、

KernelReady和 NetworkingReady，Karpenter 自動節點修復會使用這些條件來決定何時取代運作狀態不佳的節點。

驗證叢集是否正常運作：

```
kubectl get pods --all-namespaces
```

預期的輸出包括 Karpenter、CoreDNS、kube-proxy、aws-node (VPC CNI)、EKS Pod Identity Agent 和 EKS 節點監控代理程式。

```
NAMESPACE     NAME                                READY   STATUS    RESTARTS   AGE
karpenter     karpenter-567547464c-s6vkx          1/1     Running   0          3m40s
karpenter     karpenter-567547464c-x7gmw          1/1     Running   0          3m40s
kube-system   aws-node-b6gf2                      2/2     Running   0          12m
kube-system   aws-node-lcphh                      2/2     Running   0          12m
kube-system   coredns-7d4dcbf4fb-ccvrr            1/1     Running   0          16m
kube-system   coredns-7d4dcbf4fb-qbhk2            1/1     Running   0          16m
kube-system   eks-node-monitoring-agent-h79vm     1/1     Running   0          9m45s
kube-system   eks-node-monitoring-agent-tf4dw     1/1     Running   0          9m45s
kube-system   eks-pod-identity-agent-5jbtc        1/1     Running   0          12m
kube-system   eks-pod-identity-agent-rwcrc        1/1     Running   0          12m
kube-system   kube-proxy-p4bmq                    1/1     Running   0          12m
kube-system   kube-proxy-v5nwr                    1/1     Running   0          12m
kube-system   metrics-server-5b966ff79c-hr58p     1/1     Running   0          9m22s
kube-system   metrics-server-5b966ff79c-szs2d     1/1     Running   0          9m22s
```

啟用 Karpenter 功能閘道

EKS Auto Mode 預設會啟用自動節點修復和靜態容量。在自我管理的 Karpenter 上，您必須明確啟用NodeRepair特徵閘道，以及靜態 NodePools 的StaticCapacity閘道，其會維護透過 的固定節點計數，spec.replicas而不是擴展以回應待定的 Pod。在 Karpenter 中，這兩個閘道預設為 Alpha 和 off。下列命令會修補 Karpenter 部署以啟用兩個閘道。更新部署環境會觸發推出 Karpenter Pod：

```
kubectl set env deployment/karpenter -n karpenter \
  FEATURE_GATES=NodeRepair=true,StaticCapacity=true
```

預期的輸出結果：

```
deployment.apps/karpenter env updated
```

等待 Karpenter Pod 推出：

```
kubectl rollout status deployment/karpenter -n karpenter
```

安裝 NVIDIA 裝置外掛程式

EKS 最佳化 AL2023 AMI 不包含 NVIDIA 裝置外掛程式（不同於 EKS Auto Mode 使用的 Bottlerocket AMI)。透過 Helm 安裝它，讓 GPU 資源可與叢集上的 Pod 搭配使用。

```
helm repo add nvdp https://nvidia.github.io/k8s-device-plugin
helm repo update
```

```
cat << 'EOF' > /tmp/nvdp-values.yaml
mofedEnabled: false
nodeSelector:
  amiFamily: al2023
gfd:
  enabled: true
nfd:
  worker:
    tolerations:
      - operator: "Exists"
EOF
```

```
helm install nvidia-device-plugin nvdp/nvidia-device-plugin \
  --namespace kube-system \
  -f /tmp/nvdp-values.yaml
```

- `mofedEnabled: false`：停用不使用的 Mellanox OFED (InfiniBand) 檢查 AWS
- `nodeSelector.amiFamily: al2023`：僅將 DaemonSet 範圍限定為 AL2023 節點 (Bottlerocket 已內建外掛程式）
- `gfd.enabled: true`：啟用 GPU Feature Discovery 標籤 (`nvidia.com/gpu.product`、`nvidia.com/gpu.memory`等）

確認已安裝 NVIDIA 裝置外掛程式。預期在佈建具有相符標籤的 GPU NodePool 之前，沒有任何裝置外掛程式 Pod。

```
kubectl get daemonset nvidia-device-plugin -n kube-system
```

預期的輸出結果：

```
NAME                   DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE
 SELECTOR      AGE
nvidia-device-plugin   0         0         0       0            0
 amiFamily=al2023   2m5s
```

接著，建立一般用途 NodePool。system 受管節點群組會執行附加元件和控制器。新增一般用途 KarpenterNodePool，讓非 GPU 工作負載可以隨需擴展，而無需登陸system節點。GPU NodePool稍後會在步驟 2 中建立。

一般用途 EC2NodeClass 和 NodePool YAML

```
cat << EOF | kubectl apply -f -
apiVersion: karpenter.k8s.aws/v1
kind: EC2NodeClass
metadata:
  name: general-purpose
spec:
  role: "eksctl-KarpenterNodeRole-${CLUSTER_NAME}"
  amiSelectorTerms:
    - alias: al2023@latest
  subnetSelectorTerms:
    - tags:
        karpenter.sh/discovery: ${CLUSTER_NAME}
  securityGroupSelectorTerms:
    - tags:
        karpenter.sh/discovery: ${CLUSTER_NAME}
  tags:
    karpenter.sh/discovery: ${CLUSTER_NAME}
---
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: general-purpose
spec:
  template:
    spec:
      nodeClassRef:
        group: karpenter.k8s.aws
        kind: EC2NodeClass
        name: general-purpose
      requirements:
```

```
        - key: karpenter.sh/capacity-type
          operator: In
          values: ["on-demand"]
        - key: karpenter.k8s.aws/instance-category
          operator: In
          values: ["c", "m", "r"]
        - key: karpenter.k8s.aws/instance-generation
          operator: Gt
          values: ["4"]
        - key: kubernetes.io/arch
          operator: In
          values: ["amd64"]
  limits:
    cpu: 1000
    memory: 1000Gi
EOF
```

此 NodePool 沒有污點，因此任何不容忍 GPU 污點的 Pod 都可以在其上排程。

建立預設 StorageClass

GPU 工作負載通常需要持久性儲存，例如快取模型權重，因此 Kubernetes 不會在每次重新啟動 Pod 時重新下載它們。Amazon EBS CSI 驅動程式已在此步驟稍早的叢集組態中由 addons：區塊安裝，因此您只需要建立 StorageClass：

```
cat << EOF | kubectl apply -f -
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gp3
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: ebs.csi.aws.com
volumeBindingMode: WaitForFirstConsumer
parameters:
  type: gp3
  encrypted: "true"
EOF
```

此命令會建立 gp3StorageClass，並使用 storageclass.kubernetes.io/is-default-class註釋將其標記為叢集預設值。省略的任何 PersistentVolumeClaim storageClassName都會使用此 StorageClass。volumeBindingMode：

WaitForFirstConsumer 設定會延遲磁碟區建立，直到 Kubernetes 排程 Pod，因此 EBS 磁碟區會登陸與 Pod 相同的可用區域。

> **Warning**
>
> 對於 EKS Auto Mode 和自我管理 Karpenter 路徑，自動節點修復的行為與 NodePools 佈建的節點相同。EKS Auto Mode 和 Karpenter 中的自動節點修復是一種強而有力的中斷方法，可略過 PodDisruptionBudgets、karpenter.sh/do-not-disrupt註釋和 terminationGracePeriod。自動節點修復會等待 10 分鐘，再將條件AcceleratedHardwareReady設為 的節點取代為 ，False將其他修復條件取代為 30 分鐘。

設定負載平衡

本指南的後續步驟透過 an AWS Application Load Balancer (ALB) 公開 Grafana （以及推論演練中的模型端點）。立即設定負載平衡先決條件，讓這些區段無需進一步設定Ingress即可建立 。

ALB 會根據 AWS 資源標籤放置在 VPC 子網路中。用於面向網際網路的負載平衡器的公有子網路必須加上標籤 kubernetes.io/role/elb = 1，而用於內部負載平衡器的私有子網路必須加上標籤 kubernetes.io/role/internal-elb = 1。由於您使用 建立此叢集eksctl，因此這些標籤已存在於eksctl建立的子網路上。如果您使用自己的 VPC 或匯入的子網路，請手動標記它們。如需詳細資訊，請參閱the section called “標記子網路”。

建立負載平衡器 IngressClass

其餘設定會因路徑而有所不同。EKS Auto Mode 包含內建的 ALB 控制器，因此您可以IngressClass自行建立 。自我管理的 Karpenter 需要使用 Helm 安裝 AWS Load Balancer控制器，Helm Chart IngressClass會為您建立 。

EKS Auto Mode

EKS Auto Mode 包含內建的 ALB 控制器，因此您不會安裝任何其他元件。建立指向內建控制器alb的 IngressClass。Ingress 您稍後建立的每個 都會依名稱參考此類別，並透過註釋設定自己的方案和其他選項。

```
cat << EOF | kubectl apply -f -
apiVersion: networking.k8s.io/v1
kind: IngressClass
```

```
metadata:
  name: alb
spec:
  controller: eks.amazonaws.com/alb
EOF
```

Self-managed Karpenter

AWS Load Balancer控制器不是 EKS 受管的附加元件，因此無法像 EBS CSI 驅動程式那樣進入`eksctl`叢集組態`addons:`區塊。使用 Helm 安裝它。

下載 IAM 政策

控制器需要 IAM 許可才能建立和管理 ALBs。下載官方 IAM 政策 JSON：

```
curl -o iam_policy.json https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/main/docs/install/iam_policy.json
```

建立 IAM 政策：

```
aws iam create-policy \
  --policy-name AWSLoadBalancerControllerIAMPolicy \
  --policy-document file://iam_policy.json
```

建立 Pod Identity 關聯

叢集已在步驟 1 中安裝`eks-pod-identity-agent`附加元件，因此可使用 EKS Pod Identity。建立將 IAM 政策繫結至控制器服務帳戶的 Pod Identity 關聯：

```
eksctl create podidentityassociation \
  --cluster ${CLUSTER_NAME} \
  --region ${AWS_REGION} \
  --namespace kube-system \
  --service-account-name aws-load-balancer-controller \
  --permission-policy-arns arn:aws:iam::$(aws sts get-caller-identity --query
 Account --output text):policy/AWSLoadBalancerControllerIAMPolicy
```

使用 Helm 安裝

新增 EKS 圖表 Helm 儲存庫：

```
helm repo add eks https://aws.github.io/eks-charts
helm repo update
```

查詢叢集的 VPC ID。Helm 值需要它，以便控制器可以探索子網路：

```
VPC_ID=$(aws eks describe-cluster \
  --name ${CLUSTER_NAME} \
  --region ${AWS_REGION} \
  --query 'cluster.resourcesVpcConfig.vpcId' \
  --output text)
```

安裝控制器。nodeSelector 和 容錯能力會將控制器與叢集關鍵基礎設施放在system受管節點群組上，而不是與 GPU 或一般用途 Karpenter 啟動的節點競爭容量：

```
helm install aws-load-balancer-controller eks/aws-load-balancer-controller \
  --namespace kube-system \
  --version 3.4.2 \
  --set clusterName=${CLUSTER_NAME} \
  --set region=${AWS_REGION} \
  --set vpcId=${VPC_ID} \
  --set serviceAccount.create=true \
  --set serviceAccount.name=aws-load-balancer-controller \
  --set "nodeSelector.node-role=system" \
  --set "tolerations[0].key=CriticalAddonsOnly" \
  --set "tolerations[0].operator=Exists"
```

驗證控制器是否正在執行。它應該報告兩個可用的複本：

```
kubectl get deployment aws-load-balancer-controller -n kube-system
```

驗證 IngressClass

Helm Chart 會建立IngressClass名為 的 alb作為控制器安裝的一部分，並指向自我管理的控制器 ingress.k8s.aws/alb。您不需要手動建立它。確認它存在：

```
kubectl get ingressclass alb
```

預期的輸出結果：

```
NAME   CONTROLLER            PARAMETERS   AGE
```

```
alb    ingress.k8s.aws/alb   <none>       22s
```

這兩個路徑現在都具有 alb IngressClass。Grafana 和推論區段透過每個Ingress註釋參考ingressClassName: alb和設定 ALB 結構描述、source-IP 允許清單和其他行為。

步驟 2：建立動態 GPU NodePool

定義 NodePool，使用 Spot 容量搭配隨需做為備用，以動態方式佈建產生大於 4 且小於 7 的 G 系列 GPU 執行個體。EKS Auto Mode 和 Karpenter 路徑都使用相同的 NodePool API，唯一的區別在於其指向的 NodeClass。在 EKS Auto 模式中，綁定的 default NodeClass 已選取正確的 AMI 並設定 SOCI 平行提取，因此 NodePool 是您建立的唯一物件。在自我管理的 Karpenter 中，您也需要可EC2NodeClass固定 AMI 並調校 SOCI 的自訂。

> **Important**
>
> G7 EC2 執行個體類型需要 NVIDIA 驅動程式 595 版或更新版本。EKS 最佳化加速 AMIs 目前包含不支援 G7 執行個體的 NVIDIA 驅動程式 580 版。此步驟中的 NodePool 會將執行個體產生限制為小於 7，讓 Karpenter 不會選取 G7 執行個體。若要搭配 Amazon EKS 使用 G7 執行個體，您必須使用 NVIDIA 驅動程式 595 版建置自訂 AMI。如需詳細資訊，請參閱the section called “使用 EKS Linux GPU AMIs”。

EKS Auto Mode

在 EKS Auto 模式中，綁定的 default NodeClass 會自動選取 GPU 執行個體的 Bottlerocket AMI，其中包括預先安裝的 NVIDIA 驅動程式、NVIDIA 裝置外掛程式和 SOCI 平行提取。您只需要套用參考 NodeClass 的 default NodePool： NodeClass

```
cat << 'EOF' | kubectl apply -f -
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: gpu-inf
spec:
  template:
    metadata:
      labels:
        guide: ai-eks-docs
    spec:
```

```
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default
      taints:
        - key: nvidia.com/gpu
          effect: NoSchedule
      requirements:
        - key: karpenter.sh/capacity-type
          operator: In
          values: ["spot", "on-demand"]
        - key: eks.amazonaws.com/instance-category
          operator: In
          values: ["g"]
        - key: eks.amazonaws.com/instance-generation
          operator: Gt
          values: ["4"]
        - key: eks.amazonaws.com/instance-generation
          operator: Lt
          values: ["7"]
        - key: kubernetes.io/arch
          operator: In
          values: ["amd64"]
  limits:
    cpu: 1000
    memory: 5000Gi
EOF
```

此 NodePool 會佈建產生大於 4 且小於 7 的 G 系列 GPU 執行個體 (G5、G6e 等）。EKS Auto Mode 會在這些限制內選擇執行個體類型。

Self-managed Karpenter

自我管理的 Karpenter 不包含預設 NodeClass。您首先會建立 `EC2NodeClass` 來鎖定 EKS 最佳化 NVIDIA AL2023 AMI 別名、透過`FastImagePull`功能閘道啟用 SOCI，並設定 `instanceStorePolicy: RAID0`將容器化映像快取移至本機 NVMe。然後，您可以建立參考它的 NodePool。

建立 EC2NodeClass

Example

```
cat << EOF | kubectl apply -f -
```

```
apiVersion: karpenter.k8s.aws/v1
kind: EC2NodeClass
metadata:
  name: gpu-inf
  labels:
    guide: ai-eks-docs
spec:
  role: "eksctl-KarpenterNodeRole-${CLUSTER_NAME}"
  amiSelectorTerms:
    - alias: al2023@latest
  subnetSelectorTerms:
    - tags:
        karpenter.sh/discovery: ${CLUSTER_NAME}
  securityGroupSelectorTerms:
    - tags:
        karpenter.sh/discovery: ${CLUSTER_NAME}
  tags:
    karpenter.sh/discovery: ${CLUSTER_NAME}
  instanceStorePolicy: RAID0
  userData: |
    apiVersion: node.eks.aws/v1alpha1
    kind: NodeConfig
    spec:
      featureGates:
        FastImagePull: true
EOF
```

`instanceStorePolicy: RAID0` 將本機 NVMe 磁碟組合成 RAID-0 陣列。`al2023@latest` AMI 別名解析為 EKS 最佳化 AL2023 AMI。當 Karpenter 啟動 GPU 執行個體類型時，會自動選取 AL2023_x86_64_NVIDIA 加速變體，其中包含預先安裝的 NVIDIA 驅動程式。

`FastImagePull` 功能閘道會啟用 SOCI 快照器平行提取模式，同時下載和解壓縮映像層。這符合 G、P 和 Trn 執行個體系列中的 EKS 自動模式行為。

如需更多 SOCI 調校選項（每個映像的並行下載、區塊大小等），請參閱 Karpenter SOCI 藍圖。

驗證 EC2NodeClass：

```
kubectl get ec2nodeclass gpu-inf
```

預期的輸出：`READY True`。如果為 `False`，請執行 `kubectl describe ec2nodeclass gpu-inf`並檢查缺少子網路或安全群組標籤的條件。

建立 GPU NodePool

Example

```
cat << EOF | kubectl apply -f -
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: gpu-inf
spec:
  template:
    metadata:
      labels:
        guide: ai-eks-docs
        amiFamily: al2023
    spec:
      nodeClassRef:
        group: karpenter.k8s.aws
        kind: EC2NodeClass
        name: gpu-inf
      taints:
        - key: nvidia.com/gpu
          effect: NoSchedule
      requirements:
        - key: karpenter.sh/capacity-type
          operator: In
          values: ["spot", "on-demand"]
        - key: karpenter.k8s.aws/instance-category
          operator: In
          values: ["g"]
        - key: karpenter.k8s.aws/instance-generation
          operator: Gt
          values: ["4"]
        - key: karpenter.k8s.aws/instance-generation
          operator: Lt
          values: ["7"]
        - key: kubernetes.io/arch
          operator: In
          values: ["amd64"]
  limits:
    cpu: 1000
    memory: 5000Gi
EOF
```

節點範本上的amiFamily: al2023標籤是 NVIDIA 裝置外掛程式 DaemonSet 用來選取這些節點的標籤。Karpenter 會在這些限制內選擇執行個體類型。nvidia.com/gpu:NoSchedule 污點可確保只會在這些節點上排程符合 GPU 資格的 Pod。

驗證已建立 NodePool：

```
kubectl get nodepool gpu-inf
```

預期的輸出結果：

```
NAME      NODECLASS   NODES   READY   AGE
gpu-inf   default     0       True    8s
```

在自我管理 Karpenter 路徑上，NODECLASS 欄會顯示 ，gpu-inf而非 default。

步驟 3：使用範例 Pod 進行測試

使用 Pod 測試 GPU NodePool nvidia-smi 設定。

```
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: nvidia-smi
  labels:
    guide: ai-eks-docs
spec:
  tolerations:
    - key: "nvidia.com/gpu"
      operator: "Exists"
      effect: "NoSchedule"
  containers:
    - name: nvidia-smi
      image: public.ecr.aws/amazonlinux/amazonlinux:2023-minimal
      command: ["nvidia-smi"]
      resources:
        limits:
          nvidia.com/gpu: 1
  restartPolicy: OnFailure
EOF
```

確認 Pod 已排程並成功完成。

```
kubectl get pods
```

預期的輸出結果：

```
NAME         READY   STATUS      RESTARTS   AGE
nvidia-smi   0/1     Completed   0          67s
```

STATUS： 已完成表示 nvidia-smi 命令已執行並結束。檢查 Pod 日誌以查看節點偵測到的 GPU。

```
kubectl logs nvidia-smi
```

預期的輸出結果：

```
+-----------------------------------------------------------------------------------------
+
| NVIDIA-SMI 580.159.03             Driver Version: 580.159.03     CUDA Version: 13.0
   |
+-----------------------------------------+------------------------
+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr.
 ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute
 M. |
|                                         |                        |               MIG
 M. |
|=========================================+========================
+======================|
|   0  NVIDIA L4                      On  |   00000000:35:00.0 Off |
 0 |
| N/A   42C    P8             16W /   72W |       0MiB /  23034MiB |      0%
 Default |
|                                         |                        |
 N/A |
+-----------------------------------------+------------------------
+----------------------+
```

輸出會顯示 GPU 模型、驅動程式版本、CUDA 版本和可用的記憶體。在此範例中，Karpenter 佈建了具有 NVIDIA L4 GPU 和 24 GB 記憶體的 G6 執行個體。42C 是目前的 GPU 溫度，而 P8 是低功耗閒置狀態，這是預期的，因為沒有工作負載正在使用 GPU。16W / 72W 顯示相較於最大功率容量

的目前耗電量，而 0MiB / 23034MiB 顯示相較於總可用容量使用的目前 GPU 記憶體。由於 Pod 剛執行 nvidia-smi 並結束，因此沒有工作負載使用 GPU，因此記憶體為 0，且電源處於閒置狀態。NVIDIA GPU 驅動程式版本 (580.159.03) 來自 Bottlerocket AMI，而 CUDA 版本 (13.0) 來自容器映像。GPU 模型和記憶體會根據 Karpenter 選取的執行個體類型而有所不同。G5 執行個體具有 NVIDIA A10G GPUs (24 GB)、G6 執行個體具有 NVIDIA L4 GPUs (24 GB)，而 G6e 執行個體具有 NVIDIA L40S GPUs (48 GB)。

若要了解 Karpenter 和 Kubernetes 排程器如何協調以佈建節點並放置 Pod，請檢查 Pod 的生命週期事件：

```
kubectl describe po nvidia-smi
```

預期的輸出結果：

```
Events:
  Type     Reason                  Age   From                     Message
  ----     ------                  ----  ----                     -------
  Warning  FailedScheduling        60s   default-scheduler      0/2 nodes are
 available: 2 node(s) had untolerated taint(s). no new claims to deallocate,
 preemption: 0/2 nodes are available: 2 Preemption is not helpful for scheduling.
  Normal   Nominated               59s   eks-auto-mode/compute  Pod should schedule on:
 nodeclaim/gpu-inf-vxcnj
  Normal   Scheduled               24s   default-scheduler      Successfully assigned
 default/nvidia-smi to i-0fb17a09bc4203164
  Warning  FailedCreatePodSandBox  21s   kubelet                Failed to create
 pod sandbox: rpc error: code = Unknown desc = failed to setup network for sandbox
 "7f85e25b220c8fb245187758dbbbc8efb3d40f3e49e13054404880daf4c3b2f0": plugin type="aws-
cni" name="aws-cni" failed (add): add cmd: failed to setup network policy
  Normal   Pulling                  7s   kubelet                spec.containers{nvidia-
smi}: Pulling image "public.ecr.aws/amazonlinux/amazonlinux:2023-minimal"
  Normal   Pulled                   5s   kubelet                spec.containers{nvidia-
smi}: Successfully pulled image "public.ecr.aws/amazonlinux/amazonlinux:2023-minimal"
 in 1.237s (1.237s including waiting). Image size: 37442701 bytes.
  Normal   Created                  5s   kubelet                spec.containers{nvidia-
smi}: Container created
  Normal   Started                  5s   kubelet                spec.containers{nvidia-
smi}: Container started
```

這些事件會顯示 Pod 排程序列：Pod 一開始無法排程，因為沒有 GPU 節點 (FailedScheduling)、Karpenter 會指定新的 NodeClaim (Nominated)、排程器會在節點就緒時指派 Pod （排程），然後提取並啟動容器映像。EKS Auto Mode 在 G、P 和 Trn 執行個體上安裝並設定

SOCI （可擴展的 OCI) 平行提取。請注意，由於 SOCI 平行提取，容器映像在 2 秒 (1.237 秒） 內從 ECR 提取。

NodeClaim 是 Karpenter 建立來佈建特定節點的請求。它會顯示執行個體類型、容量類型、AZ，以及節點是否已就緒。

```
kubectl get nodeclaims
```

預期的 NodeClaim 輸出：

```
NAME            TYPE         CAPACITY   ZONE         NODE                   READY   AGE
gpu-inf-vxcnj   g6.4xlarge   spot       us-east-2c   i-0fb17a09bc4203164    True    51s
```

執行個體類型和 AZ 會有所不同。產生大於 4 且小於 7 的任何 G 系列執行個體都符合資格。

中的`FailedCreatePodSandBox`警告`kubectl describe pod nvidia-smi`是暫時性且預期的。VPC CNI 會在節點聯結後以非同步方式初始化，而 kubelet 會自動重試。如果 Pod 保留在中`ContainerCreating`，請使用 檢查節點事件`kubectl describe node <node-name>`。

Tip

如果沒有出現節點，請檢查容量不足錯誤：

```
kubectl get events | grep InsufficientCapacityError
```

Karpenter 快取無法使用的方案 3 分鐘。擴大 NodePool 中允許的執行個體類型和可用AZs會增加登陸容量的機會。

Note

Karpenter 啟動的 Spot 執行個體不會出現在 EC2 Spot 請求主控台中。Karpenter 搭配 使用 EC2 `CreateFleet` APItype: instant。執行個體會以spot生命週期顯示在 EC2 執行個體主控台中。

步驟 4：將預留容量新增至 NodePool （選用）

若要先搭配 Spot/隨需備用使用預留容量，請建立隨需容量保留 (ODCR)，並將其連接至 NodeClass，然後從步驟 2 更新動態 NodePool，以允許`reserved`容量。兩個路徑的保留 API 呼叫相同；NodeClass 連接不同，因為 EKS Auto Mode 和自我管理 Karpenter 使用不同的 NodeClass 類型。

> **Warning**
>
> 下列命令會產生預留執行個體類型的費用，直到您使用 將其取消為止`aws ec2 cancel-capacity-reservation --capacity-reservation-id <id>`。

建立容量保留：

```
CR_AZ="us-east-2a"
INSTANCE_TYPE="g6e.4xlarge"

aws ec2 create-capacity-reservation \
  --instance-type $INSTANCE_TYPE \
  --instance-platform Linux/UNIX \
  --availability-zone "$CR_AZ" \
  --instance-count 1 \
  --instance-match-criteria open \
  --end-date-type unlimited
```

如果您收到`InsufficientInstanceCapacity`錯誤，`CR_AZ`請變更為不同的可用區域，然後重試。

查詢容量保留 ID，並將其存放在 shell 變數中，以進行下列步驟：

```
CAPACITY_RESERVATION_ID=$(aws ec2 describe-capacity-reservations \
  --filters "Name=state,Values=active" "Name=instance-type,Values=${INSTANCE_TYPE}" \
  --query 'CapacityReservations[0].CapacityReservationId' \
  --output text \
  --region ${AWS_REGION})
echo "Capacity reservation ID: ${CAPACITY_RESERVATION_ID}"
```

然後為您的路徑套用 NodeClass 和 NodePool 變更：

EKS Auto Mode

在 EKS Auto 模式中，綁定的 `default` NodeClass 是唯讀的，因此請建立參考保留的自訂 NodeClass，然後更新 NodePool 以指向 NodeClass，並將`reserved`容量新增至`capacity-type`清單。

Example自訂 NodeClass YAML

```
NODE_ROLE=$(kubectl get nodeclass default -o jsonpath='{.spec.role}')

cat << EOF | kubectl apply -f -
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: gpu-inf
  labels:
    guide: ai-eks-docs
spec:
  role: "$NODE_ROLE"
  subnetSelectorTerms:
    - tags:
        alpha.eksctl.io/cluster-name: "$CLUSTER_NAME"
        kubernetes.io/role/internal-elb: "1"
  securityGroupSelectorTerms:
    - tags:
        aws:eks:cluster-name: "$CLUSTER_NAME"
  capacityReservationSelectorTerms:
    - id: "$CAPACITY_RESERVATION_ID"
EOF
```

`kubernetes.io/role/internal-elb: "1"` 標籤可確保節點僅在私有子網路中啟動。

更新 NodePool 以使用 ODCR 支援的 NodeClass，並包含 `reserved`做為容量類型：

Example已更新 NodePool YAML

```
cat << EOF | kubectl apply -f -
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: gpu-inf
spec:
  template:
```

```
    metadata:
      labels:
        guide: ai-eks-docs
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: gpu-inf
      taints:
        - key: nvidia.com/gpu
          effect: NoSchedule
      requirements:
        - key: karpenter.sh/capacity-type
          operator: In
          values: ["spot", "on-demand", "reserved"]
        - key: eks.amazonaws.com/instance-category
          operator: In
          values: ["g"]
        - key: eks.amazonaws.com/instance-generation
          operator: Gt
          values: ["4"]
        - key: eks.amazonaws.com/instance-generation
          operator: Lt
          values: ["7"]
        - key: kubernetes.io/arch
          operator: In
          values: ["amd64"]
  limits:
    cpu: 1000
    memory: 5000Gi
EOF
```

Self-managed Karpenter

對於自我管理的 Karpenter，重新套用EC2NodeClass您在步驟 2 中建立並capacityReservationSelectorTerms新增的 。欄位名稱和形狀符合其他索引標籤中NodeClass顯示的 EKS Auto 模式。

Example已更新 EC2NodeClass YAML

```
cat << EOF | kubectl apply -f -
apiVersion: karpenter.k8s.aws/v1
kind: EC2NodeClass
```

```
metadata:
  name: gpu-inf
  labels:
    guide: ai-eks-docs
spec:
  role: "eksctl-KarpenterNodeRole-${CLUSTER_NAME}"
  amiSelectorTerms:
    - alias: al2023@latest
  subnetSelectorTerms:
    - tags:
        karpenter.sh/discovery: ${CLUSTER_NAME}
  securityGroupSelectorTerms:
    - tags:
        karpenter.sh/discovery: ${CLUSTER_NAME}
  tags:
    karpenter.sh/discovery: ${CLUSTER_NAME}
  instanceStorePolicy: RAID0
  capacityReservationSelectorTerms:
    - id: "$CAPACITY_RESERVATION_ID"
  userData: |
    apiVersion: node.eks.aws/v1alpha1
    kind: NodeConfig
    spec:
      featureGates:
        FastImagePull: true
EOF
```

從步驟 2 的唯一變更是新`capacityReservationSelectorTerms`欄位。所有其他欄位保持不變。

更新 NodePool 以包含 `reserved`做為容量類型：

Example已更新 NodePool YAML

```
cat << EOF | kubectl apply -f -
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: gpu-inf
spec:
  template:
    metadata:
      labels:
```

```
        guide: ai-eks-docs
        amiFamily: al2023
    spec:
      nodeClassRef:
        group: karpenter.k8s.aws
        kind: EC2NodeClass
        name: gpu-inf
      taints:
        - key: nvidia.com/gpu
          effect: NoSchedule
      requirements:
        - key: karpenter.sh/capacity-type
          operator: In
          values: ["spot", "on-demand", "reserved"]
        - key: karpenter.k8s.aws/instance-category
          operator: In
          values: ["g"]
        - key: karpenter.k8s.aws/instance-generation
          operator: Gt
          values: ["4"]
        - key: karpenter.k8s.aws/instance-generation
          operator: Lt
          values: ["7"]
        - key: kubernetes.io/arch
          operator: In
          values: ["amd64"]
  limits:
    cpu: 1000
    memory: 5000Gi
EOF
```

Karpenter `reserved`將 視為最具成本效益的選項，並優先啟動。一旦保留已滿，就會回到 Spot 或隨需。

驗證預留優先順序和 Spot 備用

套用變更後，請驗證 Karpenter 是否排定保留容量的優先順序，並回到 Spot 或隨需。部署 2 個複本部署，每個 Pod 請求 1 個 GPU。ODCR 適用於 1 個執行個體，因此第一個 Pod 會觸發 Karpenter 啟動預留節點。第二個 Pod 無法容納在預留節點上，並觸發 Karpenter 從 Spot 或隨需容量啟動另一個節點。

```
cat << 'EOF' | kubectl apply -f -
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: gpu-overflow-test
  labels:
    guide: ai-eks-docs
spec:
  replicas: 2
  selector:
    matchLabels:
      app: gpu-overflow-test
  template:
    metadata:
      labels:
        app: gpu-overflow-test
        guide: ai-eks-docs
    spec:
      tolerations:
        - key: nvidia.com/gpu
          operator: Exists
          effect: NoSchedule
      containers:
        - name: nvidia-smi
          image: public.ecr.aws/amazonlinux/amazonlinux:2023-minimal
          command: ["sh", "-c", "nvidia-smi && sleep infinity"]
          resources:
            limits:
              nvidia.com/gpu: 1
EOF
```

與執行和結束步驟 3 `nvidia-smi`的測試 Pod 不同，此部署會讓 Pod 保持執行 (`sleep infinity`)，以便保留 GPU 且不釋出節點。

驗證在不同節點上排程的 Pod：

```
kubectl get pods -l app=gpu-overflow-test -o wide
```

預期的輸出結果：

```
NAME                                READY   STATUS    RESTARTS   AGE     IP
     NODE                   NOMINATED NODE   READINESS GATES
gpu-overflow-test-59b97944fb-lq56c   1/1     Running   0          2m42s
 192.168.186.240   i-057692590480155da   <none>           <none>
```

```
gpu-overflow-test-59b97944fb-z4zcx   1/1     Running   0          2m42s
 192.168.130.64    i-0521ecd1849fa0578   <none>           <none>
```

兩個 Pod 正在執行中，每個在不同的節點上。

檢查 NodeClaims 以查看容量類型：

```
kubectl get nodeclaims
```

預期的輸出結果：

```
NAME            TYPE          CAPACITY    ZONE         NODE                   READY
 AGE
gpu-inf-shg5w   g6e.xlarge    reserved    us-east-2a   i-0ea91fdeef65b8cb6   True
 2m2s
gpu-inf-ssnqf   g6e.2xlarge   spot        us-east-2b   i-00ccf7ce65cf3f6ca   True
 112s
```

預留節點會先啟動，然後在保留已滿時，再啟動 Spot 或隨需節點。

清除測試部署：

```
kubectl delete deployment gpu-overflow-test
```

監控

將收集叢集、節點和 GPU 指標的監控堆疊安裝到 Amazon Managed Service for Prometheus (AMP)，並使用 Grafana 將其視覺化。kube-prometheus-stack Helm Chart 會將 Prometheus 部署到 AMP 的湊集和遠端寫入指標，以及儀表板的自我管理 Grafana。NVIDIA DCGM Exporter 新增 GPU 特定的指標（使用率、記憶體、溫度、功率、NVLink、張量活動）。

Prometheus、Grafana 和運算子預設會落在非 GPU 節點上，因為 GPU `nvidia.com/gpu:NoSchedule` 節點帶有污點。Node-exporter 和 DCGM Exporter 都會在 GPU 節點上執行，因此我們可以在整個機群中抓取主機和 GPU 指標。

如果您開啟新的終端機，請設定叢集名稱和區域：

```
export CLUSTER_NAME=ai-eks-docs
export AWS_REGION=us-east-2
```

建立 AMP 工作區

建立 AMP 工作區以存放指標：

```
aws amp create-workspace \
  --alias "amp-ws-${CLUSTER_NAME}" \
  --region ${AWS_REGION}
```

取得工作區 ID：

```
AMP_WORKSPACE_ID=$(aws amp list-workspaces \
  --alias "amp-ws-${CLUSTER_NAME}" \
  --query 'workspaces[0].workspaceId' \
  --output text \
  --region ${AWS_REGION})

echo "AMP Workspace ID: ${AMP_WORKSPACE_ID}"
```

取得遠端寫入端點：

```
AMP_ENDPOINT=$(aws amp describe-workspace \
  --workspace-id ${AMP_WORKSPACE_ID} \
  --query 'workspace.prometheusEndpoint' \
  --output text \
  --region ${AWS_REGION})

echo "AMP Endpoint: ${AMP_ENDPOINT}"
```

建立 IAM 政策和 EKS Pod 身分關聯

建立允許 Prometheus 遠端寫入指標和 Grafana 查詢它們的 IAM 政策：

```
ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)

AMP_POLICY_ARN=$(aws iam create-policy \
  --policy-name "${CLUSTER_NAME}-amp-grafana-policy" \
  --policy-document "{\"Version\": \"2012-10-17\", \"Statement\": [{\"Sid\":
 \"AllowAMPReadWrite\", \"Effect\": \"Allow\", \"Action\": [\"aps:ListWorkspaces
\", \"aps:DescribeWorkspace\", \"aps:GetMetricMetadata\", \"aps:GetSeries
\", \"aps:QueryMetrics\", \"aps:RemoteWrite\", \"aps:GetLabels\"],
 \"Resource\": \"arn:aws:aps:${AWS_REGION}:${ACCOUNT_ID}:workspace/*\"},
 {\"Sid\": \"AllowCloudWatchMetrics\", \"Effect\": \"Allow\", \"Action
```

```
\": [\"cloudwatch:DescribeAlarmsForMetric\", \"cloudwatch:ListMetrics\",
 \"cloudwatch:GetMetricData\", \"cloudwatch:GetMetricStatistics\"], \"Resource\": \"*
\"}]}" \
  --query 'Policy.Arn' \
  --output text)

echo "AMP Policy ARN: ${AMP_POLICY_ARN}"
```

建立 Prometheus 和 Grafana 的監控命名空間和服務帳戶：

```
kubectl create namespace monitoring
kubectl create serviceaccount amp-iamproxy-ingest-service-account -n monitoring
kubectl create serviceaccount grafana-sa -n monitoring
```

建立 EKS Pod 身分關聯，將服務帳戶連結至 IAM 政策：

```
eksctl create podidentityassociation \
  --cluster ${CLUSTER_NAME} \
  --namespace monitoring \
  --service-account-name amp-iamproxy-ingest-service-account \
  --role-name "${CLUSTER_NAME}-amp-ingest-role" \
  --permission-policy-arns ${AMP_POLICY_ARN} \
  --region ${AWS_REGION}

eksctl create podidentityassociation \
  --cluster ${CLUSTER_NAME} \
  --namespace monitoring \
  --service-account-name grafana-sa \
  --role-name "${CLUSTER_NAME}-grafana-role" \
  --permission-policy-arns ${AMP_POLICY_ARN} \
  --region ${AWS_REGION}
```

確認已建立兩個 EKS Pod 身分關聯：

```
eksctl get podidentityassociation --cluster ${CLUSTER_NAME} --region ${AWS_REGION}
```

預期的輸出應在`monitoring`命名空間`grafana-sa`中同時包含 `amp-iamproxy-ingest-service-account`和 。

安裝 kube-prometheus-stack

新增 Helm 儲存庫：

```
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm repo update
```

此值檔案會省略 Prometheus、Grafana 和運算子的 nodeSelector：GPU `nvidia.com/gpu:NoSchedule` 節點的污點會使其遠離 GPU 節點，因此預設會落在系統或一般用途集區上。Node-exporter 使用萬用字元公差，因此它在每個節點上執行，包括 GPU 節點，以在整個機群中收集指標。

值檔案也會設定 Grafana，`Ingress`因此圖表會使用`albIngressClass`您在[設定負載平衡中建立的 ，為瀏覽器存取佈建面向網際網路的](#) ALB。`alb.ingress.kubernetes.io/inbound-cidrs` 註釋會將負載平衡器限制為您自己的 IP 地址，因此請在安裝之前進行設定。

> **⚠ Grafana 可透過 HTTP 使用預設登入資料公開存取**
>
> Grafana 包含預設的管理員登入資料，並提供監控資料的完整管理存取權。面向網際網路的負載平衡器會透過純文字 HTTP 將此界面放置在公有網際網路上。自動化掃描器會在幾分鐘內探索公有負載平衡器。您必須使用 `alb.ingress.kubernetes.io/inbound-cidrs`註釋限制存取，並將 source-IP 允許清單視為最低保護，而非完整保護。同時變更預設的 Grafana 管理員密碼。如需更強的姿勢，請將配置設定為 ，並新增 TLS 憑證`internal`，以使用內部配置（只能在 VPC 或連線的 VPN 內存取）。

尋找您的公有 IP 地址，並將其儲存為 /32 CIDR。值檔案會從`MY_CIDR`環境變數讀取：

```
export MY_CIDR="$(curl -s https://checkip.amazonaws.com)/32"
echo $MY_CIDR
```

結果看起來像 `203.0.113.4/32`。如果您的網路動態指派地址，您的 IP 地址可能會隨著時間而變更，因此請考慮設定更廣泛的範圍，例如 `203.0.113.0/24`。

建立值檔案：

Example kube-prometheus-stack 值檔案

```
cat << EOF > /tmp/kube-prometheus-values.yaml
alertmanager:
  enabled: false

prometheus-adapter:
  enabled: false
```

```
prometheus:
  serviceAccount:
    create: false
    name: amp-iamproxy-ingest-service-account
  prometheusSpec:
    serviceAccountName: amp-iamproxy-ingest-service-account
    enableRemoteWriteReceiver: true
    retention: 2h
    scrapeInterval: 30s
    evaluationInterval: 30s
    podMonitorSelectorNilUsesHelmValues: false
    serviceMonitorSelectorNilUsesHelmValues: false
    resources:
      requests:
        cpu: 500m
        memory: 1Gi
      limits:
        memory: 8Gi
    remoteWrite:
      - url: "${AMP_ENDPOINT}api/v1/remote_write"
        sigv4:
          region: "${AWS_REGION}"
        queueConfig:
          maxSamplesPerSend: 1000
          maxShards: 200
          capacity: 2500

prometheusOperator:
  resources:
    requests:
      cpu: 100m
      memory: 128Mi
    limits:
      memory: 256Mi

kube-state-metrics:
  resources:
    requests:
      cpu: 50m
      memory: 128Mi
    limits:
      memory: 512Mi
```

```
grafana:
  enabled: true
  ingress:
    enabled: true
    ingressClassName: alb
    labels:
      guide: ai-eks-docs
    annotations:
      alb.ingress.kubernetes.io/scheme: internet-facing
      alb.ingress.kubernetes.io/target-type: ip
      alb.ingress.kubernetes.io/inbound-cidrs: ${MY_CIDR}
      alb.ingress.kubernetes.io/load-balancer-name: grafana-ai-eks-docs
      alb.ingress.kubernetes.io/healthcheck-path: /api/health
  serviceAccount:
    create: false
    name: grafana-sa
  resources:
    requests:
      cpu: 100m
      memory: 256Mi
    limits:
      memory: 1Gi
  grafana.ini:
    auth.sigv4:
      enabled: true
  sidecar:
    datasources:
      defaultDatasourceEnabled: false
  plugins:
    - grafana-amazonprometheus-datasource
  additionalDataSources:
    - name: Amazon-Managed-Prometheus
      type: grafana-amazonprometheus-datasource
      access: proxy
      url: "${AMP_ENDPOINT}"
      isDefault: true
      jsonData:
        sigV4Auth: true
        defaultRegion: "${AWS_REGION}"
        sigV4Region: "${AWS_REGION}"
      editable: true
  dashboardProviders:
    dashboardproviders.yaml:
      apiVersion: 1
```

```
      providers:
        - name: default
          orgId: 1
          folder: 'GPU Monitoring'
          type: file
          disableDeletion: false
          editable: true
          options:
            path: /var/lib/grafana/dashboards/default
  dashboards:
    default:
      nvidia-dcgm:
        gnetId: 25261
        revision: 1
        datasource:
          - name: DS_PROMETHEUS
            value: Amazon-Managed-Prometheus
      vllm:
        gnetId: 25263
        revision: 1
        datasource:
          - name: DS_PROMETHEUS
            value: Amazon-Managed-Prometheus
      vllm-load-analysis:
        gnetId: 25494
        revision: 1
        datasource:
          - name: DS_PROMETHEUS
            value: Amazon-Managed-Prometheus

prometheus-node-exporter:
  resources:
    requests:
      cpu: 50m
      memory: 64Mi
    limits:
      memory: 128Mi
  tolerations:
    - operator: Exists
EOF
```

驗證變數是否正確填入：

```
grep -E "url:|region:" /tmp/kube-prometheus-values.yaml
```

您應該會看到完整的 AMP 端點 URL（以 開頭https://aps-workspaces…）和您的區域。如果其中一個是空的，請重新匯出變數並重新建立檔案。

安裝圖表：

```
helm install kube-prometheus-stack prometheus-community/kube-prometheus-stack \
  --namespace monitoring \
  -f /tmp/kube-prometheus-values.yaml
```

驗證 Pod 正在執行：

```
kubectl get pods -n monitoring
```

預期的輸出結果：

```
NAME                                                     READY   STATUS    RESTARTS
 AGE
kube-prometheus-stack-grafana-7c58f54f77-rftrj           3/3     Running   0
 4m
kube-prometheus-stack-kube-state-metrics-d68dcbc84-5smxq 1/1     Running   0
 4m
kube-prometheus-stack-operator-5895df479f-ttm47          1/1     Running   0
 4m
kube-prometheus-stack-prometheus-node-exporter-t9q7s     1/1     Running   0
 4m
kube-prometheus-stack-prometheus-node-exporter-x6vfb     1/1     Running   0
 4m
prometheus-kube-prometheus-stack-prometheus-0            2/2     Running   0
 4m
```

堆疊會部署下列元件：

- Prometheus (StatefulSet)：抓取指標並將其遠端寫入 AMP
- Grafana：儀表板和視覺化，使用 AMP 資料來源預先設定
- kube-state-metrics：產生有關 Kubernetes 物件狀態 (Pod 狀態、資源請求/限制、NodeClaim 狀態）的指標
- node-exporter (DaemonSet，每個節點一個）：收集主機層級指標 (CPU、記憶體、磁碟、網路）

- 運算子：管理 Prometheus 和 Alertmanager 自訂資源

警示管理員在此設定中已停用。

存取 Grafana

您可以透過面向網際網路 AWS Application Load Balancer(ALB) 存取 Grafana。您不會在此處建立它。kube-prometheus-stack 圖表使用`albIngressClass`來自設定負載平衡的 ，從值檔案中的`grafana.ingress`組態佈建。Grafana Helm Chart 具有內建`Ingress`支援，因此不需要單獨的資訊清單。ALB 名為 `grafana-ai-eks-docs`，帶有標籤 `guide: ai-eks-docs`，且僅限於您在`MY_CIDR`安裝時在 中設定的 IP 地址。

> **Grafana ALB 目標群組需要運作狀態檢查路徑**
>
> 值檔案中的`alb.ingress.kubernetes.io/healthcheck-path`註釋會將負載平衡器的運作狀態檢查指向 Grafana `/api/health`端點，這會傳回 `200`。如果沒有它，ALB 運作狀態檢查預設為 `/`，Grafana 會以`302`重新導向至 的方式回答該檢查`/login`，導致目標群組將端點報告為運作狀態不佳。

列印負載平衡器 URL。負載平衡器是以非同步方式建立，因此請允許一兩分鐘：

```
echo "http://$(kubectl get ingress kube-prometheus-stack-grafana -n monitoring -o
 jsonpath='{.status.loadBalancer.ingress[0].hostname}')"
```

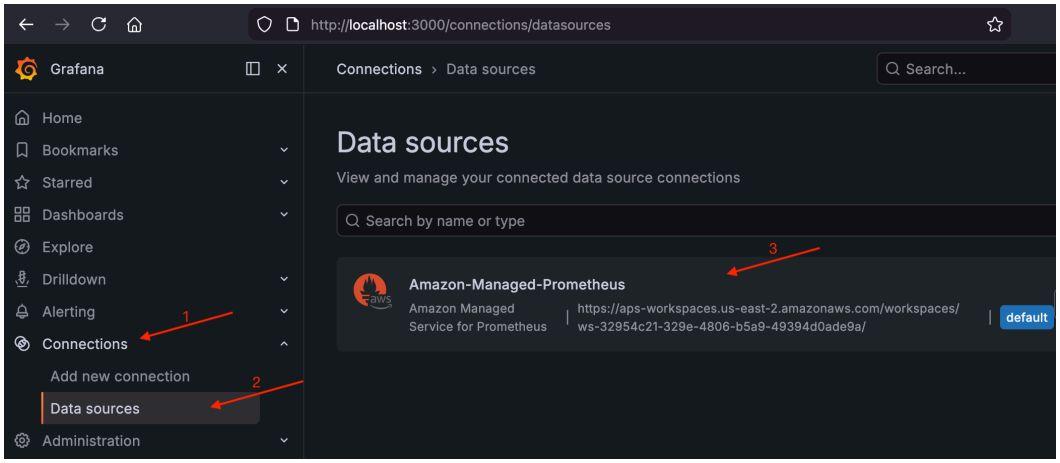
在瀏覽器中開啟 URL。從`admin`下列命令使用使用者名稱和密碼登入：

```
kubectl --namespace monitoring get secrets kube-prometheus-stack-grafana -o
 jsonpath="{.data.admin-password}" | base64 -d ; echo
```

若要驗證指標管道是否端對端運作：

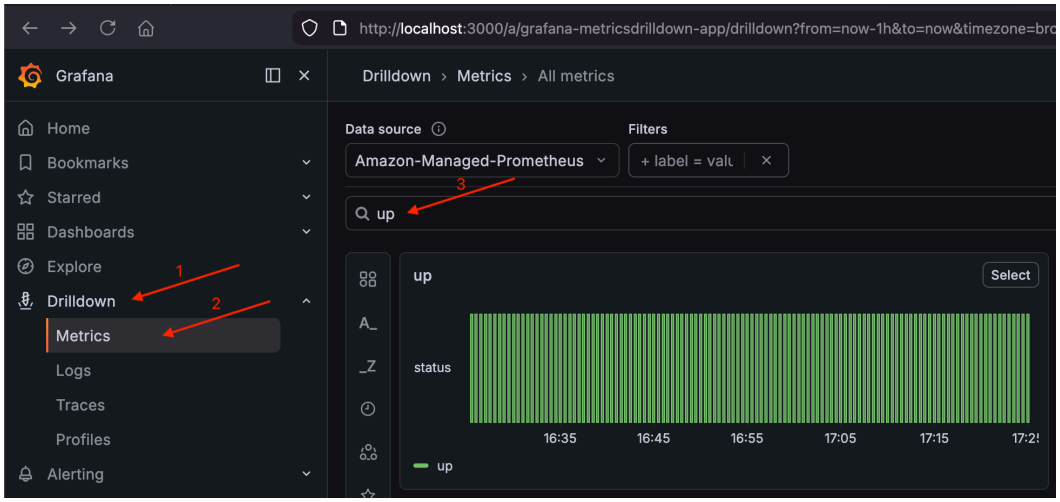
1. 導覽至連線 > 資料來源，並確認 `Amazon-Managed-Prometheus` 列為預設資料來源。

   驗證 Grafana 中的 AMP 資料來源



2. 導覽至向下切入 > 指標並搜尋指標。 up您應該會看到叢集湊集目標的結果。

   驗證 **up** Grafana 中的指標



如果 up顯示結果，則管道 （ 叢集 → Prometheus → AMP → Grafana) 正在運作。

部署適用於 GPU 指標的 DCGM Exporter

kube-prometheus-stack 會收集節點層級的 CPU 和記憶體指標，但不會收集 GPU 指標。NVIDIA DCGM Exporter 新增 GPU 使用率、記憶體用量、溫度、耗電量、NVLink 頻寬和張量活動。

```
helm repo add gpu-helm-charts https://nvidia.github.io/dcgm-exporter/helm-charts
helm repo update
```

為您的路徑設定 GPU 節點選擇器索引鍵。EKS Auto Mode 和自我管理 Karpenter 為 GPU 製造商使用不同的標籤索引鍵。

EKS Auto Mode

```
GPU_NODE_SELECTOR_KEY="eks.amazonaws.com/instance-gpu-manufacturer"
```

Self-managed Karpenter

```
GPU_NODE_SELECTOR_KEY="karpenter.k8s.aws/instance-gpu-manufacturer"
```

建立 DCGM 匯出工具值檔案：

Example dcgm-exporter 值檔案

```
cat << EOF > /tmp/dcgm-exporter-values.yaml
resources:
  requests:
    memory: "512Mi"
    cpu: "100m"
  limits:
    memory: "1Gi"
    cpu: "500m"

serviceMonitor:
  enabled: true
  additionalLabels:
    release: kube-prometheus-stack

nodeSelector:
  ${GPU_NODE_SELECTOR_KEY}: nvidia

tolerations:
  - key: "nvidia.com/gpu"
    operator: "Exists"
    effect: "NoSchedule"

customMetrics: |
  # Clocks
  DCGM_FI_DEV_SM_CLOCK,  gauge, SM clock frequency (in MHz).
  DCGM_FI_DEV_MEM_CLOCK, gauge, Memory clock frequency (in MHz).
  # Temperature
  DCGM_FI_DEV_MEMORY_TEMP, gauge, Memory temperature (in C).
  DCGM_FI_DEV_GPU_TEMP,    gauge, GPU temperature (in C).
  # Power
```

```
 DCGM_FI_DEV_POWER_USAGE,              gauge, Power draw (in W).
 DCGM_FI_DEV_TOTAL_ENERGY_CONSUMPTION, counter, Total energy consumption since boot
(in mJ).
 # PCIe
 DCGM_FI_PROF_PCIE_TX_BYTES,  counter, Number of bytes transmitted through PCIe TX (in
KB) via NVML.
 DCGM_FI_PROF_PCIE_RX_BYTES,  counter, Number of bytes received through PCIe RX (in
KB) via NVML.
 DCGM_FI_DEV_PCIE_REPLAY_COUNTER, counter, Total number of PCIe retries.
 # Utilization (the sample period varies depending on the product)
 DCGM_FI_DEV_GPU_UTIL,      gauge, GPU utilization (in %).
 DCGM_FI_DEV_MEM_COPY_UTIL, gauge, Memory utilization (in %).
 DCGM_FI_DEV_ENC_UTIL,      gauge, Encoder utilization (in %).
 DCGM_FI_DEV_DEC_UTIL,      gauge, Decoder utilization (in %).
 # Errors and violations
 DCGM_FI_DEV_XID_ERRORS,            gauge, Value of the last XID error encountered.
 DCGM_EXP_XID_ERRORS_COUNT,         gauge, Value of count of XID errors encountered.
 DCGM_FI_DEV_POWER_VIOLATION,       counter, Throttling duration due to power
constraints (in us).
 DCGM_FI_DEV_THERMAL_VIOLATION,     counter, Throttling duration due to thermal
constraints (in us).
 DCGM_FI_DEV_SYNC_BOOST_VIOLATION,  counter, Throttling duration due to sync-boost
constraints (in us).
 DCGM_FI_DEV_BOARD_LIMIT_VIOLATION, counter, Throttling duration due to board limit
constraints (in us).
 DCGM_FI_DEV_LOW_UTIL_VIOLATION,    counter, Throttling duration due to low
utilization (in us).
 DCGM_FI_DEV_RELIABILITY_VIOLATION, counter, Throttling duration due to reliability
constraints (in us).
 # Memory usage
 DCGM_FI_DEV_FB_FREE, gauge, Framebuffer memory free (in MiB).
 DCGM_FI_DEV_FB_USED, gauge, Framebuffer memory used (in MiB).
 # Retired pages
 DCGM_FI_DEV_RETIRED_SBE,     counter, Total number of retired pages due to single-bit
errors.
 DCGM_FI_DEV_RETIRED_DBE,     counter, Total number of retired pages due to double-bit
errors.
 DCGM_FI_DEV_RETIRED_PENDING, counter, Total number of pages pending retirement.
 # NVLink
 DCGM_FI_DEV_NVLINK_BANDWIDTH_TOTAL, counter, Total number of NVLink bandwidth
counters for all lanes.
 DCGM_FI_PROF_NVLINK_TX_BYTES,       counter, The rate of data transmitted over NVLink
not including protocol headers in bytes per second.
```

```
  DCGM_FI_PROF_NVLINK_RX_BYTES,       counter, The rate of data received over NVLink
 not including protocol headers in bytes per second.
  # DCP metrics
  DCGM_FI_PROF_GR_ENGINE_ACTIVE,   gauge, Ratio of time the graphics engine is active
 (in %).
  DCGM_FI_PROF_SM_ACTIVE,          gauge, The ratio of cycles an SM has at least 1 warp
 assigned (in %).
  DCGM_FI_PROF_SM_OCCUPANCY,       gauge, The ratio of number of warps resident on an
 SM (in %).
  DCGM_FI_PROF_PIPE_TENSOR_ACTIVE, gauge, Ratio of cycles the tensor (HMMA) pipe is
 active (in %).
  DCGM_FI_PROF_DRAM_ACTIVE,        gauge, Ratio of cycles the device memory interface
 is active sending or receiving data (in %).
  DCGM_FI_DEV_CLOCK_THROTTLE_REASONS, gauge, Current clock throttle reasons (bitmask of
 DCGM_CLOCKS_THROTTLE_REASON_*).
  DCGM_FI_DEV_GPU_NVLINK_ERRORS,      gauge, Identifies a GPU NVLink error type
 returned by DCGM_FI_DEV_GPU_NVLINK_ERRORS.
  ## NVLink
  DCGM_FI_DEV_NVLINK_BANDWIDTH_L0, counter, The number of bytes of active NVLink rx or
 tx data including both header and payload.
  ## Remapped rows
  DCGM_FI_DEV_UNCORRECTABLE_REMAPPED_ROWS, counter, Number of remapped rows for
 uncorrectable errors.
  DCGM_FI_DEV_CORRECTABLE_REMAPPED_ROWS, counter, Number of remapped rows for
 correctable errors.
  DCGM_FI_DEV_ROW_REMAP_FAILURE, gauge, whether remapping of rows has failed.
  ## Profiling metrics
  DCGM_FI_PROF_PIPE_FP64_ACTIVE, gauge, Ratio of cycles the fp64 pipes are active (in
 %).
  DCGM_FI_PROF_PIPE_FP32_ACTIVE, gauge, Ratio of cycles the fp32 pipes are active (in
 %).
  DCGM_FI_PROF_PIPE_FP16_ACTIVE, gauge, Ratio of cycles the fp16 pipes are active (in
 %).
  # ECC
  DCGM_FI_DEV_ECC_SBE_VOL_TOTAL, counter, Total number of single-bit volatile ECC
 errors.
  DCGM_FI_DEV_ECC_DBE_VOL_TOTAL, counter, Total number of double-bit volatile ECC
 errors.
EOF
```

customMetrics 欄位會使用延伸指標集覆寫 DCGM 匯出者的預設指標集，其中包含 NVLink 頻寬、張量活動、PCIe 輸送量、ECC 錯誤和熱調節。對於推論工作負載，這有助於您了解 GPU 運算單位

是否充分利用、GPU 是否因為批次大小低而在請求之間閒置、CPU 和 GPU 之間的資料傳輸是否為瓶頸、熱調節是否造成延遲峰值，以及大型批次的 GPU 記憶體空間剩餘多少。

安裝 DCGM 匯出工具：

```
helm install dcgm-exporter gpu-helm-charts/dcgm-exporter \
  --namespace monitoring \
  -f /tmp/dcgm-exporter-values.yaml
```

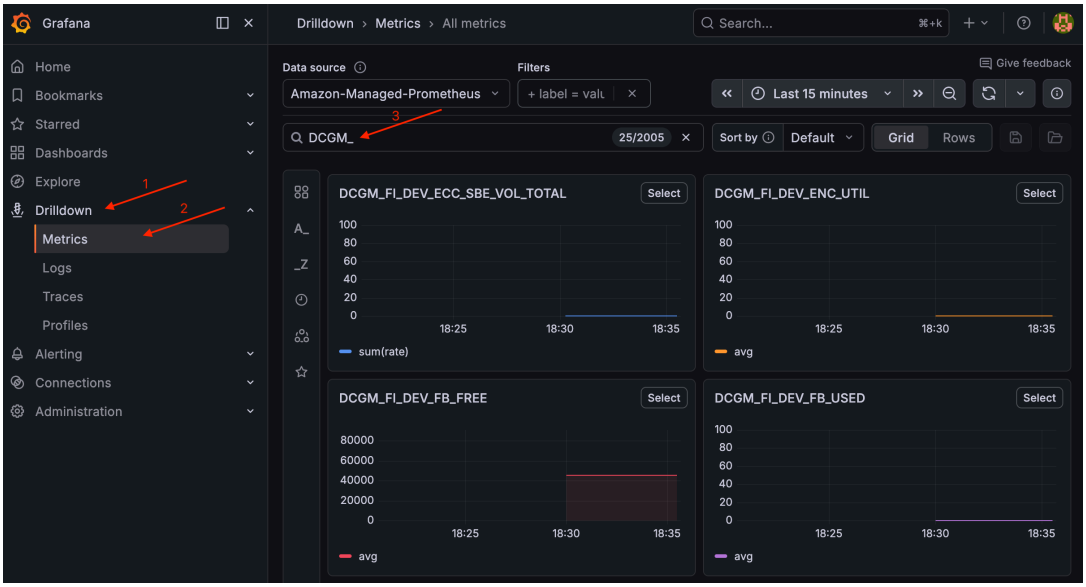
允許匯出程式`tolerations`在您於步驟 2 佈建的 GPU 污點節點上執行。`serviceMonitor` 具有 `release: kube-prometheus-stack`標籤的 可確保 Prometheus 自動探索和抓取它。

驗證 DCGM 匯出工具 DaemonSet：

```
kubectl get daemonset dcgm-exporter -n monitoring
```

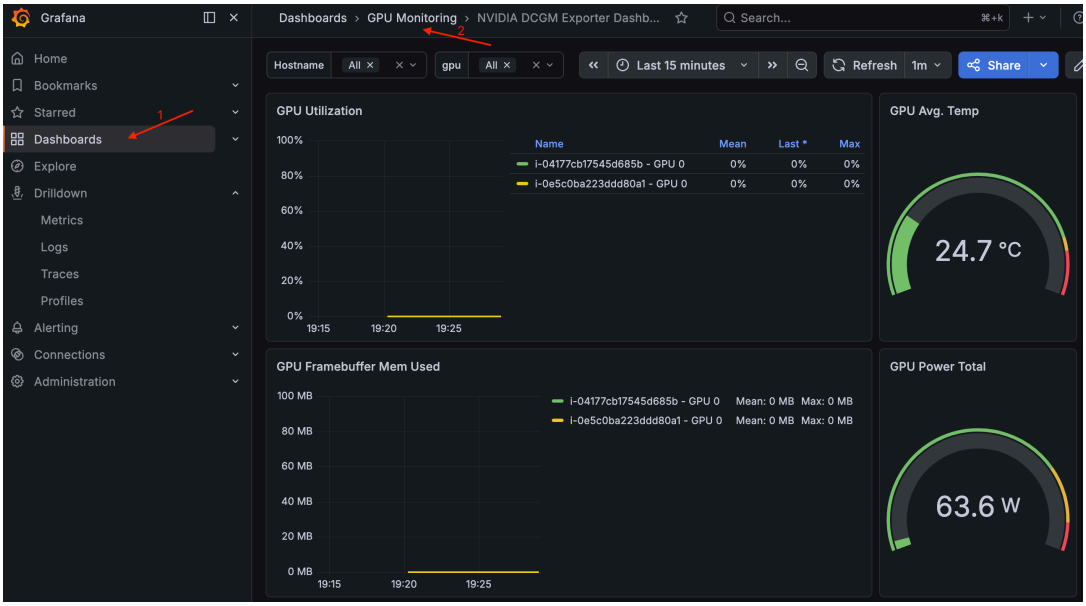
GPU 節點執行後，您應該會看到一個就緒的 Pod。若要驗證 DCGM 指標，請導覽至 Grafana 中的向下切入 > 指標，並搜尋 `DCGM_`。

在 Grafana 中驗證 DCGM 指標



若要檢視儀表板，請導覽至儀表板 > GPU 監控 > NVIDIA DCGM 匯出工具儀表板。

Grafana 中的 NVIDIA DCGM 匯出工具儀表板



模型權重 S3 儲存貯體

建立 Amazon S3 儲存貯體來存放模型權重，並設定 EKS Pod Identity Association，以便工作負載 Pod 可以讀取和寫入模型權重。

如果您開啟新的終端機，請設定叢集名稱和區域：

```
export CLUSTER_NAME=ai-eks-docs
export AWS_REGION=us-east-2
```

建立 S3 儲存貯體

使用隨機尾碼建立儲存貯體，以避免名稱衝突：

```
BUCKET_SUFFIX=$(head -c 4 /dev/urandom | od -An -tx1 | tr -d ' \n')
MODEL_BUCKET="${CLUSTER_NAME}-models-${BUCKET_SUFFIX}"

aws s3 mb s3://${MODEL_BUCKET} --region ${AWS_REGION}
```

2023 年 1 月之後建立的 S3 儲存貯體預設會啟用伺服器端加密 (AES256) 和公有存取封鎖。

設定 S3 存取的 EKS Pod 身分

在`default`命名空間中建立 `model-storage-sa` ServiceAccount、範圍為模型儲存貯體的 IAM 政策，以及連結它們的 EKS Pod 身分關聯。設定的工作負載 Pod `serviceAccountName: model-storage-sa`將能夠讀取和寫入儲存貯體。

```
kubectl create serviceaccount model-storage-sa
```

建立 IAM 政策：

```
POLICY_ARN=$(aws iam create-policy \
  --policy-name "${CLUSTER_NAME}-model-storage-policy" \
  --policy-document "{\"Version\": \"2012-10-17\", \"Statement\": [{\"Effect\":
 \"Allow\", \"Action\": [\"s3:GetObject\", \"s3:PutObject\", \"s3:ListBucket\",
 \"s3:DeleteObject\"], \"Resource\": [\"arn:aws:s3:::${MODEL_BUCKET}\", \"arn:aws:s3:::
${MODEL_BUCKET}/*\"]}]}" \
  --query 'Policy.Arn' \
  --output text)

echo "Policy ARN: ${POLICY_ARN}"
```

> **Note**
>
> 此政策s3:PutObject會針對驗證步驟授予 s3:DeleteObject和 。對於只讀取模型權重的生產推論 Pod，請移除 s3:PutObject並s3:DeleteObject遵循最低權限。

建立 EKS Pod Identity Association。 會eksctl建立具有正確信任政策的 IAM 角色，並將其連結至 ServiceAccount：

```
eksctl create podidentityassociation \
  --cluster ${CLUSTER_NAME} \
  --namespace default \
  --service-account-name model-storage-sa \
  --role-name "${CLUSTER_NAME}-model-storage-role" \
  --permission-policy-arns ${POLICY_ARN} \
  --region ${AWS_REGION}
```

驗證關聯：

```
eksctl get podidentityassociation --cluster ${CLUSTER_NAME} --region ${AWS_REGION}
```

輸出應該包含 default 命名空間中的model-storage-sa關聯。

從 Pod 驗證 S3 存取

使用 `model-storage-sa` ServiceAccount 使用 AWS CLI 映像執行一次性 Pod，以確認 EKS Pod 身分已連線且 S3 存取正常運作：

```
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: s3-test
  labels:
    guide: ai-eks-docs
spec:
  serviceAccountName: model-storage-sa
  containers:
    - name: aws-cli
      image: public.ecr.aws/aws-cli/aws-cli:2.27.0
      command:
        - sh
        - -c
        - |
          echo "=== Caller Identity ==="
          aws sts get-caller-identity
          echo ""
          echo "=== S3 Write Test ==="
          echo "pod identity works" | aws s3 cp - s3://${MODEL_BUCKET}/test.txt
          echo ""
          echo "=== S3 List Test ==="
          aws s3 ls s3://${MODEL_BUCKET}/
          echo ""
          echo "=== S3 Delete Test ==="
          aws s3 rm s3://${MODEL_BUCKET}/test.txt
  restartPolicy: Never
EOF
```

等待 Pod 完成並檢查日誌：

```
kubectl wait --for=jsonpath='{.status.phase}'=Succeeded pod/s3-test --timeout=300s
kubectl logs s3-test
```

預期的輸出結果：

```
=== Caller Identity ===
```

```
{
    "UserId": "AROA...:eks-ai-eks-docs-model-s-...",
    "Account": "123456789012",
    "Arn": "arn:aws:sts::123456789012:assumed-role/ai-eks-docs-model-storage-role/eks-
ai-eks-docs-model-s-..."
}

=== S3 Write Test ===
upload: - to s3://ai-eks-docs-models-01234567/test.txt

=== S3 List Test ===
2026-05-04 12:00:00         19 test.txt

=== S3 Delete Test ===
delete: s3://ai-eks-docs-models-01234567/test.txt
```

發起人身分會確認 Pod 透過 EKS Pod 身分擔任該`${CLUSTER_NAME}-model-storage-role`角色。S3 命令會確認讀取和寫入存取。

清除測試 Pod：

```
kubectl delete pod s3-test
```

後續步驟

叢集準備就緒後，您可以繼續載入並提供模型，以部署大型語言模型並與推論端點互動。

清除

> **Tip**
>
> 如果您打算繼續本指南的下一節，請略過完整清除。只有在完成時才會執行它。

```
export CLUSTER_NAME=ai-eks-docs
export AWS_REGION=us-east-2
```

```
kubectl delete pod nvidia-smi --ignore-not-found
kubectl delete deployment gpu-overflow-test --ignore-not-found
```

取消容量保留

如果您建立了 ODCR，請先取消它。查詢保留 ID：

```
INSTANCE_TYPE="g6e.4xlarge"
CAPACITY_RESERVATION_ID=$(aws ec2 describe-capacity-reservations \
  --filters "Name=state,Values=active" "Name=instance-type,Values=${INSTANCE_TYPE}" \
  --query 'CapacityReservations[0].CapacityReservationId' \
  --output text \
  --region ${AWS_REGION})
echo "Capacity reservation ID: ${CAPACITY_RESERVATION_ID}"
```

取消保留：

```
aws ec2 cancel-capacity-reservation --capacity-reservation-id
 ${CAPACITY_RESERVATION_ID}
```

> **Important**
>
> 取消保留不會終止執行中的執行個體。它們會以標準隨需費率繼續，直到終止為止。取消之前，請先刪除部署以耗盡預留節點。

清除監控

查詢 IAM 政策 ARN：

```
AMP_POLICY_ARN=$(aws iam list-policies \
  --scope Local \
  --query "Policies[?PolicyName=='${CLUSTER_NAME}-amp-grafana-policy'].Arn" \
  --output text)
echo "AMP Policy ARN: ${AMP_POLICY_ARN}"
```

查詢 AMP 工作區 ID：

```
AMP_WORKSPACE_ID=$(aws amp list-workspaces \
  --alias "amp-ws-${CLUSTER_NAME}" \
  --query 'workspaces[0].workspaceId' \
  --output text \
  --region ${AWS_REGION})
echo "AMP Workspace ID: ${AMP_WORKSPACE_ID}"
```

解除安裝 DCGM 匯出工具 Helm 版本：

```
helm uninstall dcgm-exporter -n monitoring
```

解除安裝 kube-prometheus-stack Helm 版本：

```
helm uninstall kube-prometheus-stack -n monitoring
```

刪除 Prometheus 擷取服務帳戶的 EKS Pod 身分關聯：

```
eksctl delete podidentityassociation \
  --cluster ${CLUSTER_NAME} \
  --namespace monitoring \
  --service-account-name amp-iamproxy-ingest-service-account \
  --region ${AWS_REGION}
```

刪除 Grafana 服務帳戶的 EKS Pod 身分關聯：

```
eksctl delete podidentityassociation \
  --cluster ${CLUSTER_NAME} \
  --namespace monitoring \
  --service-account-name grafana-sa \
  --region ${AWS_REGION}
```

刪除 Prometheus 和 Grafana 使用的 IAM 政策：

```
aws iam delete-policy --policy-arn ${AMP_POLICY_ARN}
```

刪除 AMP 工作區：

```
aws amp delete-workspace --workspace-id ${AMP_WORKSPACE_ID} --region ${AWS_REGION}
```

刪除監控命名空間：

```
kubectl delete namespace monitoring
```

清除模型權重 S3 儲存貯體

查詢模型儲存貯體名稱：

```
MODEL_BUCKET=$(aws s3api list-buckets \
  --query "Buckets[?starts_with(Name, '${CLUSTER_NAME}-models-')].Name | [0]" \
  --output text)
echo "Model bucket: ${MODEL_BUCKET}"
```

查詢 IAM 政策 ARN：

```
POLICY_ARN=$(aws iam list-policies \
  --scope Local \
  --query "Policies[?PolicyName=='${CLUSTER_NAME}-model-storage-policy'].Arn" \
  --output text)
echo "Policy ARN: ${POLICY_ARN}"
```

刪除 S3 模型儲存貯體及其所有物件：

```
aws s3 rb s3://${MODEL_BUCKET} --force
```

刪除 EKS Pod 身分關聯：

```
eksctl delete podidentityassociation \
  --cluster ${CLUSTER_NAME} \
  --namespace default \
  --service-account-name model-storage-sa \
  --region ${AWS_REGION}
```

刪除 IAM 政策：

```
aws iam delete-policy --policy-arn ${POLICY_ARN}
```

刪除 Kubernetes ServiceAccount：

```
kubectl delete serviceaccount model-storage-sa
```

刪除剩餘資源和叢集

```
kubectl delete nodepool gpu-inf --ignore-not-found
kubectl delete nodeclass gpu-inf --ignore-not-found
kubectl delete ec2nodeclass gpu-inf --ignore-not-found
eksctl delete cluster --name=$CLUSTER_NAME --region=$AWS_REGION
```

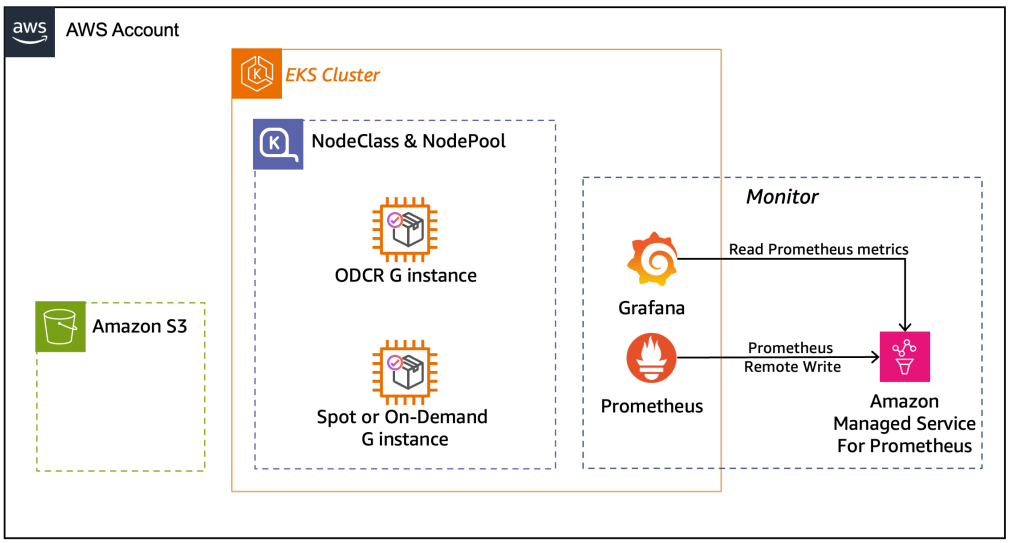
使用 Terraform 為 AI/ML 工作負載設定 Amazon EKS 叢集

> **Tip**
>
> 註冊即將舉行的 Amazon EKS AI/ML 研討會。

本節將逐步引導您建立使用 Terraform 在 Amazon EKS 上執行訓練或推論工作負載所需的基礎設施。這些步驟包括建立 EKS 叢集、使用 EKS Auto Mode 或 Karpenter 啟用 GPU 的節點、使用 Prometheus 和 Grafana 的監控堆疊，以及用於模型權重的 Amazon S3 儲存。

如需這些功能如何在 EKS 叢集中佈建和自動擴展 EC2 執行個體的詳細資訊，請參閱 EKS Auto Mode 和 Karpenter 的文件。 EC2

高階架構和工作流程



圖表顯示本節設定的 AWS 高階架構。

先決條件

> **Important**
>
> 您在本教學課程中建立的資源，包括 EKS 叢集、GPU 執行個體、Application Load Balancer 和 Amazon Managed Service for Prometheus，都會產生費用。當您完成時刪除資源，以避免持續收費。

- Terraform >= 1.15.0。如需設定指示，請參閱安裝 Terraform。
- `kubectl` >= 1.36。如需設定說明，請參閱 the section called “設定 kubectl 和 eksctl”。
- AWS CLI >= 2.27。如需設定說明，請參閱安裝。
- `jq`。 如需設定說明，請參閱下載 jq。

驗證您的工具版本：

```
terraform --version
aws --version
kubectl version --client
jq --version
```

步驟 1：下載並部署 Terraform 程式碼

本演練使用 sample-eks-docs AWS Samples GitHub 儲存庫中的 Terraform 程式碼。將儲存庫複製到工作目錄中：

```
git clone git@github.com:aws-samples/sample-eks-docs.git
cd sample-eks-docs/ai-ml/set-up-cluster
```

儲存庫在您剛變更為的`ai-ml/set-up-cluster/`目錄下具有下列結構：

```
set-up-cluster/
### scripts/
#   ### cleanup.sh
### terraform/
    ### auto-mode/
    ### karpenter/
```

儲存庫提供兩個部署路徑。請只選擇一個，並在整個指南中使用它。

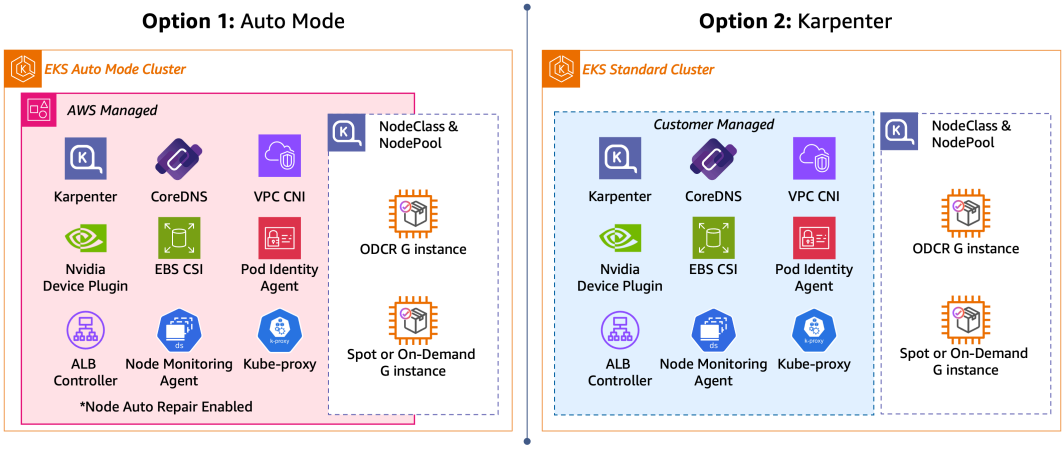
- EKS Auto Mode (`terraform/auto-mode/`) — 除了核心聯網、儲存和負載平衡附加元件之外，EKS Auto Mode 還包含和管理訓練和推論工作負載的下列功能：EKS 節點監控代理程式、自動節點修復、用於快速容器提取的 SOCI 快照器，以及預設 NodeClass 的 GPU 準備程度。NVIDIA 裝置外掛程式包含在 Bottlerocket 加速 AMI 中，EKS Auto Mode 用於已啟用 GPU 的節點。
- 自我管理 Karpenter (`terraform/karpenter/`) ： 在沒有 EKS Auto 模式的 EKS 叢集上，Terraform 程式碼會安裝和設定訓練和推論工作負載所需的元件。這包括聯網附加元件 (VPC

CNI、CoreDNS、kube-proxy)、Karpenter、EKS 節點監控代理程式、NVIDIA 裝置外掛程式，以及用於快速容器提取的 SOCI 快照器。

Important

選擇 EKS Auto Mode 或自我管理 Karpenter，並在整個指南中使用它。切換中串流需要銷毀叢集並重新開始。

EKS 叢集選項：EKS Auto Mode 和自我管理 Karpenter



Grafana 可透過 HTTP 使用預設登入資料公開存取

Grafana ALB Ingress預設為 `var.my_cidr 0.0.0.0/0`，這會透過使用預設管理員憑證的純 HTTP 向公有網際網路公開 Grafana。自動化掃描器會在幾分鐘內探索公有負載平衡器。您必須`var.my_cidr`使用自己的 IP 地址覆寫來限制存取：

```
export MY_CIDR="$(curl -s https://checkip.amazonaws.com)/32"
terraform apply -var "my_cidr=${MY_CIDR}"
```

將 source-IP 允許清單視為最低防護，而非完整防護。也請在第一次登入後變更預設的 Grafana 管理員密碼。如需更強的姿勢，請將 `alb.ingress.kubernetes.io/scheme` 變更為 `internal`（只能從您的 VPC 或連線的 VPN 內存取），然後新增 TLS 憑證。

部署叢集

將 變更為所選路徑的 目錄，初始化 Terraform，然後套用：

這兩個變體預設為 us-east-2區域。若要在不同區域中部署，請在下列步驟中將 -var "region=*region-code*"新增至 terraform apply命令，其中 *region-code* 是 AWS 您要部署的區域。

Terraform 程式碼使用目標區域中所有可用的可用區域，但 use1-az3、 和 除外usw1-az2，cac1-az3因為 Amazon EKS 不支援在這些區域中放置控制平面。

EKS Auto Mode

```
cd terraform/auto-mode
terraform init
terraform apply
```

此命令需要幾分鐘的時間才能完成。

Self-managed Karpenter

```
cd terraform/karpenter
terraform init
terraform apply
```

此命令大約需要 15 分鐘。它使用專用於託管附加元件和 Karpenter 控制器的受管節點群組來建立 EKS 叢集。Terraform 會在啟用 Spot 中斷佇列的情況下安裝 Karpenter，並開啟 NodeRepair和 StaticCapacity功能閘道。它也會安裝 NVIDIA 裝置外掛程式、 AWS Load Balancer控制器和監控堆疊。

檢閱 Terraform 輸出

當套用完成時，Terraform 會列印下列輸出（值根據您的組態而有所不同）：

```
Apply complete! Resources: 74 added, 0 changed, 0 destroyed.

Outputs:

cluster_name         = "ai-eks-docs"
configure_kubectl    = "aws eks update-kubeconfig --region us-east-2 --name ai-eks-docs
 --alias ai-eks-docs"
```

```
configure_model_bucket = "export MODEL_BUCKET=ai-eks-docs-models-20250612abc1"
model_bucket         = "ai-eks-docs-models-20250612abc1"
node_iam_role_name   = "ai-eks-docs-eks-auto-20250612..."
region               = "us-east-2"
```

configure_kubectl 輸出是kubectl指向叢集的ready-to-run命令。model_bucket 輸出包含模型權重的 S3 儲存貯體名稱。node_iam_role_name 輸出會顯示節點使用的 IAM 角色。

設定 kubectl

kubectl 指向新的叢集。configure_kubectl 輸出是ready-to-run命令：

```
eval "$(terraform output -raw configure_kubectl)"
```

驗證叢集

EKS Auto Mode

```
kubectl get pods --all-namespaces
```

預期的輸出結果：

```
NAMESPACE     NAME                                                READY
 STATUS    RESTARTS   AGE
kube-system   metrics-server-5db89f9ffd-h4mlr                     1/1
 Running   0          3m
kube-system   metrics-server-5db89f9ffd-nd748                     1/1
 Running   0          3m
monitoring    kube-prometheus-stack-grafana-ff9b5fd57-dh562       3/3
 Running   0          3m
monitoring    kube-prometheus-stack-kube-state-metrics-5dcbfdf69b-wd6qn  1/1
 Running   0          3m
monitoring    kube-prometheus-stack-operator-548c4f4485-m5wh2     1/1
 Running   0          3m
monitoring    kube-prometheus-stack-prometheus-node-exporter-p2z7l  1/1
 Running   0          3m
monitoring    prometheus-kube-prometheus-stack-prometheus-0       2/2
 Running   0          3m
```

在 EKS Auto 模式中，VPC CNI、kube-proxy 和 CoreDNS 會以受管元件的形式執行，而不會在 中顯示為 Podkube-system。

Self-managed Karpenter

```
kubectl get pods --all-namespaces
```

預期的輸出包括 Karpenter、CoreDNS、kube-proxy、aws-node (VPC CNI)、EKS Pod Identity Agent、EKS 節點監控代理程式和 NVIDIA 裝置外掛程式：

```
NAMESPACE     NAME
 READY   STATUS    RESTARTS   AGE
kube-system   aws-node-bzdcz                                                   2/2
    Running   0          5m
kube-system   aws-node-vkbhb                                                   2/2
    Running   0          5m
kube-system   coredns-7dbb8998cf-9b9wk                                         1/1
    Running   0          5m
kube-system   coredns-7dbb8998cf-pwtjd                                         1/1
    Running   0          5m
kube-system   ebs-csi-controller-748f54b69-8h7jv                               6/6
    Running   0          5m
kube-system   ebs-csi-controller-748f54b69-v2mv8                               6/6
    Running   0          5m
kube-system   eks-node-monitoring-agent-5qw4l                                  1/1
    Running   0          5m
kube-system   eks-node-monitoring-agent-7lrtm                                  1/1
    Running   0          5m
kube-system   eks-pod-identity-agent-ddvlv                                     1/1
    Running   0          5m
kube-system   eks-pod-identity-agent-q4g29                                     1/1
    Running   0          5m
kube-system   karpenter-898ff78-cndbd                                          1/1
    Running   0          5m
kube-system   karpenter-898ff78-hfbhn                                          1/1
    Running   0          5m
kube-system   kube-proxy-gcfnh                                                 1/1
    Running   0          5m
kube-system   kube-proxy-tktcf                                                 1/1
    Running   0          5m
kube-system   metrics-server-5b789db597-cm9qd                                  1/1
    Running   0          5m
kube-system   metrics-server-5b789db597-w57b6                                  1/1
    Running   0          5m
kube-system   nvidia-device-plugin-node-feature-discovery-gc-66cb7f5dc-xvftc    1/1
    Running   0          5m
```

```
kube-system   nvidia-device-plugin-node-feature-discovery-master-854d6b5hnqtp   1/1
    Running   0          5m
monitoring    kube-prometheus-stack-grafana-6797bcb59f-2zlwg                   3/3
    Running   0          5m
monitoring    kube-prometheus-stack-kube-state-metrics-8446bd549c-q6gjc        1/1
    Running   0          5m
monitoring    kube-prometheus-stack-operator-5f499b784-vf5xb                   1/1
    Running   0          5m
monitoring    kube-prometheus-stack-prometheus-node-exporter-4c6wq             1/1
    Running   0          5m
monitoring    kube-prometheus-stack-prometheus-node-exporter-l5t25             1/1
    Running   0          5m
monitoring    prometheus-kube-prometheus-stack-prometheus-0                    2/2
    Running   0          5m
```

確認已安裝 NVIDIA 裝置外掛程式。在佈建具有相符`amiFamily=al2023`標籤的 GPU NodePool 之前，會顯示零個裝置外掛程式 Pod：

```
kubectl get daemonset nvidia-device-plugin -n kube-system
```

預期的輸出結果：

```
NAME                   DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE
 SELECTOR      AGE
nvidia-device-plugin   0         0         0       0            0
 amiFamily=al2023   5m
```

步驟 2：建立動態 GPU NodePool

GPU NodePools 為選擇加入。根據預設， 會`terraform apply`建立沒有 GPU 容量且沒有 GPU 計費的叢集和監控堆疊。若要佈建 GPU 節點，請使用策略名稱傳遞 `nodepools`變數。

啟用 `spot-ondemand`策略，該策略使用 Spot 容量搭配隨需做為備用，以佈建產生大於 4 的 G 系列 GPU 執行個體：

```
terraform apply -var 'nodepools={"spot-ondemand"={}}'
```

此命令會從 `nodepools/spot-ondemand/`目錄套用 NodePool 和 NodeClass 範本。兩個路徑都使用相同的 NodePool API，但它們在 NodeClass 中的 NodePool 參考不同。

EKS Auto Mode

NodePool 參考受管 `default` NodeClass，其已選取 Bottlerocket 加速 AMI、NVIDIA 驅動程式、NVIDIA 裝置外掛程式和 SOCI 平行提取。`spot-ondemand` 策略在此路徑上沒有自己的 NodeClass。

驗證 NodePool：

```
kubectl get nodepools,nodeclasses
```

預期的輸出結果。`gpu-inf` NodePool 會加入內建 `general-purpose` 和 `system` NodePools，而這三個都參考受管 `default` NodeClass：

```
NAME                                    NODECLASS   NODES   READY   AGE
nodepool.karpenter.sh/general-purpose   default     0       True    12m
nodepool.karpenter.sh/gpu-inf           default     0       True    20s
nodepool.karpenter.sh/system            default     1       True    12m

NAME                                  ROLE                        READY   AGE
nodeclass.eks.amazonaws.com/default   ai-eks-docs-eks-auto-...    True    12m
```

Self-managed Karpenter

Terraform 會`gpu-inf`EC2NodeClass搭配 NodePool 套用自訂。將 EKS 最佳化 AL2023 AMI 別名EC2NodeClass接腳，透過`FastImagePull`功能閘道啟用 SOCI，並設定 `instanceStorePolicy: RAID0`將容器化映像快取移至本機 NVMe。

驗證 NodePool 和 EC2NodeClass：

```
kubectl get nodepools,ec2nodeclasses
```

預期的輸出結果。`gpu-inf` 配對會加入 Terraform 為非 GPU 工作負載建立的 `general-purpose` NodePool 和 EC2NodeClass：

```
NAME                                    NODECLASS         NODES   READY   AGE
nodepool.karpenter.sh/general-purpose   general-purpose   1       True    14m
nodepool.karpenter.sh/gpu-inf           gpu-inf           0       True    25s

NAME                                          READY   AGE
```

```
ec2nodeclass.karpenter.k8s.aws/general-purpose   True    14m
ec2nodeclass.karpenter.k8s.aws/gpu-inf           True    25s
```

兩個路徑都會顯示 的0節點，gpu-inf直到排定 GPU 工作負載為止。EKS Auto Mode 和 Karpenter 只會在待定 Pod 需要時啟動節點。

步驟 3：使用範例 Pod 進行測試

使用 Pod 測試 GPU NodePool nvidia-smi 設定：

```
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: nvidia-smi
  labels:
    guide: ai-eks-docs
spec:
  restartPolicy: OnFailure
  tolerations:
    - key: "nvidia.com/gpu"
      operator: "Exists"
      effect: "NoSchedule"
  containers:
    - name: nvidia-smi
      image: public.ecr.aws/amazonlinux/amazonlinux:2023-minimal
      command: ["nvidia-smi"]
      resources:
        limits:
          nvidia.com/gpu: 1
EOF
```

確認 Pod 已排程並成功完成：

```
kubectl get pods nvidia-smi
```

預期的輸出結果：

```
NAME         READY   STATUS      RESTARTS   AGE
nvidia-smi   0/1     Completed   0          67s
```

STATUS: Completed 表示nvidia-smi命令已執行並結束。檢查 Pod 日誌以查看節點偵測到的 GPU：

```
kubectl logs nvidia-smi
```

預期的輸出結果：

```
+-----------------------------------------------------------------------------------------
+
| NVIDIA-SMI 580.159.03             Driver Version: 580.159.03     CUDA Version: 13.0
   |
+-----------------------------------------+------------------------
+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr.
 ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute
 M. |
|                                         |                        |               MIG
 M. |
|=========================================+========================
+======================|
|   0  NVIDIA L4                      On  |   00000000:31:00.0 Off |
 0 |
| N/A   41C    P8             13W /   72W |       0MiB /  23034MiB |      0%
 Default |
|                                         |                        |
 N/A |
+-----------------------------------------+------------------------
+----------------------+
```

輸出會顯示 GPU 模型、驅動程式版本、CUDA 版本和可用的記憶體。在此範例中，Karpenter 佈建了具有 NVIDIA L4 GPU 和 24 GB 記憶體的 G6 執行個體。GPU 模型和記憶體會根據 Karpenter 選取的執行個體類型而有所不同。G5 執行個體具有 NVIDIA A10G GPUs (24 GB)、G6 執行個體具有 NVIDIA L4 GPUs (24 GB)，而 G6e 執行個體具有 NVIDIA L40S GPUs (48 GB)。

若要了解 Karpenter 和 Kubernetes 排程器如何協調以佈建節點並放置 Pod，請檢查 Pod 的生命週期事件：

```
kubectl describe pod nvidia-smi
```

預期的輸出結果：

```
Events:
  Type     Reason            Age   From                   Message
  ----     ------            ----  ----                   -------
  Warning  FailedScheduling  75s   default-scheduler      0/2 nodes are available: 2
 node(s) had untolerated taint(s).
  Normal   Nominated         74s   eks-auto-mode/compute  Pod should schedule on:
 nodeclaim/gpu-inf-z6q75
  Normal   Scheduled         35s   default-scheduler      Successfully assigned
 default/nvidia-smi to i-0eb897a8302551589
  Normal   Pulling           27s   kubelet                spec.containers{nvidia-smi}:
 Pulling image "public.ecr.aws/amazonlinux/amazonlinux:2023-minimal"
  Normal   Pulled            22s   kubelet                spec.containers{nvidia-smi}:
 Successfully pulled image "public.ecr.aws/amazonlinux/amazonlinux:2023-minimal" in
 5.625s (5.626s including waiting). Image size: 37440620 bytes.
  Normal   Created           22s   kubelet                spec.containers{nvidia-smi}:
 Container created
  Normal   Started           21s   kubelet                spec.containers{nvidia-smi}:
 Container started
```

這些事件會顯示 Pod 排程序列：Pod 一開始無法排程，因為 GPU 節點不存在 (`FailedScheduling`)，Karpenter 會指定新的 NodeClaim (`Nominated`)，排程器會在節點就緒時指派 Pod (`Scheduled`)，然後提取並啟動容器映像。EKS Auto Mode 預設會在 G、P 和 Trn 執行個體上安裝和設定 SOCI （可尋求 OCI) 平行提取，而自我管理 Karpenter 路徑會透過`FastImagePull`功能閘道明確設定。

> **Note**
>
> 在自我管理的 Karpenter 叢集上，`Nominated`事件會顯示 `karpenter/compute`而非 `eks-auto-mode/compute`。

NodeClaim 是 Karpenter 建立來佈建特定節點的請求。它會顯示執行個體類型、容量類型、AZ，以及節點是否已就緒：

```
kubectl get nodeclaims
```

預期的輸出結果：

```
NAME            TYPE        CAPACITY   ZONE         NODE                   READY   AGE
gpu-inf-z6q75   g6.xlarge   spot       us-east-2a   i-0eb897a8302551589   True    5m
```

執行個體類型和 AZ 會有所不同。產生大於 4 的任何 G 系列執行個體都符合資格。

> **Tip**
>
> 如果沒有出現節點，請檢查容量不足錯誤：
>
> ```
> kubectl get events | grep InsufficientCapacityError
> ```
>
> Karpenter 快取無法使用的方案 3 分鐘。擴大 NodePool 中允許的執行個體類型和可用AZs會增加登陸容量的機會。

> **Note**
>
> Karpenter 啟動的 Spot 執行個體不會出現在 EC2 Spot 請求主控台中。Karpenter 搭配 使用 EC2 `CreateFleet` APItype: instant。執行個體會以spot生命週期顯示在 EC2 執行個體主控台中。

步驟 4：將預留容量新增至 NodePool （選用）

雖然步驟 2 的 GPU NodePool 動態佈建 Spot 或隨需執行個體，但某些使用案例需要保證容量。您可以建立隨需容量保留 (ODCR)，以確保 GPU 容量在需要時可用。

使用 Terraform 時，單一命令會建立 ODCR，這是依標籤參考保留的自訂 NodeClass，並更新 NodePool 以包含 `reserved`做為容量類型。Terraform 使用 標記 ODCR，`nodepool=reserved-spot-ondemand`NodeClass 會依該標籤選取它。

> **Warning**
>
> 下列命令會建立 ODCR，該 會立即計費並繼續計費，直到您使用 `terraform destroy`或清除指令碼將其銷毀，無論節點是否在其上執行。

使用預設值 (`g6e.4xlarge`、1 個執行個體、第一個叢集 AZ)：

```
terraform apply -var 'nodepools={"reserved-spot-ondemand"={reservation={}}}'
```

選擇執行個體類型、計數和可用區域：

```
terraform apply -var 'nodepools={"reserved-spot-
ondemand"={reservation={instance_type="g6e.2xlarge",instance_count=1,az="us-
east-2a"}}}'
```

reservation 物件支援下列欄位：

- instance_type — 要保留的 GPU 執行個體類型。預設：g6e.4xlarge。
- instance_count — 要保留的執行個體數量。預設：1。
- az — 保留的可用區域。預設： "" （使用第一個叢集 AZ)。

> **Important**
>
> spot-ondemand 和 reserved-spot-ondemand策略是互斥的。您最多可以在 nodepools變數中啟用一個 。如果您之前在步驟 2 spot-ondemand中使用過 ，reserved-spot-ondemand命令會取代它，因為兩者都會管理相同的 gpu-inf NodePool。

如果您收到InsufficientInstanceCapacity錯誤，則無法在指定的 AZ 中完成保留。取消 Terraform 操作 (Ctrl+C)，然後使用不同的az值重新執行：

```
terraform apply -var 'nodepools={"reserved-spot-
ondemand"={reservation={instance_type="g6e.4xlarge",az="us-east-2b"}}}'
```

套用後，Terraform 會將 NodePool 更新為在容量類型需求on-demand中包含 spot、reserved和 。Karpenter reserved將 視為最具成本效益的選項，並優先啟動。一旦保留已滿，就會回到 Spot 或隨需。

在 EKS Auto Mode 路徑上，Terraform 會建立自訂 gpu-inf NodeClass （因為 Bundled default NodeClass 為唯讀），透過 標籤參考 ODCRcapacityReservationSelectorTerms。
在自我管理的 Karpenter 路徑上，Terraform 會重新套用 gpu-inf EC2NodeClass，並capacityReservationSelectorTerms新增和更新 NodePool 以包含 reserved。

驗證是否已建立 ODCR：

```
aws ec2 describe-capacity-reservations \
  --filters "Name=state,Values=active" "Name=tag:nodepool,Values=reserved-spot-
ondemand" \
```

```
  --query 'CapacityReservations[0].
{Id:CapacityReservationId,State:State,InstanceType:InstanceType,AvailableCount:AvailableInstanc
 \
  --output table \
  --region $(terraform output -raw region)
```

驗證 NodeClass 參考 ODCR：

EKS Auto Mode

```
kubectl get nodeclasses gpu-inf -o yaml | grep 'id: cr-.*'
```

預期的輸出結果：

```
        id: cr-xxxxxxxxxxxxxxxxx
```

Self-managed Karpenter

```
kubectl get ec2nodeclasses gpu-inf -o yaml | grep 'id: cr-.*'
```

預期的輸出結果：

```
        id: cr-xxxxxxxxxxxxxxxxx
```

驗證 NodePool 已就緒：

```
kubectl get nodepools gpu-inf
```

預期的輸出結果：

```
NAME      NODECLASS   NODES   READY   AGE
gpu-inf   gpu-inf     0       True    30s
```

確認已先使用預留容量，搭配 Spot 備用

套用變更後，請驗證 Karpenter 是否排定保留容量的優先順序，並回到 Spot 或隨需。部署 2 個複本部署，每個 Pod 請求 1 個 GPU。ODCR 適用於 1 個執行個體 (1 個 GPU)，因此第一個 Pod 會觸發

Karpenter 啟動預留節點。第二個 Pod 無法容納在預留節點上，並觸發 Karpenter 從 Spot 或隨需容量啟動另一個節點。

```
cat << 'EOF' | kubectl apply -f -
apiVersion: apps/v1
kind: Deployment
metadata:
  name: gpu-overflow-test
  labels:
    guide: ai-eks-docs
spec:
  replicas: 2
  selector:
    matchLabels:
      app: gpu-overflow-test
  template:
    metadata:
      labels:
        app: gpu-overflow-test
        guide: ai-eks-docs
    spec:
      tolerations:
        - key: nvidia.com/gpu
          operator: Exists
          effect: NoSchedule
      containers:
        - name: nvidia-smi
          image: public.ecr.aws/amazonlinux/amazonlinux:2023-minimal
          command: ["sh", "-c", "nvidia-smi && sleep infinity"]
          resources:
            limits:
              nvidia.com/gpu: 1
EOF
```

與執行和結束步驟 3 `nvidia-smi`的測試 Pod 不同，此部署會讓 Pod 保持執行 (`sleep infinity`)，以便保留 GPU 並防止節點合併。

驗證在不同節點上排程的 Pod：

```
kubectl get pods -l app=gpu-overflow-test -o wide
```

預期的輸出結果：

```
NAME                                  READY   STATUS    RESTARTS   AGE     IP
 NODE                   NOMINATED NODE   READINESS GATES
gpu-overflow-test-55d55ff5b9-dvg52   1/1     Running   0          4m42s   10.0.75.210
 i-08741a36089ff2088   <none>           <none>
gpu-overflow-test-55d55ff5b9-hw4m9   1/1     Running   0          4m43s   10.0.82.49
 i-0f50cdbacb2017202   <none>           <none>
```

檢查 NodeClaims 以查看容量類型：

```
kubectl get nodeclaims
```

預期的輸出結果：

```
NAME            TYPE          CAPACITY   ZONE         NODE                   READY   AGE
gpu-inf-vw99m   g6e.4xlarge   reserved   us-east-2c   i-0f50cdbacb2017202   True    6m
gpu-inf-s65s6   g6.xlarge     spot       us-east-2b   i-08741a36089ff2088   True
 5m59s
```

預留節點會先啟動，然後在保留已滿時啟動 Spot 或隨需節點。

清除測試部署：

```
kubectl delete deployment gpu-overflow-test
```

監控

在步驟 1 `terraform apply`中，Terraform 已在 期間佈建完整的監控堆疊。堆疊包含 Amazon Managed Service for Prometheus (AMP) 工作區、IAM 政策和 EKS Pod Identity Associations for Prometheus remote-write and Grafana 查詢存取、kube-prometheus-stack Helm Chart (Prometheus、Grafana、kube-state-metrics、 node-exporter)，以及適用於 GPU 指標的 NVIDIA DCGM Exporter。

本節涵蓋已部署監控元件的驗證。

驗證監控 Pod

等待所有監控 Pod 就緒：

```
kubectl wait --for=condition=Ready pod --all -n monitoring --timeout=300s
```

```
kubectl get pods -n monitoring
```

預期的輸出結果：

```
NAME                                                     READY   STATUS    RESTARTS
 AGE
kube-prometheus-stack-grafana-7c58f54f77-rftrj           3/3     Running   0
 5m
kube-prometheus-stack-kube-state-metrics-d68dcbc84-5smxq 1/1     Running   0
 5m
kube-prometheus-stack-operator-5895df479f-ttm47          1/1     Running   0
 5m
kube-prometheus-stack-prometheus-node-exporter-t9q7s     1/1     Running   0
 5m
kube-prometheus-stack-prometheus-node-exporter-x6vfb     1/1     Running   0
 5m
prometheus-kube-prometheus-stack-prometheus-0            2/2     Running   0
 5m
```

存取 Grafana

Grafana 透過 Internet-facing AWS Application Load Balancer (ALB) 公開，僅限於您在 中設定的 CIDR`var.my_cidr`。列印負載平衡器 URL （ 允許 ALB 佈建一兩分鐘 ）：

```
echo "http://$(kubectl get ingress kube-prometheus-stack-grafana -n monitoring -o
 jsonpath='{.status.loadBalancer.ingress[0].hostname}')"
```

在瀏覽器中開啟 URL。從`admin`下列命令使用使用者名稱和密碼登入：

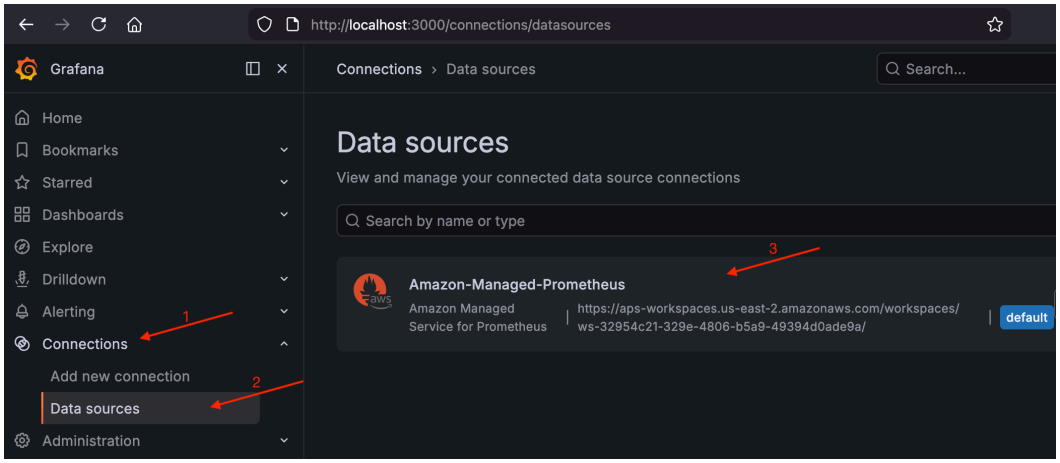
```
kubectl --namespace monitoring get secrets kube-prometheus-stack-grafana -o
 jsonpath="{.data.admin-password}" | base64 -d ; echo
```

驗證指標管道

若要驗證指標管道是否端對端運作：

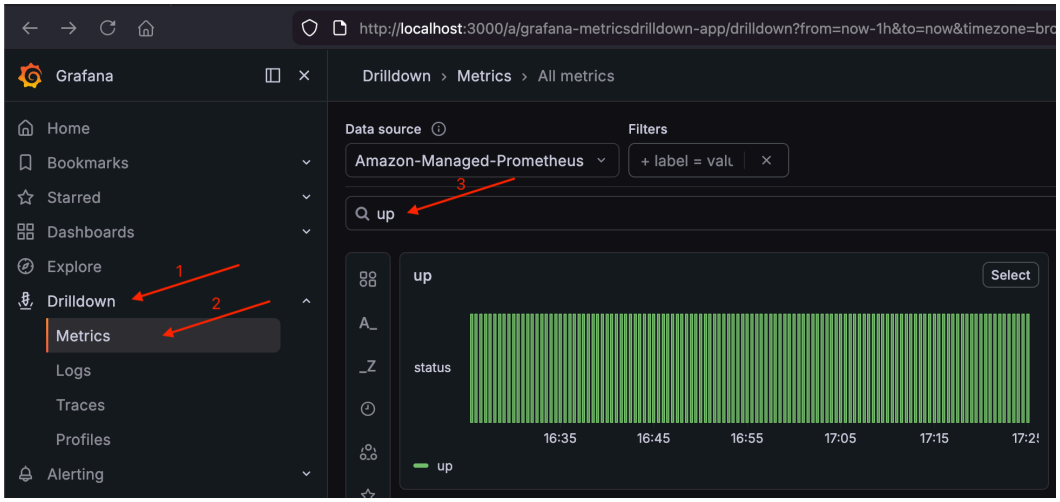
1. 導覽至連線 > 資料來源，並確認 Amazon-Managed-Prometheus 列為預設資料來源。

   驗證 Grafana 中的 AMP 資料來源



2. 導覽至向下切入 > 指標並搜尋指標。 up您應該會看到叢集湊集目標的結果。

   驗證 **up** Grafana 中的指標



如果 up顯示結果，則管道 （叢集 → Prometheus → AMP → Grafana) 正在運作。

驗證 DCGM GPU 指標

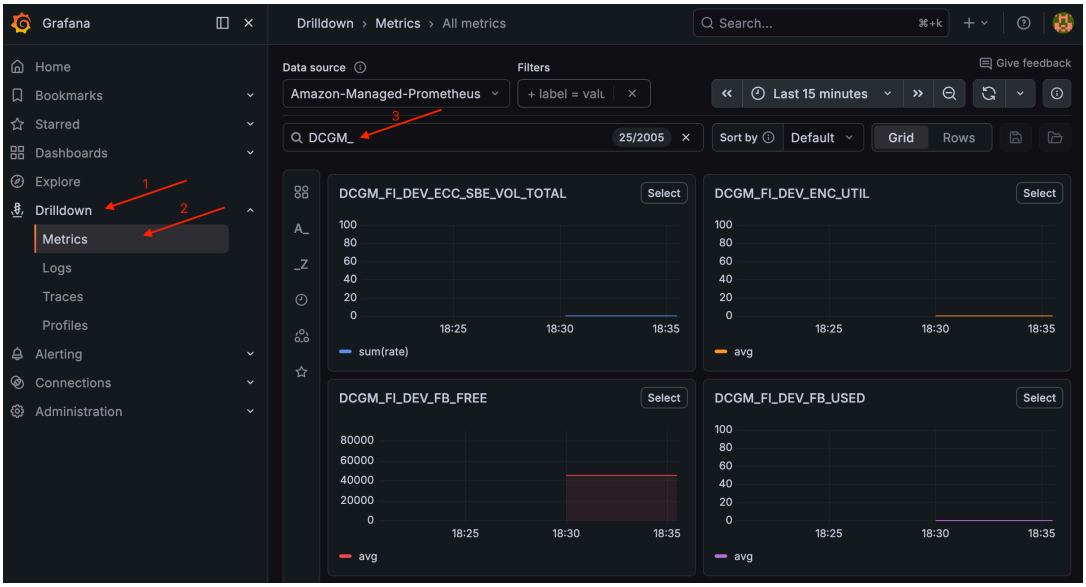
DCGM Exporter DaemonSet 會在 GPU 節點上執行，並報告 GPU 使用率、記憶體、溫度、耗電量、NVLink 頻寬和張量活動指標。

驗證 DCGM 匯出工具 DaemonSet：

```
kubectl get daemonset dcgm-exporter -n monitoring
```

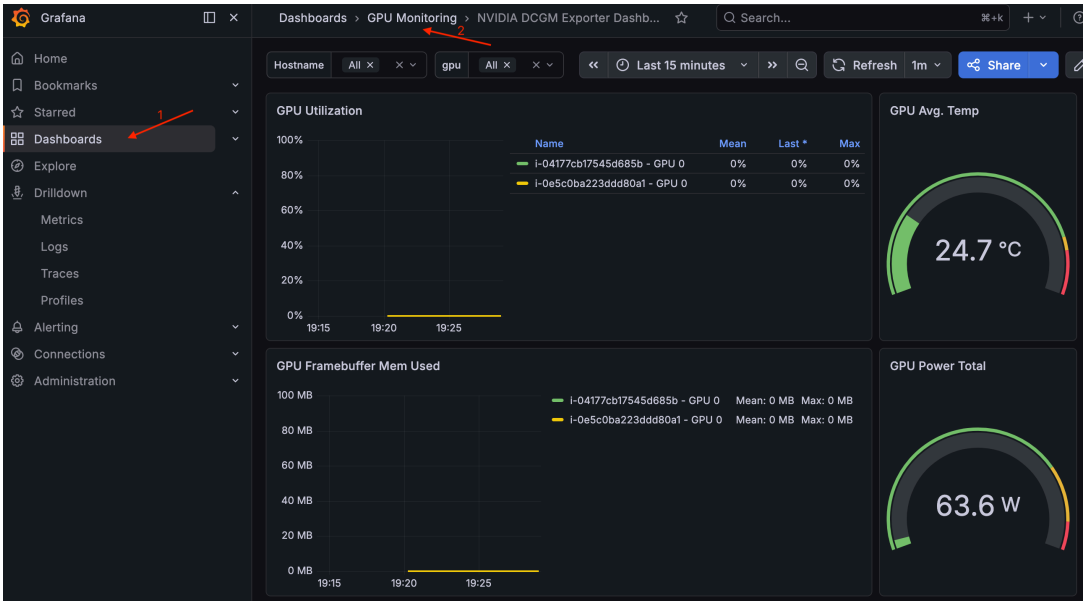
一旦 GPU 節點執行 （從步驟 2 或步驟 4)，您應該會看到一或多個就緒的 Pod。若要驗證 DCGM 指標，請導覽至 Grafana 中的向下切入 > 指標，並搜尋 DCGM_。

在 Grafana 中驗證 DCGM 指標



若要檢視儀表板，請導覽至儀表板 > GPU 監控 > NVIDIA DCGM 匯出工具儀表板。

Grafana 中的 NVIDIA DCGM 匯出工具儀表板



模型權重 S3 儲存貯體

Terraform 已建立用於存放模型權重的 Amazon S3 儲存貯體、`default`命名空間中的 `model-storage-sa` ServiceAccount、範圍為儲存貯體的 IAM 政策，以及連結它們的 EKS Pod Identity Association。設定的工作負載 Pod `serviceAccountName: model-storage-sa`可以讀取和寫入儲存貯體。

驗證儲存貯體

從 Terraform 輸出擷取儲存貯體名稱：

```
MODEL_BUCKET=$(terraform output -raw model_bucket)
echo ${MODEL_BUCKET}
```

驗證儲存貯體是否存在：

```
aws s3api head-bucket --bucket ${MODEL_BUCKET}
```

預期的輸出結果：

```
{
    "BucketArn": "arn:aws:s3:::ai-eks-docs-models-20250612abc1",
    "BucketRegion": "us-east-2",
    "AccessPointAlias": false
}
```

從 Pod 驗證 S3 存取

使用 `model-storage-sa` ServiceAccount 使用 AWS CLI 映像執行一次性 Pod，以確認 EKS Pod 身分已連線且 S3 存取正常運作：

```
cat << EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: s3-test
  labels:
    guide: ai-eks-docs
spec:
  serviceAccountName: model-storage-sa
  containers:
    - name: aws-cli
      image: public.ecr.aws/aws-cli/aws-cli:2.27.0
      command:
        - sh
        - -c
        - |
          echo "=== Caller Identity ==="
```

```
          aws sts get-caller-identity
          echo ""
          echo "=== S3 Write Test ==="
          echo "pod identity works" | aws s3 cp - s3://${MODEL_BUCKET}/test.txt
          echo ""
          echo "=== S3 List Test ==="
          aws s3 ls s3://${MODEL_BUCKET}/
          echo ""
          echo "=== S3 Delete Test ==="
          aws s3 rm s3://${MODEL_BUCKET}/test.txt
  restartPolicy: Never
EOF
```

等待 Pod 完成並檢查日誌：

```
kubectl wait --for=jsonpath='{.status.phase}'=Succeeded pod/s3-test --timeout=300s
kubectl logs s3-test
```

預期的輸出結果：

```
=== Caller Identity ===
{
    "UserId": "AROA...:eks-ai-eks-docs-model-s-...",
    "Account": "123456789012",
    "Arn": "arn:aws:sts::123456789012:assumed-role/ai-eks-docs-models-.../eks-ai-eks-
docs-..."
}

=== S3 Write Test ===
upload: - to s3://ai-eks-docs-models-20250612abc1/test.txt

=== S3 List Test ===
2026-07-15 12:00:00         19 test.txt

=== S3 Delete Test ===
delete: s3://ai-eks-docs-models-20250612abc1/test.txt
```

發起人身分確認 Pod 透過 EKS Pod 身分擔任模型儲存角色。S3 命令會確認讀取和寫入存取。

清除測試 Pod：

```
kubectl delete pod s3-test
```

後續步驟

叢集準備就緒後，您可以繼續載入並提供模型，以部署大型語言模型並與推論端點互動。

清除

> **Tip**
>
> 如果您打算繼續本指南的下一節，請略過完整清除。只有在完成時才會執行它。

刪除測試工作負載，讓 Pod 不會保留 GPU 節點：

```
kubectl delete pod nvidia-smi --ignore-not-found
kubectl delete deployment gpu-overflow-test --ignore-not-found
```

取消容量保留而不銷毀叢集

如果您只想釋放 ODCR 並返回 Spot 和隨需容量，請將`nodepools`變數切換回`spot-ondemand`策略：

```
terraform apply -var 'nodepools={"spot-ondemand"={}}'
```

這`reserved`會從 NodePool 容量類型需求中刪除，並銷毀 ODCR，並將叢集、監控堆疊和 S3 儲存貯體留在原處。

> **Important**
>
> 取消保留不會終止已在其中執行的執行個體。這些執行個體會持續以標準隨需費率執行，直到終止為止。首先刪除 GPU 工作負載，如上所示，因此保留的節點會在釋出保留之前耗盡。

銷毀叢集和所有 Terraform 受管資源

在銷毀之前耗盡 Karpenter 受管節點，因此沒有任何傳輸中節點生命週期會封鎖銷毀。刪除任何會阻止耗盡的 PodDisruptionBudgets，然後刪除 NodeClaims：

```
kubectl delete pdb -A --all --ignore-not-found
```

```
kubectl delete nodeclaim --all --wait=true --timeout=900s
```

然後銷毀建立的所有 Terraform，包括 EKS 叢集、VPC、監控堆疊、NodePools 和 NodeClasses、S3 模型儲存貯體，以及任何 ODCR：

```
terraform destroy
```

> **Warning**
>
> 模型權重 S3 儲存貯體是使用 建立`force_destroy = true`的，因此 會`terraform destroy`刪除儲存貯體以及您上傳到儲存貯體的任何模型權重。先將您想要保留的任何內容複製到另一個位置。

> **Note**
>
> 儲存庫也會提供執行上述耗盡和銷毀步驟的`scripts/cleanup.sh`協助程式，然後掃描任何以叢集名稱標記的孤立 EBS 磁碟區。從您套用的`terraform/<mode>/`目錄中執行它，然後傳遞 `--auto-approve`以略過 Terraform 確認提示。

確認保留已移除

確認叢集沒有剩餘的作用中容量保留：

```
aws ec2 describe-capacity-reservations \
  --filters "Name=state,Values=active" "Name=tag:nodepool,Values=reserved-spot-
ondemand" \
  --query 'CapacityReservations[].CapacityReservationId' \
  --output text
```

空白結果表示沒有作用中的保留，也不會產生其他費用。

在 Amazon EKS 上執行 AI/ML 推論工作負載

> **Tip**
>
> 註冊即將舉行的 Amazon EKS AI/ML 研討會。

推論是執行訓練 AI 模型的程序，以從輸入資料產生預測或輸出。這包括提供產生文字或程式碼的大型語言模型 (LLMs)、產生影像的擴散模型、用於語音合成和轉錄的語音模型，以及用於即時分析或生成的影片模型。Amazon EKS 提供可擴展的可攜式解決方案，可讓您使用 NVIDIA GPUs 或 AWS Trainium 加速器部署這些推論工作負載、動態擴展，以及與更廣泛的 Kubernetes 和工具和服務的 AWS 領域整合。

為什麼使用 Amazon EKS 進行推論

Amazon EKS 結合了 Kubernetes 協同運作與 AWS 基礎設施，以解決大規模執行推論的主要挑戰：

- 動態 GPU 擴展 — Karpenter 根據 Pod 資源請求隨需佈建適當大小的 GPU 執行個體，在閒置時從零擴展，並在流量增加時增加容量。這可避免過度佈建昂貴的 GPU 資源。
- 快速冷啟動 — SOCI （可擴展的 OCI) 平行提取下載並同時解壓縮大型容器映像層，將映像提取時間從幾分鐘縮短到幾秒鐘。結合直接從 Amazon S3 到 GPU 記憶體的模型串流，Pod 可以在兩分鐘內開始提供服務。
- 自動 GPU 故障復原 — EKS 節點監控代理程式會偵測 GPU 硬體故障，並觸發自動節點替換，將停機時間降至最低，無需手動介入。
- 成本最佳化 — 具有隨需備用的 Spot 執行個體、適當大小的執行個體選擇，以及scale-to-zero的行為，可降低 GPU 成本。預留容量 ODCRs) 可為穩定狀態工作負載提供額外節省。
- 開放標準和可攜性 — 使用 vLLM 或 SGLang 等開放原始碼模型伺服器，在標準 Kubernetes APIs （部署、服務、HPAs) 上執行推論工作負載，提供跨環境的可攜性。
- 整合式監控：來自 vLLM 或 SGLang 和 NVIDIA DCGM Exporter 的 Prometheus 指標可透過 Grafana 儀表板提供請求延遲、字符輸送量、GPU 使用率和記憶體用量的可見性。

詞彙表

本節使用下列術語：

- 推論：執行訓練模型以從輸入資料產生輸出 （文字、內嵌、分類） 的程序。
- 模型伺服器 — 容器化服務，可將模型載入記憶體、接收推論請求，以及傳回預測。範例包括 vLLM、SGLang、Triton 推論伺服器和文字產生推論 (TGI)。
- 模型權重 — 訓練模型的學習參數，儲存為模型伺服器載入 GPU 記憶體的檔案 （通常為 SafeTensors 或 GGUF 格式）。
- 加速器 — 特殊硬體，例如 NVIDIA GPUs 或 AWS Trainium/Inferentia 晶片，可加速推論所需的矩陣操作。

- Tensor 平行處理 — 在同一節點上跨多個 GPUs 分割模型，為超過單一 GPU 記憶體的模型提供服務。
- KV 快取 — 記憶體緩衝區，可在文字產生期間存放先前計算的鍵值對，避免對每個新字符進行備援運算。

Amazon EKS 上的推論如何運作

在高層級上，在 Amazon EKS 上部署推論工作負載需要下列步驟：

步驟	說明
設定叢集	使用啟用 GPU 的節點、監控和儲存來建立 EKS 叢集。請參閱 AI/ML on EKS 文件中的叢集設定。
存放模型權重	從模型登錄檔（例如 Hugging Face) 下載模型權重，並將其存放在 Amazon S3 中，以便在冷啟動或擴展事件期間快速載入模型。
部署模型伺服器	建立在 GPU 節點上執行模型伺服器的 Kubernetes 部署（例如 vLLM)。模型伺服器會將權重從 S3 串流到 GPU 記憶體，並公開與 OpenAI 相容的 API。
公開推論端點	建立 Kubernetes 服務以提供穩定的網路端點。使用連接埠轉送進行測試，或使用負載平衡器，例如生產流量的 AWS Application Load Balancer (ALB)。
監控和擴展	使用 Prometheus 指標（請求率、字符輸送量、延遲、KV 快取使用率）來監控效能並設定自動擴展。

推論效能指標

了解推論效能需要跨延遲和輸送量追蹤指標：

延遲指標

- 到第一個權杖的時間 (TTFT) — 從請求到達到第一個產生的權杖的時間。對互動式應用程式至關重要。
- 每個輸出字符的時間 (TPOT) — 在第一個字符之後產生每個後續字符的平均時間。

- End-to-end請求延遲 — 從提交請求到完成完整回應的總時間。

輸送量指標

- 每秒請求數 — 所有複本每秒提供的推論請求總數。
- 每秒輸出字符 — 字符產生速率，以總輸出字符除以經過時間來衡量。
- GPU 使用率 — 主動用於推論的 GPU 運算週期百分比。
- KV 快取使用率 — 配置使用中 KV 快取記憶體的百分比，指出伺服器與容量的接近程度。

縮短冷啟動時間

大型推論容器 (8-15 GB) 和模型權重 (10-100+ GB) 可能會導致 Pod 啟動緩慢。下列技術可將冷啟動延遲降至最低：

- SOCI 平行提取 — 同時下載和解壓縮容器映像層，而不是循序執行。根據預設，GPU 執行個體的 EKS 自動模式已啟用。
- 從 S3 的模型串流 — Run：ai Model Streamer 等工具會將模型權重直接從 Amazon S3 串流到 GPU 記憶體，繞過本機磁碟並將載入時間從幾分鐘縮短到幾秒鐘。
- 將映像儲存在 Amazon ECR 中 — 透過 VPC 端點從區域 ECR 儲存庫提取可避免大型容器映像的網際網路延遲。
- 執行個體存放區快取 — 具有本機 NVMe 磁碟的 G 系列執行個體可以快取容器層和模型成品，加速對相同節點的後續提取。

您將部署的內容

Load & Serve 模型演練會引導您部署end-to-end推論應用程式：

1. 下載模型權重 — Kubernetes 任務會從 Hugging Face 下載 Ministral-3-8B-Instruct 模型，並將其上傳至 S3 儲存貯體。
2. 部署 vLLM — 部署使用 Run：ai Model Streamer 執行 vLLM，將權重直接從 S3 串流到 GPU 記憶體，提供與 OpenAI 相容的 API。
3. 使用 Grafana 進行監控 — ServiceMonitor 將 vLLM Prometheus 指標連接到即時儀表板的監控堆疊。
4. 部署聊天前端 — Open WebUI 提供連線至 vLLM 端點的瀏覽器型聊天介面。

演練使用 the section called “設定叢集”區段中的叢集基礎設施，並使用 EKS Auto Mode 和自我管理 Karpenter 路徑。

在 Amazon EKS 上載入並提供模型

> **Tip**
>
> 註冊即將舉行的 Amazon EKS AI/ML 研討會。

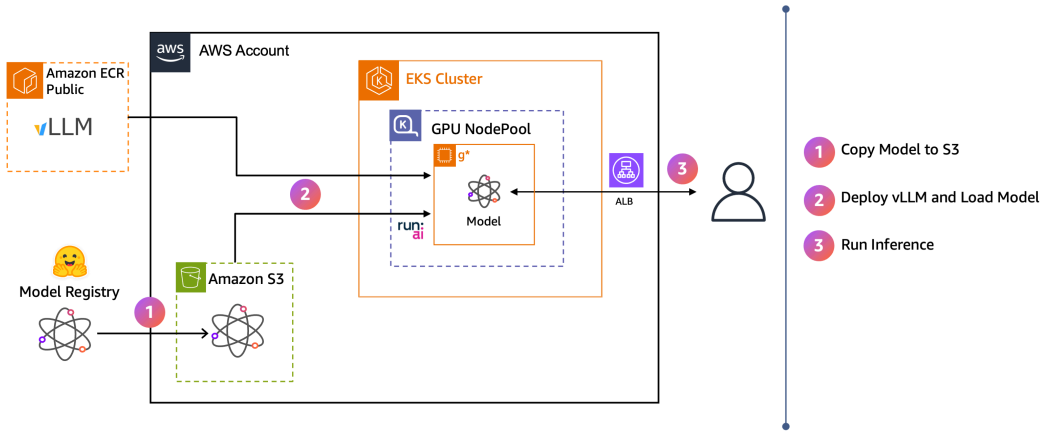
本節中的步驟會在 Amazon EKS 上部署大型語言模型 (LLM)、提供 vLLM，以及與推論端點互動。

演練使用以下工具：

- vLLM — 針對 LLM 服務和 GPU 記憶體管理進行最佳化的高輸送量推論引擎。
- Run：ai 模型串流器 — 將模型權重直接從 Amazon S3 串流到 GPU 記憶體，將載入時間從幾分鐘縮短到幾秒鐘。
- 開啟 WebUI — 連接至 vLLM OpenAI 相容 API 的自我託管聊天前端。

本節使用 Ministral-3-8B-Instruct-2512 模型，但您可以部署 vLLM 支援的任何 AI 模型。如需支援的模型清單，請參閱 vLLM 文件中的支援的模型。

> **Important**
>
> 使用您在 the section called “設定叢集”區段中建立的叢集。此演練中的指示適用於 EKS Auto Mode 和自我管理 Karpenter。



架構圖顯示end-to-end流程：

1. 模型權重會從 Hugging Face 下載到 Amazon S3。
2. vLLM 使用 Run：ai Model Streamer，將模型直接從 S3 串流到 GPU 記憶體。
3. 使用者將推論請求傳送至 vLLM 端點。

當您完成這些步驟時，您會有一個 vLLM 推論端點，可用來透過聊天前端應用程式與 Ministral 模型互動。如需在 Amazon EKS 上最佳化模型載入時間的詳細資訊，請參閱 the section called “加速模型載入”。

先決條件

完成叢集設定區段中的步驟。

如果您開啟新的終端機，請透過 CLI 區段設定您在叢集設定中使用的叢集名稱和區域：

```
export CLUSTER_NAME=ai-eks-docs
export AWS_REGION=us-east-2
```

查詢您在模型權重 S3 儲存貯體步驟中建立的模型權重儲存貯體：

```
MODEL_BUCKET=$(aws s3api list-buckets \
  --query "Buckets[?starts_with(Name, '${CLUSTER_NAME}-models-')].Name | [0]" \
  --output text)
echo "Model bucket: ${MODEL_BUCKET}"
```

步驟 1：從 Hugging Face 下載模型

在此步驟中，您會部署從 Hugging Face 下載模型的 Kubernetes 任務，並將其上傳至您在先決條件區段中建立的 S3 儲存貯體。

若要下載模型，請套用下列任務資訊清單：

Example模型下載任務資訊清單

```
cat << EOF | kubectl apply -f -
apiVersion: batch/v1
kind: Job
metadata:
  name: model-download
  namespace: default
```

```
  labels:
    guide: ai-eks-docs
spec:
  backoffLimit: 10
  activeDeadlineSeconds: 3600
  ttlSecondsAfterFinished: 86400
  template:
    spec:
      restartPolicy: Never
      serviceAccountName: model-storage-sa
      containers:
      - name: downloader
        image: python:3.11-slim
        command: ["/bin/bash", "-c"]
        args:
        - |
          set -e
          pip install -q huggingface_hub boto3
          echo "Downloading Ministral-3-8B-Instruct-2512 from Hugging Face..."
          python3 -c "from huggingface_hub import snapshot_download;
 snapshot_download('mistralai/Ministral-3-8B-Instruct-2512', local_dir='/tmp/
mistral', allow_patterns=['*.json', '*.txt', '*.md', 'consolidated.safetensors'],
 ignore_patterns=['model-*.safetensors', 'model.safetensors.index.json'])"
          echo "Uploading to S3 bucket: \${MODEL_BUCKET}"
          python3 << 'PYTHON'
          import boto3
          import os
          from pathlib import Path

          s3 = boto3.client('s3')
          bucket = os.environ.get('MODEL_BUCKET')
          local_dir = Path("/tmp/mistral")

          for file_path in local_dir.rglob("*"):
              if file_path.is_file():
                  if '.cache' in file_path.parts:
                      continue
                  s3_key = f"Ministral-3-8B-Instruct-2512/
{file_path.relative_to(local_dir)}"
                  print(f"Uploading {file_path.name}...")
                  s3.upload_file(str(file_path), bucket, s3_key)
          print("Upload complete!")
          PYTHON
        env:
```

```
        - name: MODEL_BUCKET
          value: "${MODEL_BUCKET}"
        - name: HF_HUB_DISABLE_XET
          value: "1"
        resources:
          requests:
            memory: "2Gi"
            cpu: "1"
          limits:
            memory: "4Gi"
            cpu: "2"
EOF
```

等待任務完成。模型權重 (consolidated.safetensors) 約為 10.4 GB，此步驟通常需要 3-5 分鐘。

```
kubectl wait --for=condition=complete job/model-download --timeout=600s
```

預期的輸出結果：

```
job.batch/model-download condition met
```

確認模型權重已上傳至 S3：

```
aws s3 ls s3://$(kubectl get job model-download -o
 jsonpath='{.spec.template.spec.containers[0].env[?(@.name=="MODEL_BUCKET")].value}')/
Ministral-3-8B-Instruct-2512/ --recursive
```

預期的輸出結果：

```
2026-05-18 10:29:53      20311 Ministral-3-8B-Instruct-2512/README.md
2026-05-18 10:29:53       2361 Ministral-3-8B-Instruct-2512/SYSTEM_PROMPT.txt
2026-05-18 10:29:53       1903 Ministral-3-8B-Instruct-2512/config.json
2026-05-18 10:29:54 10420633176 Ministral-3-8B-Instruct-2512/consolidated.safetensors
2026-05-18 10:29:53        131 Ministral-3-8B-Instruct-2512/generation_config.json
2026-05-18 10:29:53       1185 Ministral-3-8B-Instruct-2512/params.json
2026-05-18 10:29:53        976 Ministral-3-8B-Instruct-2512/processor_config.json
2026-05-18 10:29:53   16753777 Ministral-3-8B-Instruct-2512/tekken.json
2026-05-18 10:29:53   17077402 Ministral-3-8B-Instruct-2512/tokenizer.json
2026-05-18 10:29:53      21168 Ministral-3-8B-Instruct-2512/tokenizer_config.json
```

consolidated.safetensors 檔案包含模型權重 （ 約 10.4 GB)。其餘檔案是 vLLM 為模型提供服務所需的組態和權杖化器檔案。

步驟 2：部署推論容器

在本節中，您將 vLLM 部署為 Kubernetes 部署，以提供您上傳至 Amazon S3 的模型。

本節使用AWS 深度學習容器 (DLCs)，這是預先安裝深度學習架構的 Docker 映像，並針對 AWS 基礎設施的效能進行最佳化。DLCs包括安全修補程式、經過驗證的架構版本和最佳化的 GPU 驅動程式組態。

此部署使用下列 AWS DLC 搭配 SOCI 支援 vLLM 0.21.0：`public.ecr.aws/deep-learning-containers/vllm:0.21.0-gpu-py312-cu130-ubuntu22.04-ec2-v1.0-soci`。

映像標籤指出 vLLM 0.21.0 搭配 GPU 支援、Python 3.12、CUDA 13.0、Ubuntu 22.04、針對 EC2-based工作負載最佳化，以及啟用 SOCI 的容器啟動速度更快。

此資訊清單會建立在 GPU 節點上執行 vLLM 的部署，並使用 Run：ai Model Streamer 將模型直接從 S3 串流至 GPU 記憶體。資訊清單也會建立 ClusterIP 服務，公開連接埠 8000 上的 vLLM 端點，以進行叢集內存取。

如需在 Amazon EKS 上最佳化模型載入時間的詳細資訊，請參閱 the section called “加速模型載入”。下列範例使用 `--enforce-eager`來加速入門案例中的載入時間。我們建議您使用其他技術來加速模型載入時間，如 中所述the section called “--enforce-eager 權衡”。

套用資訊清單：

Example vLLM 部署和服務 YAML

```
cat << EOF | kubectl apply -f -
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vllm-inference-app
  labels:
    guide: ai-eks-docs
spec:
  replicas: 1
  selector:
    matchLabels:
      app: vllm-inference-app
  template:
    metadata:
```

```
      labels:
        app: vllm-inference-app
        guide: ai-eks-docs
    spec:
      serviceAccountName: model-storage-sa
      tolerations:
      - key: nvidia.com/gpu
        operator: Exists
        effect: NoSchedule
      nodeSelector:
        karpenter.sh/nodepool: gpu-inf
      containers:
      - name: vllm-inference
        image: public.ecr.aws/deep-learning-containers/vllm:0.21.0-gpu-py312-cu130-
ubuntu22.04-ec2-v1.0-soci
        ports:
        - containerPort: 8000
        args:
        - "--model=s3://${MODEL_BUCKET}/Ministral-3-8B-Instruct-2512/"
        - "--host=0.0.0.0"
        - "--port=8000"
        - "--tensor-parallel-size=1"
        - "--gpu-memory-utilization=0.9"
        - "--max-model-len=8192"
        - "--max-num-seqs=128"
        - "--load-format=runai_streamer"
        - "--enforce-eager"
        - "--tokenizer_mode=mistral"
        - "--config_format=mistral"
        - "--enable-auto-tool-choice"
        - "--tool-call-parser=mistral"
        resources:
          limits:
            nvidia.com/gpu: 1
          requests:
            memory: "40Gi"
            cpu: "8"
---
apiVersion: v1
kind: Service
metadata:
  name: vllm-inference-svc
  namespace: default
  labels:
```

```
    app: vllm-inference-app
spec:
  selector:
    app: vllm-inference-app
  ports:
  - name: http
    port: 8000
    targetPort: 8000
    protocol: TCP
EOF
```

檢查 vLLM Pod 是否處於就緒狀態：

```
kubectl get pod -l app=vllm-inference-app -w
```

預期的輸出結果：

```
NAME                                  READY   STATUS    RESTARTS   AGE
vllm-inference-app-65df5fddc8-5kmjm   1/1     Running   0          86s
```

容器映像提取 和 vLLM 將模型權重從 S3 串流到 GPU 記憶體可能需要約 2 分鐘。請等到 Pod 在 READY 1/1 欄中顯示後再繼續。

EKS、SOCI 和 Run：ai Model Streamer 的組合可快速啟動 Pod。若要檢查每個階段的啟動時間，請檢視 Pod 事件：

```
kubectl describe pod -l app=vllm-inference-app | grep -A 20 "Events:"
```

預期的輸出結果：

```
Events:
  Type     Reason            Age   From                   Message
  ----     ------            ----  ----                   -------
  Warning  FailedScheduling  86s   default-scheduler      0/2 nodes are available: 2
 node(s) had untolerated taint(s).
  Normal   Nominated         85s   eks-auto-mode/compute  Pod should schedule on:
 nodeclaim/gpu-inf-kqkq6
  Normal   Scheduled         55s   default-scheduler      Successfully assigned
 default/vllm-inference-app-d9d54586d-csmd7 to i-04f8792414384d2d3
  Normal   Pulling           52s   kubelet                Pulling image
 "public.ecr.aws/deep-learning-containers/vllm:0.21.0-gpu-py312-cu130-ubuntu22.04-ec2-
v1.0-soci"
```

```
  Normal   Pulled            4s    kubelet                Successfully pulled image
 "public.ecr.aws/deep-learning-containers/vllm:0.21.0-gpu-py312-cu130-ubuntu22.04-ec2-
v1.0-soci" in 48.376s (48.376s including waiting). Image size: 8802823997 bytes.
  Normal   Created           4s    kubelet                Created container vllm-
inference
  Normal   Started           4s    kubelet                Started container vllm-
inference
```

在此範例中，GPU 節點在 30 秒內佈建，並使用 SOCI 在大約 48 秒內提取 8.8 GB 容器映像。快速映像提取可減少大型推論容器的冷啟動時間，這可讓您動態擴展 GPU Pod，而不是過度佈建閒置 GPU 容量。

接著，檢查 vLLM 日誌以驗證模型載入時間：

```
kubectl logs $(kubectl get pod -l app=vllm-inference-app -o
 jsonpath='{.items[0].metadata.name}') | grep -i 'Model loading took'
```

預期的輸出結果：

```
INFO 05-18 18:41:49 [gpu_model_runner.py:4959] Model loading took 9.81 GiB memory and
 5.023344 seconds
```

日誌確認 Run：ai Model Streamer 在大約 5 秒內將 10.4 GB 模型權重直接從 S3 載入 GPU 記憶體，耗用 9.8 GiB 的 GPU 記憶體。

此範例中的映像下載時間是使用 g6e.4xlarge 執行個體，其具有 20 Gbps 的持續網路頻寬。映像提取和模型載入時間會因其他執行個體類型而有所不同，具體取決於可用的網路頻寬。

步驟 3：執行推論

執行 vLLM 部署後，驗證推論端點並部署聊天前端以與模型互動。

執行模型驗證測試

透過連接埠轉送公開推論端點：

```
kubectl port-forward svc/vllm-inference-svc 8000:8000
```

開啟新的終端機視窗，然後驗證推論容器是否正在回應：

```
curl -sI -X GET http://localhost:8000/health
```

預期的輸出結果：

```
HTTP/1.1 200 OK
date: Fri, 18 May 2026 00:39:23 GMT
server: uvicorn
content-length: 0
```

步驟 4：監控 vLLM

vLLM 開箱即用地公開 Prometheus 指標，包括請求率、字符輸送量、end-to-end延遲和 GPU KV 快取使用率。在本節中，您可以將這些指標與您在叢集設定步驟中設定的監控堆疊搭配使用，並在預先佈建的 Grafana 儀表板上檢視它們。

> **Important**
>
> 您必須透過 CLI 完成叢集設定的監控子區段，才能繼續。此步驟取決於安裝的 kube-prometheus-stack，以及值檔案中已佈建的 vLLM Grafana 儀表板。

套用 vLLM ServiceMonitor

ServiceMonitor 會告知 Prometheus 在何處抓取 vLLM 指標。

```
cat << EOF | kubectl apply -f -
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: vllm-inference-app
  namespace: default
  labels:
    release: kube-prometheus-stack
spec:
  selector:
    matchLabels:
      app: vllm-inference-app
  endpoints:
  - port: http
    path: /metrics
```

```
    interval: 15s
EOF
```

確認 ServiceMonitor 已建立：

```
kubectl get servicemonitor vllm-inference-app
```

預期的輸出結果：

```
NAME                 AGE
vllm-inference-app   5s
```

產生推論流量

若要將指標填入儀表板，請針對您在驗證步驟中已透過連接埠向前公開的 vLLM 端點產生推論流量。

探索提供的模型名稱：

```
MODEL_NAME=$(curl -s http://localhost:8000/v1/models | jq -r '.data[0].id')
echo "Using model: $MODEL_NAME"
```

平行傳送 50 個聊天完成請求：

```
for i in $(seq 1 50); do
  curl -s -X POST http://localhost:8000/v1/chat/completions \
    -H "Content-Type: application/json" \
    -d "{\"model\": \"$MODEL_NAME\", \"messages\": [{\"role\": \"user\", \"content\":
 \"Write a short poem about Kubernetes.\"}], \"max_tokens\": 128}" \
    > /dev/null &
done
wait
```

當流量正在流動時（或之後），請直接從 vLLM `/metrics`端點檢查字符輸送量指標：

```
curl -s http://localhost:8000/metrics | grep -E '^vllm:(prompt_tokens_total|
generation_tokens_total|avg_generation_throughput_toks_per_s|
avg_prompt_throughput_toks_per_s)' | head
```

`vllm:prompt_tokens_total` 和 `vllm:generation_tokens_total`指標單調增加提供的輸入和輸出字符計數器。`vllm:avg_prompt_throughput_toks_per_s` 和

`vllm:avg_generation_throughput_toks_per_s`指標是滾動平均輸送量計量。這些相同的指標為您在下列小節中開啟的 Grafana 儀表板提供動力。

檢視 vLLM Grafana 儀表板

來自監控區段的 kube-prometheus-stack 值檔案已在 GPU 監控資料夾下佈建社群 vLLM 儀表板 (gnetId 25263)，因此不需要額外的匯入。

透過您在存取 Grafana 區段中設定的負載平衡器存取 Grafana。列印其 URL：

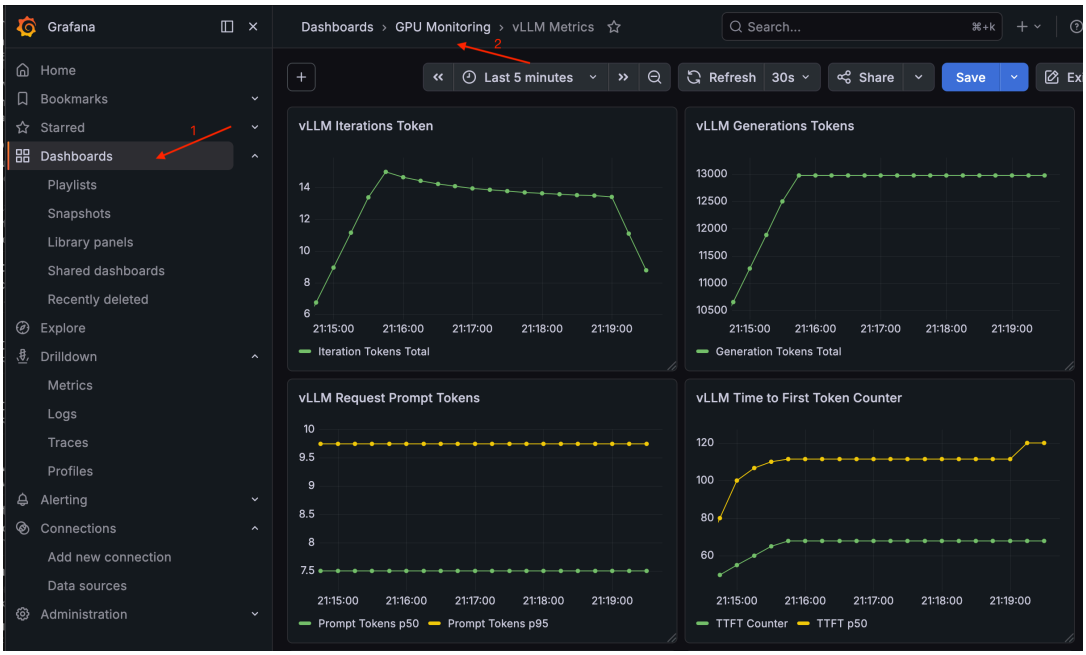
```
echo "http://$(kubectl get ingress kube-prometheus-stack-grafana -n monitoring -o
 jsonpath='{.status.loadBalancer.ingress[0].hostname}')"
```

在瀏覽器中開啟 URL，並從`admin`下列命令使用使用者名稱和密碼登入：

```
kubectl --namespace monitoring get secrets kube-prometheus-stack-grafana -o
 jsonpath="{.data.admin-password}" | base64 -d ; echo
```

導覽至儀表板 > GPU 監控 > vLLM 指標。

vLLM Grafana 儀表板



儀表板會顯示 vLLM 推論端點的請求率、提示和產生字符輸送量、延遲百分位數和 GPU KV 快取使用率。

步驟 5：部署聊天應用程式

在此步驟中，您將 Open WebUI 部署為聊天前端，以與模型互動。Open WebUI 是一種開放原始碼、自我託管的 AI 介面，支援 OpenAI 相容 APIs，並提供具有對話歷史記錄和 Markdown 轉譯的聊天介面。由於 vLLM 公開與 OpenAI 相容的 API，因此 Open WebUI 會以後端直接與其連線。

若要部署 Open WebUI 應用程式，請套用下列資訊清單：

Example開啟 WebUI 部署和服務 YAML

```
cat << 'EOF' | kubectl apply -f -
apiVersion: apps/v1
kind: Deployment
metadata:
  name: open-webui
  namespace: default
  labels:
    app: open-webui
    guide: ai-eks-docs
spec:
  replicas: 1
  selector:
    matchLabels:
      app: open-webui
  template:
    metadata:
      labels:
        app: open-webui
        guide: ai-eks-docs
    spec:
      containers:
      - name: open-webui
        image: ghcr.io/open-webui/open-webui:v0.9.2
        ports:
        - containerPort: 8080
        resources:
          requests:
            cpu: "500m"
            memory: "500Mi"
          limits:
            cpu: "1000m"
            memory: "2Gi"
        env:
```

```
        - name: OPENAI_API_BASE_URLS
          value: "http://vllm-inference-svc:8000/v1"
        - name: OPENAI_API_KEY
          value: "dummy"
        - name: WEBUI_AUTH
          value: "False"
        - name: ENABLE_OLLAMA_API
          value: "False"
        - name: ENABLE_EVALUATION_ARENA_MODELS
          value: "False"
        - name: RAG_EMBEDDING_ENGINE
          value: ""
        volumeMounts:
        - name: webui-volume
          mountPath: /app/backend/data
      volumes:
      - name: webui-volume
        emptyDir: {}
---
apiVersion: v1
kind: Service
metadata:
  name: open-webui
  namespace: default
  labels:
    app: open-webui
spec:
  type: ClusterIP
  selector:
    app: open-webui
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8080
EOF
```

等待開放 WebUI Pod 就緒：

```
kubectl wait --for=condition=ready pod -l app=open-webui --timeout=300s
```

預期的輸出結果：

```
pod/open-webui-6cbfc9867f-jf9w9 condition met
```

若要存取應用程式，請使用albIngressClass您在設定負載平衡中建立的 ，透過面向網際網路 AWS Application Load Balancer (ALB) 公開開放 WebUI。

> **⚠ Open WebUI 無需身分驗證即可公開存取**
>
> 開放式 WebUI 執行時已停用身分驗證 (WEBUI_AUTH: "False")，因此任何達到負載平衡器的人都會取得 GPU 推論端點支援的未驗證聊天介面，並且可以使用 GPU 容量。自動化掃描器可在幾分鐘內找到公有負載平衡器。您必須使用 alb.ingress.kubernetes.io/inbound-cidrs註釋限制存取，並將 source-IP 允許清單視為最低保護，而非完整保護。如需更強的姿勢，請使用內部配置、新增 TLS 憑證，以及啟用開放式 WebUI 身分驗證。

尋找您的公有 IP 地址，並將其儲存為 /32 CIDR：

```
export MY_CIDR="$(curl -s https://checkip.amazonaws.com)/32"
echo $MY_CIDR
```

結果看起來像 203.0.113.4/32。如果您的網路動態指派地址，您的 IP 地址可能會變更，在這種情況下，您可能需要更廣泛的範圍，例如 203.0.113.0/24。

套用下列項目Ingress來建立 ALB：

```
cat << EOF | kubectl apply -f -
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: open-webui
  namespace: default
  labels:
    guide: ai-eks-docs
  annotations:
    alb.ingress.kubernetes.io/load-balancer-name: ai-eks-docs-chat-app
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/target-type: ip
    alb.ingress.kubernetes.io/inbound-cidrs: ${MY_CIDR}
    alb.ingress.kubernetes.io/healthcheck-path: /health
spec:
```

```
  ingressClassName: alb
  rules:
  - http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: open-webui
            port:
              number: 80
EOF
```

運作狀態檢查路徑

`healthcheck-path` 註釋會將負載平衡器運作狀態檢查指向開放 WebUI `/health`端點，因為 ALB 預設運作狀態檢查配對器預期會有 200 個回應。

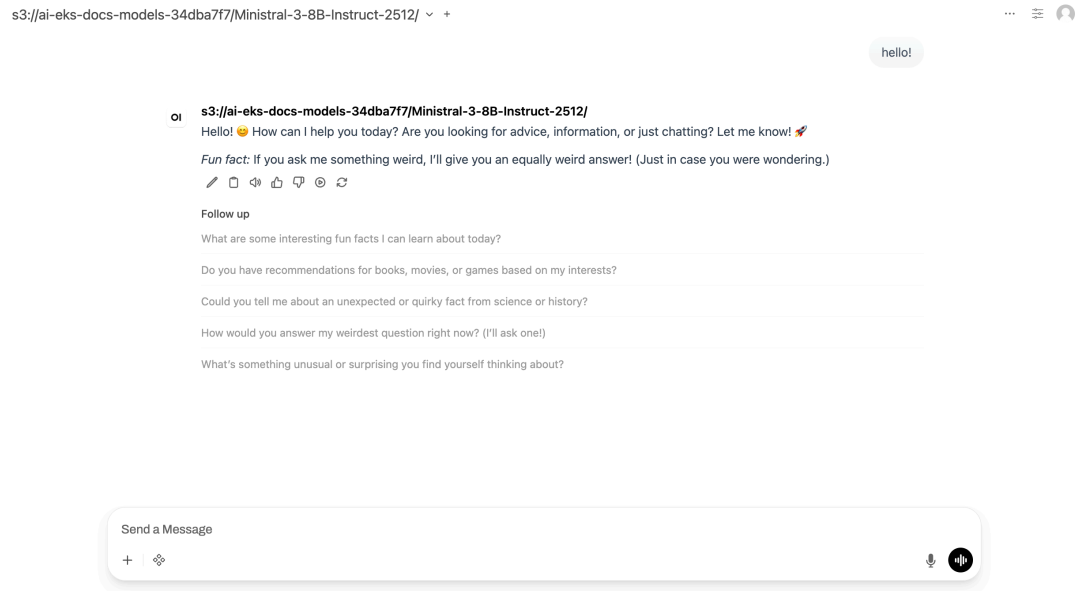
負載平衡器是以非同步方式建立，需要一兩分鐘的時間。列印 URL：

```
echo "http://$(kubectl get ingress open-webui -o
 jsonpath='{.status.loadBalancer.ingress[0].hostname}')"
```

在瀏覽器中開啟 URL。聊天界面隨即出現，您可以與 Ministral 模型互動。

連接埠轉送做為替代方案

連接埠轉送 (`kubectl port-forward svc/open-webui 8080:80`) 仍然是本機測試的選項，無需佈建負載平衡器。



清除

若要移除您在本節中建立的工作負載資源，請刪除 Open WebUI 應用程式、vLLM 推論伺服器和模型下載任務：

```
kubectl delete ingress open-webui
kubectl delete deployment open-webui
kubectl delete service open-webui
kubectl delete deployment vllm-inference-app
kubectl delete service vllm-inference-svc
kubectl delete servicemonitor vllm-inference-app
kubectl delete job model-download
```

如需移除叢集、NodePool 和 S3 儲存貯體等基礎設施資源的說明，請參閱叢集設定清除。

使用 Amazon EKS 在 GPUs 上自動擴展 AI 模型推論

> **Tip**
>
> 註冊即將舉行的 Amazon EKS AI/ML 研討會。

本節涵蓋使用 Amazon EKS 在 GPUs 上自動擴展 AI 推論工作負載的概念和程序。

您在上一個 Load & Serve Models 區段中部署的單一 vLLM 模型複本，可以在新請求開始等待之前提供有限數量的並行請求，這會增加延遲並可能導致請求逾時。若要跟上流量的步伐，您可以自動水平擴

展部署。Autoscaling 會在需求增加時新增工作負載複本，並在需求下降時將其移除，因此您可以避免過度佈建 GPU 容量。

推論自動調整規模分為兩個階段。

- 節點擴展 — 當 Kubernetes 排程器沒有空間在現有節點上放置新的 Pod 時，Pod 會保留 `Pending`。Karpenter `Pending` 會透過佈建 GPU 節點以符合它們來回應 Pod，並在工作負載複本縮減時移除節點。
- Pod 擴展 — Horizontal Pod Autoscaler (HPA) 會監看設定的指標。當該指標超過設定的閾值時，HPA 會增加工作負載複本計數，並建立新的 vLLM Pod。

節點擴展階段提供執行所需的 GPU 容量工作負載複本。Pod 擴展階段會調整工作負載複本的數量，將您選擇的指標保持在接近其目標的位置。

使用 Karpenter 佈建 GPU 節點

Karpenter 會管理節點生命週期、在工作負載需要節點時佈建節點，以及在節點不需要時合併或移除節點。它適用於 Kubernetes 排程器，而不是取代它。排程器會將 Pod 放置在節點上。Karpenter 會為排程器無法容納的 Pod 佈建容量，然後讓排程器在節點為 時繫結它們`Ready`。

Karpenter 處理 GPU 節點的方式與 CPU 節點機群不同。

GPUs 是以整個單位請求

除非您使用時間分割或 MIG 等 GPU 共用技術，否則 Pod 會取得一或多個整個 GPUs，而不是 CPU 或記憶體使用方式的一小部分。每個 GPU 都有自己的專用記憶體，因此相同節點上的複本不會共用 GPU 記憶體。映射到節點的方式取決於執行個體類型：

- 單一 GPU 執行個體 （例如 `g6.xlarge`) — 節點只保留一個複本。複本、GPUs 和節點one-to-one擴展，而且每個複本都會在自己的執行個體上隔離。
- 多 GPU 執行個體 （例如 `g6.12xlarge`，擁有 4 個 GPUs) — 裝置外掛程式公告完整的 GPU 計數，因此 Karpenter 可以將多個複本封裝到一個節點，每個 GPU 一個節點。
- 多 GPU Pod — Pod 可以請求多個 GPU。例如，將 `nvidia.com/gpu: 2`設定為跨兩個 GPUs 分割大型模型，而 Karpenter 會將 Pod 放置在可用多個 GPUs 的節點上。

Karpenter 最佳化 GPU 可用性和成本

Karpenter 以經濟實惠的方式平衡登陸稀少的 GPU 容量與執行容量：

- 執行個體選擇 — 根據彈性的 NodePool，Karpenter 會選取符合 Pod `Pending` 的最低成本執行個體類型。當啟動達到容量不足時，會自動重試其他執行個體類型和可用區域，以改善登陸 GPU 容量的機率。
- 合併 – 當需求下降且複本縮減時，Karpenter 會回收釋放的節點。會`WhenEmptyOrUnderutilized`移除空的節點，並將複本重新封裝到較少的節點上，而只有在沒有排程節點之後才會`WhenEmpty`回收節點。
- 保護傳輸中的 Pod — 若要避免在整合期間中斷 Pod，請新增`karpenter.sh/do-not-disrupt`註釋、設定 PodDisruptionBudget，或使用 `consolidateAfter`設定延遲整合。

一旦 Karpenter 具備 GPU 容量，Pod 擴展階段就會接管：HPA 會根據您設定的指標新增和移除複本。本節其餘部分說明如何選擇推論工作負載的擴展指標。

GPU 與 CPU 自動擴展

在 Kubernetes 中，傳統的 CPU 型工作負載out-of-the-box自動擴展，但 GPU 型推論不會。有三個主要差異：

自動擴展的指標可用性

Kubernetes 提供立即可用的 HPA out-of-the-box和記憶體指標

每個節點上的 kubelet 會將 CPU 和記憶體用量報告給 Kubernetes 指標 API，因此 HPA 可以直接讀取這些值，並擴展工作負載，而無需額外的元件。

加速的工作負載需要額外的指標設定、匯出程式和轉接器

kubelet 不會報告 GPU 或推論指標。若要擴展 GPU 型推論，您必須新增發佈相關訊號的匯出工具，例如來自 DCGM 匯出工具的 vLLM 應用程式指標和 GPU 指標，以及向 HPA 公開這些指標的轉接器。

資源分配和共用

傳統工作負載以一小部分的速度共用 CPU 和記憶體

Linux cgroups 會將 CPU 和記憶體劃分為部分資源，允許多個 Pod 共用相同的節點。Pod 可以請求節點的一部分容量，例如 100 毫秒的 CPU 或 512 MiB 的記憶體。

加速工作負載透過 GPUs 配置架構配置和共用 GPU

GPUs 不會透過 CPU 和記憶體等 cgroup 共用。目前最常見的 Kubernetes 設定使用 NVIDIA 裝置外掛程式，可將整個 GPUs 配置給 Pod （例如 `nvidia.com/gpu: 1`）。動態資源分配 (DRA) 是

較新的 Kubernetes API，可透過資源宣告配置 GPUs，提供更靈活的排程和資源管理。裝置外掛程式和 DRA 都可以支援 GPU 共用技術。時間分割可讓多個 Pod 隨著時間的推移將存取權互斥，而 NVIDIA A100 和 H100 等多執行個體 GPU (MIG) 分割區支援的 GPUs 則整合至硬體隔離的 GPU 執行個體。

擴展訊號

傳統工作負載通常會隨著 CPU 和記憶體使用率而擴展

使用率是 CPU 和記憶體工作負載的標準自動調整規模訊號。這是一個延遲指標，因為它只有在工作負載已處於負載狀態後才會上升，但對於許多傳統應用程式而言，它足以推動擴展決策。

加速的工作負載應依據領導指標進行擴展，而非 GPU 使用率

GPU 使用率 (`DCGM_FI_DEV_GPU_UTIL`) 會顯示 GPU 是否忙碌，而不是多忙碌。由於 vLLM 等推論伺服器可以在各種傳入請求中保持接近 100%，因此使用率是一種不可靠的自動擴展訊號。改為根據隨需求增加的領導指標擴展，例如請求佇列深度、到第一個字符的時間 (TTFT) 和請求延遲。

在上述涵蓋的這三個差異中，仔細考慮您的擴展訊號。由於 GPU 使用率不是剩餘推論容量的高可信度訊號，因此您需要識別和收集追蹤模型推論伺服器上實際需求的訊號。

下一節涵蓋要用作擴展指標的訊號，以及它們如何一起運作。

選擇擴展指標

沒有單一指標會告訴您何時擴展模型推論複本，因此您可以根據來自推論伺服器的訊號組合進行擴展。這些訊號可一起運作，每個建置在先前的訊號上，以比任何單一指標更早且更可靠地擷取需求：

Signal	指標	擴展中的角色
佇列	請求佇列深度（等待處理的請求數量）	領導指標和最佳預設觸發，因為佇列形成的時刻需求超過容量。
延遲	請求延遲 (p95 延遲或 TTFT)	即使佇列尚未建立，當回應緩慢時觸發擴展，將使用者面向的延遲保持在目標內。
GPU 記憶體	KV 快取使用率 (GPU 請求記憶體的完整程度）	保護 GPU 記憶體不會用盡；當伺服器接近其限制時，伺服器會開始拒絕或排入佇列請求。

它們共同形成防禦層：請求佇列深度是觸發瞬間需求超過容量的主要觸發條件，請求延遲會在佇列形成之前新增慢速回應的檢查，而 KV 快取使用率是防止 GPU 記憶體耗盡的後端。

將一切付諸實踐

以下小節說明如何將這些訊號用於特定的自動擴展器。

- 尋找擴展指標閾值。負載測試單一 vLLM 複本，以尋找其容量上限，以及要擴展的佇列深度和延遲閾值。從這裡開始。
- 使用 HPA 和 KEDA 擴展。使用 KEDA 和 Horizontal Pod Autoscaler 自動擴展 vLLM 複本，使用請求佇列深度和end-to-end延遲作為擴展訊號。

識別 AI 推論的擴展指標閾值

> **Tip**
>
> 註冊即將舉行的 Amazon EKS AI/ML 研討會。

本節說明如何在 Amazon EKS 上載入測試 vLLM 推論伺服器，以識別何時飽和。使用負載測試結果在下the section called “HPA 和 KEDA”一節中設定水平自動調整規模的佇列深度和延遲閾值。

演練使用以下工具：

- k6 (Grafana k6) 是一種開放原始碼負載測試工具，以`grafana/k6`容器映像的形式執行，以受控請求速率將推論請求傳送至複本。
- `g6e.4xlarge` 執行個體上的 Ministral-3-8B-Instruct-2512 模型。您可以將相同的方法套用至任何開放原始碼模型和 GPU 類型。

先決條件

本節以文件中的兩個先前區段為基礎。開始之前，請先完成這兩個步驟：

- 叢集設定為 EKS 叢集提供 GPU NodePool、kube-prometheus-stack 監控堆疊 (Prometheus 和 Grafana)，以及用於模型權重的 Amazon S3 儲存貯體。
- Load & Serve Models 會部署 vLLM 推論伺服器並建立 `vllm-inference-app ServiceMonitor`。

在開始之前確認以下每個項目，如果檢查失敗，請先完成連結區段。

如果您開啟新的終端機，請設定您先前使用的叢集名稱和區域：

```
export CLUSTER_NAME=ai-eks-docs
export AWS_REGION=us-east-2
```

擷取叢集設定步驟中建立的 Amazon S3 模型儲存貯體名稱。負載產生器會將此值傳遞至 vLLM。

```
MODEL_BUCKET=$(aws s3api list-buckets \
  --query "Buckets[?starts_with(Name, '${CLUSTER_NAME}-models-')].Name | [0]" \
  --output text)
echo "Model bucket: ${MODEL_BUCKET}"
```

確認監控堆疊正在執行

來自監控設定的 kube-prometheus-stack 會在 `monitoring` 命名空間中執行。

```
kubectl get pods -n monitoring
```

Prometheus 和 Grafana Pod 應該是 `Running`：

```
NAME                                                READY   STATUS    RESTARTS   AGE
kube-prometheus-stack-grafana-6f9c8b7d5c-2xk9p      3/3     Running   0          3h
kube-prometheus-stack-operator-7b8c9d6f4-q4m7n      1/1     Running   0          3h
prometheus-kube-prometheus-stack-prometheus-0       2/2     Running   0          3h
```

如果缺少這些項目，請先完成監控設定再繼續。

確認 vLLM 模型正在執行

從 Load & Serve Models 的`vllm-inference-app`部署`vllm-inference-svc`和服務會在 `default` 命名空間中執行。

```
kubectl get deployment vllm-inference-app
```

部署應該報告`1/1`就緒：

```
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
vllm-inference-app  1/1     1            1           3h
```

如果部署遺失或尚未就緒，請先完成載入並提供模型再繼續。

存取 Grafana

您可以在預先載入的 vLLM 儀表板中觀看負載測試，因此請確定您在存取 Grafana 區段中設定的 Grafana 負載平衡器可以連線。列印其 URL：

```
echo "http://$(kubectl get ingress kube-prometheus-stack-grafana -n monitoring -o
 jsonpath='{.status.loadBalancer.ingress[0].hostname}')"
```

在瀏覽器中開啟 URL，並從admin下列命令使用使用者名稱和密碼登入：

```
kubectl --namespace monitoring get secrets kube-prometheus-stack-grafana -o
 jsonpath="{.data.admin-password}" | base64 -d ; echo
```

確認 vLLM ServiceMonitor 存在

在監控 vLLM 步驟中建立的 vllm-inference-app ServiceMonitor 在default命名空間中執行，並告知 Prometheus 在何處抓取 vLLM 指標。

```
kubectl get servicemonitor vllm-inference-app
```

預期的輸出結果：

```
NAME                 AGE
vllm-inference-app   3h
```

如果遺失，請完成監控 vLLM 步驟。

步驟 1：暖機 GPU

即使模型已載入 GPU 記憶體，且複本報告為就緒，但第一個請求所需的時間會比之後的請求更長，因為 vLLM 和 GPU 在第一個推論上完成初始化。vLLM 會擷取 CUDA 圖形、編譯和自動調校 GPU 核心，並配置其 KV 快取記憶體集區，同時 GPU 會將其時脈速度從閒置提升到提升範圍。

這些是一次性成本，因此在暖機記錄啟動工作前測量閾值，而不是穩定狀態服務。下列暖機會傳送 100 個循序請求以完成此初始化，因此後續的負載測試會測量穩定狀態容量。

首先，建立保留暖機指令碼的 ConfigMap (warmup.js)。指令碼會從MODEL環境變數讀取目標模型，任務會從您先前擷取MODEL_BUCKET的值設定：

Example k6 暖機 ConfigMap

```
cat << 'EOF' | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  name: vllm-warmup-script
  namespace: default
data:
  warmup.js: |
    import http from 'k6/http';

    export default function () {
      const payload = JSON.stringify({
        model: __ENV.MODEL,
        prompt: 'Write a brief technical explanation about cloud computing concepts.',
        max_tokens: 100,
        temperature: 0.7,
      });
      const params = { headers: { 'Content-Type': 'application/json' }, timeout:
 '240s' };
      http.post('http://vllm-inference-svc:8000/v1/completions', payload, params);
    }
EOF
```

然後執行暖機任務。單一用戶端會傳送 100 個循序請求：

Example暖機任務

```
cat << EOF | kubectl apply -f -
apiVersion: batch/v1
kind: Job
metadata:
  name: vllm-warmup
  namespace: default
spec:
  backoffLimit: 0
  template:
    spec:
      restartPolicy: Never
      containers:
        - name: k6
          image: grafana/k6:latest
```

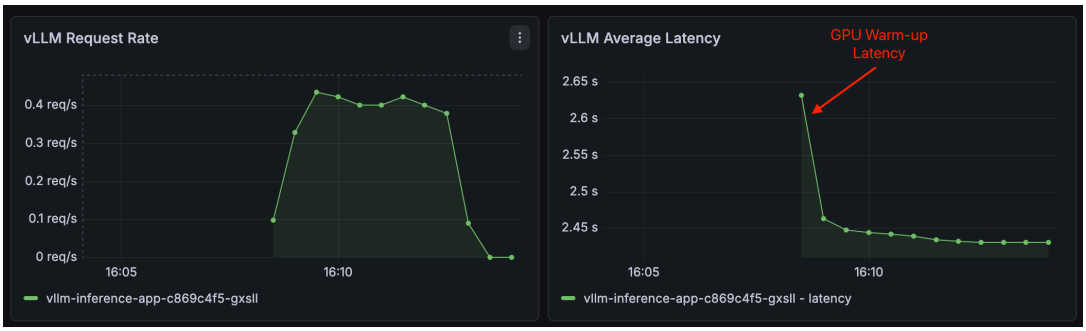
```
          env:
            - name: MODEL
              value: "s3://${MODEL_BUCKET}/Ministral-3-8B-Instruct-2512/"
          args: ["run", "--vus", "1", "--iterations", "100", "/scripts/warmup.js"]
          volumeMounts:
            - name: script
              mountPath: /scripts
      volumes:
        - name: script
          configMap:
            name: vllm-warmup-script
EOF
```

等待暖機完成（需要幾分鐘）：

```
kubectl wait --for=condition=complete job/vllm-warmup --timeout=600s
```

開啟 Grafana（開啟負載平衡器主機名稱；如需詳細資訊，請參閱存取 Grafana)，然後導覽至儀表板 > GPU 監控 > 效能測試 - vLLM 負載分析。vLLM 請求率面板會顯示暖機流量爆增。vLLM 平均延遲面板顯示第一個請求較慢，這是一次性的 GPU 暖機成本。

vLLM 儀表板顯示 GPU 暖機對延遲的影響



然後刪除暖機任務：

```
kubectl delete job vllm-warmup
```

步驟 2：以提高的請求率傳送負載

在此步驟中，您會執行負載測試，以更高的速率將請求傳送至單一 vLLM 複本：每秒 10、20、30、40、50、60 和 70 個請求。這會模擬不斷增加的使用者需求。測試在constant-arrival-rate模型中使用 k6，TARGET_RPS環境變數會在其中設定目標請求率。每個速率會執行 60 秒以允許指標穩定，然後在下一次速率增加之前暫停 30 秒。

此測試的目標是識別飽和點，這是單一複本無法再跟上傳入請求的請求率。您可以監控佇列深度 (vllm:num_requests_waiting) 和end-to-end(vllm:e2e_request_latency_seconds)。增加佇列深度和延遲表示複本即將飽和。

首先，建立包含 k6 負載測試指令碼的 ConfigMap (script.js)。指令碼會從MODEL環境變數讀取目標模型，每個 Pod 都會從您先前擷取MODEL_BUCKET的值設定：

Example k6 負載測試 ConfigMap

```
cat << 'EOF' | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  name: vllm-loadtest-script
  namespace: default
data:
  script.js: |
    import http from 'k6/http';

    // Dictionary of 10-letter words for variation (prevents prompt caching)
    const words = [
      'revolution', 'technology', 'javascript', 'kubernetes', 'basketball',
      'strawberry', 'watermelon', 'blackberry', 'throughout', 'background',
      'playground', 'understand', 'everything', 'protection', 'complexity'
    ];

    export const options = {
      noConnectionReuse: true,
      scenarios: {
        constant_request_rate: {
          executor: 'constant-arrival-rate',
          rate: __ENV.TARGET_RPS || 10,  // Overridden by TARGET_RPS env var
          timeUnit: '1s',
          duration: __ENV.DURATION || '60s',  // Overridden by DURATION env var
          preAllocatedVUs: 50,
          maxVUs: 500,
        },
      },
    };

    export default function () {
      const randomWord = words[Math.floor(Math.random() * words.length)];
      const payload = JSON.stringify({
```

```
        model: __ENV.MODEL,
        prompt: `Starting with the word "${randomWord}", write a brief technical
 explanation about cloud computing concepts. Cover containerization, orchestration, and
 scalability in about 80 words.`,
        max_tokens: 100,
        temperature: 0.7,
      });
      const params = { headers: { 'Content-Type': 'application/json' }, timeout:
 '240s' };
      http.post('http://vllm-inference-svc:8000/v1/completions', payload, params);
    }
EOF
```

然後，以每個請求速率執行負載測試。下列迴圈會針對每個速率提交個別的任務：

Example載入測試迴圈

```
for RPS in 10 20 30 40 50 60 70; do
  echo "=== Testing at ${RPS} req/s ==="
  cat << EOF | kubectl apply -f -
apiVersion: batch/v1
kind: Job
metadata:
  name: vllm-loadtest-${RPS}
  namespace: default
  labels:
    app: vllm-loadtest
spec:
  backoffLimit: 0
  template:
    metadata:
      labels:
        app: vllm-loadtest
    spec:
      restartPolicy: Never
      containers:
        - name: k6
          image: grafana/k6:latest
          env:
            - name: K6_PROMETHEUS_RW_SERVER_URL
              value: "http://kube-prometheus-stack-
prometheus.monitoring.svc.cluster.local:9090/api/v1/write"
            - name: MODEL
              value: "s3://${MODEL_BUCKET}/Ministral-3-8B-Instruct-2512/"
```

```
            - name: TARGET_RPS
              value: "${RPS}"
            - name: DURATION
              value: "60s"
          args: ["run", "-o", "experimental-prometheus-rw", "/scripts/script.js"]
          volumeMounts:
            - name: script
              mountPath: /scripts
      volumes:
        - name: script
          configMap:
            name: vllm-loadtest-script
EOF
  kubectl wait --for=condition=complete job/vllm-loadtest-${RPS} --timeout=180s
  sleep 30
done
```

在測試執行時觀看 Grafana 中的指標 （ 請參閱步驟 3)。每個速率都會建立自己的任務 (`vllm-loadtest-10` 到 `vllm-loadtest-70`)，因此您可以使用 檢查任何執行`kubectl get jobs`。完成後，請刪除所有負載測試任務：

```
kubectl delete jobs -l app=vllm-loadtest
```

步驟 3：測量負載下的指標

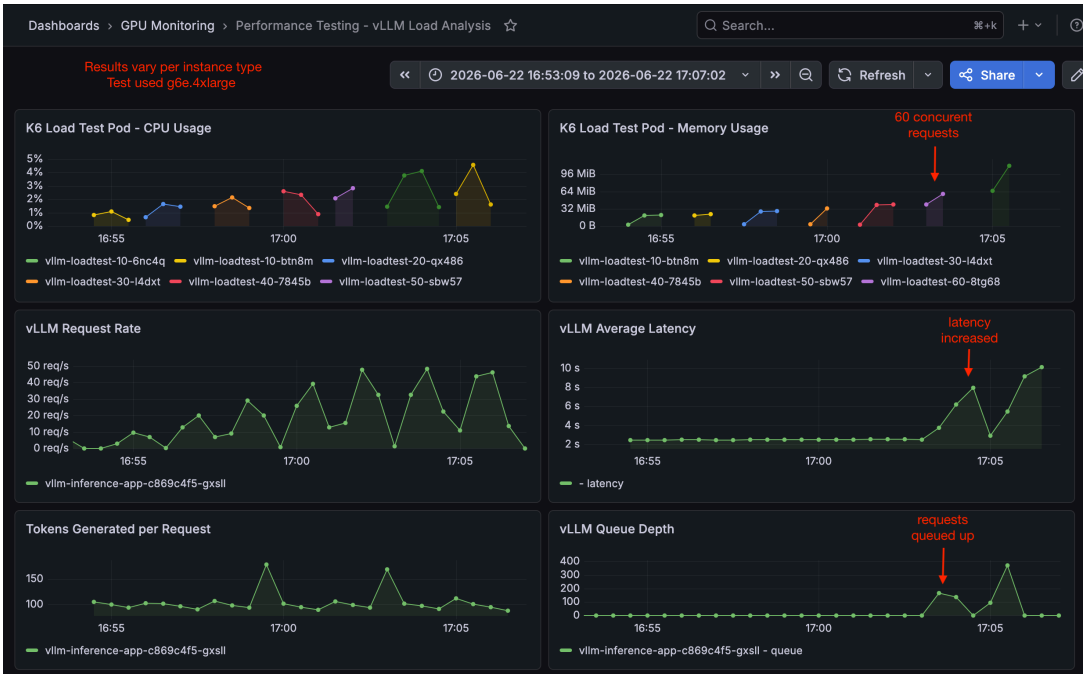
當負載執行時，請監看驅動自動擴展的兩個指標：

- 佇列深度 (`vllm:num_requests_waiting`) — 有多少請求正在等待處理。
- p95 end-to-end延遲 (`vllm:e2e_request_latency_seconds`) — 第 95 個百分位數回應時間。

在 Grafana 中檢視指標

開啟 Grafana （ 開啟負載平衡器主機名稱並以 身分登入`admin`；如需詳細資訊，請參閱存取 Grafana)，然後導覽至儀表板 > GPU 監控 > 效能測試 - vLLM 負載分析。此儀表板會從 Amazon Managed Service for Prometheus 讀取，因此面板最多可以將即時狀態延遲一分鐘，因為指標是批次遠端寫入。

效能測試 - 負載測試期間的 vLLM 負載分析儀表板



根據負載測試結果，佇列深度保持接近零到 50 個並行請求。然後，它會在 60 個並行請求中大幅增加，達到大約 150 到 350 個排入佇列的請求。同時，p95 end-to-end延遲從大約 2.5 秒的穩定基準提升為 6–10 秒。此行為表示持續過載的發生，並建議在佇列深度達到導致延遲接近 10 秒的層級之前，自動擴展應該開始。實際的起點是在佇列深度超過 25 個請求達 30-60 秒時觸發擴展。作為輔助訊號，您也可以在相同間隔內 p95 end-to-end延遲超過 5 秒時觸發擴展。這有助於考慮工作負載模式，在佇列深度累積之前，延遲會增加。

步驟 4：選擇要擴展的指標和閾值

使用在飽和點觀察到的指標，您可以在下一節中判斷用於設定自動調整規模的閾值。

- 佇列深度 (`vllm:num_requests_waiting`) — 複本飽和後等待處理的請求數量。將閾值設定為高於佇列首先保持正值的點，因此暫時性峰值不會觸發擴展。對於本節中的負載測試，佇列深度保持接近零到 50 個並行請求，並在 60 個並行請求中大幅增加到大約 150–350 個佇列請求。實際的起點是在佇列深度超過 25 個請求達 30-60 秒時觸發擴展。
- 延遲 (`vllm:e2e_request_latency_seconds`) — 複本接近其容量限制時的end-to-end回應時間。將閾值設定為低於延遲 SLO，因此自動擴展器會在使用者可見延遲變得無法接受之前橫向擴展。對於本節中的負載測試，p95 end-to-end延遲在大約 2.5 秒時保持穩定，並在飽和時增加到 6–10 秒。作為輔助訊號，實際的起點是在 p95 end-to-end延遲超過 5 秒達 30-60 秒時觸發擴展。這有助於考慮工作負載模式，其中延遲在佇列深度建立之前增加，或延遲在不佇列的情況下增加。

- 縮減規模：自動擴展器在需求下降後移除複本的速率。向下擴展比向上擴展更慢。在此範例中，只有在佇列深度為 0且 p95 end-to-end延遲低於 3 秒達 5 分鐘時，才能縮減規模。這可防止自動擴展器在流量短暫暫停期間移除複本，並有助於避免振盪。

這些值提供下一個區段中設定自動調整規模的起點。針對您自己的模型、GPU 類型和請求模式重複此程序，以判斷部署的適當閾值。

使用 HPA 和 KEDA 自動擴展 AI 推論

> **Tip**
>
> 註冊即將舉行的 Amazon EKS AI/ML 研討會。

本節說明如何根據尋找擴展指標閾值區段中建立的閾值來擴展`vllm-inference-app`部署。

演練使用以下工具：

- KEDA (Kubernetes Event-Driven Autoscaler) 是直接查詢 Prometheus 並建立標準 Kubernetes Horizontal Pod Autoscaler (HPA) 的 CNCF 專案。它會在 vLLM 佇列深度（主要需求訊號）和 p95 end-to-end延遲 (SLO 護欄）上擴展部署，並在超過任一閾值時新增複本。

  KEDA 為 GPU 推論自動擴展提供三個主要優點：

  - 擴展到零 — KEDA 可以在閒置時將部署擴展到零個複本，並在需求傳回時擴展複本，有助於降低 GPU 成本。
  - 啟用閾值 — KEDA 會將啟用擴展的閾值與用於判斷複本計數的目標閾值分開。這可讓您忽略暫時性峰值，並避免針對幾個請求叫用 GPUs。
  - 更簡單的設定 — 對於 GPU 推論工作負載，KEDA 通常是最簡單的自動調整規模機制，具有內建的 Prometheus 整合，無需管理單獨的指標轉接器。

先決條件

此子區段會從尋找擴展指標閾值繼續。請確定您已完成 Load & Serve Models 和監控設定，因此`vllm-inference-app`部署和服務會在`default`命名空間中執行，而 kube-prometheus-stack 正在抓取 vLLM 指標。

如果您開啟新的終端機，請設定您先前使用的叢集名稱和區域：

```
export CLUSTER_NAME=ai-eks-docs
export AWS_REGION=us-east-2
```

步驟 1：安裝 KEDA

將 KEDA 與 Helm 安裝到自己的命名空間：

```
helm repo add kedacore https://kedacore.github.io/charts
helm repo update

helm install keda kedacore/keda --namespace keda --create-namespace
```

驗證 KEDA Pod 正在執行：

```
kubectl get pods -n keda
```

預期的輸出結果：

```
NAME                                      READY   STATUS    RESTARTS   AGE
keda-admission-webhooks-7d4d6c6f9-xxxxx   1/1     Running   0          40s
keda-operator-6b8f9c5d7c-xxxxx            1/1     Running   0          40s
keda-operator-metrics-apiserver-xxxxx     1/1     Running   0          40s
```

步驟 2：建立 KEDA ScaledObject

會`ScaledObject`告知 KEDA 要擴展的部署、複本邊界和觸發條件。此組態使用兩個觸發：佇列深度作為主要需求訊號，而 p95 end-to-end延遲作為 SLO 護欄。`metricType: AverageValue` 設定可讓 KEDA 將每個指標分割為多個複本，因此每個 Pod 都會解譯閾值。

此演練使用尋找擴展指標閾值區段中的範例閾值：當每個 Pod 的平均佇列深度超過 25 個等待請求，或 p95 end-to-end延遲超過 5 秒時，向上擴展。取代您為自己的模型、GPU 和請求形狀測量的值。

```
cat << EOF | kubectl apply -f -
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: vllm-inference-app
spec:
  scaleTargetRef:
```

```
    apiVersion: apps/v1
    kind: Deployment
    name: vllm-inference-app
  minReplicaCount: 1
  maxReplicaCount: 5
  advanced:
    horizontalPodAutoscalerConfig:
      name: vllm-inference-app
      behavior:
        scaleUp:
          stabilizationWindowSeconds: 30
          policies:
            - type: Pods
              value: 2
              periodSeconds: 60
          selectPolicy: Max
        scaleDown:
          stabilizationWindowSeconds: 300
          policies:
            - type: Pods
              value: 1
              periodSeconds: 120
          selectPolicy: Min
  triggers:
    # Primary: queue depth (demand exceeds capacity)
    - type: prometheus
      metricType: AverageValue
      metadata:
        serverAddress: http://kube-prometheus-stack-prometheus.monitoring.svc:9090
        query: sum(vllm:num_requests_waiting) or vector(0)
        threshold: "25"
        activationThreshold: "1"
    # SLO guardrail: p95 end-to-end request latency (seconds)
    - type: prometheus
      metricType: AverageValue
      metadata:
        serverAddress: http://kube-prometheus-stack-prometheus.monitoring.svc:9090
        query: histogram_quantile(0.95,
 sum(rate(vllm:e2e_request_latency_seconds_bucket[1m])) by (le)) or vector(0)
        threshold: "5"
EOF
```

金鑰設定：

- 佇列深度觸發：query傳回所有 vLLM Pod 的等待請求總數。當沒有請求等待時，or vector(0)子句會傳回 0 而不是空的結果，這可防止觸發進入非作用中狀態。threshold: "25"是您在尋找擴展指標閾值中建立的每個 Pod 佇列深度目標，並將部署activationThreshold: "1"保持在 ，minReplicaCount直到至少一個請求等待為止，因此不會在閒置訊號上擴展。
- 延遲觸發 — query傳回 p95 end-to-end請求延遲，以秒為單位，從 1 分鐘時段內的 vLLM 延遲長條圖計算。 threshold: "5"是來自尋找擴展指標閾值的延遲 SLO 護欄。當 p95 延遲超過此延遲時，即使佇列尚未建立，KEDA 也會縱向擴展。
- 觸發器評估 — KEDA 擴展以滿足需要最多複本的任何觸發器，因此違反任一訊號都會觸發擴展。
- 向上擴展和向下擴展行為 — 向上擴展具有回應能力 (30 秒時段，每分鐘最多 2 個 Pod)，因為佇列或增加延遲表示使用者已經在等待。縮減規模是保守的 (5 分鐘時段，每 2 分鐘 1 個 Pod)，因為 GPU Pod 啟動速度緩慢。此方法可避免移除稍後可能需要的容量。

驗證 ScaledObject 已就緒：

```
kubectl get scaledobject vllm-inference-app
```

預期的輸出結果：

```
NAME                SCALETARGETKIND      SCALETARGETNAME      MIN   MAX   READY
 ACTIVE
vllm-inference-app   apps/v1.Deployment   vllm-inference-app   1     5     True
 False
```

READY: True 表示觸發程序有效，且 KEDA 正在管理部署。 ACTIVE: False 預期在沒有流量流動時執行，因為尚未超過觸發條件閾值。

當您建立 ScaledObject 時，KEDA 會自動建立和管理具有相同名稱的備份 Horizontal Pod Autoscaler (HPA)。您不會自行建立或編輯此 HPA。確認它存在：

```
kubectl get hpa vllm-inference-app
```

預期的輸出結果：

```
NAME                REFERENCE                          TARGETS               MINPODS
 MAXPODS   REPLICAS
vllm-inference-app   Deployment/vllm-inference-app   0/25 (avg), 0/5 (avg)  1         5
         1
```

TARGETS 資料欄會根據其閾值（佇列深度和 p95 延遲）顯示每個指標的目前值。 MINPODS 和 MAXPODS 來自 ScaledObject maxReplicaCount中的 minReplicaCount和 ，而 REPLICAS是目前的 vLLM Pod 數量。

步驟 3：產生載入

負載測試會重複使用您在尋找擴展指標閾值中建立的 vllm-loadtest-script ConfigMap。確認它仍然存在：

```
kubectl get configmap vllm-loadtest-script
```

如果遺失，請從尋找擴展指標閾值重新建立。

以每秒 60 個請求執行單一負載測試 10 分鐘，以推動擴展。任務會從您先前設定MODEL_BUCKET的環境變數讀取模型儲存貯體，並透過負載測試指令碼讀取的 和 DURATION變數傳遞速率TARGET_RPS和持續時間。

```
cat << EOF | kubectl apply -f -
apiVersion: batch/v1
kind: Job
metadata:
  name: vllm-loadtest-scaleup
  labels:
    app: vllm-loadtest-scaleup
spec:
  backoffLimit: 0
  template:
    spec:
      restartPolicy: Never
      containers:
        - name: k6
          image: grafana/k6:latest
          env:
            - name: MODEL
              value: "s3://${MODEL_BUCKET}/Ministral-3-8B-Instruct-2512/"
            - name: TARGET_RPS
              value: "60"
            - name: DURATION
              value: "10m"
          args: ["run", "/scripts/script.js"]
          volumeMounts:
            - name: script
```

```
              mountPath: /scripts
      volumes:
        - name: script
          configMap:
            name: vllm-loadtest-script
EOF
```

任務執行 10 分鐘。移至下一個步驟，在測試執行時觀看部署橫向擴展。

步驟 4：觀看部署擴展

觀看自動擴展器回應上一個步驟的負載。在不同的終端機中執行下列每個命令。

觀看 HPA。資料TARGETS欄會爬升到設定的目標以上，並REPLICAS增加：

```
kubectl get hpa vllm-inference-app -w
```

觀看 vLLM Pod。新複本會在模型載入 GPU 記憶體Ready時出現並到達：

```
kubectl get pods -l app=vllm-inference-app -w
```

確認 ScaledObject 處於作用中狀態。一旦違反觸發閾值True， 會ACTIVE變更為 ：

```
kubectl get scaledobject vllm-inference-app
```

預期的行為：

- 在約 30 秒內，佇列深度會上升至超過閾值 （或 p95 end-to-end延遲超過 5 秒），而 HPA TARGETS資料欄則會上升至超過設定的目標。
- KEDA 會向上擴展部署，如果沒有 GPU 節點可用，Karpenter 會佈建新的 GPU 節點。新複本會在模型載入 GPU 記憶體Ready時到達。
- 隨著流量分散到複本，佇列會耗盡、延遲復原，而且這兩個指標都會降至低於閾值。
- 負載停止後，部署會在 5 分鐘冷卻時間內保留較高的複本計數，然後一次向 縮減一個 PodminReplicaCount。

當負載測試完成時，刪除負載測試任務。將 vllm-loadtest-script ConfigMap 保留在原處，因為它與閾值區段共用：

```
kubectl delete jobs -l app=vllm-loadtest-scaleup --ignore-not-found
```

清除

> **Note**
>
> 如果您打算繼續使用自動擴展，請略過此清除。只有在完成時才會執行它。

若要移除您在本節中建立的自動調整規模資源，請刪除 `ScaledObject`並解除安裝 KEDA：

```
kubectl delete scaledobject vllm-inference-app --ignore-not-found
helm uninstall keda -n keda
kubectl delete namespace keda
```

刪除 `ScaledObject`也會移除 KEDA 建立的 HPA。您的 vLLM 部署會返回其靜態複本計數。

若要移除 vLLM 推論伺服器和相關工作負載資源，請參閱載入並提供模型。如需移除叢集、NodePool 和 S3 儲存貯體等基礎設施資源的說明，請參閱叢集設定清除。

在 Amazon EKS 上加速模型載入

當您在 Amazon EKS 上部署大型語言模型 (LLMs) 時，模型載入時間會直接影響 Pod 開始提供推論請求的速度。在擴展事件期間，當新的 Pod 或節點必須在處理流量之前載入模型時，尤其如此。模型啟動有兩個階段，您可以透過調校和編譯成品快取來改善這些階段：

- 權重載入：使用 Run：ai Model Streamer，將模型權重檔案從 Amazon S3 串流至 GPU 記憶體。https://github.com/run-ai/runai-model-streamer
- torch.compile — 將模型的運算圖形編譯為最佳化的融合 CUDA/Triton 核心。此編譯會在第一次啟動時執行，並根據模型大小大幅影響開始時間。

本主題說明如何最佳化 Run：ai Model Streamer 效能和 torch.compile 快取，以減少模型載入程序的兩個階段。如需在 Amazon EKS 上部署 vLLM 以進行推論的完整程序，請參閱 the section called “載入並服務模型”。

最佳化 EKS Auto 模式上的 S3 網路路徑

如果您在 EKS Auto Mode 上執行，且私有子網路中有 GPU 節點，建議您使用適用於 S3 的閘道 VPC 端點來最佳化節點與 S3 之間的網路路徑。使用閘道 VPC 端點時，對 S3 的流量會保留在 AWS 網路上，並完全略過 NAT 閘道，因此沒有共用頻寬上限和每 GB NAT 資料處理費用。如果沒有閘道 VPC 端點，當流量周遊 NAT 閘道時，NAT 閘道會在多個節點同時提取模型時，在擴展期間成為共用瓶頸。

EKS Auto Mode 通常會將節點放置在私有子網路中，因此 S3 的流量預設會流經 NAT Gateway。NAT Gateway 為每個目的地提供高達 100 Gbps 的頻寬和 55，000 個同時連線。不過，該頻寬會跨私有子網路中的每個節點共用。在擴展事件期間，多個節點一次下載完整模型會爭用相同的 NAT Gateway 頻寬，這可能會減慢載入所有模型的模型權重。

Run：ai 模型串流器效能調校

vLLM 和 SGLang 等推論引擎使用 Run：ai Model Streamer 作為在推論啟動期間載入權重的替代機制。

根據預設，從 S3 下載模型權重檔案時，Run：ai Model Streamer 會使用保守並行設定。增加下載並行和區塊大小可減少模型權重載入時間，方法是平行下載更多資料。

計算最佳並行

將最佳並行值計算為：

```
concurrency = ceil(total_model_size_gb / chunk_size_gb)
```

根據模型大小和區塊大小來取代您使用`concurrency`的值。如需一些範例，請參閱下表。例如，使用 67 GB 模型和 4 GB 區塊大小：`ceil(67 / 4) = 17`。

模型大小	區塊大小	並行
10 GB	4 GB	3
67 GB	4 GB	17
140 GB	4 GB	35

套用組態

將下列引數和環境變數新增至您的推論容器規格：

- `--tensor-parallel-size` – 張量平行 (TP) 程度通常是根據模型大小在 GPUs 記憶體中擬合模型所需的最小 GPU 數量。例如，`p5.48xlarge`執行個體類型的 67 GB 模型至少需要 2 個 GPUs，因此請將它設定為 2。
- `concurrency` 和 `distributed` (在 中`--model-loader-extra-config`) – `concurrency`設定為模型的計算值。`true` 只有在使用張量平行處理 (TP > 1) 時，才將 `distributed`設定為 。啟用時，每個張量平行排名會直接從 Amazon S3 串流自己的權重碎片，而不是排名 0 載入所有權

重並廣播到其他排名。這可大幅改善多 GPU 部署的載入效能。針對 TP=1 將其保留為未設定（或 `false`），其中沒有提供好處。此選項需要 vLLM V1 架構。它與 不相容`--enforce-eager`，這會強制 V0 路徑；同時使用兩者將錯誤或無提示地返回非分佈載入。

- `RUNAI_STREAMER_CHUNK_BYTESIZE` – 4 GB 區塊大小。此值一致地顯示跨基準的最佳效能。較大的區塊可減少 S3 請求的數量，並改善高頻寬執行個體的輸送量。
- `RUNAI_STREAMER_S3_REQUEST_TIMEOUT_MS` – 依請求逾時，以毫秒為單位。允許對慢速 S3 回應進行更快速的重試。
- `RUNAI_STREAMER_S3_LOW_SPEED_LIMIT` – 在將請求視為緩慢並重試之前，每秒以位元組為單位的最低傳輸速度。

```
containers:
- name: vllm-inference
  image: vllm/vllm-openai:v0.21.0
  command:
    - python3
    - -m
    - vllm.entrypoints.openai.api_server
  args:
  # ... your existing args ...
  - --model=s3://<MODEL_PATH>
  - --tensor-parallel-size=2
  - --load-format=runai_streamer
  - --model-loader-extra-config={"concurrency":17,"distributed":true}
  env:
  - name: RUNAI_STREAMER_CHUNK_BYTESIZE
    value: "4294967296"
  - name: RUNAI_STREAMER_S3_REQUEST_TIMEOUT_MS
    value: "3000"
  - name: RUNAI_STREAMER_S3_LOW_SPEED_LIMIT
    value: "1048576"
```

最佳化 torch.compile 冷啟動

推論服務程序中`torch.compile`的步驟會追蹤模型的運算圖形（數學運算序列），並將其編譯為最佳化的融合 CUDA/Triton 核心。它不會編譯模型權重，只會編譯轉換模型權重的操作。

推論引擎使用 torch.compile，因為它會自動提供顯著的輸送量改善，而不需要每個模型架構的自訂核心工程：

- 核心融合 — 將多個小型操作（剩餘新增、 layernorm、啟用）融合到單一核心中，減少 GPU 記憶體往返。
- 啟動較少的核心 — 轉換器層從 ~15-30 個單獨的 CUDA 核心下降到 ~10 個融合核心，在每次啟動時節省 CPU 額外負荷。
- Python 從熱路徑中移除 — 整個向前傳遞會成為 C++ 執行計劃，消除操作之間的 Python 解譯器額外負荷。
- 更好的 CUDA 圖形相容性 — 編譯的靜態圖形會以接近零的 CPU 額外負荷進行擷取和重播。
- 自動最佳化 — 適用於任何模型架構。相較於急迫模式，輸送量可改善 5-30%。

下表顯示使用 torch.compile 的常見推論服務引擎。

引擎	torch.compile 用量
vLLM	預設啟用 (V1 架構)
SGLang	透過 選用 `--enable-torch-compile`
TensorRT-LLM	新路徑與舊版引擎建置一起支援它

torch.compile 冷啟動問題

torch.compile 的權衡是第一個推論 Pod 必須編譯，才能處理請求。此編譯可能需要幾分鐘的時間，視模型大小而定。編譯的成品很小 (60 GB 模型約為 15 MB)，但可大幅增加冷啟動時間。由於成品是小型且決定性的，因此您可以快取和重複使用它們，以消除後續 Pod 和節點啟動時的冷啟動懲罰。

成品包含：

- 產生的 Triton 核心來源檔案
- 編譯的核心二進位檔 (`.cubin`)
- 序列核心呼叫的圖形結構

--enforce-eager 權衡

vLLM 預設會啟用 `torch.compile`和 CUDA 圖形擷取。`--enforce-eager` 標記會同時關閉並在急迫模式下執行模型，其中每個操作都會立即透過 Python 解譯器執行。由於急切模式會同時略過編譯和

圖形擷取，因此一些快速入門指南 - 包括 中的基本部署 the section called “載入並服務模型” - 用來更快--enforce-eager地啟動 Pod。

--enforce-eager 是偵錯、記憶體限制部署或模型架構的有效選擇，這些架構無法順利編譯，因此您可以在生產環境中使用它。雖然它可以減輕torch.compile冷啟動懲罰，但我們建議您使用其他方法，例如本頁後續章節中詳述的方法，以在生產環境中維持執行期效能。

在生產環境中部署--enforce-eager之前，了解 的啟動時間與執行時間效能權衡。

面向	使用 **--enforce-eager**（急切模式）	預設 (torch.compile + CUDA 圖形）
啟動時間	快速 — 無編譯或圖形擷取步驟	冷啟動緩慢 （ 由 the section called “快取相同節點上的 torch.compile 成品”和 解決的問題the section called “新節點上的暖機前 torch.compile 快取”)
穩定狀態輸送量	基準	比核心融合高約 5–30%
每個金鑰延遲 （ 小批次 ）	CPU 啟動額外負荷較高	更低：CUDA 圖形會以單一單位重播核心啟動
記憶體	更低且更可預測	較高 — 圖形擷取預先配置緩衝集區
偵錯能力	清除每個操作堆疊追蹤	產生的核心內部出現錯誤

快取相同節點上的 torch.compile 成品

當您使用支援 的推論引擎時torch.compile，例如 vLLM （ 預設為啟用 ） 或 SGLang （ 透過 啟用 ），就會套用此技術--enable-torch-compile。只有在 torch.compile 處於作用中狀態時，也就是--enforce-eager未設定 時，才會運作。

> **Important**
>
> 如果停用 torch.compile ，則此改進沒有作用。在 vLLM 中，--enforce-eager旗標會torch.compile完全停用，因此不會編譯或快取任何成品。如果您the section called “載入

並服務模型”使用 在 中遵循基本部署--enforce-eager，vLLM 會建立快取目錄，但絕不會寫入快取目錄。套用此技術--enforce-eager之前，請先移除 。

當推論引擎在第一次啟動時編譯模型的運算圖表時，您可以在節點的本機儲存體上快取產生的最佳化核心。相同節點上的後續 Pod 會重複使用快取的成品，並完全略過編譯步驟，進而大幅縮短啟動時間。

新增快取環境變數

將下列環境變數新增至推論容器規格，以將 torch.compile 和 Triton 快取導向持久性主機路徑。我們建議使用節點的本機 NVMe 執行個體存放區，而不是 的根 Amazon Elastic Block Store (Amazon EBS) 磁碟區hostPath。如需範例，請參閱下一節。

```
containers:
- name: vllm-inference
  env:
  # torch.compile cache
  - name: XDG_CACHE_HOME
    value: "/compile-cache"
  - name: TORCHINDUCTOR_CACHE_DIR
    value: "/compile-cache/inductor"
  - name: TRITON_CACHE_DIR
    value: "/compile-cache/triton"
  volumeMounts:
  - name: compile-cache
    mountPath: /compile-cache
volumes:
- name: compile-cache
  hostPath:
    path: /mnt/k8s-disks/0/compile-cache
    type: DirectoryOrCreate
```

將快取路徑設定為 NVMe 執行個體存放區

編譯的成品和串流權重受益於快速的本機儲存。在具有 NVMe 執行個體存放區的 GPU 執行個體 （ 例如 G-family 和 P-family 執行個體 ） 上，執行個體存放區提供大約 30 GB/s，相較於根 Amazon EBS 磁碟區大約 1 GB/s。將快取導向 NVMe hostPath 掛載點，以最佳化輸送量。

> **Important**
>
> hostPath 磁碟區使用 type: DirectoryOrCreate。如果您將其指向非 NVMe 執行個體存放區支援的路徑，Kubernetes 會改為在根 Amazon EBS 磁碟區上以無提示方式建立目錄。快取仍然有效，但您會失去 NVMe 效能優勢，沒有錯誤或警告。

NVMe 掛載點，以及 EKS Auto Mode 和自我管理節點之間的啟用方式：

運算	NVMe 掛載點	如何啟用 NVMe 執行個體存放區
EKS 自動模式	/mnt/.ephemeral	根據請求的暫時性儲存體動態啟用。EKS Auto Mode NVMe 只有在 NodeClass 中ephemeralStorage.size 請求的 小於執行個體的可用 NVMe NVMe 容量時，才會將執行個體存放區格式化並掛載為 RAID 0 陣列。如果請求的 ephemeralStorage.size 等於或大於 NVMe 容量，EKS Auto Mode 不會使用執行個體存放區，而是由根 EBS 磁碟區支援路徑。
自我管理 Karpenter	/mnt/k8s-disks/0	在 Karpenter instanceStorePolicy: RAID0 中設定 EC2NodeClass 。如果沒有它，Karpenter 會忽略執行個體存放區磁碟區，且路徑不受 NVMe 支援。

對於 EKS Auto 模式，在您的容器規格/mnt/.ephemeral/compile-cache中將 hostPath 設定為：

```
volumes:
- name: compile-cache
  hostPath:
    path: /mnt/.ephemeral/compile-cache
    type: DirectoryOrCreate
```

對於自我管理的 Karpenter，請在容器規格/mnt/k8s-disks/0/compile-cache中將 hostPath 設定為 ：

```
volumes:
- name: compile-cache
  hostPath:
    path: /mnt/k8s-disks/0/compile-cache
    type: DirectoryOrCreate
```

範例結果

1. 節點上的第一個 Pod 會執行 torch.compile，並將編譯的核心寫入主機/compile-cache上的 。
2. 相同節點上的後續 Pod 會掛載現有的快取並完全略過編譯，將 torch.compile 冷啟動從 ~50–80 秒縮短為 ~4–6 秒。

下表顯示不同模型大小的相同節點快取改進：

模型大小	第一個 Pod torch.compile （無快取）	後續 Pods torch.compile （相同節點）
60 GB	~53 秒	~6 秒
140 GB	~60 秒	~6 秒
640 GB	~80 秒	~6 秒

新節點上的暖機前 torch.compile 快取

當您跨節點使用同質 GPUs、張量平行處理、模型和 PyTorch 版本執行多節點推論時，就會套用此技術。中的技術the section called “快取相同節點上的 torch.compile 成品”著重於單一節點案例，但在擴展事件期間新增的新節點會從空的 torch.compile 快取開始。若要進一步縮短新初始化節點的冷啟動時間，您可以實作快取機制，將編譯的 torch.compile 成品存放在 S3 中，並在它們加入叢集時預先將其下載到新節點。

一般方法為：

1. 第一個 Pod 在第一個節點編譯模型後，將 torch.compile 成品 (~15 MB) 上傳至 S3 儲存貯體。
2. 當新節點加入叢集時，請在排定推論 Pod 之前，將快取成品下載至節點的本機儲存體。

例如，您可以實作在封裝 torch.compile 快取並將其上傳至 S3 的 GPU 節點上執行的 DaemonSet。它也可以將該快取同步到本機節點儲存，然後再將推論 Pod 排程在新節點上。

跨節點快取的考量事項

當您跨節點快取 torch.compile 成品時，只有在產生快取的節點與使用快取的節點之間符合這些參數時，編譯的核心才有效。您的快取機制必須考慮所有這些項目。任何參數不相符都會產生無效的快取，強制重新編譯或導致執行時間錯誤。您的快取工具必須透過這些參數來區分成品，例如，將成品整合到 S3 物件金鑰或快取目錄結構中。

參數	為什麼它很重要
GPU 類型	編譯的核心是 GPU-architecture-specific （例如，H100 為 sm_90，L4 為 sm_89)。
Tensor 平行處理 (TP)	不同的 TP 度會產生不同的運算圖形分割區。
模型	每個模型架構和大小會編譯成不同的核心。
PyTorch 版本	torch.compile 和 Triton 編譯器內部可能會在版本之間帶來突破性的變更。

範例結果

使用跨節點快取預暖 (Qwen3-6-35B-A3B、67 GB、2x GPU 搭配 p5.48xlarge 上的 TP=2) 進行定向測試時，新擴展節點上的第一個 Pod 與已暖節點上後續 Pod 的啟動時間相同：

案例	第一個 Pod	第二個 Pod
沒有跨節點快取 （新節點）	65 秒	16 秒
使用跨節點快取 （新節點）	16 秒	16 秒

部署範例

下列範例將 Run：ai Model Streamer 效能調校和 torch.compile 快取合併為單一 vLLM 部署資訊清單。將預留位置值取代為您自己的組態：

- `serviceAccountName` – 具有對模型儲存貯體具有 Amazon S3 讀取存取權之 IAM 角色的服務帳戶。

- nodeSelector (karpenter.sh/nodepool) – GPU 節點集區名稱 （ 例如 gpu-nodepool-g6e-12xlarge)。
- --model – 模型權重的 Amazon S3 路徑。
- --model-loader-extra-config – concurrency根據您的模型大小進行設定：ceil(total_model_size_gb / chunk_size_gb)。例如，大小為 4 GB 的 67 GB 模型會提供 ceil(67 / 4) = 17。
- --tensor-parallel-size – 將張量平行 (TP) 程度設定為符合記憶體中模型所需的最小 GPUs 數量。
- hostPath path – 適用於使用 Karpenter 自我管理節點的 NVMe 執行個體存放區掛載點。在 EKS Auto 模式下，請/mnt/.ephemeral/compile-cache改用 。如需詳細資訊，請參閱 the section called “快取相同節點上的 torch.compile 成品”。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vllm-inference
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: vllm-inference
  template:
    metadata:
      labels:
        app: vllm-inference
    spec:
      serviceAccountName: <SERVICE_ACCOUNT_NAME>
      nodeSelector:
        karpenter.sh/nodepool: <GPU_NODEPOOL>
      tolerations:
        - key: nvidia.com/gpu
          operator: Exists
          effect: NoSchedule
      containers:
        - name: vllm
          image: vllm/vllm-openai:v0.21.0
          command:
            - python3
```

```
        - -m
        - vllm.entrypoints.openai.api_server
      args:
        - --model=s3://<BUCKET_NAME>/<MODEL_PATH>
        - --load-format=runai_streamer
        - --model-loader-extra-config={"concurrency":17,"distributed":true}
        - --tensor-parallel-size=2
        - --max-model-len=8192
        - --host=0.0.0.0
        - --port=8000
      ports:
        - containerPort: 8000
          name: http
      env:
        # Run:ai streamer tuning
        - name: RUNAI_STREAMER_CHUNK_BYTESIZE
          value: "4294967296"
        - name: RUNAI_STREAMER_S3_REQUEST_TIMEOUT_MS
          value: "3000"
        - name: RUNAI_STREAMER_S3_LOW_SPEED_LIMIT
          value: "1048576"
        # torch.compile cache
        - name: XDG_CACHE_HOME
          value: "/compile-cache"
        - name: TORCHINDUCTOR_CACHE_DIR
          value: "/compile-cache/inductor"
        - name: TRITON_CACHE_DIR
          value: "/compile-cache/triton"
      resources:
        requests:
          cpu: "12"
          memory: 80Gi
          nvidia.com/gpu: "2"
        limits:
          nvidia.com/gpu: "2"
      volumeMounts:
        - name: compile-cache
          mountPath: /compile-cache
      startupProbe:
        httpGet:
          path: /health
          port: 8000
        periodSeconds: 10
        failureThreshold: 60
```

```
            initialDelaySeconds: 30
          readinessProbe:
            httpGet:
              path: /health
              port: 8000
            periodSeconds: 5
            timeoutSeconds: 3
      volumes:
        - name: compile-cache
          hostPath:
            path: /mnt/k8s-disks/0/compile-cache
            type: DirectoryOrCreate
```

針對 AI/ML 工作負載的 Amazon EKS 叢集組態

> **Tip**
>
> 註冊即將舉行的 Amazon EKS AI/ML 研討會。

本節旨在協助您設定針對 AI/ML 工作負載最佳化的 Amazon EKS 叢集。您可以找到使用 Linux 和 Windows 最佳化 AMIs 執行 GPU 加速容器、使用 Elastic Fabric Adapter (EFA) 設定訓練叢集以實現高效能聯網，以及使用 AWS Inferentia 執行個體建立推論叢集的指引，包括先決條件、step-by-step程序和部署考量。

主題

- 針對 GPU 執行個體使用 EKS 最佳化的加速 AMIs
- 執行 GPU 加速容器 (Windows on EC2 G 系列)
- 使用 Elastic Fabric Adapter 在 Amazon EKS 上執行機器學習訓練
- 將 AWS Inferentia 執行個體與 Amazon EKS 搭配使用以進行Machine Learning

針對 GPU 執行個體使用 EKS 最佳化的加速 AMIs

Amazon EKS 支援 GPU 執行個體的 EKS 最佳化 Amazon Linux 和 Bottlerocket AMIs。EKS 最佳化的加速 AMIs 透過為加速 Kubernetes 堆疊提供預先建置且經過驗證的作業系統映像，簡化在 EKS 叢集中執行 AI 和 ML 工作負載。除了標準 EKS 最佳化 AMIs 中包含的核心 Kubernetes 元件之外，EKS 最佳化加速 AMIs 還包含執行 NVIDIA GPU 和 P EC2 執行個體所需的核心模組G和驅動程式，以及 EKS 叢集中的 AWS GPU Inferentia 和 Trainium EC2 執行個體。

下表顯示每個 EKS 最佳化加速 AMI 變體支援的 GPU 執行個體類型。如需 AMI 變體的最新更新，請參閱 GitHub 上的 EKS 最佳化 AL2023 版本和 Bottlerocket 版本。

EKS AMI 變體	EC2 執行個體類型
AL2023 x86_64 NVIDIA	p6-b300、p6-b200、p5、p5e、p5en、p4d、p4de、p3、p3dn、g7e、gr6、g6、g6e、g6f、gr6f、g5、g4dn
AL2023 ARM NVIDIA	p6e-gb200、p6e-gb300、g5g
AL2023 x86_64 Neuron	inf1、inf2、trn1、trn2
Bottlerocket x86_64 aws-k8s-nvidia	p6-b300、p6-b200、p5、p5e、p5en、p4d、p4de、p3、p3dn、g7e、gr6、g6、g6e、g6f、gr6f、g5、g4dn
Bottlerocket aarch64/arm64 aws-k8s-nvidia	g5g
Bottlerocket x86_64 aws-k8s	inf1、inf2、trn1、trn2

Important

G7 EC2 執行個體類型需要 NVIDIA 驅動程式 595 版或更新版本。EKS 最佳化加速 AMIs 目前包含不支援 G7 執行個體的 NVIDIA 驅動程式 580 版。若要搭配 Amazon EKS 使用 G7 執行個體，您必須使用 EKS AMI 建置指令碼搭配 NVIDIA 驅動程式 595 版來建置自訂 AMI。
如果您使用 Karpenter 搭配 EKS 進行節點佈建和自動擴展，建議您從使用自動 AMI 選擇的 NodePools 中排除g7執行個體系列。如需如何設定 Karpenter NodeClass 以使用g7執行個體的自訂 AMI，請參閱 Karpenter 文件`amiSelectorTerms`中的 。

EKS 最佳化 NVIDIA AMIs

使用 EKS 最佳化 NVIDIA AMIs，即表示您同意 NVIDIA 的雲端最終使用者授權合約 (EULA)。

若要尋找最新的 EKS 最佳化 NVIDIA AMIs，請參閱 the section called “獲取最新 ID”和 the section called “獲取最新 ID”。

搭配 EKS 最佳化 AL2023 或 Bottlerocket NVIDIA AMIs 使用 Amazon Elastic Fabric Adapter (EFA) 時，您必須分別安裝 EFA 裝置外掛程式。如需詳細資訊，請參閱the section called “使用 EFA 設定訓練叢集”。

EKS AL2023 NVIDIA AMIs

搭配 EKS 最佳化 AL2023 NVIDIA AMI 使用 NVIDIA GPU 運算子時，您必須停用驅動程式和工具組的運算子安裝，因為這些已包含在 EKS AMIs中。 AL2023 AMIs EKS 最佳化 AL2023 NVIDIA AMIs 不包含 NVIDIA Kubernetes 裝置外掛程式或 NVIDIA DRA 驅動程式，且必須單獨安裝。如需詳細資訊，請參閱the section called “NVIDIA GPU”。

除了標準 EKS AMI 元件之外，EKS 最佳化 AL2023 NVIDIA AMIs 還包含下列元件。

- NVIDIA 驅動程式
- NVIDIA CUDA 使用者模式驅動程式
- NVIDIA 容器工具組
- NVIDIA 布料管理員
- NVIDIA 持久性
- NVIDIA IMEX 驅動程式
- NVIDIA NVLink 子網路管理員
- NVIDIA GDRCopy 驅動程式
- EFA 最小值 （核心模組和 rdma-core)

如需 NVIDIA CUDA 使用者模式驅動程式和應用程式容器內所用 CUDA 執行時間/程式庫的詳細資訊，請參閱 NVIDIA 文件。從 顯示的 CUDA 版本`nvidia-smi`是安裝在主機上的 NVIDIA CUDA 使用者模式驅動程式版本，必須與應用程式容器中使用的 CUDA 執行時間/程式庫相容。

EKS 最佳化 AL2023 NVIDIA AMIs 支援 Kubernetes 版本 1.33 及更高版本的核心 6.12，以及所有 Kubernetes 版本的 NVIDIA 驅動程式 580 版本。使用 CUDA 13+ 需要 NVIDIA 580 驅動程式。

如需 AMIs 中元件版本的詳細資訊，請參閱 GitHub 上的 EKS 最佳化 AL2023 版本。如需 EKS AMI 如何設定 NVIDIA 相依性的詳細資訊，請參閱 EKS AL2023 NVIDIA AMIs 安裝指令碼和核心載入指令碼。您可以使用 `dnf list installed`命令，在執行中的 EC2 執行個體上找到已安裝套件及其版本的清單。

使用 EKS 最佳化 AMIs 做為基礎建置自訂 AMIs 時，不建議或支援執行作業系統升級 （即 `dnf upgrade`) 或升級 EKS 最佳化 AMIs 中包含的任何 Kubernetes 或 GPU 套件，因為這可能會破壞元

件相容性。如果您確實升級 EKS 最佳化 AMIs 中包含的作業系統或套件，建議您在部署到生產環境之前，先在開發或預備環境中徹底測試。

為 GPU 執行個體建置自訂 AMIs 時，建議您為要執行的每個執行個體類型產生和系列分別建置自訂 AMIs。EKS 最佳化加速 AMIs 會根據基礎執行個體類型產生和系列，在執行時間選擇性地安裝驅動程式和套件。如需詳細資訊，請參閱安裝和執行時間的 EKS AMI 指令碼。

EKS Bottlerocket NVIDIA AMIs

搭配 EKS 最佳化 Bottlerocket NVIDIA AMIs 使用 NVIDIA GPU 運算子時，您必須停用驅動程式、工具組和裝置外掛程式的運算子安裝，因為這些已包含在 EKS AMIs 中。

除了標準 EKS AMI 元件之外，EKS 最佳化 Bottlerocket NVIDIA AMIs 還包含下列元件。EFA （核心模組和 rdma-core) 的最低相依性安裝在所有 Bottlerocket 變體中。

- NVIDIA Kubernetes 裝置外掛程式
- NVIDIA 驅動程式
- NVIDIA CUDA 使用者模式驅動程式
- NVIDIA 容器工具組
- NVIDIA 布料管理員
- NVIDIA 持久性
- NVIDIA IMEX 驅動程式
- NVIDIA NVLink 子網路管理員
- NVIDIA MIG 管理員

如需 NVIDIA CUDA 使用者模式驅動程式和應用程式容器內所用 CUDA 執行時間/程式庫的詳細資訊，請參閱 NVIDIA 文件。從 顯示的 CUDA 版本`nvidia-smi`是安裝在主機上的 NVIDIA CUDA 使用者模式驅動程式版本，必須與應用程式容器中使用的 CUDA 執行時間/程式庫相容。

如需已安裝套件及其版本的詳細資訊，請參閱 Bottlerocket 文件中的 Bottlerocket 版本資訊。EKS 最佳化 Bottlerocket NVIDIA AMIs支援 Kubernetes 版本 1.33 及更高版本的核心 6.12，以及 Kubernetes 版本 1.34 及更高版本的 NVIDIA 驅動程式 580 版本。使用 CUDA 13+ 需要 NVIDIA 580 驅動程式。

EKS 最佳化 Neuron AMIs

如需如何使用 Neuron 搭配 Amazon EKS 執行訓練和推論工作負載的詳細資訊，請參閱下列參考：

- Containers - Kubernetes - Neuron 文件入門 AWS
- GitHub 上 AWS Neuron EKS 範例中的訓練範例
- 在 Amazon EKS 上使用 Inferentia 部署 ML 推論工作負載

若要尋找最新的 EKS 最佳化 Neuron AMIs，請參閱 the section called “獲取最新 ID”和 the section called “獲取最新 ID”。

搭配 EKS 最佳化 AL2023 或 Bottlerocket Neuron AMIs 使用 Amazon Elastic Fabric Adapter (EFA) 時，您必須分別安裝 EFA 裝置外掛程式。如需詳細資訊，請參閱the section called “使用 EFA 設定訓練叢集”。

EKS AL2023 Neuron AMIs

EKS 最佳化 AL2023 Neuron AMIs 不包含 Neuron DRA 驅動程式、Neuron Kubernetes 裝置外掛程式或 Neuron Kubernetes 排程器延伸模組，且這些必須單獨安裝。如需詳細資訊，請參閱the section called “Neuron 加速器”。

除了標準 EKS AMI 元件之外，EKS 最佳化 AL2023 Neuron AMIs 還包含下列元件。

- Neuron 驅動程式 (aws-neuronx-dkms)
- Neuron 工具 (aws-neuronx-tools)
- EFA 最小值 （ 核心模組和 rdma-core)

如需 EKS AMI 如何設定 Neuron 相依性的詳細資訊，請參閱 EKS AL2023 Neuron AMIs 安裝指令碼。請參閱 GitHub 上的 EKS 最佳化 AL2023 版本，以查看 AMIs 中包含的元件版本。您可以使用 `dnf list installed`命令，在執行中的 EC2 執行個體上找到已安裝套件及其版本的清單。

EKS Bottlerocket Neuron AMIs

標準 Bottlerocket 變體 (aws-k8s) 包含在 AWS Inferentia 或 Trainium EC2 執行個體上執行時自動偵測和載入的 Neuron 相依性。

EKS 最佳化 Bottlerocket AMIs 不包含 Neuron DRA 驅動程式、Neuron Kubernetes 裝置外掛程式或 Neuron Kubernetes 排程器延伸模組，且必須單獨安裝。如需詳細資訊，請參閱the section called “Neuron 加速器”。

除了標準 EKS AMI 元件之外，EKS 最佳化 Bottlerocket Neuron AMIs 還包含下列元件。

- Neuron 驅動程式 (aws-neuronx-dkms)
- EFA 最小值 （ 核心模組和 rdma-core)

搭配 Neuron 執行個體使用 EKS 最佳化 Bottlerocket AMIs 時，必須在 Bottlerocket 使用者資料中設定下列項目。此設定允許容器根據工作負載規格中提供的 `runAsUser`和 `runAsGroup`值取得掛載 Neuron 裝置的擁有權。如需 Bottlerocket 中 Neuron 支援的詳細資訊，請參閱 GitHub 上的 Quickstart on EKS 讀我檔案。

```
[settings]
[settings.kubernetes]
device-ownership-from-security-context = true
```

如需 EKS 最佳化 Bottlerocket AMI 中 Neuron 驅動程式版本的資訊，請參閱 Bottlerocket 核心套件變更日誌。 AMIs

執行 GPU 加速容器 (Windows on EC2 G 系列)

> **Important**
>
> Kubernetes Device Plugin for DirectX by TensorWorks 是一種第三方工具，未經 背書、支援或維護 AWS。 AWS 對此外掛程式的安全性、可靠性或效能不負任何責任。

了解如何使用 NVIDIA GPU 搭配 TensorWorks 的 DirectX 專用 Kubernetes 裝置外掛程式，在 Amazon EKS (Elastic Kubernetes Service) 上執行 GPU 加速 Windows 容器工作負載。如需詳細資訊，請參閱 DirectX 專用 Kubernetes 裝置外掛程式。

對於為 Windows 容器設定 GPU 加速，我們提供兩種主要方法：

- 選項 1：使用預先安裝的必要 GPU 驅動程式，建置自訂 EKS Windows 最佳化 AMI。
  - 當您需要一個一致且預先設定的環境來執行 GPU 加速 Windows 容器，並且您能夠投入額外努力來建置和維護自訂 AMI 時，請使用此方法。
- 選項 2：啟動執行個體後，在 EKS 工作節點上安裝必要的 GPU 驅動程式。
  - 當您想要更簡單的設定程序並且不介意在每個新的工作節點上安裝 GPU 驅動程式時，請使用此方法。當您評估 GPU 加速工作負載並對此建立原型時，則更適合開發環境。

您可以使用本指南中詳述的步驟，以藉此利用這兩種方法。

考量事項

本指南提供使用 NVIDIA GPU、NVIDIA GRID 驅動程式和 TensorWorks 的 DirectX 專用 Kubernetes 裝置外掛程式，為您的 Windows 容器安裝和設定 GPU 加速的步驟。這些步驟均已經過測試和驗證，可為 Amazon EKS 上的 Windows 容器工作負載提供 GPU 加速。如需有關相容驅動程式和裝置外掛程式的詳細資訊，請參閱 the section called “已知限制”。在繼續之前，請注意下列事項：

- 僅具有 NVIDIA GRID 驅動程式的 G 系列執行個體類型已經過測試和驗證，可搭配本指南使用。雖然其他執行個體類型和驅動程式組合或許也能執行 GPU 加速 Windows 容器，但他們可能需要本指南中未涵蓋的其他組態步驟。
- 僅 DirectX 型工作負載已經過測試和驗證，可搭配本指南使用。雖然 OpenGL、Vulkan 和 OpenCL 等其他 GPU API 可能相容於執行 GPU 加速 Windows 容器，但他們可能需要本指南中未涵蓋的其他組態步驟。
- 在執行 GPU 加速 Windows 容器之前，需要注意一些已知限制。如需詳細資訊，請參閱 the section called “已知限制” 一節。

先決條件

若要在 Amazon EKS 上啟用適用於 Windows 容器的 GPU 加速，您必須先準備好下列要求，才能繼續：

- 使用 Kubernetes v1.27 或更新版本啟動 Amazon EKS 叢集。
- 為 Windows 節點 佈建 Windows Server 2022 或更新版本。
- 以 G 系列執行個體類型佈建 Windows 節點，例如 G4 或 G5。
- 使用 containerd `1.7.x` 或 `2.x.x` 為 Windows 節點佈建容器執行時期。(請參閱 the section called “獲取版本資訊”，以驗證 Amazon EKS 最佳化 AMI 中的 containerd 版本。)

在每個 Windows 節點上安裝 GPU 驅動程式

若要在 EKS 工作節點上安裝 NVIDIA GRID 驅動程式，請遵循 Amazon EC2 執行個體的 NVIDIA 驅動程式中概述的步驟。導覽至安裝選項 - 選項 3：GRID 驅動程式，並遵循安裝步驟。

為 Windows Server Core 安裝

對於沒有桌面體驗的 Windows Server Core，請使用下列命令，以無提示的方式安裝 NVIDIA GRID 驅動程式：

```
$nvidiaInstallerFilePath = nvidia-driver-installer.exe # Replace with path to installer
```

```
$installerArguments = "-s -clean -noreboot -noeula"
Start-Process -FilePath $nvidiaInstallerFilePath -ArgumentList $installerArguments -
Wait -NoNewWindow -PassThru
```

驗證您的安裝

執行下列 PowerShell 命令，以顯示有關執行個體上 GPU 的診斷資訊：

```
nvidia-smi
```

此命令會顯示 NVIDIA 驅動程式版本，以及有關 GPU 硬體的資訊。請確保此命令的輸出與您預期安裝的 NVIDIA GRID 驅動程式版本相符。

在每個節點上部署 GPU 裝置外掛程式

若要啟用探索並將 GPU 資源公開至 Windows 節點上的容器，您需要一個裝置外掛程式。透過在 EKS 叢集中將 Tensorworks 的 DirectX 裝置外掛程式作為 DaemonSet 執行，請在每個工作節點上予以部署。請遵循 README.md 中指定的安裝指南，其將包含下列步驟。建議：

- 在 `kube-system` 命名空間中部署裝置外掛程式。
- 為 DaemonSet 設定適當的資源限制，以確保它不會消耗節點上過多的資源。

> **Note**
>
> 裝置外掛程式 DaemonSet 將作為具有更高權限的主機程序容器，在每個節點上執行。建議實作 RBAC 控制，以限制對此 DaemonSet 的存取，因此只有授權使用者才能執行特殊權限命令。

執行 GPU 加速容器時，裝置外掛程式可支援兩種模式：

- 單一租用模式：此模式會將所有 GPU 資源專用於執行個體上的單一容器。使用下列命令，安裝具有單一租用支援的裝置外掛程式。如需詳細資訊，請參閱 README.md。

  ```
  kubectl apply -f "https://raw.githubusercontent.com/TensorWorks/directx-device-
  plugins/main/deployments/default-daemonsets.yml"
  ```

- 多租用模式：此模式允許在執行個體上的多個容器之間共用 GPU 資源。使用下列命令，安裝具有多租用支援的裝置外掛程式。如需詳細資訊，請參閱 README.md。

```
kubectl apply -f "https://raw.githubusercontent.com/TensorWorks/directx-device-
plugins/main/deployments/multitenancy-inline.yml"
```

或者，使用 ConfigMap 來指定多租用。

```
kubectl apply -f "https://raw.githubusercontent.com/TensorWorks/directx-device-
plugins/main/deployments/multitenancy-configmap.yml"
```

驗證裝置外掛程式部署

部署裝置外掛程式之後，請取代 <namespace>並執行下列命令，以確認 DirectX 裝置外掛程式在所有 Windows 節點上正確執行。

```
kubectl get ds device-plugin-wddm -n <namespace>
```

驗證容器是否已準備好部署

裝置外掛程式 DaemonSet 在採用 GPU 技術的 Windows 工作節點上執行後，請使用下列命令，以驗證每個節點是否具有可配置的 GPU。對應數量應與每個節點上的 DirectX 裝置數量相符。

```
kubectl get nodes "-o=custom-
columns=NAME:.metadata.name,DirectX:.status.allocatable.directx\.microsoft\.com/
display"
```

使用 GPU 加速執行 Windows 容器

啟動 Pod 之前，請在 directx.microsoft.com/display 中指定資源名稱 .spec.containers[].resources。這表示您的容器需要已啟用 GPU 的功能，而 kube-scheduler 會嘗試將 Pod 放置在具有可用 GPU 資源且預先設定的 Windows 節點上。

例如，請參閱以下範例命令，而該命令會啟動 Job，以執行 Monte Carlo 模擬來估算 pi 的值。此範例來自 DirectX 專用 Kubernetes 裝置外掛程式 GitHub 儲存庫，而該儲存庫具有多個範例可供選擇，而您可以執行這些範例來測試您的 Windows 節點 GPU 功能。

```
cat <<EOF | kubectl apply -f -
apiVersion: batch/v1
kind: Job
metadata:
  name: example-cuda-montecarlo-wddm
```

```
spec:
  template:
    spec:
      containers:
      - name: example-cuda-montecarlo-wddm
        image: "index.docker.io/tensorworks/example-cuda-montecarlo:0.0.1"
        resources:
          limits:
            directx.microsoft.com/display: 1
      nodeSelector:
        "kubernetes.io/os": windows
      restartPolicy: Never
  backoffLimit: 0
EOF
```

已知限制

所有 GPU 皆可使用

該執行個體上的所有 GPU 皆可供主機上的每個執行中的容器使用，即使您為指定容器請求特定數量的 GPU。此外，預設行為是，即使節點上有多個 GPU 可用，在主機上執行的所有容器都會使用具有索引 0 的 GPU。因此，若要讓多 GPU 任務正常運作，您必須明確指定要在應用程式程式碼中使用的特定 GPU 裝置。

配置裝置以用於應用程式的確切實作，將取決於您使用的程式設計語言或架構。例如，如果您使用 CUDA 程式設計，若要選取特定 GPU，您可以使用 函數 cudaSetDevice() 明確指定要在應用程式程式碼中使用的裝置。

由於存在會影響 Windows 容器的已知問題，需要明確指定裝置。您可以在 microsoft/Windows-Containers 問題 #333 中追蹤解決此問題的進度。下表代表此 GPU 配置行為的視覺化呈現和實際範例。

假設存在 EC2 執行個體類型 `g4dn.12xlarge` 的單一 Windows 節點，且其中隨附四個 GPU。假設在此執行個體上啟動三個 Pod。此表格顯示，無論每個容器請求的 GPU 數量為何，這三個 Pod 都可以存取執行個體上的所有四個 GPU，並且根據預設，會使用具有裝置索引 0 的 GPU。

Pod	請求的 GPU	實際 GPU 存取	預設 GPU 用量	可用的 GPU 索引	執行個體 GPU 總數
Pod 1	1 個 GPU	所有 4 個 GPU	具有索引 0 的 GPU	0、1、2、3	4

Pod	請求的 GPU	實際 GPU 存取	預設 GPU 用量	可用的 GPU 索引	執行個體 GPU 總數
Pod 2	2 GPU	所有 4 個 GPU	具有索引 0 的 GPU	0、1、2、3	4
Pod 3	1 個 GPU	所有 4 個 GPU	具有索引 0 的 GPU	0、1、2、3	4

Kubernetes 裝置外掛程式支援

Kubernetes 裝置外掛程式的 NVIDIA 的正式實作不支援 Windows。您可以在 NVIDIA/k8s-device-plugin 問題 #419 中追蹤新增官方 Windows 支援的進度。

GPU 運算執行個體限制

視 AWS 您的帳戶組態而定，您可以啟動的 Amazon EC2 GPU 運算執行個體數量和類型可能會有服務限制。如果您需要額外容量，您可以請求提高配額。

必須建置 Windows GPU 最佳化 AMI

Amazon EKS 並未提供 EKS Windows GPU 最佳化 AMI 或 EC2 Image Builder 受管元件。您需要遵循本指南中的步驟，以使用預先安裝的必要 GPU 驅動程式建置自訂 EKS Windows 最佳化 AMI，或在啟動執行個體後，在 EKS 工作節點上安裝必要的 GPU 驅動程式。

不支援 Inferentia 和 Trainium

Windows 不支援 AWS Inferentia 和 AWS Trainium 型工作負載。

使用 Elastic Fabric Adapter 在 Amazon EKS 上執行機器學習訓練

本主題會說明如何將 Elastic Fabric Adapter (EFA) 與部署在 Amazon EKS 叢集中的 Pod 整合。Elastic Fabric Adapter (EFA) 是 Amazon EC2 執行個體的網路介面，可讓您在 AWS上大規模執行需要高層級節點間通訊的應用程式。其自訂的作業系統略過硬體介面可增強執行個體間通訊的效能，這對於擴展這些應用程式至關重要。藉由 EFA，使用訊息傳遞介面 (MPI) 的高效能運算 (HPC) 應用程式和使用 NVIDIA 集體通訊程式庫 (NCCL) 的機器學習 (ML) 應用程式可擴展到數千個 CPU 或 GPU。因此，您可以使用 AWS 雲端的隨需彈性和彈性，獲得現場部署 HPC 叢集的應用程式效能。將 EFA 與 Amazon EKS 叢集上執行的應用程式整合，可減少完成大規模分散式訓練工作負載的時間，而無需在叢集中新增其他執行個體。如需 EFA 的詳細資訊，請參閱 Elastic Fabric Adapter。

如需在 EKS 叢集中安裝和管理 EFA 裝置的資訊，包括 EFA DRA 驅動程式、EFA 裝置外掛程式、拓撲感知配置和網路介面組態，請參閱 the section called “EFA 裝置”。

具有 EFA 的執行個體類型

AWS EFA Kubernetes 裝置外掛程式支援所有具有 EFA 的 Amazon EC2 執行個體類型。若要查看具有 EFA 的所有執行個體類型的清單，請參閱《Amazon EC2 使用者指南》中的支援的執行個體類型。不過，為了快速執行 ML 應用程式，除了 EFA 之外，我們建議執行個體具有硬體加速晶片，例如 NVIDIA GPUs、AWS Inferentia 晶片或 AWS Trainium 晶片。若要查看具有硬體加速晶片和 EFA 的執行個體類型的清單，請參閱《Amazon EC2 使用者指南》中的加速運算。

當您比較執行個體類型以在它們之間進行選擇時，請考慮該執行個體類型可用的 EFA 網路卡數量，以及加速器卡數量、CPU 數量和記憶體數量。對於每張網路卡您最多可以指派一個 EFA。EFA 視為網路介面。若要查看每個具有 EFAs可用 EFA 數量，請參閱《Amazon EC2 使用者指南》中的網路卡清單。

EFA 和僅限 EFA 介面

Elastic Fabric Adapter (EFA) 是一種網路介面，結合了彈性網路轉接器 (ENA) 和 OS-bypass 介面的功能，採用 AWS 可擴展可靠資料包 (SRD) 通訊協定。EFA 功能可讓應用程式直接與硬體通訊，以實現低延遲傳輸。您可以選擇只使用僅限 EFA 介面存取 EFA 功能，以限制與相同可用區域內介面的通訊。

若要建立具有僅限 EFA 介面的節點，您必須使用自訂 EC2 啟動範本，並將 `InterfaceType` 設定為 `efa-only`。在自訂啟動範本中，您無法將網路卡 `0` 設定為僅限 EFA 介面，因為這是 EC2 執行個體的主要網路卡和網路介面。對於僅限 EFA 介面，您必須使用 VPC CNI 版本 `1.18.5` 或更新版本。如果您使用的是 Amazon Linux 2，則僅限 EFA 界面的 ami 版本必須是 `v20240928`或更新版本。

下列程序會引導您使用具有 NVIDIA GPUs 和 EFA 介面的節點來建立 `eksctl` EKS 叢集。您無法使用 `eksctl` 來建立使用僅限 EFA 介面的節點和節點群組。

先決條件

- 現有 Amazon EKS 叢集。如果您沒有現有的叢集，請使用 建立一個叢集開始使用。您的叢集必須部署在具有至少一個私有子網路的 VPC 中，其中具有足夠的可用 IP 地址，以便部署節點。私有子網路必須具有外部裝置 (例如 NAT 閘道) 所提供的傳出網際網路存取權。

  如果計劃使用 `eksctl` 來建立您的節點群組，`eksctl` 也會為您建立叢集。
- 在您的裝置或 AWS CloudShell 上安裝和設定的 AWS 命令列界面 (AWS CLI) 版本 `1.27.160` `2.12.3`或更新版本。若要檢查您目前的版本，請使用 `aws --version | cut -d / -f2 | cut -d ' ' -f1`。適用於 macOS 的 `yum`、 `apt-get`或 Homebrew 等套件管理員通常是最新版

本 CLI AWS 後面的幾個版本。若要安裝最新版本，請參閱《 AWS 命令列界面使用者指南》中的使用 aws 設定安裝 和 快速組態。 https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-quickstart.html#cli-configure-quickstart-config安裝在 AWS CloudShell 中的 AWS CLI 版本也可能是最新版本後面的幾個版本。若要更新它，請參閱《CloudShell AWS 使用者指南》中的將 CLI 安裝到您的主目錄。 AWS CloudShell

- kubectl 命令列工具安裝在您的裝置或 AWS CloudShell 上。版本可以與您的叢集 Kubernetes 版本相同，或是為最多比該版本更舊一版或更新一版的次要版本。例如，如果您的叢集版本為 1.29，則可以搭配使用 kubectl 1.28、1.29 或 1.30 版。若要安裝或升級 kubectl，請參閱 the section called “設定 kubectl 和 eksctl”。
- 在啟動支援多個 Elastic Fabric Adapter 的工作節點 (例如 p4d 或 p5) 之前，您必須先安裝版本 1.7.10 或更新版本的 Kubernetes 專用 Amazon VPC CNI 外掛程式。如需有關更新 Kubernetes 專用 Amazon VPC CNI 外掛程式版本的詳細資訊，請參閱 the section called “Amazon VPC CNI”。
- 若為 p6-b200 執行個體，您必須使用 EFA 裝置外掛程式版本 v0.5.6 或更新版本。

> **Important**
>
> 搭配 Kubernetes 採用 EFA 所需的一項重要考量，就是將大型頁面作為叢集中的資源進行設定和管理。如需詳細資訊，請參閱 Kubernetes 文件中的管理大型頁面。已安裝 EFA 驅動程式的 Amazon EC2 執行個體會預先分配 5128 2MiB 大型頁面，您可以請求作為資源在任務規格中耗用。

建立節點群組

下列程序可協助您建立具有 EFA 介面和 GPUDirect RDMA 之 p4d.24xlarge 支援的節點群組，並針對使用 EFA 的多節點 NCCL 效能執行範例 NVIDIA 集體通訊程式庫 (NCCL) 測試。此範例可用作使用 EFAs範本。

1. 決定您要在其中部署節點的 AWS 區域中可使用哪些支援 EFA 的 Amazon EC2 執行個體類型。將 *region-code* 取代為您要部署節點群組 AWS 的區域。

```
aws ec2 describe-instance-types --region region-code \
    --filters Name=network-info.efa-supported,Values=true \
    --query "InstanceTypes[*].[InstanceType]" --output text
```

   當您部署節點時，您要部署的執行個體類型必須在叢集所在的 AWS 區域中可用。

2. 判斷您想要部署的執行個體類型所在的哪一個 Availability Zone (可用區域) 為可用。在本教學課程中，會使用p5.48xlarge執行個體類型，且必須在您在上一個步驟中指定的 AWS 區域的輸出中傳回。在生產叢集中部署節點時，請使用上一個步驟中傳回的任何執行個體類型取代 *p5.48xlarge*。

```
aws ec2 describe-instance-type-offerings --region region-code \
    --location-type availability-zone --filters Name=instance-
type,Values=p4d.24xlarge,p5.48xlarge \
    --query 'InstanceTypeOfferings[*].Location' --output text
```

範例輸出如下。

```
us-west-2a    us-west-2c    us-west-2b
```

請注意傳回的 Availability Zone (可用區域) 以供稍後步驟使用。將節點部署到叢集時，您的 VPC 必須在輸出中傳回的其中一個 Availability Zone (可用區域) 中具有可用 IP 位址的子網路。

3. 使用 eksctl 建立節點群組。您需要在裝置0.215.0或 AWS CloudShell 上安裝版本 或更新版本的eksctl命令列工具。如需有關安裝或更新 eksctl 的指示，請參閱 eksctl 文件中的安裝一節。

   a. 將下列內容複製到名為 *efa-cluster.yaml* 的檔案。使用自己的取代範例值。您可以使用不同的執行個體取代 p5.48xlarge，但是如果您這樣做，請確保 availabilityZones 值是針對步驟 1 中執行個體類型傳回的可用區域。

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: my-efa-cluster
  region: region-code
  version: "1.XX"

iam:
  withOIDC: true

availabilityZones: ["us-west-2a", "us-west-2c"]

managedNodeGroups:
  - name: my-efa-ng
    instanceType: p5.48xlarge
```

```
    minSize: 1
    desiredCapacity: 2
    maxSize: 3
    availabilityZones: ["us-west-2a"]
    volumeSize: 300
    privateNetworking: true
    efaEnabled: true
```

b. 在現有叢集中建立受管節點群組。

```
eksctl create nodegroup -f efa-cluster.yaml
```

如果您沒有現有的叢集，則可以執行下列命令來建立叢集和節點群組。

```
eksctl create cluster -f efa-cluster.yaml
```

> **Note**
>
> 由於此範例中使用的執行個體類型具有 GPU，`eksctl` 會在使用 Amazon Linux 2. 時，為您自動在每個執行個體上安裝 NVIDIA Kubernetes 裝置外掛程式。Bottlerocket 不需要執行此操作，因為 NVIDIA 裝置外掛程式內建於 Bottlerocket 的 EKS NVIDIA 變體中。在節點群組組態中將 `efaEnabled` 設定為 `true` 時，`eksctl` 也會自動在節點上部署 EFA 裝置外掛程式。

搭配 EFA 使用 Bottlerocket

Bottlerocket AMI 版本 1.28.0 及更新版本包含對 EFA 的官方支援。若要將 Bottlerocket 用於已啟用 EFA 的節點，請在您的組態中指定 `amiFamily: Bottlerocket`。如果您需要使用自訂 AMI ID，則必須使用標準 `nodeGroups` 而不是 `managedNodeGroups`。

以下是範例組態：

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: my-efa-bottlerocket-cluster
  region: region-code
  version: "1.XX"
```

```
iam:
  withOIDC: true

availabilityZones: ["us-west-2a", "us-west-2c"]

managedNodeGroups:
  - name: my-efa-bottlerocket-ng
    instanceType: p5.48xlarge
    minSize: 1
    desiredCapacity: 2
    maxSize: 3
    availabilityZones: ["us-west-2a"]
    volumeSize: 300
    privateNetworking: true
    efaEnabled: true
    amiFamily: Bottlerocket
    bottlerocket:
      enableAdminContainer: true
      settings:
        kernel:
          sysctl:
            "vm.nr_hugepages": "3000"  # Configures 3000 * 2Mi = 6000Mi hugepages
```

上述 vm.nr_hugepages sysctl 設定會設定 2Mi 巨型分頁的數量。在此範例中，3000 表示 3000 * 2Mi = 6000Mi 的巨型分頁。

驗證 EFA 裝置外掛程式安裝

當您使用 efaEnabled: true 建立節點群組時，eksctl 會自動為您部署 EFA Kubernetes 裝置外掛程式。您可以驗證裝置外掛程式是否已安裝並正常運作：

1. 檢查 DaemonSet 狀態：

```
kubectl get daemonsets -n kube-system
```

輸出範例：

```
NAME                                  DESIRED   CURRENT   READY   UP-TO-DATE
 AVAILABLE   NODE SELECTOR   AGE
aws-efa-k8s-device-plugin-daemonset   2         2         2       2            2
      <none>          6m16s
```

```
...
```

在此，EFA 裝置外掛程式 DaemonSet 會在兩個節點上執行。兩者狀態皆為就緒及可用。

2. 接著，驗證 DaemonSet 建立的 Pod：

```
kubectl get pods -n kube-system -l name=aws-efa-k8s-device-plugin
```

輸出範例：

```
NAME                                          READY   STATUS    RESTARTS   AGE
aws-efa-k8s-device-plugin-daemonset-d68bs   1/1     Running   0          6m16s
aws-efa-k8s-device-plugin-daemonset-w4l8t   1/1     Running   0          6m16s
```

EFA 裝置外掛程式 Pod 處於「執行中」狀態，確認外掛程式已成功部署並可運作。

3. 驗證資源註冊：

您可以描述節點，進而確認 `vpc.amazonaws.com/efa` 資源已向 kubelet 註冊：

```
kubectl describe nodes
```

如果 EFA 資源已正確註冊，您即會看到該資源列於節點的容量和可配置資源下。例如：

```
Capacity:
  ...
  vpc.amazonaws.com/efa:  4
Allocatable:
  ...
  vpc.amazonaws.com/efa:  4
```

此輸出會確認節點可辨識 EFA 資源，使其可供請求它的 Pod 使用。

(選用) 測試 EFA 的效能

建議您測試 EFA 設定。您可以在 GitHub 上的 `aws-samples/awsome-distributed-training` 儲存庫中使用 NCCL 測試。NCCL 測試會使用 NVIDIA Collective Communications Library 評估網路的效能。下列步驟會在 Amazon EKS 上提交 NCCL 測試。

1. 部署 Kubeflow MPI 運算子：

對於 NCCL 測試，您可以套用 Kubeflow MPI 運算子。MPI 運算子可以很容易地在 Kubernetes 上執行 Allreduce 式分散式訓練。如需詳細資訊，請參閱 GitHub 上的 MPI 運算子。

2. 執行多節點 NCCL 效能測試，以驗證 GPUDirectRDMA/EFA：

   要透過 EFA 使用 GPUDirectRDMA 驗證 NCCL 效能，請執行標準 NCCL 效能測試。如需詳細資訊，請參閱 GitHub 上的官方 NCCL 測試儲存庫。

   請完成以下步驟，以執行雙節點 NCCL 效能測試。在 NCCL 測試任務範例中，每個工作者會請求八個 GPUs、5120Mi 的 hugepages-2Mi、四個 EFAs 和 8000Mi 的記憶體，這實際上表示每個工作者都會耗用p5.48xlarge執行個體的所有資源。

   a. 建立 MPIJob 資訊清單：

      將下列內容複製到名為 nccl-tests.yaml 的檔案：

```
apiVersion: kubeflow.org/v2beta1
kind: MPIJob
metadata:
  name: nccl-tests
spec:
  runPolicy:
    cleanPodPolicy: Running
    backoffLimit: 20
  slotsPerWorker: 8
  mpiReplicaSpecs:
    Launcher:
      replicas: 1
      template:
         spec:
          restartPolicy: OnFailure
          containers:
          - image: public.ecr.aws/hpc-cloud/nccl-tests:latest
            imagePullPolicy: IfNotPresent
            name: test-nccl-launcher
            env:
             - name: PATH
               value: $PATH:/opt/amazon/efa/bin:/usr/bin
            command:
            - /opt/amazon/openmpi/bin/mpirun
            - --allow-run-as-root
            - --tag-output
            - -np
```

```
            - "16"
            - -N
            - "8"
            - --bind-to
            - none
            - -x
            - PATH
            - -x
            - LD_LIBRARY_PATH
            - -x
            - NCCL_DEBUG=INFO
            - -x
            - NCCL_BUFFSIZE=8388608
            - -x
            - NCCL_P2P_NET_CHUNKSIZE=524288
            - -x
            - NCCL_TUNER_PLUGIN=/opt/amazon/ofi-nccl/lib/x86_64-linux-gnu/libnccl-
ofi-tuner.so
            - --mca
            - pml
            - ^cm,ucx
            - --mca
            - btl
            - tcp,self
            - --mca
            - btl_tcp_if_exclude
            - lo,docker0,veth_def_agent
            - /opt/nccl-tests/build/all_reduce_perf
            - -b
            - "8"
            - -e
            - "16G"
            - -f
            - "2"
            - -g
            - "1"
            - -c
            - "1"
            - -n
            - "100"
    Worker:
      replicas: 2
      template:
        spec:
```

```
      nodeSelector:
        node.kubernetes.io/instance-type: "p5.48xlarge"
      containers:
      - image: public.ecr.aws/hpc-cloud/nccl-tests:latest
        imagePullPolicy: IfNotPresent
        name: nccl-tests-worker
        volumeMounts:
        - name: shmem
          mountPath: /dev/shm
        resources:
          limits:
            nvidia.com/gpu: 8
            hugepages-2Mi: 5120Mi
            vpc.amazonaws.com/efa: 32
            memory: 32000Mi
          requests:
            nvidia.com/gpu: 8
            hugepages-2Mi: 5120Mi
            vpc.amazonaws.com/efa: 32
            memory: 32000Mi
      volumes:
      - name: shmem
        hostPath:
          path: /dev/shm
```

b. 套用 NCCL 測試 MPIJob：

套用資訊清單，以提交 MPIJob。這將建立兩個 p5.48xlarge Amazon EC2 執行個體。

```
kubectl apply -f nccl-tests.yaml
```

範例輸出如下。

```
mpijob.kubeflow.org/nccl-tests created
```

c. 確認任務已啟動 Pod：

檢視您的執行中 Pod。

```
kubectl get pods
```

範例輸出如下。

```
NAME                              READY   STATUS     RESTARTS   AGE
nccl-tests-launcher-nbql9    0/1     Init:0/1   0          2m49s
nccl-tests-worker-0          1/1     Running    0          2m49s
nccl-tests-worker-1          1/1     Running    0          2m49s
```

MPI 運算子會建立啟動器 Pod 和 2 個工作者 Pod (每個節點上有一個)。

d. 使用日誌確認任務已成功執行：

檢視 `nccl-tests-launcher` Pod 的日誌。使用輸出的值取代 *nbql9*。

```
kubectl logs -f nccl-tests-launcher-nbql9
```

如果測試成功完成，您可以部署使用 NVIDIA Collective Communications Library 的應用程式。

將 AWS Inferentia 執行個體與 Amazon EKS 搭配使用以進行Machine Learning

本主題描述了如何建立 Amazon EKS 叢集，其中具有執行 Amazon EC2 Inf1 執行個體的節點，以及 (選用) 如何部署範例應用程式。Amazon EC2 Inf1 執行個體採用 AWS Inferentia 晶片，由 自訂，AWS 可在雲端提供高效能和最低成本的推論。機器學習模型使用 AWS Neuron 部署到容器，這是一個專門的軟體開發套件 (SDK)，由編譯器、執行時間和分析工具組成，可最佳化 Inferentia 晶片的機器學習推論效能。 AWS Neuron 支援熱門的機器學習架構，例如 TensorFlow、PyTorch 和 MXNet。

> **Note**
>
> Neuron 裝置邏輯 ID 必須是連續的。如果請求多個 Neuron 裝置的 Pod 已排程在某個 `inf1.6xlarge` 或 `inf1.24xlarge` 執行個體類型 (具有多個 Neuron 裝置) 上，則如果 Kubernetes 排程器選取非連續的裝置 ID，該 Pod 將無法啟動。如需詳細資訊，請參閱 GitHub 上的裝置邏輯 ID 必須是連續的。

先決條件

- 將 `eksctl` 安裝在您的電腦上。如果您尚未安裝，請參閱 `eksctl` 文件中的安裝一節。
- 將 `kubectl` 安裝在您的電腦上。如需詳細資訊，請參閱the section called “設定 kubectl 和 eksctl”。
- (選用) 將 `python3` 安裝在您的電腦上。如果您尚未安裝，請參閱 Python 下載以取得安裝指示。

建立 叢集

1. 建立具有 Inf1 Amazon EC2 執行個體之節點的叢集。您可以使用任何 Inf1 執行個體類型取代 *inf1.2xlarge*。eksctl 公用程式偵測到您正在啟動具有 Inf1 執行個體類型的節點群組，並將使用 Amazon EKS 最佳化加速 Amazon Linux AMI 之一來啟動您的節點。

   > **Note**
   >
   > 您無法搭配 TensorFlow Serving 使用服務帳戶的 IAM 角色。

   ```
   eksctl create cluster \
       --name inferentia \
       --region region-code \
       --nodegroup-name ng-inf1 \
       --node-type inf1.2xlarge \
       --nodes 2 \
       --nodes-min 1 \
       --nodes-max 4 \
       --ssh-access \
       --ssh-public-key your-key \
       --with-oidc
   ```

   > **Note**
   >
   > 請注意下列輸出行的值。它將用於稍後的 (選用) 步驟。

   ```
   [9]  adding identity "arn:aws: iam::111122223333:role/eksctl-inferentia-nodegroup-ng-in-NodeInstanceRole-FI7HIYS3BS09" to auth ConfigMap
   ```

   使用Inf1執行個體啟動節點群組時， eksctl會自動安裝 AWS Neuron Kubernetes 裝置外掛程式。這個外掛程式會將 Neuron 裝置做為系統資源公告給 Kubernetes 排程器，如此就能由容器請求。除了預設 Amazon EKS 節點 IAM 政策之外，還會新增 Amazon S3 唯讀存取政策，以便稍後步驟介紹的範例應用程式可以從 Amazon S3 載入訓練過的模型。

2. 請確保所有的 Pod 都已正確啟動。

```
kubectl get pods -n kube-system
```

縮寫輸出：

```
NAME                                   READY   STATUS    RESTARTS   AGE
[...]
neuron-device-plugin-daemonset-6djhp   1/1     Running   0          5m
neuron-device-plugin-daemonset-hwjsj   1/1     Running   0          5m
```

(選用) 部署 TensorFlow Serving 應用程式映像

經過訓練的模型必須編譯至 Inferentia 目標，才能部署到 Inferentia 執行個體上。若要繼續，您需要儲存在 Amazon S3 中的 Neuron 最佳化 TensorFlow 模型。如果您尚未擁有 SavedModel，請依照建立 Neuron 相容的 ResNet50 模型教學，並將產生的 SavedModel 上傳到 S3。ResNet-50 是一種熱門的機器學習模型，用於圖像識別任務。如需編譯 Neuron 模型的詳細資訊，請參閱 AWS 深度學習 AMI 開發人員指南中的 AWS Inferentia Chip with DLAMI。 AMIs

範例部署資訊清單會管理 AWS 深度學習容器為 TensorFlow 提供的預先建置推論服務容器。容器內部是 AWS Neuron 執行期和 TensorFlow Serving 應用程式。針對 Neuron 最佳化的預建置 Deep Learning Containers 的完整清單會在 GitHub 上進行維護，位於可用圖像下。啟動時，DLC 會從 Amazon S3 中擷取您的模型、使用儲存的模型啟動 Neuron TensorFlow Serving，並等待預測請求。

分配給服務應用程式的 Nesuron 裝置數量可以進行調整，方法是變更部署 yaml 中的 `aws.amazon.com/neuron` 資源。請注意，TensorFlow Serving 和 Neuron 執行時間之間的通訊透過 GRPC 進行，這需要將 `IPC_LOCK` 功能傳遞給容器。

1. 將 `AmazonS3ReadOnlyAccess` IAM 政策新增至在建立叢集的步驟 1 中建立的節點執行個體角色。這是必要步驟，以便範例應用程式可以從 Amazon S3 載入訓練過的模型。

```
aws iam attach-role-policy \
    --policy-arn arn:aws: iam::aws:policy/AmazonS3ReadOnlyAccess \
    --role-name eksctl-inferentia-nodegroup-ng-in-NodeInstanceRole-FI7HIYS3BS09
```

2. 使用下列內容建立名為 `rn50_deployment.yaml` 的檔案。更新區域碼和模型路徑以符合您想要的設定。當用戶端向 TensorFlow 伺服器發出請求時，模型名稱用於識別目的。此範例使用模型名稱來比對範例 ResNet50 用戶端指令碼，該指令碼將在稍後步驟中用於傳送預測請求。

```
aws ecr list-images --repository-name neuron-rtd --registry-id 790709498068 --region
 us-west-2
```

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: eks-neuron-test
  labels:
    app: eks-neuron-test
    role: master
spec:
  replicas: 2
  selector:
    matchLabels:
      app: eks-neuron-test
      role: master
  template:
    metadata:
      labels:
        app: eks-neuron-test
        role: master
    spec:
      containers:
        - name: eks-neuron-test
          image: 763104351884.dkr.ecr.us-east-1.amazonaws.com/tensorflow-inference-
neuron:1.15.4-neuron-py37-ubuntu18.04
          command:
            - /usr/local/bin/entrypoint.sh
          args:
            - --port=8500
            - --rest_api_port=9000
            - --model_name=resnet50_neuron
            - --model_base_path=s3://${your-bucket-of-models}/resnet50_neuron/
          ports:
            - containerPort: 8500
            - containerPort: 9000
          imagePullPolicy: IfNotPresent
          env:
            - name: AWS_REGION
              value: "us-east-1"
            - name: S3_USE_HTTPS
              value: "1"
```

```
            - name: S3_VERIFY_SSL
              value: "0"
            - name: S3_ENDPOINT
              value: s3.us-east-1.amazonaws.com
            - name: AWS_LOG_LEVEL
              value: "3"
          resources:
            limits:
              cpu: 4
              memory: 4Gi
              aws.amazon.com/neuron: 1
            requests:
              cpu: "1"
              memory: 1Gi
          securityContext:
            capabilities:
              add:
                - IPC_LOCK
```

3. 部署模型。

```
kubectl apply -f rn50_deployment.yaml
```

4. 使用下列內容建立名為 rn50_service.yaml 的檔案。HTTP 和 gRPC 連接埠會開啟以接受預測請求。

```
kind: Service
apiVersion: v1
metadata:
  name: eks-neuron-test
  labels:
    app: eks-neuron-test
spec:
  type: ClusterIP
  ports:
    - name: http-tf-serving
      port: 8500
      targetPort: 8500
    - name: grpc-tf-serving
      port: 9000
      targetPort: 9000
  selector:
    app: eks-neuron-test
```

```
    role: master
```

5. 為您的 TensorFlow 模型服務應用程式建立 Kubernetes 服務。

```
kubectl apply -f rn50_service.yaml
```

(選用) 針對您的 TensorFlow Serving 服務進行預測

1. 若要在本機測試，請將 gRPC 連接埠轉送至 eks-neuron-test 服務。

```
kubectl port-forward service/eks-neuron-test 8500:8500 &
```

2. 建立一個叫做 tensorflow-model-server-infer.py 的 Python 指令碼，具有以下內容。此指令碼會透過 gRPC 執行推論，這是一種服務架構。

```
import numpy as np
import grpc
import tensorflow as tf
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.resnet50 import preprocess_input
from tensorflow_serving.apis import predict_pb2
from tensorflow_serving.apis import prediction_service_pb2_grpc
from tensorflow.keras.applications.resnet50 import decode_predictions

if __name__ == '__main__':
   channel = grpc.insecure_channel('localhost:8500')
   stub = prediction_service_pb2_grpc.PredictionServiceStub(channel)
   img_file = tf.keras.utils.get_file(
       "./kitten_small.jpg",
       "https://raw.githubusercontent.com/awslabs/mxnet-model-server/master/docs/
images/kitten_small.jpg")
   img = image.load_img(img_file, target_size=(224, 224))
   img_array = preprocess_input(image.img_to_array(img)[None, ...])
   request = predict_pb2.PredictRequest()
   request.model_spec.name = 'resnet50_neuron'
   request.inputs['input'].CopyFrom(
       tf.make_tensor_proto(img_array, shape=img_array.shape))
   result = stub.Predict(request)
   prediction = tf.make_ndarray(result.outputs['output'])
   print(decode_predictions(prediction))
```

3. 執行指令碼，將預測提交至您的服務。

```
python3 tensorflow-model-server-infer.py
```

範例輸出如下。

```
[[(u'n02123045', u'tabby', 0.68817204), (u'n02127052', u'lynx', 0.12701613),
 (u'n02123159', u'tiger_cat', 0.08736559), (u'n02124075', u'Egyptian_cat',
 0.063844085), (u'n02128757', u'snow_leopard', 0.009240591)]]
```

在 Amazon EKS 上管理 AI/ML 工作負載的加速運算

> **Tip**
>
> 註冊即將舉行的 Amazon EKS AI/ML 研討會。

本節說明如何使用 Amazon EKS 購買和佈建適用於 AI/ML 訓練和推論工作負載的 EC2 加速運算執行個體。無論您是訓練大規模模型、執行即時推論，還是部署生成式 AI 應用程式，使用正確的 NVIDIA GPU 或 AWS Trainium 容量都是工作負載效能的基礎。

從 EC2 執行個體類型中選擇

如需可用 Amazon EC2 加速運算執行個體的詳細資訊，請參閱 Amazon EC2 加速運算執行個體的規格。 Amazon EC2 其中包括來自 P 系列和 G 系列，以及 Trainium 和 Inferentia AWS設計加速器的 NVIDIA GPU 執行個體。

了解 EC2 購買選項

了解工作負載所需的加速執行個體後，下一步是了解可用於取得這些加速執行個體類型的購買選項。AWS 提供四種運算容量購買選項：隨需執行個體、Spot 執行個體、ML 的容量區塊，以及隨需容量保留 ODCRs)。每個選項都提供不同的工作負載模式、成本設定檔和可用性需求。Amazon EC2 執行個體購買選項文件說明每個選項的運作方式、定價模式，以及使用時機。

- 隨需執行個體：容量存在時，以秒為單位付款，無需承諾且立即可用。最適合用於開發、原型設計、無法預測的推論擴展，以及需要立即運算而不會造成中斷風險的任何工作負載。

- Spot 執行個體：使用備用 EC2 容量，與隨需相比可節省高達 90%，並提供 2 分鐘的中斷通知。最適合用於對持久儲存進行檢查點的容錯工作負載：超參數調校、具有定期檢查點的分散式訓練、批次和離線推論，以及資料預先處理管道。
- ML 的容量區塊：預留 P-family 和 Trainium 執行個體固定時段 (24 小時，最長 6 個月），最多可提前 8 週預訂。將容量區塊用於規劃的大規模訓練執行、有時間限制的微調實驗，以及具有已知時間軸且需要可預測 GPU 叢集存取權的研究專案。
- 隨需容量保留 (ODCRs)：在沒有長期承諾的情況下，將加速容量保留在特定可用區域中，無論是否使用容量，都會以標準隨需費率計費。最適合用於生產推論、SLA 繫結服務，以及無法接受排程延遲或容量無法使用的業務關鍵應用程式。與容量區塊不同，ODCRs 支援 P 系列和 G 系列執行個體。

將購買選項與工作負載需求配對

現在您已了解加速執行個體類型和購買選項，下一步是將正確的購買選項與您的工作負載特定需求配對。跨執行個體類型、區域和時間具有更大彈性的工作負載符合更多購買選項和更低定價的資格。

根據下列因素做出決策：

- 策略重要性和 SLA 承諾
- 需求可預測性和排程彈性
- 願意預先承諾預留容量
- 執行個體類型、區域和時間的彈性
- 可容忍中斷與節省成本

實際上，團隊採用混合方法，結合多個購買選項，以平衡其工作負載產品組合的成本、可用性和可靠性。文章如何在 上取得 GPU 容量 AWS，提供決策樹、定價比較，以及為不同類型的工作負載選擇正確購買選項的真實範例。

驗證您的 EC2 服務配額

在 EKS 叢集上實作任何容量購買選項之前，請確認 AWS 您的帳戶有足夠的 vCPU 配額可供您計劃使用的 GPU 執行個體系列使用。如果沒有足夠的配額，Karpenter NodePools、EKS Auto Mode 佈建和 EKS 節點群組將無法啟動加速運算節點，無論您選取哪個購買選項。

AWS 會強制執行每個執行個體系列和購買模型的個別 vCPU 配額。檢閱 Amazon EC2 執行個體類型配額，以了解加速運算執行個體的預設配額。

這些配額是根據 vCPU 計數，而不是執行個體計數。例如，啟動 10 個 p6-b300.48xlarge 執行個體需要 1，920 vCPUs (10 × 192)。新帳戶的預設 GPU 配額通常設定為 0，因此在嘗試部署執行個體之前請求增加。

如果您在建立容量區塊保留、啟動隨需執行個體或提交 Spot 請求時遇到配額限制，請聯絡 AWS Support 或您的 AWS 客戶團隊以討論您的需求，並探索保護最符合您需求的加速運算容量的選項。

搭配 Amazon EKS 使用 EC2 購買選項

選取 EC2 加速運算購買選項後，請將 Amazon EKS 叢集設定為使用容量。Amazon EKS 提供三種佈建方法，每個方法都有不同的控制和自動化平衡：

- Amazon EKS Auto Mode：自動佈建、擴展和修補節點的 AWS受管運算。使用內建 Karpenter 來佈建和 Bottlerocket 作業系統，其中包含 NVIDIA 驅動程式和裝置外掛程式。當您想要具有最少營運開銷的受管基礎設施時最佳。支援靜態和動態容量佈建。
- Karpenter（自我管理）：您在 Amazon EKS 叢集中安裝和操作的開放原始碼上游專案。提供與 EKS Auto Mode 相同的佈建模型，您可以完全控制作業系統、AMIs、核心調校和節點生命週期。最適合要求 EKS Auto Mode 未提供out-of-the-box平台團隊。
- 節點群組（受管和自我管理）：由 EC2 Auto Scaling Groups (ASG) 支援，透過 EC2 啟動範本預先定義容量。最適合具有現有 EKS 受管或自我管理節點群組的平台團隊，並使用已知的靜態加速運算足跡，以可預測的大小訓練工作負載。

以下頁面詳細說明每個佈建選項。

- the section called “EKS 自動模式和 Karpenter”
- the section called “受管節點群組和自我管理節點”

混合策略：合併購買選項

在單一 Amazon EKS 叢集中結合多個容量購買選項是很常見的。這種方法透過將不同的工作負載路由到最適當的容量來源，同時最佳化成本、可用性和可靠性。客戶使用三種 EKS 運算管理方法 (EKS Auto Mode、Karpenter 或 Node Groups) 中的任何一種實作此混合策略，或在相同的叢集中結合這些策略。

EKS Auto Mode 和 Karpenter 一律先佈建預留容量 (ODCRs和容量區塊），接著是 Spot 或隨需。您可以透過在預留容量上排定關鍵工作負載，以及在 Spot 或隨需執行個體上排定彈性工作負載，來耦合此執行個體佈建優先順序。您可以透過 Kubernetes 原生排程基本概念控制工作負載路

由：nodeSelector鎖定特定容量類型、污點和容錯，隔離 NVIDIA GPU 或 AWS Trainium 節點，並將工作負載topologySpreadConstraints分散到可用區域以獲得高可用性。

精心設計的 Amazon EKS 叢集會將加速運算 NodePools 或節點群組組織成兩個類別：預留和爆量，每個類別都符合最適合容量策略的工作負載模式。範例說明如下。

- 預留容量：gpu-reservedNodePool 或節點群組會對 SLA 繫結服務和計劃運算密集型任務的預留容量 (ODCRs 和容量區塊） 執行生產推論和排定的大規模訓練。此 NodePool 或節點群組提供推論和生產工作負載：即時推論端點、生產模型服務，以及需要全年無休 GPU 可用性且效能可預測的業務關鍵應用程式。它還支援排定的運算密集型任務：規劃的分散式訓練、大規模微調實驗、有時間限制的研究專案，以及您事先知道開始時間和持續時間的任何工作負載。
- 高載容量：gpu-burstNodePool 或節點群組會處理實驗、臨機操作工作負載和批次處理。它使用 Spot 執行個體做為具有隨需備用的主要容量類型。此組合可最大限度地節省容錯工作負載的成本，並確保 Spot 無法使用時的容量。此 NodePool 或節點群組提供批次離線推論、資料預先處理管道、模型評估任務、開發和原型設計、無法預測的推論擴展、短期偵錯工作階段，以及任何可實作檢查點且可處理 Spot 中斷或不合理保留，但無法等待預留時段的工作負載。此 NodePool 或節點群組上的工作負載實作檢查點和正常關機，以在 2 分鐘 Spot 中斷時段內處理節點遺失。

使用 nodeSelector 指定工作負載所需的容量類型：karpenter.sh/capacity-type: [spot, on-demand, reserved]。以權重為基礎的佈建可在所有容量集區中有效率地擴展叢集。透過此架構，您可以在單一 Amazon EKS 叢集中執行各種 AI/ML 工作負載，從實驗筆記本到生產推論，同時最佳化成本。

使用 EKS Auto Mode 和 Karpenter 管理 AI/ML 工作負載的運算

> **Tip**
>
> 註冊即將舉行的 Amazon EKS AI/ML 研討會。

本節說明如何使用 Amazon EKS Auto Mode 或自我管理 Karpenter 管理 AI 訓練和推論工作負載的加速運算 (AWS Trainium、NVIDIA GPUs)。

EKS Auto Mode 和 Karpenter 支援兩種佈建模式：動態佈建和靜態佈建。使用動態佈建時，EKS Auto Mode 和 Karpenter 會在叢集上排程工作負載時佈建和擴展加速運算執行個體。使用靜態佈建時，EKS Auto Mode 和 Karpenter 會佈建和維護固定數量的節點。動態和靜態佈建可用於相同叢集，以維持固定的基準容量集區，同時隨著工作負載需求進行擴展。

EKS Auto Mode 和 Karpenter 支援所有四個容量購買選項（隨需、Spot、容量區塊和 ODCRs），並一律先佈建預留容量，接著是 Spot 或隨需。

EKS 自動模式與 Karpenter

這兩種方法共用 NodePool API，但在操作擁有權、資源 APIs、作業系統支援、Spot 中斷處理和組態彈性方面有所不同。

功能	EKS 自動模式	自我管理 Karpenter
最適合	偏好營運負荷最少的受管基礎設施團隊	偏好完全控制節點生命週期、AMIs、作業系統調校和修補的團隊。
操作模型	AWS 佈建和管理 Karpenter 控制器、GPU/Trainium 驅動程式、裝置外掛程式、作業系統修補和 Spot 中斷處理。	您可以在叢集中安裝和操作 Karpenter 控制器，以及自己的 GPU/Trainium 驅動程式、裝置外掛程式、AMI 生命週期、修補和 Spot 中斷處理。
運算選項	隨需、Spot、ODCRs、ML 的容量區塊	隨需、Spot、ODCRs、ML 的容量區塊
資源 APIs	`NodePool` (`karpenter.sh/v1`), `NodeClass` (`eks.amazonaws.com/v1`).	`NodePool` (`karpenter.sh/v1`), `EC2NodeClass` (`karpenter.k8s.aws/v1`).
節點作業系統	僅限 Bottlerocket。包含 NVIDIA GPU、 AWS Trainium 和 EFA 相依性。	AL2023、 Bottlerocket、Windows 或您自己的 AMI。
節點生命週期	安全性修補的 21 天最大節點生命週期。工作負載必須容忍節點輪換。	您可以透過 NodePool `expireAfter` 和中斷預算定義節點生命週期。
Spot 中斷處理	原生。不需要 SQS 佇列或節點終止處理常式。	您設定和啟用 的責任。
快速容器提取	所有 G、P 和 Trn 系列執行個體中都包含 SOCI 平行提取	您設定和啟用 的責任。
EC2 置放群組	叢集、分割區、分散	叢集、分割區、分散

功能	EKS 自動模式	自我管理 Karpenter
網路界面組態	類型 `interface` 或 的每個界面組態 `EFA-only`	類型 `interface` 或 的每個界面組態 `EFA-only`
節點修復	根據預設，包含 EKS 節點監控代理程式	選擇性地啟用 EKS 節點監控代理程式自我管理
定價	除了基礎 EC2 執行個體成本外，還需支付 EKS Auto Mode 管理費。	開放原始碼。您支付基礎 EC2 執行個體的費用。

常見的 AI/ML 已知標籤

EKS Auto Mode 和 Karpenter 會公開您可以在 NodePool `requirements`和 Pod 中使用的執行個體標籤`nodeAffinity`，`nodeSelector`或在沒有硬式編碼執行個體類型的目標工作負載。標籤字首在兩者之間不同：EKS Auto Mode 使用 `eks.amazonaws.com/`，而自我管理的 Karpenter 使用 `karpenter.k8s.aws/`。

下表顯示可在 NodePools 中使用的相關標籤。EKS Auto Mode 和 Karpenter 也會將 Karpenter 文件中列出的標籤套用至節點，做為佈建程序的一部分，可進一步用於工作負載目標。

EKS Auto Mode

如需完整清單，請參閱 EKS Auto Mode 支援的標籤。

標籤	範例值	說明
`eks.amazonaws.com/instance-family`	`p5`	類似屬性的執行個體類型，但資源數量不同。
`eks.amazonaws.com/instance-category`	`p`	執行個體類別，通常是產生編號之前的字母。
`eks.amazonaws.com/instance-generation`	`5`	類別內的執行個體類型產生號碼。
`eks.amazonaws.com/instance-gpu-name`	`h100`	執行個體上的 GPU 名稱。

標籤	範例值	說明
eks.amazonaws.com/instance-gpu-manufacturer	nvidia	GPU 製造商的名稱。
eks.amazonaws.com/instance-gpu-count	8	執行個體上的 GPUs 數量。
eks.amazonaws.com/instance-gpu-memory	81920	每個 GPU 的記憶體 MB 數。
karpenter.sh/capacity-type	reserved	容量類型：spot、 on-demand 或 reserved。
topology.kubernetes.io/zone	us-east-1a	可用區域。

Self-managed Karpenter

如需完整清單，請參閱 Karpenter Well-Known 標籤。

標籤	範例值	說明
karpenter.k8s.aws/instance-family	p5	類似屬性的執行個體類型，但資源數量不同。
karpenter.k8s.aws/instance-category	p	執行個體類別，通常是產生編號之前的字母。
karpenter.k8s.aws/instance-generation	5	類別內的執行個體類型產生號碼。
karpenter.k8s.aws/instance-gpu-name	h100	執行個體上的 GPU 名稱。

標籤	範例值	說明
karpenter.k8s.aws/instance-gpu-manufacturer	nvidia	GPU 製造商的名稱。
karpenter.k8s.aws/instance-gpu-count	8	執行個體上的 GPUs 數量。
karpenter.sh/capacity-type	reserved	容量類型：spot、 on-demand 或 reserved。
topology.kubernetes.io/zone	us-east-1a	可用區域。
kubernetes.io/arch	amd64	CPU 架構。

排程預留容量的標籤

當 EKS Auto Mode 或 Karpenter 在保留中啟動節點時，它會新增下列標籤。在 nodeSelector、節點親和性或 NodePool 需求中使用它們來路由工作負載。

- karpenter.sh/capacity-type：reserved、 on-demand或 spot。指出備份節點的容量。
- karpenter.k8s.aws/capacity-reservation-id：啟動節點的特定保留 ID。
- karpenter.k8s.aws/capacity-reservation-type：default適用於 ODCRs，capacity-block適用於容量區塊。

下列範例顯示常見的排程模式：

將 Pod 固定到一個特定的保留 （無備用）：

```
spec:
  nodeSelector:
    karpenter.sh/capacity-type: reserved
    karpenter.k8s.aws/capacity-reservation-id: "cr-0123456789abcdef0"
```

僅限目標 ODCR 節點（任何 ODCR，而非容量區塊）：

```
spec:
  nodeSelector:
    karpenter.sh/capacity-type: reserved
    karpenter.k8s.aws/capacity-reservation-type: default
```

針對任何預留容量 (ODCR 或容量區塊）：

```
spec:
  nodeSelector:
    karpenter.sh/capacity-type: reserved
```

偏好預留，但如果無法使用，請回到 Spot 或隨需：

```
spec:
  affinity:
    nodeAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
        - weight: 100
          preference:
            matchExpressions:
              - key: karpenter.sh/capacity-type
                operator: In
                values: ["reserved"]
```

保留過期行為

保留結束時ODCRs 和容量區塊的行為會有所不同。確保您的排程和檢查點策略符合您工作負載的保留類型。

ODCRs

在 ODCR 中啟動的執行個體無限期不在該 ODCR 中。ODCR 可能會過期、取消，或可從 ODCR 手動移除執行個體。如果發生上述任何情況，且 EKS Auto Mode / Karpenter 偵測到執行個體不再屬於 ODCR，則會將節點的`karpenter.sh/capacity-type`標籤從 更新`reserved`為 `on-demand`。執行個體會繼續做為標準隨需容量執行，而現有的 Pod 會繼續不間斷地執行。

> **Note**
>
> 如果已重新標記任何以嚴格 排程的 Pod，`nodeSelector: karpenter.sh/capacity-type: reserved`將不會排程到節點。若要讓工作負載在 ODCR 到期或取消後存活，

請使用上述preferredDuringSchedulingIgnoredDuringExecution模式，而非 nodeSelector。

容量區塊

與 ODCRs 不同，容量區塊一律有結束時間，EC2 會在結束時間前 30 分鐘終止容量區塊執行個體 (UltraServer 執行個體類型為 60 分鐘）。在保留時段關閉之前，規劃訓練和推論任務以完成或儲存狀態。在區塊過期且不會重新排程其他位置Pending時，對特定capacity-reservation-id執行使用嚴格 nodeSelector 的 Pod。如果您需要工作負載在容量區塊到期期間移至其他容量，請將檢查點與上述的彈性親和性模式結合。

- 您可以使用預留執行個體，直到大多數執行個體類型的容量區塊結束時間前 30 分鐘，或 UltraServer 執行個體類型的結束時間前 60 分鐘為止。
- EKS Auto Mode 和 Karpenter 會在 EC2 開始終止前 10 分鐘先開始耗盡容量區塊中的節點，讓工作負載有時間進行檢查點並正常關閉。

靜態容量 NodePools

EKS Auto Mode 和 Karpenter 支援靜態容量 NodePools 都會維持固定數量的節點。靜態集區可減少冷啟動延遲敏感推論的延遲，並可讓您為叢集預留最低基礎設施使用量。

透過在 NodePool 上設定 replicas 欄位來設定靜態容量。

考量

- 在 NodePool 上replicas設定 後，您就無法將其移除。單一 NodePool 無法在靜態和動態容量佈建之間切換。
- 靜態容量 NodePools 不會考慮合併。limits.nodes 上述設定replicas以允許在 AMI 偏離或過期期間暫時擴展。
- 針對可預測的可用區域 (AZ) 分佈，請為每個可用區域建立一個靜態容量 NodePool，而不是在單一集區中跨越多個區域。

EKS Auto Mode

以下範例顯示使用預設 EKS Auto Mode NodeClass 的靜態容量 NodePool，並建立具有 4 個節點 (replicas) 的靜態 NodePool，最多可達 6 個節點 ()limits.nodes。

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: gpu-static-inference
spec:
  replicas: 4
  template:
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default
      requirements:
        - key: "karpenter.sh/capacity-type"
          operator: In
          values: ["on-demand"]
        - key: "eks.amazonaws.com/instance-family"
          operator: In
          values: ["g6e"]
        - key: "topology.kubernetes.io/zone"
          operator: In
          values: ["us-east-1a"]
      taints:
        - key: nvidia.com/gpu
          value: "true"
          effect: NoSchedule
  limits:
    nodes: 6   # Allow temporary headroom during node replacement
```

Self-managed Karpenter

使用自我管理的 Karpenter 時，靜態容量由 alpha `StaticCapacity`功能 （ 在 Karpenter 1.8 版中啟動） 鎖定，必須在 Helm 值中啟用：

```
settings:
  featureGates:
    staticCapacity: true
```

NodePool 參考`EC2NodeClass`名為 的自訂`my-nodeclass`，並建立具有 4 個節點 (`replicas`) 的靜態 NodePool，最多可達 6 個節點 ()`limits.nodes`。

```
apiVersion: karpenter.sh/v1
```

```
kind: NodePool
metadata:
  name: gpu-static-inference
spec:
  replicas: 4
  template:
    spec:
      nodeClassRef:
        group: karpenter.k8s.aws
        kind: EC2NodeClass
        name: my-nodeclass
      requirements:
        - key: "karpenter.sh/capacity-type"
          operator: In
          values: ["on-demand"]
        - key: "karpenter.k8s.aws/instance-family"
          operator: In
          values: ["g6e"]
        - key: "topology.kubernetes.io/zone"
          operator: In
          values: ["us-east-1a"]
      taints:
        - key: nvidia.com/gpu
          value: "true"
          effect: NoSchedule
  limits:
    nodes: 6   # Allow temporary headroom during node replacement
```

ML 的容量區塊

ML 的容量區塊可讓您為定義的未來時段保留 P 系列和 Trainium 執行個體。它們是預付的，因此 EKS Auto Mode 和 Karpenter 會將它們建模為免費，並優先於隨需和 Spot。ML 的容量區塊的保留期間可以是 1-14 天或 7 天的倍數，最長可達 182 天 (6 個月）。

若要搭配 EKS Auto Mode 或 Karpenter 使用 ML 的容量區塊，請在 NodeClass 中使用`capacityReservationSelectorTerms`容量保留 ID 來設定 。您無法將開放保留比對與 ML 的容量區塊搭配使用。術語可以指定 ID、一組標籤或執行個體比對條件以供選取。指定標籤時，它會選取所有可從具有相符標籤的帳戶存取的容量保留。這可以透過指定擁有者帳戶 ID 進一步限制。

如需更多範例，請參閱 Karpenter 文件。

EKS Auto Mode

建立參考容量區塊保留NodeClass的 ，然後建立使用它的 NodePool。

consolidateAfter: Never 設定後，Karpenter 不會嘗試取代、合併或終止節點，以更有效地降低成本或封裝工作負載。這建議用於容量區塊，因為容量已預付。

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: capacity-block-gpu
spec:
  capacityReservationSelectorTerms:
    - id: "cr-0123456789abcdef0"   # Your Capacity Block reservation ID
    # Alternative: select by tags
    # - tags:
    #     role: "production-inference"
    #   owner: "012345678901"
---
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: gpu-capacity-block
spec:
  disruption:
    consolidationPolicy: WhenEmpty
    consolidateAfter: Never
  template:
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: capacity-block-gpu
      requirements:
        - key: "karpenter.sh/capacity-type"
          operator: In
          values: ["reserved"]
        - key: "eks.amazonaws.com/instance-family"
          operator: In
          values: ["p5", "p5e", "p5en", "p4d"]
      taints:
        - key: nvidia.com/gpu
          value: "true"
```

```
          effect: NoSchedule
```

Self-managed Karpenter

除了 之外，建立EC2NodeClass包含 AMI、子網路和安全群組選取器的 capacityReservationSelectorTerms，然後建立使用它的 NodePool。

consolidateAfter: Never 設定後，Karpenter 不會嘗試取代、合併或終止節點，以更有效地降低成本或封裝工作負載。這建議用於容量區塊，因為容量已預付。

```
apiVersion: karpenter.k8s.aws/v1
kind: EC2NodeClass
metadata:
  name: capacity-block-gpu
spec:
  amiSelectorTerms:
    - alias: al2023@latest
  subnetSelectorTerms:
    - tags:
        karpenter.sh/discovery: ml-cluster
  securityGroupSelectorTerms:
    - tags:
        karpenter.sh/discovery: ml-cluster
  capacityReservationSelectorTerms:
    - id: "cr-0123456789abcdef0" # Your Capacity Block reservation ID
    # Alternative: select by tags
    # - tags:
    #     role: "production-inference"
    #   owner: "012345678901"
---
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: gpu-capacity-block
spec:
  disruption:
    consolidationPolicy: WhenEmpty
    consolidateAfter: Never
  template:
    spec:
      nodeClassRef:
        group: karpenter.k8s.aws
        kind: EC2NodeClass
```

```
        name: capacity-block-gpu
    requirements:
      - key: "karpenter.sh/capacity-type"
        operator: In
        values: ["reserved"]
      - key: "karpenter.k8s.aws/instance-family"
        operator: In
        values: ["p5", "p5e", "p5en", "p4d"]
    taints:
      - key: nvidia.com/gpu
        value: "true"
        effect: NoSchedule
```

隨需容量保留 ODCRs)

ODCRs可保證特定可用區域 (AZ) 中的容量，無需長期承諾。無論容量是否使用，您都會按標準隨需費率計費。ODCRs支援所有 NVIDIA GPU 系列，包括 ML 的容量區塊不支援的 G 系列執行個體。ODCRs是預付的，因此 EKS Auto Mode 和 Karpenter 會將它們建模為免費，並優先於隨需和 Spot。

保留結束時ODCRs 的行為與 ML 的容量區塊不同。當 ODCR 過期或取消時，執行個體會繼續做為標準隨需執行。如需詳細資訊，請參閱 the section called “保留過期行為”。

若要搭配 EKS Auto Mode 或 Karpenter 使用 ODCRs，請在 NodeClass 中`capacityReservationSelectorTerms`以容量保留條件設定 。術語可以指定 ID、一組標籤或執行個體比對條件以供選取。指定標籤時，它會選取所有可從具有相符標籤的帳戶存取的容量保留。指定執行個體比對條件時，它會依相符行為選取保留：開啟（符合所有相容的執行個體）或目標（僅符合明確的目標執行個體）。這可以透過指定擁有者帳戶 ID 進一步限制。

如需更多範例，請參閱 Karpenter 文件。

EKS Auto Mode

NodeClass 使用 `capacityReservationSelectorTerms` 和以`on-demand`備用`reserved`為優先順序的 NodePool 建立 。釘`topology.kubernetes.io/zone`選至 ODCR 的 AZ：

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: odcr-gpu-production
spec:
```

```
  capacityReservationSelectorTerms:
    - id: "cr-0987654321fedcba0"
    # Alternative: select by tags
    # - tags:
    #     Purpose: "production-inference"
    #   owner: "012345678901"
---
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: gpu-reserved-production
spec:
  disruption:
    consolidationPolicy: WhenEmpty
    consolidateAfter: Never
  template:
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: odcr-gpu-production
      requirements:
        - key: "karpenter.sh/capacity-type"
          operator: In
          values: ["reserved", "on-demand"]
        - key: "eks.amazonaws.com/instance-family"
          operator: In
          values: ["p5", "g6e"]
        - key: "topology.kubernetes.io/zone"
          operator: In
          values: ["us-east-1a"]
      taints:
        - key: nvidia.com/gpu
          value: "true"
          effect: NoSchedule
```

Self-managed Karpenter

除了 之外，EC2NodeClass使用 AMI、子網路和安全群組選取器建立 `capacityReservationSelectorTerms`，然後建立 NodePool：

```
apiVersion: karpenter.k8s.aws/v1
kind: EC2NodeClass
```

```
metadata:
  name: odcr-gpu-production
spec:
  amiSelectorTerms:
    - alias: al2023@latest
  subnetSelectorTerms:
    - tags:
        karpenter.sh/discovery: ml-cluster
  securityGroupSelectorTerms:
    - tags:
        karpenter.sh/discovery: ml-cluster
  capacityReservationSelectorTerms:
    - id: "cr-0987654321fedcba0"
---
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: gpu-reserved-production
spec:
  disruption:
    consolidationPolicy: WhenEmpty
    consolidateAfter: Never
  template:
    spec:
      nodeClassRef:
        group: karpenter.k8s.aws
        kind: EC2NodeClass
        name: odcr-gpu-production
      requirements:
        - key: "karpenter.sh/capacity-type"
          operator: In
          values: ["reserved", "on-demand"]
        - key: "karpenter.k8s.aws/instance-family"
          operator: In
          values: ["p5", "g6e"]
        - key: "topology.kubernetes.io/zone"
          operator: In
          values: ["us-east-1a"]
      taints:
        - key: nvidia.com/gpu
          value: "true"
          effect: NoSchedule
```

On-Demand

隨需是預設容量類型，可與 EKS Auto Mode 和 Karpenter 中的靜態或動態佈建搭配使用。您可以在 NodePool `karpenter.sh/capacity-type: on-demand`中設定 ，明確請求隨需執行個體。EKS Auto Mode 和 Karpenter 選取滿足 Pod 資源請求的最低價執行個體。使用隨需進行開發、原型設計、不可預測的推論擴展，以及任何需要立即可用性且沒有中斷風險的工作負載。

EKS Auto Mode

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: gpu-ondemand
spec:
  template:
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default
      requirements:
        - key: "karpenter.sh/capacity-type"
          operator: In
          values: ["on-demand"]
        - key: "eks.amazonaws.com/instance-family"
          operator: In
          values: ["g6", "g6e", "g7e"]
        - key: "eks.amazonaws.com/instance-gpu-manufacturer"
          operator: In
          values: ["nvidia"]
      taints:
        - key: nvidia.com/gpu
          value: "true"
          effect: NoSchedule
```

Self-managed Karpenter

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: gpu-ondemand
spec:
```

```
  template:
    spec:
      nodeClassRef:
        group: karpenter.k8s.aws
        kind: EC2NodeClass
        name: default
      requirements:
        - key: "karpenter.sh/capacity-type"
          operator: In
          values: ["on-demand"]
        - key: "karpenter.k8s.aws/instance-family"
          operator: In
          values: ["g6", "g6e", "g7e"]
        - key: "karpenter.k8s.aws/instance-gpu-manufacturer"
          operator: In
          values: ["nvidia"]
      taints:
        - key: nvidia.com/gpu
          value: "true"
          effect: NoSchedule
```

Spot

Spot 使用備用 EC2 容量，相較於隨需節省高達 90% 的費用。 AWS 可以透過 2 分鐘的中斷通知回收 Spot 執行個體。透過列出 NodePool 上的多個執行個體系列，將可用性最大化。將具有 PodDisruptionBudget和 檢查點的 Spot 工作負載定期與耐用的儲存體 (Amazon S3 或 Amazon EFS) 配對，以便 Pod 在耗盡時段期間儲存狀態。

Spot 非常適合容錯、可重複使用的訓練和推論工作負載，其中偶爾中斷是可以接受的，以換取顯著的成本節省。

常見的候選項目包括：

- 超參數調校和掃描：許多簡短的平行試驗可在中斷時重試。
- 具有檢查點的分散式訓練：長時間執行的任務會定期將狀態儲存到 S3 或 FSx，並在節點遺失後從最後一個檢查點恢復。
- 批次和離線推論：針對以小時為單位而非秒為單位測量end-to-end延遲的資料集進行大規模評分任務。
- 資料預先處理和特徵工程管道：大型資料集的平行轉換。
- 模型評估和基準測試：產生等冪性結果的可重複任務。

- 開發、原型設計和筆記本：互動式實驗，使用者可以容忍偶爾重新啟動。

避免 Spot 用於對延遲敏感的即時推論、SLA 繫結的生產端點，以及不檢查點或無法容忍重新啟動的工作負載。

您可以在 NodePool `karpenter.sh/capacity-type: spot`中設定 以明確請求 Spot 執行個體。

EKS Auto Mode

EKS Auto 模式會以原生方式處理 Spot 中斷。不需要 SQS 佇列或節點終止處理常式。

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: gpu-spot
spec:
  disruption:
    budgets:
      - nodes: 10%
    consolidationPolicy: WhenEmpty
    consolidateAfter: 1h
  template:
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default
      requirements:
        - key: "karpenter.sh/capacity-type"
          operator: In
          values: ["spot"]
        - key: "eks.amazonaws.com/instance-family"
          operator: In
          values: ["g6", "g6e", "g7e"]
        - key: "eks.amazonaws.com/instance-gpu-manufacturer"
          operator: In
          values: ["nvidia"]
      taints:
        - key: nvidia.com/gpu
          value: "true"
          effect: NoSchedule
  limits:
    resources:
```

```
      nvidia.com/gpu: "64"
```

Self-managed Karpenter

自我管理的 Karpenter 要求您透過設定中斷佇列，在 Karpenter 控制器 （而非 NodePool) 上啟用原生中斷處理：接收 EC2 Spot 中斷和重新平衡建議事件的 SQS 佇列。您可以在安裝時設定一次。

如果您直接使用 Helm 安裝 Karpenter，請在 `settings.interruptionQueue`中設定 `values.yaml`：

```
# karpenter values.yaml (Helm)
settings:
  clusterName: my-cluster
  interruptionQueue: my-queue   # Name of the SQS queue receiving Spot events
```

如果您使用 引導 Karpenter`eksctl`，`withSpotInterruptionQueue: true`請在叢集組態檔案中設定 。 `eksctl`會建立 SQS 佇列和 EventBridge 規則，並將 Karpenter 控制器設定為使用它們。

```
# eksctl ClusterConfig
karpenter:
  version: "${KARPENTER_VERSION}"
  withSpotInterruptionQueue: true
```

一旦控制器設定為使用您的佇列，個別 NodePool 資源就不需要額外的組態。中斷處理會套用至整個叢集。

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: gpu-spot
spec:
  disruption:
    budgets:
      - nodes: 10%
    consolidationPolicy: WhenEmpty
    consolidateAfter: 1h
  template:
    spec:
      nodeClassRef:
```

```
        group: karpenter.k8s.aws
        kind: EC2NodeClass
        name: default
      requirements:
        - key: "karpenter.sh/capacity-type"
          operator: In
          values: ["spot"]
        - key: "karpenter.k8s.aws/instance-family"
          operator: In
          values: ["g6", "g6e", "g7e"]
        - key: "karpenter.k8s.aws/instance-gpu-manufacturer"
          operator: In
          values: ["nvidia"]
      taints:
        - key: nvidia.com/gpu
          value: "true"
          effect: NoSchedule
  limits:
    resources:
      nvidia.com/gpu: "64"
```

使用節點群組在 Amazon EKS 上管理 AI/ML 工作負載的運算

> **Tip**
>
> 註冊即將舉行的 Amazon EKS AI/ML 研討會。

本節說明如何使用 Amazon EKS 受管節點群組或自我管理節點，管理 AI 訓練和推論工作負載的加速運算 (AWS Trainium、NVIDIA GPUs)。

EKS 受管節點群組和自我管理節點使用 EC2 Auto Scaling 群組 (ASG)。EKS 受管節點群組具有建立、更新和刪除節點的專用 EKS APIs，也具有內建的節點修復功能和生命週期終止掛鉤。EKS 自我管理節點是透過 EC2 APIs直接部署和管理。

使用這些選項，您可以預先定義執行個體類型、所需計數、擴展邊界和 EC2 啟動範本。如果您也有非 EKS 工作負載，並偏好透過 EC2 啟動範本的組態一致性，請考慮使用 EKS 受管節點群組或自我管理節點。EKS 節點群組適合預先已知加速運算足跡的訓練和微調工作負載。請注意，EKS Auto Mode 和 Karpenter 都支援靜態容量佈建，the section called “EKS 自動模式和 Karpenter”如需詳細資訊，請參閱 。

EKS 受管節點群組和自我管理節點支援所有加速運算購買選項（隨需、Spot、隨需容量預留、ML 的容量區塊）。您可以為每個容量類型建立個別的受管或自我管理節點群組，每個群組都有自己的啟動範本、執行個體類型和擴展組態。這可讓您在沒有異質動態佈建邏輯的情況下，對每個容量集區進行明確且 ASG 支援的控制。

EKS 受管節點群組與自我管理節點

在 EKS 受管節點群組和自我管理節點之間進行選擇，取決於您需要的自訂和控制層級。EKS 受管節點群組允許 EC2 啟動範本自訂的子集，而自我管理節點支援 EC2 啟動範本的完整廣度。如果您沒有自行自訂和管理節點生命週期的特定原因，請從 EKS 受管節點群組開始，並只在特定需求強制時移至自我管理節點。

在下列情況下使用受管節點群組：您希望 EKS 代表您處理 AMI 選擇、節點引導、滾動更新、節點修復和正常耗盡工作流程。如果您不想使用 EKS Auto Mode 或 Karpenter 進行訓練和推論工作負載，建議使用 EKS 受管節點群組。使用適用於 ML 的容量區塊時，EKS 受管節點群組會自動建立排程擴展政策，在保留結束前 40 分鐘耗盡節點群組，無需使用 AWS Node Termination Handler 或您自己的縮減規模自動化。當您使用支援的 EKS 最佳化 AMI、不需要核心層級或深入 EC2 啟動範本自訂，以及想要更簡單的 Kubernetes 版本節點升級路徑時，請使用 EKS 受管節點群組。

在下列情況下使用自我管理節點群組：您需要完全控制 EC2 啟動範本、AMI、核心參數、容器執行期組態或自訂引導指令碼。常見的 ML 案例包括使用 Elastic Fabric Adapter (EFA) 進行分散式訓練的調校核心和 NIC 設定，或與自訂節點生命週期控制器整合。自我管理節點可讓您靈活地運送所需的任何使用者資料和 IAM 執行個體描述檔，但您需要負責更新、排程擴展政策和生命週期關聯，例如AWS 節點終止處理常式。

為 ML 預留具有容量區塊的 GPUs

機器學習 (ML) 的容量區塊可讓您在未來日期為有時間限制的訓練或推論工作負載保留 GPU 執行個體。如需詳細資訊，請參閱《Amazon EC2 使用者指南》中的 ML 的容量區塊。

您可以透過 EKS 受管節點群組和自我管理節點使用容量區塊保留。EC2 啟動範本組態在這兩種情況下都相同。用於終止工作負載的節點建立工作流程、縮減規模行為和生命週期關聯會因佈建選項而有所不同。

考量事項

- 容量區塊僅適用於特定 Amazon EC2 執行個體類型和 AWS 區域。如需詳細資訊，請參閱使用容量區塊先決條件。
- 容量區塊是區域區塊。在建立節點群組期間，您必須使用與容量區塊保留位於相同可用區域 (AZ) 的子網路。

- 如果您在容量區塊保留變成作用中之前建立節點群組，請在節點群組建立0期間將所需的容量設定為 。
- 若要允許正常的工作負載耗盡時間，請在容量區塊保留結束前超過 30 分鐘排程scale-to-zero。EC2 會在保留結束前 30 分鐘開始關閉執行個體。

使用 ML 的容量區塊建立節點群組

EKS 受管節點群組和自我管理節點需要使用以容量區塊保留為目標的自訂 EC2 啟動範本。以下顯示 EKS 受管節點群組和自我管理節點的最低必要欄位。自我管理節點需要其他欄位，如以下自我管理節點步驟所示。

`LaunchTemplateData` 必須包含：

- `InstanceMarketOptions` 並將 `MarketType` 設為 `"capacity-block"`
- `CapacityReservationSpecification: CapacityReservationTarget` 將 `CapacityReservationId`設定為容量區塊 ID。例如，*`cr-0123456789abcdef0`*。
- `InstanceType` 設定為容量區塊保留的執行個體類型。例如，*`p5.48xlarge`*。

這些需求會顯示在下列範例，以建立 EKS 受管節點群組和自我管理節點的啟動範本。

Managed node groups

1. 使用下列內容建立名為 `eks-capacity-block-lt.json` 的檔案。

   使用容量區塊`InstanceType`的值取代 `CapacityReservationId`和 的內容。如需其他 EC2 啟動範本欄位的詳細資訊，請參閱 the section called “啟動範本”和使用機器學習工作負載的容量區塊。

   ```
   {
     "LaunchTemplateData": {
       "InstanceMarketOptions": {
         "MarketType": "capacity-block"
       },
       "CapacityReservationSpecification": {
         "CapacityReservationTarget": {
           "CapacityReservationId": "cr-0123456789abcdef0"
         }
       },
       "InstanceType": "p5.48xlarge"
   ```

```
  }
}
```

2. 建立啟動範本。

```
aws ec2 create-launch-template \
    --launch-template-name EKS-Capacity-Block-Launch-Template \
    --launch-template-data file://eks-capacity-block-lt.json
```

3. 使用啟動範本來建立 EKS 受管節點群組。將以下命令中的預留位置取代為適用於您環境的值。以下命令會將 `--ami-type`設為 AL2023 EKS 最佳化 NVIDIA AMIs。the section called “使用 EKS Linux GPU AMIs” 如需可用 EKS 最佳化 AMIs 的詳細資訊，請參閱 。如果您使用自訂 AMI 搭配 EKS 受管節點群組，請在啟動範本中指定您的 AMI ID。

   建立使用容量區塊的 EKS 受管節點群組時，請執行下列動作：

   - 將 `--capacity-type` 設定為 "CAPACITY_BLOCK"。
   - 只在與容量保留相同的可用區域中指定子網路。
   - 如果您在保留作用中`desiredSize`之前指定非零，Auto Scaling 群組會報告啟動錯誤，直到保留變成作用中為止。啟用後，執行個體會啟動，而 ASG 會擴展至請求的 `desiredSize`。

```
aws eks create-nodegroup \
    --cluster-name my-eks-cluster \
    --nodegroup-name eks-cb-nodes \
    --node-role "arn:aws: iam::111122223333:role/myNodeRole" \
    --region region-code \
    --subnets subnet-ExampleID1 \
    --ami-type "AL2023_x86_64_NVIDIA" \
    --scaling-config minSize=0,maxSize=2,desiredSize=0 \
    --capacity-type "CAPACITY_BLOCK" \
    --launch-template name="EKS-Capacity-Block-Launch-Template"
```

4. 如果您在`0`建立時`desiredSize`將 設定為 ，請在保留變成作用中時，使用下列其中一項來擴展節點群組：

   - ASG 上與保留開始時間一致的排程擴展政策。如需詳細資訊，請參閱《Amazon EC2 Auto Scaling 使用者指南》中的為 Amazon EC2 Auto Scaling 排程擴展。
   - Amazon EKS 主控台或 `aws eks update-nodegroup-config`來更新擴展組態。

5. 驗證節點在擴展後加入叢集。

6. EKS 會在保留結束前自動建立名為 Amazon EKS 節點群組容量縮減的排程擴展政策，將節點群組縮減至保留結束前 0 40 分鐘。這可讓 Pod 在 EC2 以 30 分鐘標記開始終止執行個體之前，有時間正常耗盡。請勿編輯或刪除此排程動作。

Self-managed nodes

1. 使用下列內容建立名為 eks-capacity-block-lt.json 的檔案。

   使用容量區塊InstanceType的值取代 CapacityReservationId和 的內容。如需其他 EC2 啟動範本欄位的詳細資訊，請參閱 the section called “啟動範本”和使用機器學習工作負載的容量區塊。

   將 IamInstanceProfile、ImageId、UserData、 SecurityGroupIds的內容取代為您環境KeyName的值。

   ```
   {
     "LaunchTemplateData": {
       "InstanceMarketOptions": {
         "MarketType": "capacity-block"
       },
       "CapacityReservationSpecification": {
         "CapacityReservationTarget": {
           "CapacityReservationId": "cr-0123456789abcdef0"
         }
       },
       "IamInstanceProfile": {
         "Arn": "arn:aws: iam::111122223333:role/myNodeRole"
       },
       "ImageId": "image-id",
       "InstanceType": "p5.48xlarge",
       "KeyName": "key-name",
       "SecurityGroupIds": "sg-05b1d815d1EXAMPLE"
       ],
       "UserData": "user-data"
     }
   }
   ```

2. 建立啟動範本。

   ```
   aws ec2 create-launch-template \
       --launch-template-name EKS-Capacity-Block-Launch-Template \
   ```

```
    --launch-template-data file://eks-capacity-block-lt.json
```

3. 使用啟動範本，依照中的步驟建立 Auto Scaling 群組the section called “Amazon Linux”。如果保留尚未啟用，請將 `DesiredCapacity`設定為 `0`。僅在預留容量的可用區域中指定子網路。
4. 在建立自我管理節點並將 `DesiredCapacity`設定為 之後`0`，請在 Auto Scaling 群組上建立與容量區塊保留時間一致的排程擴展政策。如需詳細資訊，請參閱 Amazon EC2 Auto Scaling 的排程擴展。

   您可以使用預留執行個體，直到保留結束時間前 30 分鐘為止。在結束時間前超過 30 分鐘排定 scale-to-zero，讓 Pod 有時間耗盡。

   如果您想要手動擴展，請在保留開始時間更新 ASG 所需的容量，並在結束時間之前超過 30 分鐘再次更新。
5. 若要正常耗盡 Pod，請設定 AWS 節點終止處理常式。它會使用 EventBridge 監控來自 Amazon EC2 Auto Scaling 的 ASG 縮減生命週期事件，並讓 Kubernetes 控制平面在執行個體無法使用之前採取行動。如果沒有它，Pod 和 Kubernetes 物件可能會卡在待定狀態。如需詳細資訊，請參閱 GitHub 上的 AWS 節點終止處理常式。

   如果您未設定節點終止處理常式，請在 30 分鐘時段之前手動耗盡 Pod，讓它們有時間正常耗盡。

搭配 Amazon EKS 使用 P6e-GB200 UltraServers

本主題說明如何設定和使用 Amazon EKS 搭配 P6e-GB200 UltraServers。具有 4 個 NVIDIA Blackwell GPUs 的`p6e-gb200.36xlarge`執行個體類型僅提供 P6e-GB200 UltraServers。P6e-GB200 UltraServers 有兩種類型。`u-p6e-gb200x36` UltraServer 有 9 個`p6e-gb200.36xlarge`執行個體，`u-p6e-gb200x72`UltraServer 有 18 個`p6e-gb200.36xlarge`執行個體。

若要進一步了解，請參閱 Amazon EC2 P6e-GB200 UltraServers 網頁。

考量事項

- Amazon EKS 支援 P6e-GB200 UltraServers for Kubernetes 1.33 版及更新版本。此 Kubernetes 版本支援動態資源配置 (DRA)，預設為在 EKS 和 AL2023 EKS 最佳化加速 AMIs 中啟用。DRA 是搭配 EKS 使用 P6e-GB200 UltraServers 的要求。Karpenter 或 EKS Auto Mode 不支援 DRA，建議在搭配 EKS 使用 P6e-GB200 UltraServers 時使用 EKS 自我管理節點群組或 EKS 受管節點群組。
- P6e-GB200 UltraServers 可透過適用於 ML 的 EC2 容量區塊提供。the section called “容量管理” 如需如何使用容量區塊啟動 EKS 節點的資訊，請參閱 。

- 搭配容量區塊使用 EKS 受管節點群組時，您必須使用自訂啟動範本。使用 P6e-GB200 UltraServers 升級 EKS 受管節點群組時，您必須先將節點群組的所需大小設定為 ，0才能升級。
- 建議使用 EKS 最佳化加速 AMIs 的 AL2023 ARM NVIDIA 變體。此 AMI 包含使用 P6e-GB200 UltraServers 所需的節點元件和組態。如果您決定建置自己的 AMI，您必須負責安裝和驗證節點和系統軟體的相容性，包括驅動程式。如需詳細資訊，請參閱the section called “使用 EKS Linux GPU AMIs”。
- 建議使用 EKS 最佳化 AMI 發行版本 v20251103 或更新版本，其中包括 NVIDIA 驅動程式 580 版。此 NVIDIA 驅動程式版本可讓 Coherent 驅動程式型記憶體 (CDMM) 處理潛在的記憶體過度報告。啟用 CDMM 時，不支援下列功能：NVIDIA Multi-Instance GPU (MIG) 和 vGPU。如需 CDMM 的詳細資訊，請參閱 NVIDIA Coherent 驅動程式型記憶體管理 (CDMM)。
- 搭配 EKS 最佳化 AL2023 NVIDIA AMI 使用 NVIDIA GPU 運算子時，您必須停用驅動程式和工具組的運算子安裝，因為這些都已包含在 AMI 中。 AL2023 EKS 最佳化 AL2023 NVIDIA AMIs 不包含 NVIDIA Kubernetes 裝置外掛程式或 NVIDIA DRA 驅動程式，且必須單獨安裝。
- 每個p6e-gb200.36xlarge執行個體最多可以設定 17 個網路卡，並可以利用 EFA 在 UltraServers 之間進行通訊。工作負載網路流量可以跨 UltraServers，但為了取得最高效能，建議在利用 IMEX 進行 UltraServer GPU 內通訊的相同 UltraServer 中排程工作負載。 UltraServer 如需詳細資訊，請參閱 P6e-GB200 執行個體的 EFA 組態。
- 每個p6e-gb200.36xlarge執行個體都有 3x 7.5TB 執行個體存放區儲存空間。根據預設，EKS 最佳化 AMI 不會格式化和掛載執行個體存放區。節點的暫時性儲存可以在請求暫時性儲存的 Pod 和下載到節點的容器映像之間共用。如果使用 AL2023 EKS 最佳化 AMI，您可以透過將 NodeConfig 中的執行個體本機儲存政策設定為 RAID0，將其設定為使用者資料中節點引導的一部分。將 設定為 RAID0 會分割執行個體存放區，並設定容器執行時間和 kubelet，以利用此暫時性儲存。

元件

建議使用 P6e-GB200 UltraServers 在 EKS 上執行工作負載時，建議使用下列元件。您可以選擇性地使用 NVIDIA GPU 運算子來安裝 NVIDIA 節點元件。搭配 EKS 最佳化 AL2023 NVIDIA AMI 使用 NVIDIA GPU 運算子時，您必須停用驅動程式和工具組的運算子安裝，因為這些已包含在 AMI 中。

堆疊	元件
EKS 最佳化加速 AMI	核心 6.12
	NVIDIA GPU 驅動程式
	NVIDIA CUDA 使用者模式驅動程式

堆疊	元件
	NVIDIA 容器工具組
	NVIDIA 布料管理員
	NVIDIA IMEX 驅動程式
	NVIDIA NVLink 子網路管理員
	EFA 驅動程式
在節點上執行的元件	VPC CNI
	EFA DRA 驅動程式或 EFA 裝置外掛程式
	NVIDIA K8s 裝置外掛程式
	NVIDIA DRA 驅動程式
	NVIDIA 節點功能探索 (NFD)
	NVIDIA GPU 功能探索 (GFD)

上表中的節點元件會執行下列函數：

- VPC CNI：將 VPC IPs 配置為在 EKS 上執行之 Pod 的主要網路介面
- EFA DRA 驅動程式或 EFA 裝置外掛程式：將 EFA 裝置配置為在 EKS 上執行的 Pod 的次要網路。負責跨 P6e-GB200 UltraServers 的網路流量。對於多節點工作負載，UltraServer 內的 GPU-to-GPU 流量可以流經多節點 NVLink。EFA DRA 驅動程式建議用於 Kubernetes 1.34 和更新版本，並提供拓撲感知配置和裝置共用。所有 Kubernetes 版本都支援 EFA 裝置外掛程式。如需詳細資訊，請參閱the section called “EFA 裝置”。
- NVIDIA Kubernetes 裝置外掛程式：將 GPUs 配置為在 EKS 上執行之 Pod 的裝置。建議使用 NVIDIA Kubernetes 裝置外掛程式，直到 NVIDIA DRA 驅動程式 GPU 配置功能從實驗畢業。如需更新的資訊，請參閱 NVIDIA DRA 驅動程式版本。
- NVIDIA DRA 驅動程式：啟用 ComputeDomain 自訂資源，以促進在 P6e-GB200 UltraServers 上執行工作負載之後建立 IMEX 網域。

  - ComputeDomain 資源說明 Internode Memory Exchange (IMEX) 網域。當具有 ComputeDomain ResourceClaim 的工作負載部署到叢集時，NVIDIA DRA 驅動程式會自動建立在相符節點上執行的 IMEX DaemonSet，並在工作負載啟動之前在節點之間建立 IMEX 通道 (IMEX)。若要進一步了解 IMEX，請參閱適用於多節點 NVLink 系統的 NVIDIA IMEX 概觀。
  - NVIDIA DRA 驅動程式使用 NVIDIA GFD 套用的 clique ID 標籤 (`nvidia.com/gpu.clique`)，可轉送網路拓撲和 NVLink 網域的知識。
  - 最佳實務是為每個工作負載任務建立 ComputeDomain。
- NVIDIA 節點功能探索 (NFD)：GFD 根據探索到的節點層級屬性套用節點標籤所需的相依性。
- NVIDIA GPU 功能探索 (GFD)：將稱為 的 NVIDIA `nvidia.com/gpu.clique` 標準拓撲標籤套用至節點。相同 內的節點`nvidia.com/gpu.clique`具有多節點 NVLink 連線能力，您可以在應用程式中使用 Pod 親和性，將 Pod 排程至相同的 NVlink 網域。

程序

下節假設您有一個執行 Kubernetes 1.33 版或更新版本的 EKS 叢集，其中包含一或多個執行 AL2023 ARM NVIDIA EKS 最佳化加速 AMI 的 P6e-GB200 UltraServers 節點群組。如需 EKS 自我管理節點和受管節點群組的先決條件步驟the section called “容量管理”，請參閱 中的連結。

下列程序使用下列元件。

名稱	版本	說明
NVIDIA GPU Operator	25.3.4+	用於必要外掛程式的生命週期管理，例如 NVIDIA Kubernetes 裝置外掛程式和 NFD/GFD。
NVIDIA DRA 驅動程式	25.8.0+	對於 ComputeDomain CRDs和 IMEX 網域管理。
EFA DRA 驅動程式 (DRANET)	最新	用於與拓撲感知配置進行跨 UltraServer 通訊。建議用於 Kubernetes 1.34+。
EFA 裝置外掛程式	0.5.14+	用於跨 UltraServer 通訊。支援所有 Kubernetes 版本。

安裝 NVIDIA GPU 運算子

NVIDIA GPU 運算子可簡化在 Kubernetes 叢集中使用 GPUs 所需的元件管理。由於 NVIDIA GPU 驅動程式和容器工具組已安裝為 EKS 最佳化加速 AMI 的一部分，因此必須在 Helm 值組態`false`中將這些項目設為 。

1. 使用下列組態建立名為 gpu-operator-values.yaml的 Helm 值檔案。

```
devicePlugin:
  enabled: true
nfd:
  enabled: true
gfd:
  enabled: true
driver:
  enabled: false
toolkit:
  enabled: false
migManager:
  enabled: false
```

2. 使用您在上一個步驟中建立gpu-operator-values.yaml的檔案，為您的叢集安裝 NVIDIA GPU 運算子。

```
helm repo add nvidia https://helm.ngc.nvidia.com/nvidia
helm repo update
```

```
helm install gpu-operator nvidia/gpu-operator \
 --namespace gpu-operator \
 --create-namespace \
 --version v25.3.4 \
 --values gpu-operator-values.yaml
```

安裝 NVIDIA DRA 驅動程式

從 NVIDIA GPU 運算子版本 開始v25.3.4，必須單獨安裝 NVIDIA DRA 驅動程式。建議追蹤 NVIDIA GPU 運算子版本備註，因為這可能會在未來版本中變更。

1. 使用下列組態建立名為 dra-values.yaml的 Helm 值檔案。請注意將 DRA 驅動程式tolerations設定為僅在具有 NVIDIA GPU 的節點上部署的 nodeAffinity和 。

```
resources:
  gpus:
    enabled: false # set to false to disable experimental gpu support
  computeDomains:
    enabled: true
```

```
controller:
  nodeSelector: null
  affinity: null
  tolerations: []

kubeletPlugin:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: "nvidia.com/gpu.present"
            operator: In
            values:
            - "true"
  tolerations:
    - key: "nvidia.com/gpu"
      operator: Exists
      effect: NoSchedule
```

2. 使用您在上一個步驟中建立dra-values.yaml的檔案，為您的叢集安裝 NVIDIA DRA 驅動程式。

```
helm repo add nvidia https://helm.ngc.nvidia.com/nvidia
helm repo update
```

```
helm install nvidia-dra-driver-gpu nvidia/nvidia-dra-driver-gpu \
  --version="25.8.0" \
  --namespace nvidia-dra-driver-gpu \
  --create-namespace \
  -f dra-values.yaml
```

3. 安裝後，DRA 驅動程式會建立可讓 Kubernetes 了解和配置ComputeDomain資源DeviceClass的資源，讓 P6e-GB200 UltraServers 上的分散式 GPU 工作負載能夠進行 IMEX 管理。

   確認 DRA 資源可透過下列命令使用。

```
kubectl api-resources | grep resource.k8s.io
```

```
deviceclasses           resource.k8s.io/v1  false        DeviceClass
resourceclaims          resource.k8s.io/v1  true         ResourceClaim
resourceclaimtemplates  resource.k8s.io/v1  true         ResourceClaimTemplate
```

```
resourceslices          resource.k8s.io/v1  false        ResourceSlice
```

```
kubectl get deviceclasses
```

```
NAME
compute-domain-daemon.nvidia.com
compute-domain-default-channel.nvidia.com
```

安裝跨 UltraServer 通訊的 EFA

若要在 UltraServers 之間使用 EFA 通訊，請安裝 EFA DRA 驅動程式 (DRANET) 或 EFA 裝置外掛程式。P6e-GB200 執行個體最多可設定 17 個網路卡，且主要 NIC （索引 0) 必須為 類型，`interface`並支援高達 100 Gbps 的 ENA 頻寬。在節點佈建期間，根據您的需求設定 EFA 和 ENA 介面。如需 EFA 組態的詳細資訊，請參閱 P6e-GB200 執行個體的 EFA 組態 AWS 文件。

> **Important**
>
> 請勿在相同的節點上安裝 EFA DRA 驅動程式和 EFA 裝置外掛程式。這兩個機制不能同時存在於同一個節點上。

選項 1：安裝 EFA DRA 驅動程式 (DRANET)

1. 使用下列組態建立名為 `efa-values.yaml`的 Helm 值檔案。

```
tolerations:
  - key: nvidia.com/gpu
    operator: Exists
    effect: NoSchedule
```

2. 新增 EKS Helm Chart 儲存庫並安裝 EFA DRA 驅動程式。

```
helm repo add eks https://aws.github.io/eks-charts
helm repo update
```

```
helm install aws-dranet eks/aws-dranet --namespace kube-system -f efa-values.yaml
```

3. 確認 DRANET DaemonSet 正在執行。

```
kubectl get daemonset -n kube-system aws-dranet
```

```
NAME          DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR
 AGE
aws-dranet    2         2         2       2            2           <none>
 60s
```

4. 確認 DeviceClass和 ResourceSlice 物件可用。

```
kubectl get deviceclass efa.networking.k8s.aws
```

```
NAME                     AGE
efa.networking.k8s.aws   60s
```

```
kubectl get resourceslices -l resource.k8s.io/driver=dra.net
```

如需使用 EFA DRA 驅動程式的詳細資訊，包括拓撲感知配置和裝置共用，請參閱 the section called “EFA 裝置”。

選項 2：安裝 EFA 裝置外掛程式

1. 使用下列組態建立名為 efa-values.yaml的 Helm 值檔案。

```
tolerations:
  - key: nvidia.com/gpu
    operator: Exists
    effect: NoSchedule
```

2. 新增 EKS Helm Chart 儲存庫並安裝 EFA 裝置外掛程式。

```
helm repo add eks https://aws.github.io/eks-charts
helm repo update
```

```
helm install efa eks/aws-efa-k8s-device-plugin -n kube-system -f efa-values.yaml
```

3. 驗證 EFA 裝置外掛程式 DaemonSet 正在執行。

```
kubectl get daemonset -n kube-system aws-efa-k8s-device-plugin-daemonset
```

```
NAME                                  DESIRED   CURRENT   READY   UP-TO-DATE
 AVAILABLE   NODE SELECTOR   AGE
aws-efa-k8s-device-plugin-daemonset   2         2         2       2
      2           <none>          60s
```

4. 確認您的節點具有可配置的 EFA 裝置。例如，如果您將執行個體設定為在每個 XYZ 群組中使用 1 個僅限 efa 的界面，則預期每個節點會看到 4 個可配置的 EFA 裝置。

```
kubectl get nodes "-o=custom-columns=NAME:.metadata.name,EFA:.status.allocatable.vpc
\.amazonaws\.com/efa"
```

```
NAME                                           EFA
ip-192-168-11-225.us-west-2.compute.internal   4
ip-192-168-24-96.us-west-2.compute.internal    4
```

透過多節點 NVLink 驗證 IMEX

對於多節點 NVLINK NCCL 測試和其他微型基準，請檢閱 awesome-distributed-training GitHub 儲存庫。下列步驟說明如何使用 nvbandwidth 執行多節點 NVLink 測試。

1. 若要跨 NVL72 網域中的兩個節點執行多節點頻寬測試，請先安裝 MPI 運算子：

```
kubectl create -f https://github.com/kubeflow/mpi-operator/releases/download/v0.7.0/
mpi-operator.yaml
```

2. 建立名為 的檔案`nvbandwidth-test-job.yaml`，以定義測試資訊清單。請注意`nvidia.com/gpu.clique`，在具有多節點 NVLink 連線能力的相同 NVLink 網域中排程工作者的 Pod 親和性。以下範例使用 cuMemcpyAsync 執行多節點device-to-device CE 讀取 memcpy 測試，並在日誌中列印結果。

   從 NVIDIA DRA 驅動程式版本 `v25.8.0` ComputeDomains 開始是彈性的，`.spec.numNodes`可以在 ComputeDomain 定義`0`中設定為 。檢閱最新的 NVIDIA DRA 驅動程式版本備註以取得更新。

   每個節點只能有一個 ComputeDomain (IMEX 頻道）。請勿將 ComputeDomain 資源`All`的 `allocationMode` 變更為 ，因為這可能會導致存取該 ComputeDomain 的 ComputeDomain 和

Pod 無法正確配置和排程。 ComputeDomain 如需詳細資訊，請參閱 NVIDIA DRA 驅動程式問題 #353。

```
---
apiVersion: resource.nvidia.com/v1beta1
kind: ComputeDomain
metadata:
  name: nvbandwidth-test-compute-domain
spec:
  numNodes: 0 # This can be set to 0 from NVIDIA DRA Driver version v25.8.0+
  channel:
    resourceClaimTemplate:
      name: nvbandwidth-test-compute-domain-channel

---
apiVersion: kubeflow.org/v2beta1
kind: MPIJob
metadata:
  name: nvbandwidth-test
spec:
  slotsPerWorker: 4 # 4 GPUs per worker node
  launcherCreationPolicy: WaitForWorkersReady
  runPolicy:
    cleanPodPolicy: Running
  sshAuthMountPath: /home/mpiuser/.ssh
  mpiReplicaSpecs:
    Launcher:
      replicas: 1
      template:
        metadata:
          labels:
            nvbandwidth-test-replica: mpi-launcher
        spec:
          affinity:
            nodeAffinity:
              requiredDuringSchedulingIgnoredDuringExecution:
                nodeSelectorTerms:
                - matchExpressions:
                  # Only schedule on NVIDIA GB200/GB300 nodes
                  - key: node.kubernetes.io/instance-type
                    operator: In
                    values:
                    - p6e-gb200.36xlarge
```

```
                - p6e-gb300.36xlarge
        containers:
        - image: ghcr.io/nvidia/k8s-samples:nvbandwidth-v0.7-8d103163
          name: mpi-launcher
          securityContext:
            runAsUser: 1000
          command:
          - mpirun
          args:
          - --bind-to
          - core
          - --map-by
          - ppr:4:node
          - -np
          - "8"
          - --report-bindings
          - -q
          - nvbandwidth
          - -t
          - multinode_device_to_device_memcpy_read_ce
    Worker:
      replicas: 2 # 2 worker nodes
      template:
        metadata:
          labels:
            nvbandwidth-test-replica: mpi-worker
        spec:
          affinity:
            nodeAffinity:
              requiredDuringSchedulingIgnoredDuringExecution:
                nodeSelectorTerms:
                - matchExpressions:
                  # Only schedule on NVIDIA GB200/GB300 nodes
                  - key: node.kubernetes.io/instance-type
                    operator: In
                    values:
                    - p6e-gb200.36xlarge
                    - p6e-gb300.36xlarge
            podAffinity:
              requiredDuringSchedulingIgnoredDuringExecution:
              - labelSelector:
                  matchExpressions:
                  - key: nvbandwidth-test-replica
                    operator: In
```

```
                values:
                - mpi-worker
            topologyKey: nvidia.com/gpu.clique
      containers:
      - image: ghcr.io/nvidia/k8s-samples:nvbandwidth-v0.7-8d103163
        name: mpi-worker
        securityContext:
          runAsUser: 1000
        env:
        command:
        - /usr/sbin/sshd
        args:
        - -De
        - -f
        - /home/mpiuser/.sshd_config
        resources:
          limits:
            nvidia.com/gpu: 4  # Request 4 GPUs per worker
          claims:
          - name: compute-domain-channel # Link to IMEX channel
      resourceClaims:
      - name: compute-domain-channel
        resourceClaimTemplateName: nvbandwidth-test-compute-domain-channel
```

3. 建立 ComputeDomain 並使用下列命令啟動任務。

```
kubectl apply -f nvbandwidth-test-job.yaml
```

4. 建立 ComputeDomain 之後，您可以看到工作負載的 ComputeDomain 有兩個節點：

```
kubectl get computedomains.resource.nvidia.com -o yaml
```

```
status:
  nodes:
  - cliqueID: <ClusterUUID>.<Clique ID>
    ipAddress: <node-ip>
    name: <node-hostname>
  - cliqueID: <ClusterUUID>.<Clique ID>
    ipAddress: <node-ip>
    name: <node-hostname>
  status: Ready
```

5. 使用下列命令檢閱任務的結果。

```
kubectl logs --tail=-1 -l job-name=nvbandwidth-test-launcher
```

成功的測試會以 GB/s 為單位顯示多節點 memcpy 測試的頻寬統計資料。成功測試輸出的範例如下所示。

```
...
nvbandwidth Version: ...
Built from Git version: ...

MPI version: ...
CUDA Runtime Version: ...
CUDA Driver Version: ...
Driver Version: ...

Process 0 (nvbandwidth-test-worker-0): device 0: NVIDIA GB200 (...)
Process 1 (nvbandwidth-test-worker-0): device 1: NVIDIA GB200 (...)
Process 2 (nvbandwidth-test-worker-0): device 2: NVIDIA GB200 (...)
Process 3 (nvbandwidth-test-worker-0): device 3: NVIDIA GB200 (...)
Process 4 (nvbandwidth-test-worker-1): device 0: NVIDIA GB200 (...)
Process 5 (nvbandwidth-test-worker-1): device 1: NVIDIA GB200 (...)
Process 6 (nvbandwidth-test-worker-1): device 2: NVIDIA GB200 (...)
Process 7 (nvbandwidth-test-worker-1): device 3: NVIDIA GB200 (...)

Running multinode_device_to_device_memcpy_read_ce.
memcpy CE GPU(row) -> GPU(column) bandwidth (GB/s)
           0         1         2         3         4         5         6         7
 0       N/A    821.45    822.18    821.73    822.05    821.38    822.61    821.89
 1    822.34       N/A    821.67    822.12    821.94    820.87    821.53    822.08
 2    821.76    822.29       N/A    821.58    822.43    821.15    821.82    822.31
 3    822.19    821.84    822.05       N/A    821.67    821.23    820.95    822.47
 4    821.63    822.38    821.49    822.17       N/A    821.06    821.78    822.22
 5    822.08    821.52    821.89    822.35    821.27       N/A    821.64    822.13
 6    821.94    822.15    821.68    822.04    821.39    820.92       N/A    822.56
 7    822.27    821.73    822.11    821.86    822.38    821.04    821.49       N/A

SUM multinode_device_to_device_memcpy_read_ce ...

NOTE: The reported results may not reflect the full capabilities of the platform.
Performance can vary with software drivers, hardware clocks, and system topology.
```

6. 測試完成時，請使用下列命令將其刪除。

```
kubectl delete -f nvbandwidth-test-job.yaml
```

在 Amazon EKS 上管理硬體裝置

Amazon EKS 支援兩種 Kubernetes 機制來管理 EKS 叢集中的特殊硬體裝置：動態資源分配 (DRA) 和裝置外掛程式。這兩種機制都可讓工作負載存取 NVIDIA GPUs 和 AWS Trainium 晶片等硬體加速器，以及 Elastic Fabric Adapter (EFA) 等高效能網路裝置。

在 Karpenter、EKS 受管節點群組或自我管理節點中使用靜態容量佈建時，我們建議將 DRA 驅動程式用於 Kubernetes 1.34 版及更新版本的新部署。EKS Auto 模式目前不支援 DRA。DRA 提供更豐富的裝置選擇、拓撲感知排程，以及裝置外掛程式無法實現的裝置共用功能。

如需這兩個 Kubernetes 功能的一般資訊，請參閱動態資源分配和裝置外掛程式的 Kubernetes 文件。

動態資源分配與裝置外掛程式的比較

Kubernetes 裝置外掛程式是向 Kubernetes 工作負載公開專用硬體的主要機制。裝置外掛程式會將裝置公告為您在容器資源請求和限制中請求的延伸資源（例如 `nvidia.com/gpu`或 `aws.amazon.com/neuroncore`)。雖然裝置外掛程式受到廣泛支援和使用，但它們具有限制：

- 系統會將裝置請求為不透明整數計數，而沒有屬性型篩選。
- 不支援容器或 Pod 之間的裝置共用。
- 沒有跨裝置類型的表達式拓撲感知配置。
- 智慧型置放通常需要自訂排程器延伸。

動態資源分配 (DRA) 是 Kubernetes 1.34 版中普遍提供的 Kubernetes 功能，可解決這些限制。透過 DRA，裝置驅動程式會透過 `ResourceSlice` 物件將豐富的裝置屬性發佈至 Kubernetes 排程器。您可以使用 `ResourceClaim`和參考`DeviceClass`類別的`ResourceClaimTemplate`物件來請求裝置。

DRA 啟用：

- 使用通用表達式語言 (CEL) 表達式的屬性型裝置選擇。
- 拓撲感知配置，可確保裝置位於相同的 PCIe 交換器或 NUMA 網域上。
- 透過共用`ResourceClaim`參考在多個容器或 Pod 之間共用裝置。
- 使用 MIG 或時間分割時動態分割和共用 NVIDIA GPUs

Amazon EKS 的 DRA 驅動程式

下列 DRA 驅動程式通常用於管理 Amazon EKS 叢集中的特殊硬體裝置。

NVIDIA DRA 驅動程式

GitHub 上適用於 GPUs 的 NVIDIA DRA 驅動程式可靈活配置和動態設定 NVIDIA GPUs。the section called “DRA 和裝置外掛程式” 如需使用 NVIDIA DRA 驅動程式管理 GPUs 的相關資訊the section called “搭配 EKS 使用 P6e-GB200 ”，以及使用 EC2 Grace-Blackwell 執行個體`ComputeDomains`處理多節點 NVLink (MNNVL) 工作負載的相關資訊，請參閱 。

EFA DRA 驅動程式

EFA DRA 驅動程式 (GitHub 上的 DRANET) 使用拓撲感知排程來管理 Elastic Fabric Adapter (EFA) 裝置配置，將 EFA 介面與其拓撲本機 GPUs 或 Neuron 裝置配對，並支援 Pod 之間的裝置共用。 GitHub 如需詳細資訊，請參閱the section called “EFA 裝置”。

Neuron DRA 驅動程式

Neuron DRA 驅動程式使用拓撲感知排程、連線裝置子集配置和邏輯 NeuronCore (LNC) 組態來管理 AWS Trainium 和 AWS Inferentia2 裝置配置，而不需要自訂排程器延伸。如需詳細資訊，請參閱the section called “Neuron 加速器”。

Amazon EKS 的裝置外掛程式

下列裝置外掛程式通常用於管理 Amazon EKS 叢集中的特殊硬體裝置。

NVIDIA 裝置外掛程式

GitHub 上的 NVIDIA 裝置外掛程式會將 NVIDIA GPUs 公告為`nvidia.com/gpu`延伸資源，並追蹤 GPUs 的運作狀態。

EFA 裝置外掛程式

EFA 裝置外掛程式會探索每個節點上所有可用的 EFA 裝置，並將 EFA 裝置公告為`vpc.amazonaws.com/efa`延伸資源。

Neuron 裝置外掛程式

Neuron 裝置外掛程式會將 Neuron 硬體公開為 `aws.amazon.com/neuroncore`和`aws.amazon.com/neuron`延伸資源。它會探索每個節點上可用的 Neuron 裝置、將它們公告為可配置的資源，並管理其生命週期。

考量事項

在 Amazon EKS 上使用 DRA 驅動程式之前，請檢閱下列考量事項：

- DRA 適用於 Amazon EKS 搭配 Kubernetes 1.33 版及更新版本，但建議用於 Kubernetes 1.34 版及更新版本，因為 GitHub 上存在上游 Kubernetes 問題。您的叢集控制平面和節點必須執行支援 DRA 的 Kubernetes 版本。
- DRA 目前與 EKS Auto 模式不相容。
- 使用動態佈建容量時，DRA 目前與 Karpenter 不相容。您必須使用 Karpenter 中的靜態容量佈建，或 EKS 受管節點群組或具有 DRA 驅動程式的自我管理節點。
- 相同裝置類型的 DRA 驅動程式和裝置外掛程式不得同時在相同節點上執行。在安裝對應的 DRA 驅動程式之前，請先解除安裝裝置外掛程式，或將其部署在不同的節點上。在相同節點上執行相同裝置的 DRA 驅動程式和裝置外掛程式，可能會導致基礎硬體裝置的無提示過度訂閱。
- DRA 使用與裝置外掛程式 (`ResourceClaim`、`DeviceClass`) `ResourceClaimTemplate`不同的 Kubernetes API 資源 (`resource.limits`、、)`resource.requests`。您可以使用 DRA 來管理裝置外掛程式延伸資源，而無需變更工作負載規格。如需 DRA 中擴充資源的詳細資訊，請參閱 Kubernetes 網站上的 Kubernetes 文件。
- 適用於 NVIDIA、EFA 和 Neuron 的 DRA 驅動程式與 EKS 最佳化 AL2023 AMIs 和 Bottlerocket AMIs。如果您使用 NVIDIA DRA 驅動程式搭配 Bottlerocket，請停用包含在 Bottlerocket NVIDIA 變體中的 NVIDIA 裝置外掛程式。
- 所有 Kubernetes 版本都完全支援裝置外掛程式。

DRA ResourceClaim 與 ResourceClaimTemplate

使用 DRA 時，您可以透過 `ResourceClaim`或 `ResourceClaimTemplate` 物件請求裝置。這兩種資源類型具有不同的用途，並具有不同的生命週期行為。

ResourceClaim

`ResourceClaim` 是您獨立於任何 Pod 建立的具名 Kubernetes 物件。您可以使用 `resourceClaimName` 欄位，在 Pod 規格中依名稱加以參考。`ResourceClaim` 具有下列特性：

- 在建立任何參考叢集的 Pod 之前，它必須存在於叢集中。如果宣告不存在，Pod 會保持待定狀態。
- 它會持續存在，直到您明確刪除它為止，無論是否有任何 Pod 參考它。
- 多個 Pod 可以參考相同的 `ResourceClaim`，以啟用裝置共用。所有參考相同宣告的 Pod 都會共用相同配置裝置的存取權，並排程到相同的節點。

ResourceClaim 當您需要多個 Pod 來共用對相同裝置的存取權，或需要宣告在單一 Pod 的生命週期之後存在時，請使用 。

ResourceClaimTemplate

ResourceClaimTemplate 定義 Kubernetes 用來為每個 Pod 自動產生唯一 ResourceClaim的範本。您可以使用 resourceClaimTemplateName 欄位在 Pod 規格中參考它。ResourceClaimTemplate 本身不繫結至任何 Pod，它是可重複使用的範本，可獨立保留。ResourceClaimTemplate 具有下列特性：

- Kubernetes 會ResourceClaim為每個參考範本的 Pod 建立新的 。每個 Pod 都會取得自己的個別裝置集。
- 每個產生的 ResourceClaim 都會繫結至觸發其建立的 Pod 生命週期。刪除 Pod 時，ResourceClaim也會刪除產生的關聯 。ResourceClaimTemplate 本身不受影響，並繼續為未來的 Pod 產生新的宣告。

  當工作負載中的每個 Pod 需要具有類似組態的專屬裝置ResourceClaimTemplate時，請使用 。例如，ResourceClaimTemplate針對使用平行執行的任務中的 Pod，使用 ，其中每個 Pod 都需要自己的 GPU 或 EFA 裝置。

下表摘要說明 ResourceClaim和 之間的差異ResourceClaimTemplate。

Behavior (行為)	ResourceClaim	ResourceClaimTemplate
建立	您在 Pod 參考它之前手動建立它	Kubernetes 會根據 Pod 自動產生宣告
生命週期	持續到您刪除為止	範本會持續存在，直到您刪除為止。產生的每個 ResourceClaim 都會繫結至觸發其建立的 Pod。
在 Pod 之間共用裝置	支援。多個 Pod 可以參考相同的宣告。	不支援。每個 Pod 都會取得個別的宣告。
Pod 規格欄位	resourceClaimName	resourceClaimTemplateName

如需使用ResourceClaim物件在 Pod 之間共用 EFA 裝置的範例，請參閱 the section called “在多個 Pod 之間共用 EFA 裝置”。如需搭配拓撲感知配置使用ResourceClaimTemplate物件的範例，請參閱 the section called “拓撲感知 EFA 和 GPU/Neuron 裝置配置”。

主題

- the section called “EFA 裝置”
- the section called “Neuron 加速器”
- the section called “NVIDIA GPU”
- the section called “DRA 和裝置外掛程式”
- the section called “多執行個體 GPUs (MIG)”
- the section called “時間分割”

在 Amazon EKS 上管理 EFA 裝置

Elastic Fabric Adapter (EFA) 是 Amazon EC2 執行個體的網路裝置，可為人工智慧、機器學習和高效能運算 (HPC) 工作負載啟用高效能節點間通訊和 RDMA （遠端直接記憶體存取）。Amazon EKS 支援兩種在 EKS 叢集中管理 EFA 裝置的機制：EFA 動態資源分配 (DRA) 驅動程式 (DRANET) 和 EFA 裝置外掛程式。

在 Karpenter、EKS 受管節點群組或自我管理節點中使用靜態容量佈建時，我們建議將 DRA 驅動程式用於 Kubernetes 1.34 版及更新版本的新部署。EKS Auto 模式目前不支援 DRA。使用 EFA DRA 驅動程式，您可以設定拓撲感知配置，將 EFA 介面與其拓撲本機 GPUs 或 Neuron 裝置配對，並在 Pod 之間共用裝置。

EFA DRA 驅動程式與 EFA 裝置外掛程式

功能	EFA DRA 驅動程式	EFA 裝置外掛程式
最低 Kubernetes 版本	1.34	所有 EKS 支援的 Kubernetes 版本
EKS 運算	Karpenter （僅限靜態容量）、受管節點群組、自我管理節點	EKS Auto Mode、Karpenter、受管節點群組、自我管理節點
EKS 最佳化 AMIs	AL2023， Bottlerocket	AL2023， Bottlerocket

功能	EFA DRA 驅動程式	EFA 裝置外掛程式
裝置廣告	透過裝置類型、拓撲和 PCIe 地區性等`ResourceSlice` 物件的豐富屬性	`vpc.amazonaws.com/efa` 延伸資源的整數計數
GPU-EFA 親和性	DRA 原生拓撲感知	自動拓撲感知（僅限 EKS 最佳化 AL2023 AMIs)
Neuron-EFA 親和性	DRA 原生拓撲感知	自動拓撲感知（僅限 EKS 最佳化 AL2023 AMIs)
裝置共用	多個 Pod 可以透過共用`ResourceClaim` 參考共用相同的 EFA 裝置	不支援。每個 EFA 裝置只會配置給一個 Pod。

使用 EFA 介面建立 EKS 節點

當您使用 EFA 介面建立 EKS 節點時，EFA 介面會在執行個體佈建期間連接至執行個體。您可以自訂每個裝置的 EFA 組態，並使用置放群組搭配 EKS Auto Mode、Karpenter、EKS 受管節點群組或 EKS 自我管理節點群組。使用 EKS Auto 模式，您可以透過 `NodeClass`下的 傳遞每個網路界面的組態`advancedNetworking.networkInterfaces`。使用 Karpenter，您可以透過 傳遞每個網路界面的組態`EC2NodeClass`。透過 EKS 受管節點群組或自我管理節點，您可以使用啟動範本傳遞每個網路介面的組態。

使用 搭配 `efaEnabled`設定`eksctl`佈建 EKS 節點時，所有介面都會設定為介面類型 EFA、建立 EFA 特定的安全群組，並在叢集上安裝 EFA 裝置外掛程式。如果您需要在使用 時自訂每個裝置的 EFA 組態`eksctl`，建議您使用 `eksctl 對啟動範本的支援。

下列範例示範如何使用 EFA 介面設定`NodeClass`和啟動範本。這有助於自訂用於 EFA 和標準 IP 型流量的介面。如需每個執行個體類型支援的 EFA 介面數量，以及如何設定它們以取得最大網路頻寬的資訊，請參閱《Amazon EC2 使用者指南》中的為啟用 EFA 的執行個體類型最大化網路頻寬。

EKS 自動模式

在 EKS Auto 模式中，您可以使用 `NodeClass()` 中的 `advancedNetworking.networkInterfaces` 欄位來設定 EFA 網路介面`eks.amazonaws.com/v1`。每個項目都會指定 `networkCardIndex`、 `deviceIndex`和 `interfaceType`。`interfaceType` 可以是`interface`標準網路介面，也可以`efa-only`是專用於未指派 IP 地址之 RDMA 流量的 EFA 介面。

設定 networkInterfaces 時，參考 啟動NodePool的執行個體NodeClass會使用此組態，無論 Pod 是否請求vpc.amazonaws.com/efa資源。對於具有靜態網路介面組態的節點，EKS Auto Mode 不會在執行個體啟動後連接其他 IPs、字首或 ENIs，只有啟動時設定的介面和 IPs 可供 Pod 使用。

此功能可與靜態容量節點集區搭配使用，以維護適用於分散式訓練和推論工作負載的預先暖機 EFA 就緒節點。

EKS Auto Mode NodeClass 範例搭配 P5 的僅限 EFA 介面

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: efa-node-class
spec:
  role: MyNodeRole
  subnetSelectorTerms:
    - tags:
        Name: "private-subnet"
  securityGroupSelectorTerms:
    - tags:
        Name: "efa-security-group"
  placementGroupSelector:
    name: "ml-training-pg"
  advancedNetworking:
    networkInterfaces:
    - deviceIndex: 0
      interfaceType: interface
      networkCardIndex: 0
      secondaryIPv4PrefixCount: 1
    - networkCardIndex: 0
      deviceIndex: 1
      interfaceType: efa-only
    - networkCardIndex: 1
      deviceIndex: 0
      interfaceType: efa-only
    - networkCardIndex: 2
      deviceIndex: 0
      interfaceType: efa-only
    - networkCardIndex: 3
      deviceIndex: 0
      interfaceType: efa-only
```

如需靜態網路介面組態限制的完整清單，請參閱 NodeClass 文件the section called “靜態網路界面組態”中的 。

Karpenter

中的每個項目都會networkInterfaces指定 networkCardIndex、 deviceIndex和 interfaceType。interfaceType 可以interface用於標準網路介面，也可以efa-only用於專用於 RDMA 流量且未指派 IP 地址的 EFA 介面。設定 networkInterfaces 時，參考 啟動NodePool的執行個體NodeClass會使用此組態，無論 Pod 是否請求vpc.amazonaws.com/efa資源。

使用 Karpenter 而不在 networkInterfaces中指定 時NodeClass，為請求的 Pod 建立的執行個體vpc.amazonaws.com/efa會設定所有介面類型 EFA。

的networkInterfaces組態EC2NodeClass已在 Karpenter 1.11 版中新增。下列範例顯示為具有 1 個 ENA 介面和 8 個僅限 EFA 介面的 P6-B200 執行個體EC2NodeClass設定的 。

範例 Karpenter EC2NodeClass 搭配 P6-B200 的僅限 EFA 介面

```
apiVersion: karpenter.k8s.aws/v1
kind: EC2NodeClass
metadata:
  name: efa-node-class
spec:
  networkInterfaces:
  - networkCardIndex: 0
    deviceIndex: 0
    interfaceType: interface
  - networkCardIndex: 0
    deviceIndex: 1
    interfaceType: efa-only
  - networkCardIndex: 1
    deviceIndex: 0
    interfaceType: efa-only
  - networkCardIndex: 2
    deviceIndex: 0
    interfaceType: efa-only
  - networkCardIndex: 3
    deviceIndex: 0
    interfaceType: efa-only
  - networkCardIndex: 4
    deviceIndex: 0
    interfaceType: efa-only
```

```
  - networkCardIndex: 5
    deviceIndex: 0
    interfaceType: efa-only
  - networkCardIndex: 6
    deviceIndex: 0
    interfaceType: efa-only
  - networkCardIndex: 7
    deviceIndex: 0
    interfaceType: efa-only
```

EKS 受管節點群組和自我管理節點

透過 EKS 受管節點群組或自我管理節點，您可以使用啟動範本傳遞每個網路介面的組態。

下列範例顯示為具有 1 個 ENA 介面和 8 個僅限 EFA 介面的 P6-B200 執行個體設定的啟動範本。主要網路介面（網路卡 0、裝置索引 0) 使用 IP 流量的標準`interface`類型，而其他介面則`efa-only`用於專用 RDMA 流量。根據您的執行個體類型調整`efa-only`介面數量。如需每個執行個體類型支援的 EFA 介面數量，請參閱《Amazon EC2 使用者指南》中的啟用 EFA 執行個體類型的網路頻寬最大化。

P6-B200 搭配僅限 EFA 介面的啟動範本範例

將 取代 *`security-group-id`* 為您的值。安全群組必須允許進出本身的所有傳入和傳出流量，才能啟用 EFA OS-bypass 功能。如需詳細資訊，請參閱《Amazon EC2 使用者指南》中的步驟 1：準備啟用 EFA 的安全群組。

> **Important**
>
> 使用 EKS 受管節點群組時，請勿在啟動範本`SubnetId`中指定 。EKS 要求透過 `CreateNodegroup` API 指定所有子網路，並拒絕包含子網路組態的啟動範本。

```
{
  "LaunchTemplateName": "efa-launch-template",
  "LaunchTemplateData": {
    "InstanceType": "p6-b200.48xlarge",
    "NetworkInterfaces": [
      {
        "NetworkCardIndex": 0,
        "DeviceIndex": 0,
        "InterfaceType": "interface",
        "Groups": ["security-group-id"]
      },
```

```
      {
        "NetworkCardIndex": 0,
        "DeviceIndex": 1,
        "InterfaceType": "efa-only",
        "Groups": ["security-group-id"]
      },
      {
        "NetworkCardIndex": 1,
        "DeviceIndex": 0,
        "InterfaceType": "efa-only",
        "Groups": ["security-group-id"]
      },
      {
        "NetworkCardIndex": 2,
        "DeviceIndex": 0,
        "InterfaceType": "efa-only",
        "Groups": ["security-group-id"]
      },
      {
        "NetworkCardIndex": 3,
        "DeviceIndex": 0,
        "InterfaceType": "efa-only",
        "Groups": ["security-group-id"]
      },
      {
        "NetworkCardIndex": 4,
        "DeviceIndex": 0,
        "InterfaceType": "efa-only",
        "Groups": ["security-group-id"]
      },
      {
        "NetworkCardIndex": 5,
        "DeviceIndex": 0,
        "InterfaceType": "efa-only",
        "Groups": ["security-group-id"]
      },
      {
        "NetworkCardIndex": 6,
        "DeviceIndex": 0,
        "InterfaceType": "efa-only",
        "Groups": ["security-group-id"]
      },
      {
        "NetworkCardIndex": 7,
```

```
        "DeviceIndex": 0,
        "InterfaceType": "efa-only",
        "Groups": ["security-group-id"]
      }
    ]
  }
}
```

搭配 EFA 使用 EKS 最佳化 AMIs

EKS 最佳化 AL2023 AMIs 和所有 Bottlerocket AMIs 包含使用 EFA 所需的主機層級元件，特別是 aws-efa-installer 安裝的元件。EKS AL2023 和 Bottlerocket AMIs 不包含 EFA DRA 驅動程式或 EFA 裝置外掛程式，而且這些必須分別安裝在叢集上，才能部署工作負載。

保留 IP 地址配置

`p5.48xlarge` 和 等已啟用 EFA 的執行個體`p6-b200.48xlarge`支援許多網路介面。根據預設，Amazon VPC CNI 會將 IP 地址分配到所有已啟用 IP 的連接 ENIs，即使 Pod 未主動使用這些地址，也會從您的子網路取用大量 IP 地址。在具有數十個網路介面的執行個體上，這可能會快速耗盡子網路的可用 IP 空間。

若要減少啟用 EFA 節點的 IP 地址使用量，請將您的網路介面設定為`efa-only`用於主要節點以外的所有介面。僅限 EFA 的介面專用於 RDMA 流量，且沒有指派 IP 地址，因此不會耗用子網路中的地址。如需範例組態，請參閱 the section called “Karpenter”和 the section called “EKS 受管節點群組和自我管理節點”。如需每個執行個體類型的建議界面配置，請參閱《Amazon EC2 使用者指南》中的為啟用 EFA 的執行個體類型最大化網路頻寬。

除了使用`efa-only`介面之外，您還可以設定 Amazon VPC CNI 來限制暖 （預先配置） IP 地址和 ENIs 的數量。根據預設，VPC CNI 會預先配置 ENIs 和 IP 地址的暖集區，以加快 Pod 啟動速度，但在大型執行個體上，可以保留數百個未使用的 IP 地址。在 `aws-node` DaemonSet 上設定`WARM_IP_TARGET`和 `WARM_ENI_TARGET`環境變數，以控制 CNI 維護的備用 IP 地址和 ENIs 數量。如需這些設定的詳細資訊，請參閱 Amazon VPC CNI 最佳實務。

> **Note**
>
> `WARM_ENI_TARGET` 和 `WARM_IP_TARGET`設定適用於整個叢集，並適用於 VPC CNI 管理的所有節點。目前無法為每個節點群組或執行個體類型設定不同的值。如果您需要更精細地控制這些設定，請在 GitHub 上提供有關容器藍圖問題 #1834 的意見回饋。

安裝 EFA DRA 驅動程式 (DRANET)

EFA DRA 驅動程式內建於 GitHub 上的上游 DRANET 專案中，可提供 Kubernetes DRA 的雲端感知網路裝置管理。本文件中會互換使用 EFA DRA 驅動程式和 DRANET，並參考相同的工具。

EFA DRA 驅動程式會將 EFA 裝置公告為具有驅動程式名稱 `dra.net`和`DeviceClass`名稱的`ResourceSlice`物件`efa.networking.k8s.aws`。EFA DRA 驅動程式會在每個節點上以 DaemonSet 身分執行，並自動探索 EFA 裝置。

先決條件

- 執行 Kubernetes 1.34 版或更新版本的 Amazon EKS 叢集，具有由 Karpenter、EKS 受管節點群組或自我管理節點群組佈建的靜態容量。
- 具有啟用 EFA 的 Amazon EC2 執行個體類型的節點。如需支援的執行個體類型清單，請參閱《Amazon EC2 使用者指南》中的支援的執行個體類型。
- 針對 EFA 安裝主機層級元件的節點，如需詳細資訊，請參閱安裝 EFA 軟體。EKS 最佳化 AL2023 NVIDIA 和 Neuron AMIs，以及 Bottlerocket AMIs 包含 EFA 主機層級元件。
- 已在命令列環境中安裝 Helm，如需詳細資訊，請參閱安裝 Helm 說明。
- `kubectl` 設定為與您的叢集通訊，the section called “安裝或更新 kubectl”如需詳細資訊，請參閱 。

程序

> **Important**
>
> 請勿在執行 EFA 裝置外掛程式的節點上安裝 EFA DRA 驅動程式。這兩個機制不能同時存在於同一個節點上。這樣做可能會導致基礎裝置的靜音過度訂閱到相同節點上的多個 Pod。

1. 新增 EKS Helm Chart 儲存庫。

```
helm repo add eks https://aws.github.io/eks-charts
```

2. 更新本機 Helm 儲存庫。

```
helm repo update
```

3. 使用 Helm 在叢集上安裝 EFA DRA 驅動程式。EFA DRA 驅動程式會自動透過執行個體中繼資料服務 (IMDS) 在 EC2 執行個體上執行，並啟用 EFA 裝置探索。根據預設，EFA DRA 驅動程式會部署為kube-system命名空間中的 DaemonSet。如需可設定的參數，請參閱 GitHub 上的 EKS Helm Chart 儲存庫中的 Helm values.yaml。

```
helm install aws-dranet eks/aws-dranet --namespace kube-system
```

4. 確認 DRANET DaemonSet 正在執行。

```
kubectl get daemonset -n kube-system aws-dranet
```

```
NAME          DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR
 AGE
aws-dranet    2         2         2       2            2           <none>
 60s
```

5. 確認DeviceClass已建立 。

```
kubectl get deviceclass
```

```
NAME                     AGE
efa.networking.k8s.aws   60s
```

6. 確認已為您的節點公告ResourceSlice物件。

```
kubectl get resourceslices --field-selector spec.driver=dra.net
```

如果上述步驟發生錯誤，您可以使用下列命令檢查 DRANET 的日誌。

```
kubectl logs -n kube-system -l app=aws-dranet
```

7. 若要使用 DRA 驅動程式請求 EFA 裝置，請建立參考 EFA ResourceClaimTemplate 的 ResourceClaim或 ，DeviceClass並在 Pod 規格中參考它。下列範例會請求單一 EFA 裝置。

```
apiVersion: resource.k8s.io/v1
kind: ResourceClaimTemplate
metadata:
  name: single-efa-claim
spec:
```

```
  spec:
    devices:
      requests:
      - name: efa
        exactly:
          deviceClassName: efa.networking.k8s.aws
          count: 1
---
apiVersion: v1
kind: Pod
metadata:
  name: efa-workload
spec:
  containers:
  - name: app
    ...
    resources:
      claims:
      - name: efa-device
  resourceClaims:
  - name: efa-device
    resourceClaimTemplateName: single-efa-claim
```

拓撲感知 EFA 和 GPU/Neuron 裝置配置

EFA DRA 驅動程式支援拓撲感知配置，可將 EFA 介面與相同 PCIe 根目錄上的 GPUs 或 Neuron 裝置配對。使用 `matchAttribute` 限制條件來對齊 EFA 和 GPU 或 Neuron 裝置配置。若要使用此功能，您還必須使用 NVIDIA 或 Neuron DRA 驅動程式。如需詳細資訊，請參閱the section called “NVIDIA GPU”及the section called “Neuron 加速器”。

下列範例會請求與 1 個 NVIDIA GPU 一致的 1 個 EFA 介面：

```
apiVersion: resource.k8s.io/v1
kind: ResourceClaimTemplate
metadata:
  name: aligned-efa-nvidia
spec:
  spec:
    devices:
      requests:
      - name: 1-efa
        exactly:
```

```
          deviceClassName: efa.networking.k8s.aws
          count: 1
      - name: 1-gpu
        exactly:
          deviceClassName: gpu.nvidia.com
          count: 1
      constraints:
      - requests: ["1-gpu", "1-efa"]
        matchAttribute: "resource.kubernetes.io/pcieRoot"
```

下列範例會請求與 4 個 Neuron 裝置對齊的 4 個 EFA 介面：

```
apiVersion: resource.k8s.io/v1
kind: ResourceClaimTemplate
metadata:
  name: aligned-efa-neuron
spec:
  spec:
    devices:
      requests:
      - name: 4-neurons
        exactly:
          deviceClassName: neuron.aws.com
          count: 4
      - name: 4-efas
        exactly:
          deviceClassName: efa.networking.k8s.aws
          count: 4
      constraints:
      - requests: ["4-neurons", "4-efas"]
        matchAttribute: "resource.aws.com/devicegroup4_id"
```

devicegroup 屬性名稱中的數字對應至已連線拓撲群組中的 Neuron 裝置數量。例如，resource.aws.com/devicegroup1_id 識別單一 Neuron 裝置、resource.aws.com/devicegroup4_id識別 4 個連線裝置的群組，resource.aws.com/devicegroup8_id以及分別resource.aws.com/devicegroup16_id識別 8 和 16 個連線裝置的群組。選擇matchAttribute符合您請求count中裝置的 ，以便配置的 Neuron 裝置和 EFA 介面屬於相同的連線拓撲群組。如需這些屬性的詳細資訊，請參閱 Neuron DRA 驅動程式文件。

您可以使用 allocationMode 簡化 EFA 裝置如何配置給對齊的 GPU 或 Neuron 加速器。allocationMode 欄位支援兩個值： ExactCount（預設值） 請求 指定的特定數量裝置count，並All請求集區中的所有相符裝置。例如，在p5.48xlarge執行個體上，有四個 EFA 裝

置與一個 GPU 共用相同的 PCIe 根。若要使用對齊的 GPUs 配置這些 EFA 裝置群組，即使您不知道確切的 EFA-GPU 裝置映射和對齊的 EFA 裝置計數，您也可以ResourceClaimTemplate使用 allocationMode: All 為 EFA 裝置設定您的 。

```
apiVersion: resource.k8s.io/v1
kind: ResourceClaimTemplate
metadata:
  name: aligned-all-efa-one-nvidia
spec:
  spec:
    devices:
      requests:
      - name: all-efas
        exactly:
          deviceClassName: efa.networking.k8s.aws
          allocationMode: All
      - name: one-gpu
        exactly:
          deviceClassName: gpu.nvidia.com
          allocationMode: ExactCount
          count: 1
      constraints:
      - requests: ["all-efas", "one-gpu"]
        matchAttribute: "resource.kubernetes.io/pcieRoot"
```

在多個 Pod 之間共用 EFA 裝置

EFA DRA 驅動程式支援使用 在多個 Pod 之間共用 EFA 裝置ResourceClaim。與為每個 Pod ResourceClaimTemplate產生個別宣告的 不同， ResourceClaim是您獨立建立並從多個 Pod 參考的具名物件。所有參考相同ResourceClaim已配置 EFA 裝置共用存取權的 Pod，並排程到提供這些裝置的相同節點。

若要在 Pod 之間共用 EFA 裝置，請建立ResourceClaim請求 EFA 裝置的 ，然後使用 在每個 Pod 的 resourceClaims 欄位中依名稱參考該宣告resourceClaimName。在建立參考叢集的 Pod 之前，叢集中ResourceClaim必須存在 。如果參考ResourceClaim不存在，Pod 會保持待定狀態，直到建立宣告為止。

下列範例會建立ResourceClaim請求 4 個 EFA 裝置的 ，以及兩個共用這些裝置存取權的 Pod。

1. 建立 ResourceClaim。

```
apiVersion: resource.k8s.io/v1
```

```
kind: ResourceClaim
metadata:
  name: shared-efa
spec:
  devices:
    requests:
    - name: efa
      exactly:
        deviceClassName: efa.networking.k8s.aws
        count: 4
```

2. 在需要存取 EFA 裝置的每個 Pod 中ResourceClaim，依名稱參考 。每個 Pod 會使用 resourceClaimName 來參考現有的宣告，而非 resourceClaimTemplateName。

```
apiVersion: v1
kind: Pod
metadata:
  name: training-worker
spec:
  containers:
  - name: worker
    image: my-training-image
    resources:
      claims:
      - name: efa-devices
  resourceClaims:
  - name: efa-devices
    resourceClaimName: shared-efa
---
apiVersion: v1
kind: Pod
metadata:
  name: training-monitor
spec:
  containers:
  - name: monitor
    image: my-monitor-image
    resources:
      claims:
      - name: efa-devices
  resourceClaims:
  - name: efa-devices
    resourceClaimName: shared-efa
```

兩個 Pod 都參考相同的 `shared-efaResourceClaim`，並排程到配置這些 EFA 裝置的節點。`ResourceClaim` 生命週期獨立於 Pod，它會持續存在直到您刪除它為止，即使所有參考它的 Pod 都被移除。

安裝 EFA Kubernetes 裝置外掛程式

EFA Kubernetes 裝置外掛程式會將 EFA 裝置公告為`vpc.amazonaws.com/efa`延伸資源。您可以在容器資源請求和限制中請求 EFA 裝置。如需使用訓練工作負載設定 EFA 的完整逐步解說，請參閱 the section called “使用 EFA 設定訓練叢集”。

> **Important**
>
> 使用 EKS 最佳化 AL2023 加速 AMIs 時，會自動配置具有 EFA 介面的 NVIDIA GPUs 或 Neuron 裝置拓撲。使用 Bottlerocket EKS 最佳化 AMIs 或自訂 AMIs 時，不會發生此自動對齊。如果您需要透過 Bottlerocket 或自訂 AMIs 配置拓撲對齊加速器和 EFA 裝置，請使用 EFA DRA 驅動程式和對應的 Neuron DRA 驅動程式。若要在 Bottlerocket 上使用 NVIDIA DRA 驅動程式，您必須先停用隨附於 Bottlerocket NVIDIA 變體的 NVIDIA 裝置外掛程式，該變體需要 Bottlerocket 1.63.0 版或更新版本。如需詳細資訊，請參閱the section called “拓撲感知 EFA 和 GPU/Neuron 裝置配置”及the section called “安裝 NVIDIA DRA 驅動程式”。

> **Important**
>
> 從 NVIDIA v0k8s-device-plugin.19.0 開始， `--mofed-enabled`旗標預設為 `true`，這會導致 NVIDIA 裝置外掛程式將所有`/dev/infiniband/uverbs*`裝置掛載到請求 GPUs 的容器。這與 EFA 裝置外掛程式發生衝突，應該是管理 中 EFA 裝置配置的元件`/dev/infiniband`。如果您使用 EKS 受管節點群組或自我管理節點搭配 NVIDIA 裝置外掛程式，則必須明確停用 MOFED。如需說明，請參閱the section called “安裝 NVIDIA Kubernetes 裝置外掛程式”。
>
> EKS Auto Mode 預設不會啟用 MOFED，且不受此問題影響。

先決條件

- Amazon EKS 叢集。
- 具有啟用 EFA 的 Amazon EC2 執行個體類型的節點。如需支援的執行個體類型清單，請參閱《Amazon EC2 使用者指南》中的支援的執行個體類型。

- 針對 EFA 安裝主機層級元件的節點，如需詳細資訊，請參閱安裝 EFA 軟體。EKS 最佳化 AL2023 AMIs 和 Bottlerocket AMIs 包含 EFA 主機層級元件。
- 已在命令列環境中安裝 Helm，如需詳細資訊，請參閱安裝 Helm 說明。
- kubectl 設定為與您的叢集通訊，the section called “安裝或更新 kubectl”如需詳細資訊，請參閱 。

程序

1. 新增 EKS Helm Chart 儲存庫。

```
helm repo add eks https://aws.github.io/eks-charts
```

2. 更新本機 Helm 儲存庫。

```
helm repo update
```

3. 安裝 EFA 裝置外掛程式。

```
helm install efa eks/aws-efa-k8s-device-plugin -n kube-system
```

4. 驗證 EFA 裝置外掛程式 DaemonSet 正在執行。

```
kubectl get daemonset -n kube-system efa-aws-efa-k8s-device-plugin
```

```
NAME                                  DESIRED   CURRENT   READY   UP-TO-DATE
 AVAILABLE   NODE SELECTOR   AGE
efa-aws-efa-k8s-device-plugin         2         2         2       2            2
      <none>          60s
```

5. 確認您的節點具有可分配的 EFA 資源。

```
kubectl get nodes "-o=custom-columns=NAME:.metadata.name,EFA:.status.allocatable.vpc
\.amazonaws\.com/efa"
```

```
NAME                                          EFA
ip-192-168-11-225.us-west-2.compute.internal   4
ip-192-168-24-96.us-west-2.compute.internal    4
```

6. 若要使用裝置外掛程式請求 EFA 裝置，請在您的容器`vpc.amazonaws.com/efa`資源請求和限制中指定資源。

```
apiVersion: v1
kind: Pod
metadata:
  name: efa-workload
spec:
  containers:
  - name: app
    ...
    resources:
      limits:
        vpc.amazonaws.com/efa: 4
        hugepages-2Mi: ...
      requests:
        vpc.amazonaws.com/efa: 4
        hugepages-2Mi: ...
```

在 Amazon EKS 上管理 Neuron 裝置

AWS Trainium 和 AWS Inferentia 是由 設計的專用機器學習晶片 AWS。Amazon EKS 支援兩種在 EKS 叢集中管理 Neuron 裝置的機制：Neuron DRA 驅動程式和 Neuron Kubernetes 裝置外掛程式。

在 Karpenter、EKS 受管節點群組或自我管理節點中使用靜態容量佈建時，我們建議將 DRA 驅動程式用於 Kubernetes 1.34 版及更新版本的新部署。EKS Auto 模式目前不支援 DRA。Neuron DRA 驅動程式提供拓撲感知配置、連線裝置子集排程、邏輯 NeuronCore (LNC) 組態，以及 UltraServer 多節點配置，而不需要自訂排程器延伸。

Neuron DRA 驅動程式與 Neuron 裝置外掛程式

功能	Neuron DRA 驅動程式	Neuron 裝置外掛程式
最低 Kubernetes 版本	1.34	所有 EKS 支援的 Kubernetes 版本
EKS 運算	Karpenter（僅限靜態容量）、受管節點群組、自我管理節點	EKS Auto Mode、Karpenter、受管節點群組、自我管理節點

功能	Neuron DRA 驅動程式	Neuron 裝置外掛程式
EKS 最佳化 AMI 支援	AL2023 (Neuron)、 Bottlerocket	AL2023 (Neuron)、 Bottlerocket
裝置廣告	透過裝置 ID、執行個體類型、拓撲、驅動程式版本和 EFA 地區性等`ResourceSlice` 物件的豐富屬性	`aws.amazon.com/neuron` 和`aws.amazon.com/neuroncore` 擴充資源的整數計數
連線的裝置子集	使用拓撲限制條件配置 1、4、8 或 16 個連線 Neuron 裝置的子集	需要 Neuron 排程器延伸以進行連續的裝置配置
LNC 組態	透過`ResourceClaimTemplate` 參數的每個工作負載邏輯 NeuronCore 組態 (LNC=1 或 LNC=2)	在 EC2 啟動範本中需要預先設定
以屬性為基礎的選擇	使用 CEL 表達式，依執行個體類型、驅動程式版本和其他屬性篩選裝置	不支援
拓撲感知 EFA 配置	DRA 原生拓撲感知	自動拓撲感知 （僅限 EKS 最佳化 AL2023 AMIs)

安裝 Neuron DRA 驅動程式

Neuron DRA 驅動程式會將 Neuron 裝置公告為`DeviceClass`名稱為 的`ResourceSlice`物件`neuron.aws.com`。驅動程式會以 DaemonSet 的形式執行，並自動探索 Neuron 裝置及其拓撲屬性。

如需 Neuron DRA 驅動程式的詳細資訊，請參閱 Neuron DRA 文件。

先決條件

- 執行 Kubernetes 1.34 版或更新版本的 Amazon EKS 叢集，具有由 Karpenter、EKS 受管節點群組或自我管理節點群組佈建的靜態容量。
- 具有 AWS Trainium 或 Inferentia2 執行個體類型的節點。
- 已在命令列環境中安裝 Helm，如需詳細資訊，請參閱安裝 Helm 說明。
- `kubectl` 設定為與您的叢集通訊，the section called “安裝或更新 kubectl”如需詳細資訊，請參閱 。

程序

> **Important**
>
> 請勿在執行 Neuron 裝置外掛程式的節點上安裝 Neuron DRA 驅動程式。這兩個機制不能同時存在於同一個節點上。這樣做可能會導致基礎裝置的靜音過度訂閱到相同節點上的多個 Pod。

1. 使用 Helm 安裝 Neuron DRA 驅動程式。

```
helm upgrade --install neuron-helm-chart oci://public.ecr.aws/neuron/neuron-helm-
chart \
    --namespace neuron-dra-driver \
    --create-namespace \
    --set "devicePlugin.enabled=false" \
    --set "npd.enabled=false" \
    --set "draDriver.enabled=true"
```

   根據預設，驅動程式會部署為 neuron-dra-driver 命名空間中的 DaemonSetDeviceClassneuron.aws.com。

2. 確認 DRA 驅動程式 DaemonSet 正在執行。

```
kubectl get ds -n neuron-dra-driver neuron-dra-driver-kubelet-plugin
```

```
NAME                              DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE
  NODE SELECTOR   AGE
neuron-dra-driver-kubelet-plugin  1         1         1       1            1
  <none>          60s
```

3. 確認DeviceClass已建立 。

```
kubectl get deviceclass neuron.aws.com
```

```
NAME            AGE
neuron.aws.com  60s
```

4. 確認已為您的節點公告ResourceSlice物件。

```
kubectl get resourceslice
```

如需可用ResourceSlice物件屬性的資訊，請參閱 Neuron DRA 文件。

在 Pod 中請求 Neuron 裝置

若要使用 DRA 驅動程式請求 Neuron 裝置，請建立參考 的 ResourceClaimTemplate，neuron.aws.comDeviceClass並在 Pod 規格中參考它。

下列範例會請求trn2.48xlarge執行個體上的所有 Neuron 裝置：

```
apiVersion: resource.k8s.io/v1
kind: ResourceClaimTemplate
metadata:
  name: all-neurons
spec:
  spec:
    devices:
      requests:
      - name: neurons
        exactly:
          deviceClassName: neuron.aws.com
          selectors:
          - cel:
              expression: "device.attributes['neuron.aws.com'].instanceType ==
 'trn2.48xlarge'"
          allocationMode: All
---
apiVersion: v1
kind: Pod
metadata:
  name: neuron-workload
spec:
  containers:
  - name: app
    ...
    resources:
      claims:
      - name: neurons
  resourceClaims:
  - name: neurons
    resourceClaimTemplateName: all-neurons
```

配置連線的裝置子集

Neuron DRA 驅動程式可以配置已連線 Neuron 裝置的子集，而不需要 Neuron 排程器延伸。支援的子集大小為 1、4、8 或 16 個裝置。使用matchAttribute限制條件搭配拓撲群組 ID，以確保裝置已連線。

下列範例會請求 4 個連線的 Neuron 裝置：

```
apiVersion: resource.k8s.io/v1
kind: ResourceClaimTemplate
metadata:
  name: 1x4-connected-neurons
spec:
  spec:
    devices:
      requests:
      - name: neurons
        exactly:
          deviceClassName: neuron.aws.com
          allocationMode: ExactCount
          count: 4
          selectors:
          - cel:
              expression: "device.attributes['neuron.aws.com'].instanceType ==
 'trn2.48xlarge'"
      constraints:
      - requests: ["neurons"]
        matchAttribute: "resource.aws.com/devicegroup4_id"
```

連線子集支援matchAttribute的值為 resource.aws.com/devicegroup1_id、resource.aws.com/devicegroup8_id、 resource.aws.com/devicegroup4_id和 resource.aws.com/devicegroup16_id。devicegroup 屬性名稱中的數字對應至已連線拓撲群組中的 Neuron 裝置數量。例如， resource.aws.com/devicegroup1_id 識別單一 Neuron 裝置、resource.aws.com/devicegroup4_id識別 4 個連線裝置的群組，resource.aws.com/devicegroup8_id以及分別resource.aws.com/devicegroup16_id識別 8 和 16 個連線裝置的群組。選擇matchAttribute符合您請求count中裝置的 ，讓配置的裝置屬於相同的連線拓撲群組。如需這些屬性的詳細資訊，請參閱 Neuron DRA 驅動程式文件。

設定邏輯 NeuronCores (LNC)

Neuron DRA 驅動程式允許透過`ResourceClaimTemplate`參數進行每個工作負載的 Logical NeuronCore 組態。這不需要在 EC2 啟動範本中預先設定 LNC。

下列範例會請求 LNC 設為 1 的所有 Neuron 裝置：

```
apiVersion: resource.k8s.io/v1
kind: ResourceClaimTemplate
metadata:
  name: all-neurons-lnc-1
spec:
  spec:
    devices:
      requests:
      - name: neurons
        exactly:
          deviceClassName: neuron.aws.com
          selectors:
          - cel:
              expression: "device.attributes['neuron.aws.com'].instanceType ==
 'trn2.48xlarge'"
          allocationMode: All
      config:
      - requests: ["neurons"]
        opaque:
          driver: neuron.aws.com
          parameters:
            apiVersion: neuron.aws.com/v1
            kind: NeuronConfig
            logicalNeuronCore: 1
```

使用對齊的 EFA 介面配置 Neuron 裝置

請參閱 the section called “拓撲感知 EFA 和 GPU/Neuron 裝置配置”

安裝 Neuron Kubernetes 裝置外掛程式

Neuron Kubernetes 裝置外掛程式會將 Neuron 裝置公告為 `aws.amazon.com/neuron`，並將 NeuronCores 公告為`aws.amazon.com/neuroncore`延伸資源。您可以在容器資源請求和限制中請求 Neuron 裝置。

先決條件

- Amazon EKS 叢集。
- 針對 AWS Trainium 或 AWS Inferentia 執行個體安裝主機層級元件的節點。如果使用 EKS AL2023 加速 AMIs 或 EKS Bottlerocket AMIs，則會包含這些項目。
- 已在命令列環境中安裝 Helm，如需詳細資訊，請參閱安裝 Helm 說明。
- `kubectl` 設定為與您的叢集通訊，the section called “安裝或更新 `kubectl`”如需詳細資訊，請參閱 。

程序

1. 使用 Helm 安裝 Neuron Kubernetes 裝置外掛程式。

```
helm upgrade --install neuron-helm-chart oci://public.ecr.aws/neuron/neuron-helm-
chart \
    --set "npd.enabled=false"
```

2. 驗證 Neuron 裝置外掛程式 DaemonSet 正在執行中。

```
kubectl get ds -n kube-system neuron-device-plugin
```

```
NAME                   DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE
 SELECTOR   AGE
neuron-device-plugin   1         1         1       1            1           <none>
       60s
```

3. 確認您的節點具有可配置的 Neuron 裝置。

```
kubectl get nodes "-o=custom-
columns=NAME:.metadata.name,NeuronDevice:.status.allocatable.aws\.amazon\.com/
neuron,NeuronCore:.status.allocatable.aws\.amazon\.com/neuroncore"
```

```
NAME                                           NeuronDevice   NeuronCore
ip-192-168-47-173.us-west-2.compute.internal   1              2
```

使用測試 Pod 驗證 Neuron 裝置

您可以在測試 Pod 中執行 `neuron-ls`工具，以驗證 Neuron 裝置是否可存取。

1. 使用下列內容建立名為 neuron-ls.yaml 的檔案。此資訊清單會啟動已安裝 neuron-ls工具的 Neuron Monitor 容器。

```
apiVersion: v1
kind: Pod
metadata:
  name: neuron-ls
spec:
  restartPolicy: Never
  containers:
  - name: neuron-container
    image: public.ecr.aws/g4h4h0b5/neuron-monitor:1.0.0
    command: ["/bin/sh"]
    args: ["-c", "neuron-ls"]
    resources:
      limits:
        aws.amazon.com/neuron: 1
      requests:
        aws.amazon.com/neuron: 1
  tolerations:
  - key: "aws.amazon.com/neuron"
    operator: "Exists"
    effect: "NoSchedule"
```

2. 套用清單檔案。

```
kubectl apply -f neuron-ls.yaml
```

3. Pod 執行完成後，請檢視其日誌。

```
kubectl logs neuron-ls
```

範例輸出如下。

```
instance-type: inf2.xlarge
instance-id: ...
+--------+--------+--------+---------+
| NEURON | NEURON | NEURON |   PCI   |
| DEVICE | CORES  | MEMORY |   BDF   |
+--------+--------+--------+---------+
| 0      | 2      | 32 GB  | 00:1f.0 |
+--------+--------+--------+---------+
```

> **Note**
>
> 使用 Neuron 裝置外掛程式時，具有多個 Neuron 裝置（例如 `trn2.48xlarge`) 的執行個體上的連續裝置配置需要 Neuron Kubernetes 排程器延伸。Neuron DRA 驅動程式會透過拓撲限制自動處理此問題。

如需搭配 Amazon EKS 使用 Neuron 裝置的詳細資訊，請參閱用於在 EKS 上執行的 Neuron 文件。

在 Amazon EKS 上管理 NVIDIA GPUs

NVIDIA GPUs 廣泛用於機器學習訓練、推論和高效能運算工作負載。Amazon EKS 支援兩種在 EKS 叢集中管理 NVIDIA GPU 裝置的機制：GPUs 的 NVIDIA DRA 驅動程式和 NVIDIA Kubernetes 裝置外掛程式。

在 Karpenter、EKS 受管節點群組或自我管理節點中使用靜態容量佈建時，我們建議將 DRA 驅動程式用於 Kubernetes 1.34 版及更新版本的新部署。EKS Auto 模式目前不支援 DRA。NVIDIA DRA 驅動程式可在容器之間啟用靈活的 GPU 配置和 GPU 共用。

如需 EKS 中 DRA 功能的可用性，請參閱 Kubernetes 版本的 EKS 版本備註。如需搭配 EKS 使用 NVIDIA DRA 驅動程式和裝置外掛程式的詳細資訊，請參閱 the section called “DRA 和裝置外掛程式”。

GPU 共用 (MIG 和時間分割）

多執行個體 GPU (MIG)

多執行個體 GPU (MIG) 是 NVIDIA GPUs 上的硬體功能，可將單一實體 GPU 分割成多個隔離的執行個體。執行個體數量上限取決於 GPU。A100、H100、H200 和 B200 等資料中心 GPUs 最多支援七個執行個體，而以 Blackwell 為基礎的 GPU g7和 g7e GPUs 支援較少。每個執行個體都有專用記憶體、運算單位和記憶體頻寬，因此在一個執行個體上執行的工作負載不會影響另一個執行個體上的工作負載。與不隔離的情況下透過軟體時間多工共用 GPU 的時間分割不同，MIG 可在 Pod 之間提供硬體層級記憶體和故障隔離。

MIG 最適合多租戶推論、需要可預測服務品質的工作負載，以及具有硬體隔離租用合規要求的環境。可在 NVIDIA Ampere (A100)、Hopper (H100 和 H200) 和 Blackwell GPUs 上使用。在 上 AWS，這些是 P 系列執行個體類型，以及 Blackwell 型g7和g7e執行個體類型。

如需搭配 NVIDIA GPUs 和 EKS 使用 MIG 的詳細資訊和步驟，請參閱 the section called “多執行個體 GPUs (MIG)”。

時間分割

時間分割可讓多個 Pod 透過設定 NVIDIA 裝置外掛程式或 DRA 驅動程式，為每個 GPU 公告多個可排程的插槽，來共用單一實體 NVIDIA GPU。Kubernetes 排程器會將多個 Pod 放置在同一個 GPU 上，GPU 的 CUDA 排程器時間乘以工作負載。

時間分割是最簡單的 GPU 共用策略。它只使用軟體，不需要特殊硬體，並且適用於每個 NVIDIA GPU 執行個體類型 AWS。時間分割不提供共用 GPU 的 Pod 之間的記憶體或運算隔離。它最適合 GPU 使用率低的工作負載，例如在多個使用者共用 GPU 的請求或開發環境之間閒置的推論服務。對於需要硬體層級記憶體和運算隔離的工作負載，請改用 MIG。

如需搭配 NVIDIA GPUs 和 EKS 使用時間分割的詳細資訊和步驟，請參閱 the section called “時間分割”。

主題

- the section called “DRA 和裝置外掛程式”
- the section called “多執行個體 GPUs (MIG)”
- the section called “時間分割”

在 Amazon EKS 上使用 NVIDIA DRA 驅動程式或裝置外掛程式

Amazon EKS 支援兩種在 EKS 叢集中管理 NVIDIA GPU 裝置的機制：GPUs 的 NVIDIA DRA 驅動程式和 NVIDIA Kubernetes 裝置外掛程式。

在 Karpenter、EKS 受管節點群組或自我管理節點中使用靜態容量佈建時，建議使用 NVIDIA DRA 驅動程式搭配 Kubernetes 1.34 版及更新版本進行新部署。EKS Auto Mode 目前不支援 NVIDIA DRA 驅動程式。使用動態容量佈建時，搭配 EKS Auto 模式或 Karpenter 使用 NVIDIA 裝置外掛程式。NVIDIA 裝置外掛程式仍然支援 EKS 受管節點群組和自我管理節點。

如果您使用多執行個體 GPUs (MIG) 或時間分割等 GPU 共用功能，建議您搭配 DRA 驅動程式使用靜態組態，或使用 NVIDIA 裝置外掛程式。動態 MIG 和時間分割在 NVIDIA DRA 驅動程式中處於 Alpha 狀態。如需最新更新，請參閱 GitHub 上的 NVIDIA DRA 驅動程式版本。

NVIDIA DRA 驅動程式與 NVIDIA 裝置外掛程式

功能	NVIDIA DRA 驅動程式	NVIDIA 裝置外掛程式
最低 Kubernetes 版本	1.34	所有 EKS 支援的 Kubernetes 版本
EKS 運算	Karpenter（僅限靜態容量）、受管節點群組、自我管理節點	EKS Auto Mode、Karpenter、受管節點群組、自我管理節點
EKS 最佳化 AMIs	AL2023 (NVIDIA)、 Bottlerocket	AL2023 (NVIDIA)、 Bottlerocket
裝置廣告	透過 GPU 模型、記憶體、驅動程式版本和拓撲等`ResourceSlice` 物件的豐富屬性	`nvidia.com/gpu` 延伸資源的整數計數
GPU 共用	(Alpha) 動態 MIG、MPS、時間分割	(GA) 靜態 MIG、MPS、時間分割
ComputeDomains	透過 `ComputeDomain` 資源管理多節點 NVLink (MNNVL)，以進行安全的多節點 GPU 通訊	不支援
以屬性為基礎的選擇	使用 CEL 表達式，依模型、記憶體或其他屬性篩選 GPUs	不支援
拓撲感知 EFA 配置	DRA 原生拓撲感知	自動拓撲感知（僅限 EKS 最佳化 AL2023 AMIs)

安裝 NVIDIA DRA 驅動程式

適用於 GPUs 的 NVIDIA DRA 驅動程式可管理兩種類型的資源：GPUs 和 ComputeDomains。它執行兩個 DRA kubelet 外掛程式： `gpu-kubelet-plugin`和 `compute-domain-kubelet-plugin`。每個 都可以在安裝期間分別啟用或停用。本指南著重於 GPU 配置。如需使用 ComputeDomains 的詳細資訊，請參閱 the section called “搭配 EKS 使用 P6e-GB200 ”。

先決條件

- 執行 Kubernetes 1.34 版或更新版本的 Amazon EKS 叢集，具有由 Karpenter、EKS 受管節點群組或自我管理節點群組佈建的靜態容量。
- 具有 NVIDIA GPU 執行個體類型的節點（例如 P或 G執行個體）。
- 針對 NVIDIA GPUs 安裝主機層級元件的節點。使用 EKS 最佳化 AL2023 或 Bottlerocket NVIDIA AMIs 時，會預先安裝主機層級 NVIDIA 驅動程式、CUDA 使用者模式驅動程式和容器工具組。
- 已在命令列環境中安裝 Helm，如需詳細資訊，請參閱安裝 Helm 說明。
- `kubectl` 設定為與您的叢集通訊，the section called “安裝或更新 `kubectl`”如需詳細資訊，請參閱 。

程序

> **Important**
>
> 使用 NVIDIA DRA 驅動程式進行 GPU 裝置管理時，請勿將其與 NVIDIA 裝置外掛程式一起部署在同一個節點上。這樣做可能會導致基礎裝置的靜音過度訂閱到相同節點上的多個 Pod。

> **在 Bottlerocket 上停用內建 NVIDIA 裝置外掛程式**
>
> EKS 最佳化 Bottlerocket NVIDIA 變體包含 NVIDIA 裝置外掛程式，並預設啟用它。DRA 驅動程式無法與相同節點上的裝置外掛程式一起執行。使用 DRA 驅動程式之前，請先停用 Bottlerocket GPU 節點上的內建裝置外掛程式。Bottlerocket 節點使用者資料`false`中的 `settings.kubelet-device-plugins.nvidia.enabled`設定為 。
>
> ```
> [settings.kubelet-device-plugins.nvidia]
> enabled = false
> ```
>
> 此`settings.kubelet-device-plugins.nvidia.enabled`設定可在 Bottlerocket 1.63.0 版及更新版本中使用。在舊版上，無法停用內建 NVIDIA 裝置外掛程式。如需詳細資訊，請參閱 GitHub 上的 bottlerocket-os/bottlerocket pull request #4856。

1. 直接從 Kubernetes SIG OCI 登錄檔安裝 NVIDIA DRA 驅動程式。若要尋找可用的版本，請參閱 GitHub 上的 NVIDIA DRA 驅動程式版本。

```
helm install dra-driver-nvidia-gpu \
    oci://registry.k8s.io/dra-driver-nvidia/charts/dra-driver-nvidia-gpu \
    --version 0.4.1 \
    --create-namespace \
    --namespace nvidia \
    --set resources.computeDomains.enabled=false \
    --set gpuResourcesEnabledOverride=true
```

如需進階組態選項，請參閱 Kubernetes SIG 網站上的 NVIDIA DRA 驅動程式 Helm Chart 值。若要查看特定圖表版本的可用值，請執行 `helm show values oci://registry.k8s.io/dra-driver-nvidia/charts/dra-driver-nvidia-gpu --version 0.4.1`。

2. （選用）若要透過 DRA 驅動程式使用時間分割，請在上一個步驟中將`TimeSlicingSettings`功能閘道新增至`helm install`命令。這是預設停用的 Alpha 功能。如需詳細資訊，請參閱the section called “搭配 NVIDIA DRA 驅動程式使用 GPU 時間分割”。

```
--set featureGates.TimeSlicingSettings=true
```

3. （選用）若要透過 DRA 驅動程式使用動態 MIG，請在上一個步驟中將`DynamicMIG`功能閘道新增至`helm install`命令。這是預設停用的 Alpha 功能。您無法將`DynamicMIG`功能閘道與`PassthroughSupport`、 `NVMLDeviceHealthCheck`或`MPSSupport`功能閘道結合。如需詳細資訊，請參閱the section called “搭配 NVIDIA DRA 驅動程式使用 MIG”。

```
--set featureGates.DynamicMIG=true
```

4. 確認 DRA 驅動程式 Pod 正在執行。

```
kubectl get pods -n nvidia
```

5. 確認`DeviceClass`物件已建立。

```
kubectl get deviceclass
```

```
NAME            AGE
gpu.nvidia.com  60s
```

6. 確認已為您的 GPU 節點發佈`ResourceSlice`物件。

```
kubectl get resourceslice
```

若要使用 DRA 驅動程式請求 NVIDIA GPUs，請建立參考 `ResourceClaimTemplate`的 ，`gpu.nvidia.com`DeviceClass並在 Pod 規格中參考它。下列範例會請求單一 GPU。the section called “拓撲感知 EFA 和 GPU/Neuron 裝置配置” 如需使用拓撲對齊 EFA 介面配置 NVIDIA GPUs 的步驟，請參閱 。

```
apiVersion: resource.k8s.io/v1
kind: ResourceClaimTemplate
metadata:
  name: single-gpu
spec:
  spec:
    devices:
      requests:
      - name: gpu
        exactly:
          deviceClassName: gpu.nvidia.com
          count: 1
---
apiVersion: v1
kind: Pod
metadata:
  name: gpu-workload
spec:
  containers:
  - name: gpu-demo
    image: public.ecr.aws/amazonlinux/amazonlinux:2023-minimal
    command: ["/bin/sh", "-c"]
    args: ["nvidia-smi && tail -f /dev/null"]
    resources:
      claims:
      - name: gpu
  resourceClaims:
  - name: gpu
    resourceClaimTemplateName: single-gpu
  tolerations:
  - key: "nvidia.com/gpu"
    operator: "Exists"
    effect: "NoSchedule"
```

安裝 NVIDIA Kubernetes 裝置外掛程式

NVIDIA Kubernetes 裝置外掛程式會將 NVIDIA GPUs 公告為`nvidia.com/gpu`延伸資源。您可以在容器資源請求和限制中請求 GPUs。

先決條件

- Amazon EKS 叢集。
- 具有 NVIDIA GPU 執行個體類型的節點 （ 例如 `P`或 `G`執行個體 ）。
- 使用 EKS 最佳化 AL2023 NVIDIA AMI 搭配 NVIDIA GPU 執行個體類型的節點。EKS 最佳化 Bottlerocket AMIs包含 NVIDIA 裝置外掛程式。您不需要單獨安裝。
- 針對 NVIDIA GPUs 安裝主機層級元件的節點。使用 EKS 最佳化 AL2023 或 Bottlerocket NVIDIA AMIs 時，會預先安裝主機層級 NVIDIA 驅動程式、CUDA 使用者模式驅動程式和容器工具組。
- 已在命令列環境中安裝 Helm，如需詳細資訊，請參閱安裝 Helm 說明。
- `kubectl` 設定為與您的叢集通訊，the section called “安裝或更新 `kubectl`”如需詳細資訊，請參閱 。

程序

1. 新增 NVIDIA 裝置外掛程式 Helm Chart 儲存庫。

```
helm repo add nvdp https://nvidia.github.io/k8s-device-plugin
```

2. 更新本機 Helm 儲存庫。

```
helm repo update
```

3. 安裝 NVIDIA Kubernetes 裝置外掛程式。

```
helm install nvdp nvdp/nvidia-device-plugin \
    --create-namespace \
    --namespace nvidia
```

> 選用：啟用 GPU 功能探索 (GFD)
>
> 裝置外掛程式會公告`nvidia.com/gpu`擴充的資源，並自行排程 GPU Pod。在 EKS 最佳化 AL2023 NVIDIA AMI 上，`nvidia.com/gpu.present=true`節點標籤已由 於開機時套用`nodeadm`，因此基本 GPU 排程不需要 GPU 功能探索 (GFD)。

`--set gfd.enabled=true` 如果您想要以詳細 GPU 屬性標記節點，例如 `nvidia.com/gpu.product`、`nvidia.com/gpu.memory`、`nvidia.com/gpu.count`、MIG 設定檔標籤和驅動程式/CUDA 版本，請使用 啟用 GFD。透過這些標籤，您可以以具有 `nodeSelector`或 節點親和性的特定 GPU 類型為目標（例如，僅將工作負載排程到 A10G GPUs 或特定 MIG 設定檔）。時間分割和 MIG 等 GPU 共用組態也會使用這些標籤。如果您不需要屬性型節點選擇或 GPU 共用，則可以省略 旗標。

```
helm install nvdp nvdp/nvidia-device-plugin \
    --create-namespace \
    --namespace nvidia \
    --set gfd.enabled=true
```

⚠ 如果您的工作負載使用 GDRCopy

`k8s-device-plugin` v0.19.0 到 v0.19.2 預設啟用 GDRCopy 和 MOFED 功能。此預設啟用已在 v0`k8s-device-plugin`.19.3 中還原。由於還原，無法再使用 Pod 規格中的`NVIDIA_GDRCOPY=enabled`環境變數為每個容器啟用 GDRCopy (`gdrdrv`)，現在會忽略該變數。
如果您的工作負載使用 GDRCopy (GPUDirect RDMA 複本），您必須在安裝時透過設定 ，在裝置外掛程式上啟用它`gdrcopyEnabled=true`：

```
helm upgrade --install nvdp nvdp/nvidia-device-plugin \
    --namespace nvidia \
    --create-namespace \
    --set gdrcopyEnabled=true
```

（如果您也想要 GPU 功能探索標籤，`--set gfd.enabled=true`也請新增 ，如前所述。）
如果您透過 GitHub 上的 NVIDIA GPU Operator 管理 NVIDIA 裝置外掛程式，該運算子會在`gdrdrv`核心模組載入節點`GDRCOPY_ENABLED=true`時動態設定。
如需詳細資訊，請參閱 GitHub 上的 NVIDIA k8s-device-plugin 問題 #1692。

> **Note**
>
> 您也可以使用 GitHub 上的 NVIDIA GPU Operator 安裝和管理 NVIDIA Kubernetes 裝置外掛程式，這會自動管理佈建 GPUs 所需的所有 NVIDIA 軟體元件。

4. 驗證 NVIDIA 裝置外掛程式 DaemonSet 正在執行。

```
kubectl get ds -n nvidia nvdp-nvidia-device-plugin
```

```
NAME                        DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE
 SELECTOR   AGE
nvdp-nvidia-device-plugin   2         2         2       2            2
 <none>          60s
```

5. 確認您的節點具有可配置的 GPUs。

```
kubectl get nodes "-o=custom-
columns=NAME:.metadata.name,GPU:.status.allocatable.nvidia\.com/gpu"
```

範例輸出如下。

```
NAME                                          GPU
ip-192-168-11-225.us-west-2.compute.internal   1
ip-192-168-24-96.us-west-2.compute.internal    1
```

在 Pod 中請求 NVIDIA GPUs

若要使用裝置外掛程式請求 NVIDIA GPUs，請在容器`nvidia.com/gpu`資源請求和限制中指定資源。

```
apiVersion: v1
kind: Pod
metadata:
  name: nvidia-smi
spec:
  restartPolicy: OnFailure
  containers:
  - name: gpu-demo
    image: public.ecr.aws/amazonlinux/amazonlinux:2023-minimal
```

```
    command: ["/bin/sh", "-c"]
    args: ["nvidia-smi && tail -f /dev/null"]
    resources:
      limits:
        nvidia.com/gpu: 1
      requests:
        nvidia.com/gpu: 1
  tolerations:
  - key: "nvidia.com/gpu"
    operator: "Equal"
    value: "true"
    effect: "NoSchedule"
```

若要執行此測試，請套用資訊清單並檢視日誌：

```
kubectl apply -f nvidia-smi.yaml
kubectl logs nvidia-smi
```

範例輸出如下。

```
+-----------------------------------------------------------------------------------------
+
| NVIDIA-SMI XXX.XXX.XX            Driver Version: XXX.XXX.XX     CUDA Version: XX.X
   |
|-----------------------------------------+------------------------
+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr.
 ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute
 M. |
|                                         |                        |               MIG
 M. |
|=========================================+========================
+======================|
|   0  NVIDIA L4                      On  |   00000000:31:00.0 Off |
 0 |
| N/A   27C    P8             11W /   72W |       0MiB /  23034MiB |      0%
 Default |
|                                         |                        |
 N/A |
+-----------------------------------------+------------------------
+----------------------+
```

```
+-----------------------------------------------------------------------------------------
+
| Processes:
   |
|  GPU   GI   CI              PID   Type   Process name                          GPU
 Memory |
|        ID   ID                                                               Usage
   |
|
=============================================================================================|
|  No running processes found
   |
+-----------------------------------------------------------------------------------------
+
```

在 Amazon EKS 上使用多執行個體 GPUs (MIG) 搭配 NVIDIA GPUs

多執行個體 GPU (MIG) 是 NVIDIA GPUs 上的硬體功能，可將單一實體 GPU 分割成多個隔離的執行個體。分割執行個體的數量上限取決於 GPU。每個執行個體都有專用記憶體、運算單位和記憶體頻寬，因此在一個執行個體上執行的工作負載不會影響另一個執行個體上的工作負載。與不隔離的情況下透過軟體時間多工共用 GPU 的時間分割不同，MIG 可在 Pod 之間提供硬體層級記憶體和故障隔離。

MIG 最適合需要可預測服務品質的多租戶推論和工作負載。可在 NVIDIA Ampere (A100)、Hopper (H100 和 H200) 和 Blackwell GPUs 上使用。在 上 AWS，這些是 P 系列執行個體類型，以及 Blackwell 型g7和g7e執行個體類型。如需完整清單，請參閱the section called “支援 MIG 的執行個體類型”。

在下列情況下，MIG 非常適合：

- 您需要記憶體隔離，以便一個工作負載無法使用另一個工作負載所需的 GPU 記憶體。
- 您可以執行多租戶推論，其中租戶共用 GPU 硬體，但需要每個租戶的服務品質。
- 您已在 A100、H100、H200 或 Blackwell 執行個體上執行訓練，並希望在訓練閒置時將這些 GPUs 重複使用於較小的推論工作負載。

在下列情況下，請考慮不同的方法：

- 您的節點使用不支援 MIG 的 GPU 執行個體類型，例如 g5、 g6或 g6e 系列。請改用時間分割。
- 您不需要記憶體隔離，而且想要最簡單的組態。請改用時間分割。
- 您需要經常變更 GPU 分割區，而不會中斷。變更 MIG 模式或分割區配置需要 GPU 重設，GPU Operator 會透過重新啟動節點來執行此重設。

- 您可以執行多 GPU 訓練，這取決於 GPUs 之間的集體或peer-to-peer通訊。MIG 不支援 NCCL 或跨 GPU P2P。

考量事項

在生產環境中使用 MIG 之前，請檢閱下列考量事項。

一般考量事項

- 變更 MIG 組態需要重設 GPU。啟用或停用 MIG 模式或變更分割區配置需要 GPU 重設，因此無法變更。例如，NVIDIA GPU Operator 中的 MIG Manager 會套用組態變更，方法是停止節點上的 GPU Pod，並在需要重新啟動以變更 MIG 模式時重新啟動節點。
- 運算與執行個體大小不成正比。1g 執行個體不會為每個工作負載提供全 GPU 輸送量的比例分配，因為記憶體頻寬和快取行為因設定檔而異。在調整分割區大小之前，在您想要使用的設定檔上為工作負載建立基準。
- 時間分割對 MIG 執行個體沒有影響。MIG 執行個體已進行硬體隔離，無法透過時間分割進一步共用。在 MIG 裝置上請求`TimeSlicing`共用策略並不會變更硬體行為。若要跨容器共用單一 MIG 執行個體，請改用 NVIDIA Multi-Process Service (MPS)。
- 有限的跨 GPU 通訊。啟用 MIG 時，不同 GPUs 上的 MIG 執行個體無法使用 GPU-to-GPU peer-to-peer(P2P) 通訊，且 NCCL 無法使用 MIG。依賴集體通訊或跨 GPUs 的 P2P 的多 GPU 工作負載，例如張量平行多 GPU 訓練，需要整個 GPUs。如需詳細資訊，請參閱 NVIDIA 網站上的 NVIDIA MIG 使用者指南中的應用程式考量事項。

NVIDIA 裝置外掛程式考量事項

- Pod 資源請求必須符合 策略。使用單一策略，Pods 請求 `nvidia.com/gpu`。使用混合策略，Pod 會請求設定檔特定的資源，例如 `nvidia.com/mig-1g.10gb`。請求節點未公告之設定檔的 Pod 會保持在 `Pending` 狀態。使用 確認公告的資源`kubectl describe node <node-name>`。
- AL2023 上的獨立裝置外掛程式。如果您單獨安裝 NVIDIA 裝置外掛程式，例如做為叢集設定的一部分，請將其排除在 MIG 節點之外，以免與 GPU Operator 管理的裝置外掛程式發生衝突。將節點親和性規則新增至獨立裝置外掛程式，該外掛程式會排除具有 `nvidia.com/mig.config`標籤的節點。
- 不支援 EKS Auto 模式。EKS Auto Mode 會管理 NVIDIA 裝置外掛程式，而不會公開其組態（請參閱 the section called “部署加速工作負載”)。您無法在 EKS Auto Mode 節點上啟用 MIG。在自我管理的 Karpenter 節點或受管節點群組上設定 MIG，您可以在其中控制 AMI 和裝置外掛程式設定。

NVIDIA DRA 驅動程式考量事項

- 靜態 MIG 需要預先建立的執行個體。使用靜態 MIG 時，DRA 驅動程式會配置現有的 MIG 執行個體，但不啟用 MIG 模式或分割 GPUs。您必須啟用 MIG 模式，並先建立執行個體，例如在 NVIDIA GPU Operator 或 中使用 MIG Managernvidia-smi。如需詳細資訊，請參閱the section called “搭配 NVIDIA DRA 驅動程式使用 MIG”。
- 動態 MIG 是一種 Alpha 功能。使用動態 MIG，驅動程式會隨需建立和銷毀 MIG 分割區，以回應工作負載請求。它需要DynamicMIG功能閘道，預設為停用。如需詳細資訊，請參閱the section called “搭配 NVIDIA DRA 驅動程式使用 MIG”。
- 在 Bottlerocket 上停用內建裝置外掛程式。DRA 驅動程式無法與相同節點上的 NVIDIA 裝置外掛程式一起執行。在 Bottlerocket 上，停用需要 Bottlerocket 1.63.0 版或更新版本的內建裝置外掛程式。如需詳細資訊，請參閱the section called “安裝 NVIDIA DRA 驅動程式”。
- 運算支援。在 Karpenter、EKS 受管節點群組或自我管理節點中，靜態容量佈建支援 NVIDIA DRA 驅動程式，EKS Auto Mode 不支援。如需詳細資訊，請參閱 Karpenter 網站上的 Karpenter 靜態 NodePool 文件。

支援 MIG 的執行個體類型

在 上 AWS，下列執行個體類型提供具備 MIG 功能的 GPUs。

執行個體類型	GPU	記憶體
p4d.24xlarge	8 個 NVIDIA A100 40 GB	320 GB
p4de.24xlarge	8 個 NVIDIA A100 80 GB	640 GB
p5.48xlarge	8 個 NVIDIA H100 80 GB	640 GB
p5e.48xlarge	8 個 NVIDIA H200	1128 GB
p5en.48xlarge	8 個 NVIDIA H200	1128 GB
p6-b200.48xlarge	8 個 NVIDIA Blackwell B200	1432 GB
p6-b300.48xlarge	8 個 NVIDIA Blackwell Ultra B300	2144 GB

執行個體類型	GPU	記憶體
g7.48xlarge	8 個 NVIDIA RTX PRO 4500 Blackwell 伺服器版本	256 GB
g7e.48xlarge	8 個 NVIDIA RTX PRO 6000 Blackwell 伺服器版本	768 GB

MIG 不適用於 g5、 g6或 g6e 系列。對於使用支援 MIG 的 NVIDIA GB200 GPU 的 p6e-gb200 UltraServers，請參閱 the section called “搭配 EKS 使用 P6e-GB200 ”。

> **Note**
>
> g7 執行個體類型需要 NVIDIA 驅動程式 595 版或更新版本。EKS 最佳化加速 AMIs 目前包含 NVIDIA 驅動程式 580 版，因此若要在 上使用 MIGg7，您必須建置驅動程式 595 版的自訂 AMI。如需詳細資訊，請參閱the section called “建置自訂 EKS 最佳化 Amazon Linux AMI”。

MIG 執行個體是由使用命名模式 的設定檔描述<slices>g.<memory>gb，其中 <slices>是運算配量的數量，而 <memory>是執行個體的記憶體，以 GB 為單位。例如，3g.40gb設定檔提供七個運算配量中的三個，以及 40 GB 的記憶體。每個 GPU 支援的設定檔由硬體修正。如需完整清單，請參閱 NVIDIA 網站上的 NVIDIA 多執行個體 GPU 使用者指南。

每個執行個體類型的 MIG 設定檔

節點上可用的 MIG 設定檔取決於執行個體類型的 GPU。下列各節列出每個具備 MIG 功能的 Amazon EC2 執行個體類型的設定檔。對於每個設定檔，最大執行個體是您可以在單一 GPU 上建立的該設定檔執行個體數量上限，而每個執行個體的記憶體是分配給每個設定檔的 GPU 記憶體。

p4d.24xlarge – NVIDIA A100 40 GB

設定檔	運算配量	每個執行個體的記憶體數量	執行個體上限
1g.5gb	第 1 頁，共 7 頁	5 GB	7
1g.10gb	第 1 頁，共 7 頁	10 GB	4

設定檔	運算配量	每個執行個體的記憶體數量	執行個體上限
2g.10gb	第 2 頁，共 7 頁	10 GB	3
3g.20gb	第 3 頁，共 7 頁	20 GB	2
4g.20gb	第 4 頁，共 7 頁	20 GB	1
7g.40gb	第 7 頁，共 7 頁	40 GB	1

p4de.24xlarge – NVIDIA A100 80 GB

設定檔	運算配量	每個執行個體的記憶體數量	執行個體上限
1g.10gb	第 1 頁，共 7 頁	10 GB	7
1g.20gb	第 1 頁，共 7 頁	20 GB	4
2g.20gb	第 2 頁，共 7 頁	20 GB	3
3g.40gb	第 3 頁，共 7 頁	40 GB	2
4g.40gb	第 4 頁，共 7 頁	40 GB	1
7g.80gb	第 7 頁，共 7 頁	80 GB	1

p5.48xlarge – NVIDIA H100 80 GB

設定檔	運算配量	每個執行個體的記憶體數量	執行個體上限
1g.10gb	第 1 頁，共 7 頁	10 GB	7
1g.20gb	第 1 頁，共 7 頁	20 GB	4
2g.20gb	第 2 頁，共 7 頁	20 GB	3

設定檔	運算配量	每個執行個體的記憶體數量	執行個體上限
3g.40gb	第 3 頁，共 7 頁	40 GB	2
4g.40gb	第 4 頁，共 7 頁	40 GB	1
7g.80gb	第 7 頁，共 7 頁	80 GB	1

p5e.48xlarge 和 **p5en.48xlarge** – NVIDIA H200 141 GB

設定檔	運算配量	每個執行個體的記憶體數量	執行個體上限
1g.18gb	第 1 頁，共 7 頁	18 GB	7
1g.35gb	第 1 頁，共 7 頁	35 GB	4
2g.35gb	第 2 頁，共 7 頁	35 GB	3
3g.71gb	第 3 頁，共 7 頁	71 GB	2
4g.71gb	第 4 頁，共 7 頁	71 GB	1
7g.141gb	第 7 頁，共 7 頁	141 GB	1

p6-b200.48xlarge – NVIDIA Blackwell B200 180 GB

設定檔	運算配量	每個執行個體的記憶體數量	執行個體上限
1g.23gb	第 1 頁，共 7 頁	23 GB	7
1g.45gb	第 1 頁，共 7 頁	45 GB	4
2g.45gb	第 2 頁，共 7 頁	45 GB	3
3g.90gb	第 3 頁，共 7 頁	90 GB	2

設定檔	運算配量	每個執行個體的記憶體數量	執行個體上限
4g.90gb	第 4 頁，共 7 頁	90 GB	1
7g.180gb	第 7 頁，共 7 頁	180 GB	1

p6-b300.48xlarge – NVIDIA Blackwell Ultra B300

p6-b300.48xlarge 使用 HGX B300，支援將每個 GPU 分割成 7 個 32 GB、4 個 67 GB、2 個 135 GB 或 1 個 270 GB 的執行個體。這些大小是初步的，可能會變更。如需設定檔詳細資訊，請參閱 NVIDIA 網站上的 NVIDIA 支援的 MIG 設定檔。

g7.48xlarge – NVIDIA RTX PRO 4500 Blackwell 伺服器版本 32 GB

設定檔	運算配量	每個執行個體的記憶體數量	執行個體上限
1g.16gb	第 1 頁，共 2 頁	16 GB	2
2g.32gb	第 2 個，共 2 個	32 GB	1

RTX PRO 4500 Blackwell 也支援圖形啟用 (+gfx) 和媒體引擎 (+me.all、-me) 設定檔變體。如需完整清單，請參閱 NVIDIA 網站上的 NVIDIA 支援的 MIG 設定檔。

g7e.48xlarge – NVIDIA RTX PRO 6000 Blackwell 伺服器版本 96 GB

設定檔	運算配量	每個執行個體的記憶體數量	執行個體上限
1g.24gb	第 1 頁，共 4 頁	24 GB	4
2g.48gb	第 2 頁，共 4 頁	48 GB	2
4g.96gb	第 4 個，共 4 個	96 GB	1

RTX PRO 6000 Blackwell Server Edition 也支援圖形啟用 (+gfx) 和媒體引擎 (+me.all、-me) 設定檔變體。如需完整清單，請參閱 NVIDIA 網站上的 NVIDIA 支援的 MIG 設定檔。

MIG 策略

NVIDIA DRA 驅動程式和 NVIDIA 裝置外掛程式會以不同的方式向 Kubernetes 公開 MIG 執行個體。裝置外掛程式使用全節點的 MIG 策略設定，而 DRA 驅動程式沒有同等設定，因為它會依其屬性選取執行個體。了解此差異是兩種模型之間選擇的關鍵。

NVIDIA DRA 驅動程式

NVIDIA DRA 驅動程式不會使用單一或混合策略概念，也沒有要設定的同等設定。驅動程式不會將 MIG 執行個體公告為計數的資源，而是將每個執行個體發佈為 中的裝置，mig.nvidia.comDeviceClass並具有其 等屬性profile。Pod 會將這些屬性與 ResourceClaim或 中的通用表達式語言 (CEL) 選擇器相符，以選取執行個體ResourceClaimTemplate，如 所示the section called “搭配 NVIDIA DRA 驅動程式使用 MIG”。

由於選取是每個執行個體，因此混合設定檔節點可在沒有策略模式切換的情況下運作。單一 GPU 可以分割成數個不同的設定檔，每個宣告都會選取所需的設定檔。對於 DRA 驅動程式而言重要的選擇不是單一或混合策略，而是靜態或動態 MIG，這會控制您是否預先建立 MIG 執行個體或驅動程式隨需建立它們。如需詳細資訊，請參閱the section called “搭配 NVIDIA DRA 驅動程式使用 MIG”。

NVIDIA 裝置外掛程式

NVIDIA 裝置外掛程式會使用兩種策略之一，向 Kubernetes 公告 MIG 執行個體。由於裝置外掛程式會將 MIG 執行個體公開為節點層級的延伸資源，只包含整數計數，而且沒有每個執行個體的屬性，因此策略會決定這些資源的命名方式。

- 單一策略 – 節點上的每個 GPU 都使用相同的 MIG 設定檔。裝置外掛程式會將每個執行個體公告為nvidia.com/gpu資源，Pod 請求與專用 GPU 的請求nvidia.com/gpu: 1相同。現有的資訊清單不會變更。Bottlerocket 和 AL2023 都支援單一策略。
- 混合策略 – 相同節點上的 GPUs 可以使用不同的 MIG 設定檔。裝置外掛程式會將每個設定檔公告為不同的資源，例如 nvidia.com/mig-1g.10gb或 nvidia.com/mig-3g.40gb，Pod 會請求所需的特定設定檔。您無法將混合策略與 Bottlerocket 的內建 NVIDIA 裝置外掛程式搭配使用。如需詳細資訊，請參閱 GitHub 上的 Bottlerocket GitHub 問題 #4483。 GitHub

搭配 NVIDIA DRA 驅動程式使用 MIG

使用 NVIDIA DRA 驅動程式配置 MIG 執行個體時，Pods 會透過 `ResourceClaim`或 `ResourceClaimTemplate`而非裝置外掛程式的`nvidia.com/mig-<profile>`延伸資源請求 MIG 執行個體。

由於 DRA 驅動程式會依執行個體的屬性而非計數的資源來描述執行個體，因此不會使用裝置外掛程式所需的單一或混合策略（請參閱 the section called “MIG 策略”)。驅動程式會將每個 MIG 執行個體公開為 `gpu.nvidia.com/type` 中的裝置`mig`，`mig.nvidia.com`DeviceClass並公告每個執行個體的屬性，例如實體 GPU `parentUUID`的 `profile`（例如 `1g.5gb`) 和 。您可以使用通用表達式語言 (CEL) 選擇器比對這些屬性，以請求特定設定檔或將多個執行個體保留在相同的 GPU 上。

DRA 驅動程式會以兩種模式之一配置 MIG 執行個體：

- 靜態 MIG – 您可以在驅動程式啟動之前啟用 MIG 模式並在節點上建立 MIG 執行個體，例如在 NVIDIA GPU Operator 中使用 MIG Manager，如中所述the section called “在 AL2023 節點上使用 MIG 搭配 NVIDIA 裝置外掛程式”。驅動程式會探索現有的執行個體並將其配置給 Pod，但不會修改節點的 MIG 組態。在 GPU kubelet 外掛程式重新啟動之前，不會發現驅動程式啟動後新增的執行個體。靜態 MIG 是預設值，不需要任何功能閘道。
- 動態 MIG – 驅動程式會隨需建立和銷毀 MIG 分割區以回應工作負載請求，因此您不會預先分割 GPUs。動態 MIG 是一種依預設停用的 Alpha 功能。您可以使用下列各節中顯示的相同`ResourceClaimTemplate`選取器來請求設定檔，而且驅動程式會分割 GPU 以滿足請求。

考量事項

- 動態 MIG 取代節點上的靜態探索。驅動程式會管理所有分割區，並銷毀 GPU kubelet 外掛程式啟動時未建立的任何 MIG 分割區。請勿在具有您要保留之預先建立分割區的節點上啟用動態 MIG，並且不要執行`mig-parted`或在外掛程式執行`nvidia-smi mig`時啟用動態 MIG，因為手動變更可能會與驅動程式的分割區狀態衝突，並導致 Pod 準備或清除失敗。
- 動態 MIG 處於 Alpha 狀態，且在安裝 NVIDIA DRA 驅動程式時需要啟用功能閘道，請參閱 the section called “安裝 NVIDIA DRA 驅動程式” 以取得說明。
- Hopper (H100 和 H200) 和更新架構會隨需啟用 MIG 模式。上一代無法隨需啟用 MIG 模式，包括 Ampere (A100) GPUs。
- 動態 MIG 取決於 Kubernetes 可分割裝置功能 (GitHub 上的 KEP-4815)，此功能預設為在 Kubernetes 1.36 版和更新版本中啟用。在舊版上，此功能預設不會啟用，因此排程器無法配置動態建立的 MIG 裝置。

先決條件

- 執行 Kubernetes 1.34 版或更新版本的 Amazon EKS 叢集，具有由 Karpenter、EKS 受管節點群組或自我管理節點群組佈建的靜態容量。
- 具備 MIG 功能的 P 系列節點，已啟用 MIG 模式，並將 GPUs 分割為 MIG 執行個體。如需靜態 MIG，請參閱 NVIDIA 網站上的 NVIDIA GPU Operator 中的 MIG Manager。
- 如中所述安裝的 NVIDIA DRA 驅動程式the section called “安裝 NVIDIA DRA 驅動程式”，如果您未使用靜態 MIG 分割，可選擇是否啟用動態 MIG。

程序

下列範例可與靜態或動態 MIG 和 NVIDIA DRA 驅動程式搭配使用。

1. 建立從 `ResourceClaimTemplate`請求 MIG `mig.nvidia.com` 執行個體的 `DeviceClass`，以及參考它的 Pod。此範例會請求任何可用的 MIG 執行個體，而不限制設定檔。

```
cat <<EOF | kubectl apply -f -
apiVersion: resource.k8s.io/v1
kind: ResourceClaimTemplate
metadata:
  name: mig-profile-any
spec:
  spec:
    devices:
      requests:
      - name: mig
        exactly:
          deviceClassName: mig.nvidia.com
          count: 1
---
apiVersion: v1
kind: Pod
metadata:
  name: mig-dra-pod
spec:
  tolerations:
    - key: nvidia.com/gpu
      operator: Exists
      effect: NoSchedule
  containers:
    - name: cuda
```

```
      image: nvidia/cuda:12.6.0-base-ubuntu22.04
      command: ["nvidia-smi", "-L"]
      resources:
        claims:
        - name: mig
  resourceClaims:
    - name: mig
      resourceClaimTemplateName: mig-profile-any
  restartPolicy: OnFailure
EOF
```

2. 確認 Pod 已配置單一 MIG 執行個體。

```
kubectl logs mig-dra-pod
```

範例輸出如下。Pod 會從分割的 GPU 看到一個 MIG 執行個體。

```
GPU 0: NVIDIA A100-SXM4-40GB (UUID: GPU-edd63844-8488-f76a-f6e0-1027a7319a88)
  MIG 2g.10gb     Device  0: (UUID: MIG-a5ad493e-e7e8-5675-9381-1d0f5311a456)
```

請求特定的 MIG 設定檔

若要請求特定設定檔，而非任何可用的執行個體，請新增符合 `profile` 屬性的 CEL 選擇器。下列 `ResourceClaimTemplate`會請求執行個體，而 Pod 會參考該`1g.5gb`執行個體。

```
cat <<EOF | kubectl apply -f -
apiVersion: resource.k8s.io/v1
kind: ResourceClaimTemplate
metadata:
  name: mig-profile-1g.5gb
spec:
  spec:
    devices:
      requests:
      - name: mig
        exactly:
          deviceClassName: mig.nvidia.com
          selectors:
          - cel:
              expression: "device.attributes['gpu.nvidia.com'].profile == '1g.5gb'"
---
```

```
apiVersion: v1
kind: Pod
metadata:
  name: mig-profile-pod
spec:
  tolerations:
    - key: nvidia.com/gpu
      operator: Exists
      effect: NoSchedule
  containers:
    - name: cuda
      image: nvidia/cuda:12.6.0-base-ubuntu22.04
      command: ["nvidia-smi", "-L"]
      resources:
        claims:
        - name: mig
  resourceClaims:
    - name: mig
      resourceClaimTemplateName: mig-profile-1g.5gb
  restartPolicy: OnFailure
EOF
```

從相同的 GPU 請求多個 MIG 執行個體

若要確保單一宣告中的多個 MIG 執行個體來自相同的實體 GPU，請使用 新增`constraints`區塊`matchAttribute: "gpu.nvidia.com/parentUUID"`。下列會從相同的 GPU `ResourceClaimTemplate`請求`1g.5gb`執行個體和`2g.10gb`執行個體，而 Pod 會參考 宣告。由於容器參考宣告而不命名特定請求，因此會收到兩個 MIG 執行個體。

```
cat <<EOF | kubectl apply -f -
apiVersion: resource.k8s.io/v1
kind: ResourceClaimTemplate
metadata:
  name: multi-mig
spec:
  spec:
    devices:
      requests:
      - name: mig-small
        exactly:
          deviceClassName: mig.nvidia.com
          selectors:
          - cel:
```

```
              expression: "device.attributes['gpu.nvidia.com'].profile == '1g.5gb'"
      - name: mig-medium
        exactly:
          deviceClassName: mig.nvidia.com
          selectors:
          - cel:
              expression: "device.attributes['gpu.nvidia.com'].profile == '2g.10gb'"
      constraints:
      - requests: []
        matchAttribute: "gpu.nvidia.com/parentUUID"
---
apiVersion: v1
kind: Pod
metadata:
  name: multi-mig-pod
spec:
  tolerations:
    - key: nvidia.com/gpu
      operator: Exists
      effect: NoSchedule
  containers:
    - name: cuda
      image: nvidia/cuda:12.6.0-base-ubuntu22.04
      command: ["nvidia-smi", "-L"]
      resources:
        claims:
        - name: mig
  resourceClaims:
    - name: mig
      resourceClaimTemplateName: multi-mig
  restartPolicy: OnFailure
EOF
```

在 Bottlerocket 節點上使用 MIG 搭配 NVIDIA 裝置外掛程式

在 Bottlerocket 上，EKS 最佳化的加速 AMI 包含 NVIDIA 裝置外掛程式。您可以透過`settings.kubelet-device-plugins.nvidia`設定以單一策略啟用 MIG。Bottlerocket 在 1.34.0 版和更新版本中支援 MIG。

先決條件

- Amazon EKS 叢集。下列程序使用 EKS 最佳化 Bottlerocket NVIDIA AMI 1.34.0 版或更新版本佈建具備 MIG 功能的 P 系列節點。

- 叢集中已安裝並設定 Karpenter，因為下列程序使用 Karpenter `EC2NodeClass` 在 Bottlerocket 節點使用者資料中提供 MIG 設定。如需詳細資訊，請參閱 Karpenter 網站上的 Karpenter 入門。
- `kubectl` 設定為與您的叢集通訊，the section called “安裝或更新 `kubectl`”如需詳細資訊，請參閱 。

程序

將 MIG 分割設定新增至 GPU 節點的 Bottlerocket 使用者資料。您提供使用者資料的方式取決於您佈建節點的方式。下列範例顯示`p4d.24xlarge`節點`EC2NodeClass`的 Karpenter。

```
cat <<EOF | kubectl apply -f -
apiVersion: karpenter.k8s.aws/v1
kind: EC2NodeClass
metadata:
  name: gpu-bottlerocket-mig
spec:
  amiFamily: Bottlerocket
  amiSelectorTerms:
    - alias: bottlerocket@latest
  role: eksctl-KarpenterNodeRole-<cluster-name>
  subnetSelectorTerms:
    - tags:
        karpenter.sh/discovery: <cluster-name>
  securityGroupSelectorTerms:
    - tags:
        karpenter.sh/discovery: <cluster-name>
  userData: |
    [settings.kubelet-device-plugins.nvidia]
    device-partitioning-strategy = "mig"

    [settings.kubelet-device-plugins.nvidia.mig.profile]
    "a100.40gb" = "2g.10gb"
EOF
```

當使用這些設定佈建的節點加入叢集時，會在 GPUs 上啟用 MIG 模式、將每個 GPU 分割為`2g.10gb`執行個體，且裝置外掛程式會將產生的執行個體公告為`nvidia.com/gpu`資源。由於 `p4d.24xlarge`有八個 A100 GPUs，且每個都支援三個`2g.10gb`執行個體，因此節點會公告 `nvidia.com/gpu: 24`。

> **Note**
>
> 此`mig.profile`設定由 GPU 模型輸入，例如 `a100.40gb`或 `h100.80gb`。如果沒有`mig.profile`設定，GPU 會啟用 MIG 模式並使用其最大的設定檔。由於 Bottlerocket 使用單一策略，節點上的每個 GPU 都使用相同的設定檔。若要在相同的節點（混合策略）上使用不同的設定檔，請使用 AL2023 路徑搭配 NVIDIA GPU Operator。

在 AL2023 節點上使用 MIG 搭配 NVIDIA 裝置外掛程式

在 AL2023 上，下列步驟使用 NVIDIA GPU Operator 安裝 NVIDIA 裝置外掛程式和 MIG Manager。MIG Manager 會啟用 MIG 模式，並根據您提供的組態分割 GPUs。然後，NVIDIA 裝置外掛程式會將產生的執行個體公告給 Kubernetes。GPU Operator 同時支援單一和混合策略。

由於 EKS 最佳化 AL2023 NVIDIA AMI 已包含 NVIDIA 驅動程式和工具組，請在 GPU Operator 中停用驅動程式管理，以避免與預先安裝的驅動程式發生衝突。或者，您可以自行安裝和管理 NVIDIA 裝置外掛程式和 MIG Manager，而無需使用 GPU Operator。

先決條件

- Amazon EKS 叢集。下列程序使用 EKS 最佳化 AL2023 NVIDIA AMI 佈建具備 MIG 功能的 P 系列節點（例如 `p4d.24xlarge`)。
- 您的叢集中已安裝並設定 Karpenter，因為程序會建立 Karpenter `EC2NodeClass`和 `NodePool`來佈建 GPU 節點。如需詳細資訊，請參閱 Karpenter 網站上的 Karpenter 入門。
- 已在命令列環境中安裝 Helm，如需詳細資訊，請參閱安裝 Helm 說明。
- `kubectl` 設定為與您的叢集通訊，the section called “安裝或更新 `kubectl`”如需詳細資訊，請參閱 。

程序

1. `NodePool` 為 AL2023 P 系列 GPU 節點建立 `EC2NodeClass`和 。在 AL2023 上，GPU Operator 會在後續步驟中套用 MIG 分割，因此這是標準 AL2023 GPU 節點類別。下列範例使用 EKS 最佳化 AL2023 NVIDIA AMI 佈建`p4d.24xlarge`節點。

```
cat <<EOF | kubectl apply -f -
apiVersion: karpenter.k8s.aws/v1
kind: EC2NodeClass
metadata:
```

```
  name: gpu-mig-al2023
spec:
  amiFamily: AL2023
  amiSelectorTerms:
    - alias: al2023@latest
  role: eksctl-KarpenterNodeRole-<cluster-name>
  subnetSelectorTerms:
    - tags:
        karpenter.sh/discovery: <cluster-name>
  securityGroupSelectorTerms:
    - tags:
        karpenter.sh/discovery: <cluster-name>
  tags:
    karpenter.sh/discovery: <cluster-name>
---
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: gpu-mig-al2023
spec:
  template:
    spec:
      nodeClassRef:
        group: karpenter.k8s.aws
        kind: EC2NodeClass
        name: gpu-mig-al2023
      taints:
        - key: nvidia.com/gpu
          effect: NoSchedule
      requirements:
        - key: karpenter.sh/capacity-type
          operator: In
          values: ["spot", "on-demand"]
        - key: node.kubernetes.io/instance-type
          operator: In
          values: ["p4d.24xlarge"]
        - key: kubernetes.io/arch
          operator: In
          values: ["amd64"]
  limits:
    cpu: 1000
    memory: 5000Gi
EOF
```

2. 新增 NVIDIA Helm 儲存庫。

```
helm repo add nvidia https://nvidia.github.io/gpu-operator
helm repo update
```

3. 建立gpu-operator-values.yaml檔案以停用驅動程式管理、選取混合策略，並定義要套用的 MIG 設定檔。下列範例定義的p4d-half-balanced組態會將p4d.24xlarge節點上八個 GPUs 中的四個 GPU 分割，並保留整個 GPU。

```
cat <<EOF > gpu-operator-values.yaml
driver:
  enabled: false
toolkit:
  enabled: false
devicePlugin:
  enabled: true
nfd:
  enabled: true
gfd:
  enabled: true
mig:
  strategy: mixed
migManager:
  enabled: true
  env:
    - name: WITH_REBOOT
      value: "true"
  config:
    create: true
    name: custom-mig-parted-configs
    default: all-disabled
    data:
      config.yaml: |-
        version: v1
        mig-configs:
          all-disabled:
            - devices: all
              mig-enabled: false
          p4d-half-balanced:
            - devices: [0, 1, 2, 3]
              mig-enabled: true
              mig-devices:
                "1g.5gb": 2
```

```
                "2g.10gb": 1
                "3g.20gb": 1
            - devices: [4, 5, 6, 7]
              mig-enabled: false
EOF
```

4. 使用值檔案安裝 GPU Operator。

```
helm install gpu-operator nvidia/gpu-operator \
    --namespace gpu-operator \
    --create-namespace \
    --values gpu-operator-values.yaml
```

5. 使用要套用的設定檔組態來標記具備 MIG 功能的節點。MIG Manager 元件會監看此標籤，並相應地分割 GPUs，重新啟動節點以套用變更。

```
kubectl label nodes -l node.kubernetes.io/instance-type=p4d.24xlarge \
    nvidia.com/mig.config=p4d-half-balanced --overwrite
```

6. GPU Operator 分割 GPUs 之後，Pods 會依其資源名稱而非 請求特定的 MIG 設定檔`nvidia.com/gpu`。下列範例會執行請求一個`1g.5gb`執行個體的 Pod。

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: mig-inference
spec:
  tolerations:
    - key: nvidia.com/gpu
      operator: Exists
      effect: NoSchedule
  containers:
    - name: cuda
      image: nvidia/cuda:12.6.0-base-ubuntu22.04
      command: ["nvidia-smi", "-L"]
      resources:
        limits:
          nvidia.com/mig-1g.5gb: 1
EOF
```

如需 P 系列執行個體的完整混合策略組態範例，包括具有容量保留和每個設定檔工作負載的受管節點群組，請參閱《Amazon EKS 最佳實務指南》中的 MIG 一節。

確認 MIG 處於作用中狀態

啟用 MIG 的節點為 之後Ready，請確認 GPUs 上的 MIG 模式處於作用中狀態，且節點會公告預期的 MIG 資源。

1. 確認節點公告 MIG 資源。使用單一策略時，節點會將執行個體報告為 nvidia.com/gpu。使用混合策略時，節點會報告設定檔特定的資源，例如 nvidia.com/mig-1g.10gb。

```
kubectl describe node <node-name> | grep nvidia.com
```

2. nvidia-smi 從已啟用 MIG 的節點上的 Pod 執行，以確認 MIG 模式處於作用中狀態。

   nvidia-smi MIG M.: Enabled報告已開啟 MIG 模式的 GPUs，並列出每個 GPU 上設定的 MIG 執行個體。以下是啟用 MIG 並分割為 3g.20gb、 2g.10gb和 1g.5gb執行個體之 A100 40 GB GPU 的輸出範例。

```
+-----------------------------------------------------------------------------------------
+
| NVIDIA-SMI 580.159.03             Driver Version: 580.159.03     CUDA Version: 13.0
     |
+-----------------------------------------+------------------------
+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr.
 ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util
 Compute M. |
|                                         |                        |
 MIG M. |
|=========================================+========================
+======================|
|   0  NVIDIA A100-SXM4-40GB          On  |   00000000:10:1C.0 Off |
  On |
| N/A   35C    P0             65W /  400W |     213MiB /  40960MiB |     N/A
 Default |
|                                         |                        |
 Enabled |
+-----------------------------------------+------------------------
+----------------------+
```

```
+-----------------------------------------------------------------------------------------
+
| MIG devices:
     |
+------------------+----------------------------------+-----------
+-----------------------+
| GPU  GI  CI  MIG |              Shared Memory-Usage |        Vol|        Shared
     |
|      ID  ID  Dev |                Shared BAR1-Usage | SM     Unc| CE ENC  DEC  OFA
 JPG |
|                  |                                  |        ECC|
     |
|==================+==================================+===========
+=======================|
|  0    1   0   0  |            107MiB / 20096MiB    | 42      0 |  3   0    2    0
   0 |
|                  |              0MiB / 12211MiB    |           |
     |
+------------------+----------------------------------+-----------
+-----------------------+
|  0    5   0   1  |             71MiB /  9984MiB    | 28      0 |  2   0    1    0
   0 |
|                  |              0MiB /  6105MiB    |           |
     |
+------------------+----------------------------------+-----------
+-----------------------+
|  0   13   0   2  |             36MiB /  4864MiB    | 14      0 |  1   0    0    0
   0 |
|                  |              0MiB /  3052MiB    |           |
     |
+------------------+----------------------------------+-----------
+-----------------------+
```

使用混合策略時，Pod 只會看到其請求的 MIG 執行個體，而不是節點的完整 GPU 配置。上一個步驟的 `mig-inference` Pod 請求一個`nvidia.com/mig-1g.5gb`執行個體，因此 Pod `nvidia-smi -L`會列出單一 MIG 裝置。

```
kubectl logs mig-inference
```

範例輸出如下。

```
GPU 0: NVIDIA A100-SXM4-40GB (UUID: GPU-5b7ce860-1951-004c-4881-6ac6997df770)
```

```
  MIG 1g.5gb      Device  0: (UUID: MIG-9b065868-21b6-5b8d-8ab9-e99089ed472c)
```

若要查看節點上每個 GPU 的完整分割區配置，請在主機`nvidia-smi -L`上執行，而不是在工作負載 Pod 內執行。下列命令會在節點上啟動特權偵錯 Pod，並執行主機的 `nvidia-smi`。以已啟用 MIG 的節點名稱取代 *node-name*。

```
kubectl debug node/<node-name> -it --profile=sysadmin --image=nvidia/cuda:12.6.0-
base-ubuntu22.04 -- chroot /host nvidia-smi -L
```

以下是`p4d-half-balanced`組態的範例輸出。前四個 GPUs 會分割為 MIG 執行個體，其餘四個 GPU 是整個 GPUs。每一`MIG`行都是硬體隔離的執行個體，具有自己的 UUID、記憶體和運算配量。

```
GPU 0: NVIDIA A100-SXM4-40GB (UUID: GPU-5b7ce860-1951-004c-4881-6ac6997df770)
  MIG 3g.20gb     Device  0: (UUID: MIG-7dc16162-7ba2-5894-abde-d753dc8ecf56)
  MIG 2g.10gb     Device  1: (UUID: MIG-56cfe1f0-0662-50e4-a5f7-111107e4d5e6)
  MIG 1g.5gb      Device  2: (UUID: MIG-1409727e-2ffa-5fd4-9586-204c9e2b36d5)
  MIG 1g.5gb      Device  3: (UUID: MIG-9b065868-21b6-5b8d-8ab9-e99089ed472c)
GPU 1: NVIDIA A100-SXM4-40GB (UUID: GPU-73692a43-dd2d-f1a1-b0df-2f4734e2a87d)
  MIG 3g.20gb     Device  0: (UUID: MIG-745fd93a-9582-57a6-8b3b-9782d289ca1b)
  MIG 2g.10gb     Device  1: (UUID: MIG-8440294e-c4a0-5687-add5-2cc506babb2f)
  MIG 1g.5gb      Device  2: (UUID: MIG-559810c0-f2bb-5ca2-b529-47228d99437f)
  MIG 1g.5gb      Device  3: (UUID: MIG-a91cf3f9-6459-59e9-97a8-dc69b9958eef)
GPU 2: NVIDIA A100-SXM4-40GB (UUID: GPU-4e56019e-84de-eef5-5ac3-85e468e93639)
  MIG 3g.20gb     Device  0: (UUID: MIG-c130392b-fd7c-59f8-9f4b-ecab27a2984b)
  MIG 2g.10gb     Device  1: (UUID: MIG-7d61cf16-ad08-57be-b2c2-ac6e515b28c3)
  MIG 1g.5gb      Device  2: (UUID: MIG-f5388dec-d841-506c-bb7a-6ac136ceee53)
  MIG 1g.5gb      Device  3: (UUID: MIG-02d54020-c6cb-5997-894b-e95af0f49388)
GPU 3: NVIDIA A100-SXM4-40GB (UUID: GPU-4fd894a0-b471-9e77-eb67-0ad15002ed5b)
  MIG 3g.20gb     Device  0: (UUID: MIG-10399b59-2625-5106-b3f8-76ae19da46e1)
  MIG 2g.10gb     Device  1: (UUID: MIG-ef81ee7d-fc48-56ed-8479-8ffa76eb4154)
  MIG 1g.5gb      Device  2: (UUID: MIG-ed9bf59d-6bf2-5e07-80fb-dd0d7ca41f9d)
  MIG 1g.5gb      Device  3: (UUID: MIG-a15d6f7d-661b-514e-8838-06fe4ffe7f75)
GPU 4: NVIDIA A100-SXM4-40GB (UUID: GPU-05b6b91b-da6e-3078-1f4f-a7bbf1ff7ed2)
GPU 5: NVIDIA A100-SXM4-40GB (UUID: GPU-078a8df1-f387-0315-6b0b-af12e082f6d5)
GPU 6: NVIDIA A100-SXM4-40GB (UUID: GPU-6cdeffe7-45f1-7e8e-bcc1-4634399ad877)
GPU 7: NVIDIA A100-SXM4-40GB (UUID: GPU-5f68814a-4e4a-5dec-79b4-8d70a61c7714)
```

在 Amazon EKS 上使用時間分割搭配 NVIDIA GPUs

時間分割可讓多個 Pod 共用單一實體 NVIDIA GPU。Kubernetes 排程器會將多個 Pod 放置在相同的 GPU 上，GPU 的 CUDA 排程器時間乘以工作。時間分割是最簡單的 GPU 共用策略。它只使用軟體，不需要特殊硬體，並且適用於每個 NVIDIA GPU 執行個體類型 AWS。時間分割不提供共用 GPU 的 Pod 之間的記憶體或運算隔離。它最適合 GPU 使用率低的工作負載，例如在多個使用者共用 GPU 的請求或開發環境之間閒置的推論服務。對於需要硬體層級記憶體和運算隔離的工作負載，請改用多執行個體 GPU (MIG)。

在 Amazon EKS 上，您可以使用 NVIDIA DRA 驅動程式或 NVIDIA 裝置外掛程式來管理時間分割。

如果您使用靜態容量佈建，則時間分割只能與 Karpenter 搭配使用。EKS Auto 模式目前無法使用時間分割。

在以下情況下，時間分割非常適合：

- GPU 使用率持續低，例如在請求之間閒置的延遲敏感推論服務。
- 數個開發人員共用單一 GPU 節點以進行筆記本或開發工作。
- 您的節點使用不支援 MIG 的 GPU 執行個體類型，例如 g5、 g6或 g6e 系列。
- 您可以接受 Pod 偶爾速度較慢，因為同一 GPU 上的另一個 Pod 忙碌中。

在下列情況下，請考慮不同的方法：

- 您需要記憶體隔離。時間分割 GPU 上的 Pod 共用相同的 GPU 記憶體，一個 Pod 可能會耗盡其他 Pod 依賴的記憶體。使用 MIG 進行記憶體隔離。
- 您需要可預測的每個 Pod 延遲或服務品質。GPU 排程器會盡最大努力跨插槽共用運算，無須保證。
- 您執行訓練工作負載。時間分割新增內容切換，可降低訓練效率。另一個 Pod 減速的檢查點任務必須從其最後一個檢查點重試。

考量事項

在生產環境中使用時間分割之前，請檢閱下列考量事項。

一般考量事項

- 無記憶體隔離：共用時間分割 GPU 的 Pod 會共用其記憶體。一個 Pod 可以配置其他 Pod 所需的記憶體，這可能會導致out-of-memory錯誤。將共用每個 GPU 的 Pod 數量與工作負載的記憶體使用量配對。如果您的 Pod 定期載入大型模型，則每個 GPU 共用較少的 Pod 或使用 MIG。

- 最佳運算共用：GPU 排程器會盡最大努力跨 Pod 共用運算，並不保證每個 Pod 的運算比例。
- 時間分割和 MPS 無法共用相同的 GPU：時間分割會將 GPU 運算模式設定為 DEFAULT，而 NVIDIA 多重程序服務 (MPS) 需要 EXCLUSIVE_PROCESS。您可以在相同的叢集中使用這兩種策略，但不能同時在相同的實體 GPU 上使用。時間分割也不會影響 MIG 執行個體。若要跨容器共用單一 MIG 執行個體，請改用 MPS。
- 每個容器指標：當時間分割作用中時，NVIDIA Data Center GPU Manager (DCGM) 無法將指標歸因於個別容器。GPU 層級指標仍然可用，但您無法識別哪些 Pod 使用了給定數量的 GPU 資源。

NVIDIA DRA 驅動程式考量事項

- Alpha 功能：透過 DRA 驅動程式的時間分割需要`TimeSlicingSettings`功能閘道，這是預設停用的 Alpha 功能。如需詳細資訊，請參閱the section called “搭配 NVIDIA DRA 驅動程式使用 GPU 時間分割”。
- 僅限使用者媒介的共用：Pod 僅透過參考相同 `ResourceClaim`或 來共用 `GPUResourceClaimTemplate`，這是命名空間範圍，因此共用無法跨命名空間。系統媒介共享是未來提議的功能。
- 在 Bottlerocket 上停用內建裝置外掛程式：DRA 驅動程式無法與相同節點上的 NVIDIA 裝置外掛程式一起執行。在 Bottlerocket 上，停用需要 Bottlerocket 1.63.0 版或更新版本的內建裝置外掛程式。如需詳細資訊，請參閱the section called “安裝 NVIDIA DRA 驅動程式”。
- 運算支援：Karpenter、EKS 受管節點群組或自我管理節點中的靜態容量佈建支援 NVIDIA DRA 驅動程式，EKS Auto Mode 不支援。如需詳細資訊，請參閱 Karpenter 網站上的 Karpenter 靜態 NodePool 文件。

NVIDIA 裝置外掛程式考量事項

- 使用插槽進行最佳運算共用：裝置外掛程式會公告每個 GPU 固定數量的插槽。如果單一 Pod 請求多個插槽，則不會收到額外的運算。啟用 `fail-requests-greater-than-one`選項以拒絕請求多個槽的 Pod。
- 使用 Karpenter 佈建：即使 GPU 上已啟用時間分割，Karpenter 仍會將每個`nvidia.com/gpu`請求計為實體 GPU。如需詳細資訊，請參閱 GitHub 上的 Karpenter 問題 #2140。
- 不支援 EKS Auto Mode：EKS Auto Mode 會管理 NVIDIA 裝置外掛程式，而不會公開其組態。由於時間分割需要裝置外掛程式組態，您無法在 EKS Auto Mode 節點上套用時間分割組態。
- 時間分割組態變更：NVIDIA 裝置外掛程式不會監控時間分割`ConfigMap`的變更。更新組態後重新啟動裝置外掛程式。

組態選項

您可以將 NVIDIA GPUs 的時間分割與 EKS 最佳化 AL2023 和 Bottlerocket NVIDIA AMIs 搭配使用。時間分割設定會根據您使用的是 NVIDIA DRA 驅動程式或 NVIDIA 裝置外掛程式而有所不同。

NVIDIA DRA driver

EKS 最佳化的加速 AMIs 不包含 NVIDIA DRA 驅動程式。如果您使用的是 Bottlerocket，您必須先停用內建 NVIDIA 裝置外掛程式，才能使用 NVIDIA DRA 驅動程式，方法是在節點使用者資料`settings.kubelet-device-plugins.nvidia.enabled = false`中設定 ，這需要 Bottlerocket 1.63.0 版或更新版本。如需詳細資訊，請參閱the section called “安裝 NVIDIA DRA 驅動程式”。

使用 NVIDIA DRA 驅動程式時，時間分割是透過 ResourceClaimTemplates 設定。`interval` 欄位控制 CUDA 時間配量持續時間。

Interval	描述
預設	使用 NVIDIA GPU 驅動程式的內建預設間隔
Short	較短的間隔；內容更頻繁切換
中	中繼間隔
Long	每個內容在被先佔之前，會先行執行更長的時間

NVIDIA device plugin

- Bottlerocket – AMI 包含預先安裝的 NVIDIA 裝置外掛程式。您可以透過 Bottlerocket 設定來設定時間分割，無需個別的裝置外掛程式安裝、Helm Chart 或 `ConfigMap`。
- AL2023 – 您可以安裝 NVIDIA 裝置外掛程式，如中所述the section called “安裝 NVIDIA Kubernetes 裝置外掛程式”，並透過 提供時間分割組態`ConfigMap`。

  下列選項控制 NVIDIA 裝置外掛程式如何公告和管理時間分割 GPUs。Bottlerocket 設定和 AL2023 之間的欄位名稱不同`ConfigMap`，如以下程序所示。

選項	建議值	說明
複本	2–8	每個實體 GPU 要公告的可排程插槽數量。較高的值允許更多共用，但會增加 Pod 之間的爭用。
預設重新命名	`false`	當 時`false`，Pods 請求 `nvidia.com/gpu`，可保持與現有資訊清單的相容性。當 時`true`，裝置外掛程式會將資源公告為 `nvidia.com/gpu.shared` ，Pod 必須請求該名稱。當您同時執行共用和專用 GPU 節點集區，並希望工作負載明確選取共用資源`true`時，請將此設定為 。
大於 1 的失敗請求	`true`	拒絕請求多個時段的 Pod。請求多個插槽的 Pod 不會接收比例運算。啟用此設定以防止常見的組態錯誤。

如需 GPU 共用選項的完整清單及其預設值，請參閱 GitHub 上的 NVIDIA Kubernetes 裝置外掛程式文件。

搭配 NVIDIA DRA 驅動程式使用 GPU 時間分割

透過 NVIDIA DRA 驅動程式，Pod 會參考`gpu.nvidia.com`DeviceClass使用時間分割 請求 `ResourceClaim`的常見 來共用 GPU`GpuConfig`。

NVIDIA DRA 驅動程式目前實作使用者媒介的時間分割：GPU 只會在您明確指向相同宣告的 Pod 和容器之間共用。由於 `ResourceClaim`和 `ResourceClaimTemplate`是命名空間範圍，因此以這種方式共用 GPU 的 Pod 必須位於相同的命名空間中。若要在單一 Pod 中跨容器共用 GPU，請讓每個容器在宣告中參考相同的請求名稱。參考不同請求名稱的容器都會收到個別的 GPU。針對您需要`ResourceClaimTemplate`的每個時間配量間隔建立個別的 ，並在您要共用 GPUs 的每個命名空間中建立個別的 。

系統媒介的時間分割是指驅動程式根據系統定義的條件而非您設定的宣告，跨獨立宣告（包括跨命名空間）共用 GPU。如需系統媒介時間分割的詳細資訊，請參閱 GitHub 上 GPUs 的系統媒介時間分割（問題編號 659) 和提取請求編號 1257。

> **Important**
>
> 透過 DRA 驅動程式的時間分割需要`TimeSlicingSettings`功能閘道，這是預設停用的 Alpha 功能。如果您在未啟用此功能閘道的情況下請求`TimeSlicing`共用策略，驅動程式將無法準備裝置，且 Pod `ContainerCreating`會與報告`FailedPrepareDynamicResources`的事件一起保持在 中`error validating GPU config: unknown GPU sharing strategy: TimeSlicing`。只有在您接受使用 alpha 功能的風險時，才開啟功能閘道。

先決條件

- 執行 Kubernetes 1.34 版或更新版本的 Amazon EKS 叢集。在 Karpenter、EKS 受管節點群組或自我管理節點中，靜態容量佈建支援 NVIDIA DRA 驅動程式。
- 使用 EKS 最佳化 AL2023 NVIDIA AMI 搭配 NVIDIA GPU 執行個體類型的節點。
- 如中所述安裝的 NVIDIA DRA 驅動程式the section called “安裝 NVIDIA DRA 驅動程式”，已啟用`TimeSlicingSettings`功能閘道。

程序

1. 使用`TimeSlicing`共用策略建立`ResourceClaim`請求 GPU 的 。參考此宣告的多個 Pod 共用相同的實體 GPU。

```
cat <<EOF | kubectl apply -f -
apiVersion: resource.k8s.io/v1
kind: ResourceClaim
metadata:
  name: shared-timeslice-gpu
spec:
  devices:
    requests:
    - name: gpu
      exactly:
        deviceClassName: gpu.nvidia.com
        count: 1
    config:
    - requests: ["gpu"]
      opaque:
        driver: gpu.nvidia.com
```

```
        parameters:
          apiVersion: resource.nvidia.com/v1beta1
          kind: GpuConfig
          sharing:
            strategy: TimeSlicing
            timeSlicingConfig:
              interval: Long
EOF
```

2. 部署參考ResourceClaim依名稱共用的兩個或多個 Pod。每個 Pod 透過 resourceClaims和 參考宣告resources.claims。

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: share-a
spec:
  tolerations:
    - key: nvidia.com/gpu
      operator: Exists
      effect: NoSchedule
  containers:
    - name: cuda
      image: nvidia/cuda:12.6.0-base-ubuntu22.04
      command: ["nvidia-smi", "-L"]
      resources:
        claims:
        - name: gpu
  resourceClaims:
    - name: gpu
      resourceClaimName: shared-timeslice-gpu
  restartPolicy: OnFailure
EOF
```

3. 確認 Pod 共用相同的實體 GPU。在每個 Pod nvidia-smi -L中執行，並確認它們報告相同的 GPU UUID。

```
kubectl logs share-a
```

範例輸出如下。參考相同宣告的第二個 Pod 會報告相同的 GPU UUID，以確認兩個 Pod 共用一個實體 GPU。

```
GPU 0: NVIDIA L4 (UUID: GPU-b41973ee-5d0a-cde8-6287-12b53f861f02)
```

在 Bottlerocket 節點上使用 GPU 時間分割搭配 NVIDIA 裝置外掛程式

在 Bottlerocket 上，EKS 最佳化的加速 AMI 包含 NVIDIA 裝置外掛程式。您可以透過`settings.kubelet-device-plugins.nvidia`設定啟用時間分割，當節點啟動時，Bottlerocket 會轉譯為裝置外掛程式組態。

先決條件

- Amazon EKS 叢集。下列程序使用 EKS 最佳化 Bottlerocket NVIDIA AMI 佈建 NVIDIA GPU 節點。
- 叢集中已安裝和設定的 Karpenter，因為下列程序使用 Karpenter `EC2NodeClass` 在 Bottlerocket 節點使用者資料中提供時間分割設定。如需詳細資訊，請參閱 Karpenter 網站上的 Karpenter 入門。
- `kubectl` 設定為與您的叢集通訊，the section called “安裝或更新 `kubectl`”如需詳細資訊，請參閱 。

程序

將時間分割設定新增至 GPU 節點的 Bottlerocket 使用者資料。下列範例為每個 GPU 設定四個插槽。您提供使用者資料的方式取決於您佈建節點的方式。下列範例顯示 Karpenter `EC2NodeClass`。

```
cat <<EOF | kubectl apply -f -
apiVersion: karpenter.k8s.aws/v1
kind: EC2NodeClass
metadata:
  name: gpu-bottlerocket-timeslicing
spec:
  amiFamily: Bottlerocket
  amiSelectorTerms:
    - alias: bottlerocket@latest
  role: eksctl-KarpenterNodeRole-<cluster-name>
  subnetSelectorTerms:
    - tags:
        karpenter.sh/discovery: <cluster-name>
  securityGroupSelectorTerms:
    - tags:
        karpenter.sh/discovery: <cluster-name>
  userData: |
    [settings.kubelet-device-plugins.nvidia]
```

```
    device-sharing-strategy = "time-slicing"

    [settings.kubelet-device-plugins.nvidia.time-slicing]
    replicas = 4
    rename-by-default = false
    fail-requests-greater-than-one = true
EOF
```

當使用這些設定佈建的節點加入叢集時，裝置外掛程式會為每個實體 GPU 公告四個nvidia.com/gpu插槽。您可以在工作負載規格中使用容器資源請求或nvidia.com/gpu延伸資源的限制來請求它們。

在 AL2023 節點上使用 GPU 時間分割搭配 NVIDIA 裝置外掛程式

在 AL2023 上，您可以安裝 NVIDIA 裝置外掛程式，如中所述the section called “安裝 NVIDIA Kubernetes 裝置外掛程式”，並在 中提供時間分割組態ConfigMap。

先決條件

- 具有使用 NVIDIA GPU 執行個體類型和 EKS 最佳化 AL2023 NVIDIA AMI 之節點的 Amazon EKS 叢集。
- 已在命令列環境中安裝 Helm，如需詳細資訊，請參閱安裝 Helm 說明。
- kubectl 設定為與您的叢集通訊，the section called “安裝或更新 kubectl”如需詳細資訊，請參閱 。

程序

1. 在nvidia命名空間ConfigMap中建立時間分割，其中您在 中安裝裝置外掛程式the section called “安裝 NVIDIA Kubernetes 裝置外掛程式”。此範例為每個 GPU 設定四個插槽。

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  name: nvidia-device-plugin-config
  namespace: nvidia
data:
  config.yaml: |
    version: v1
    sharing:
      timeSlicing:
```

```
        renameByDefault: false
        failRequestsGreaterThanOne: true
        resources:
          - name: nvidia.com/gpu
            replicas: 4
EOF
```

2. 更新現有的 NVIDIA 裝置外掛程式版本，以ConfigMap使用 config.name值參考 。--reuse-values 旗標會保留您在 中安裝裝置外掛程式時設定的值the section called “安裝 NVIDIA Kubernetes 裝置外掛程式”。

```
helm upgrade nvdp nvdp/nvidia-device-plugin \
    --namespace nvidia \
    --reuse-values \
    --set config.name=nvidia-device-plugin-config
```

> **Note**
>
> 當您變更 時，裝置外掛程式不會自動重新載入ConfigMap。更新時間分割組態後，請重新啟動裝置外掛程式 Pod 以套用變更。

驗證 GPU 時間分割是否作用中

時間分割節點為 之後Ready，請確認裝置外掛程式公告預期的插槽數量，以及 Pod 共用實體 GPU。

> **Note**
>
> 此程序中的共用 UUID 檢查會在 Pod 落在相同的實體 GPU 上時，示範最清楚的時間分割。在具有單一 GPU 的節點上，裝置外掛程式會公告四個插槽，而所有四個 Pod 都會共用該 GPU，因此它們會報告相同的 UUID。在具有多個實體 GPUs 的節點上，排程器可以將 Pod 放置在不同的 GPUs 上。即使時間分割處於作用中狀態，這些 Pod 仍會報告不同的 UUIDs。若要示範在一個 GPU 上共用，請將工作負載排程到單一 GPU 執行個體類型。例如，將 等節點選擇器node.kubernetes.io/instance-type: g6.2xlarge新增至 Pod 規格。

1. 確認節點公告設定的 GPU 插槽數量。每個 GPU 有四個插槽，具有一個實體 GPU 的節點會報告 4。

```
kubectl get nodes "-o=custom-
columns=NAME:.metadata.name,GPU:.status.allocatable.nvidia\.com/gpu"
```

範例輸出如下。

```
NAME                                           GPU
ip-192-168-11-225.us-west-2.compute.internal   4
```

2. 建立執行四個複本的部署，每個複本請求一個 GPU 插槽。

```
cat <<EOF | kubectl apply -f -
apiVersion: apps/v1
kind: Deployment
metadata:
  name: timeslicing-demo
spec:
  replicas: 4
  selector:
    matchLabels:
      app: timeslicing-demo
  template:
    metadata:
      labels:
        app: timeslicing-demo
    spec:
      tolerations:
        - key: nvidia.com/gpu
          operator: Exists
          effect: NoSchedule
      containers:
        - name: cuda
          image: nvidia/cuda:12.6.0-base-ubuntu22.04
          command: ["bash", "-c", "nvidia-smi --query-gpu=uuid --format=csv,noheader;
 sleep infinity"]
          resources:
            limits:
              nvidia.com/gpu: 1
EOF
```

3. 確認所有四個 Pod 都已排程在相同的節點上。

```
kubectl get pods -l app=timeslicing-demo -o wide
```

4. 確認所有四個 Pod 都報告相同的 GPU UUID。所有四個 Pod 之間的單一共用 UUID 會確認一個實體 GPU 正在多工。

```
kubectl logs -l app=timeslicing-demo --prefix
```

範例輸出如下。

```
[pod/timeslicing-demo-xxxxxxxxxx-aaaaa/cuda] GPU-c0583cce-87c5-c736-db7f-6d3128c84d03
[pod/timeslicing-demo-xxxxxxxxxx-bbbbb/cuda] GPU-c0583cce-87c5-c736-db7f-6d3128c84d03
[pod/timeslicing-demo-xxxxxxxxxx-ccccc/cuda] GPU-c0583cce-87c5-c736-db7f-6d3128c84d03
[pod/timeslicing-demo-xxxxxxxxxx-ddddd/cuda] GPU-c0583cce-87c5-c736-db7f-6d3128c84d03
```

針對 AI/ML 工作負載最佳化 Amazon EKS 叢集的配方

> **Tip**
>
> 註冊即將舉行的 Amazon EKS AI/ML 研討會。

本節旨在提供位元大小的配方來最佳化 Amazon EKS 叢集，尤其是針對涉及特殊硬體的 AI/ML 工作負載。您可以找到有關透過將污點新增至受管節點群組，以防止在特定節點上排程 Pod 的指引，包括先決條件、逐步程序和部署考量。

主題

- 配方：阻止在特定節點上排程 Pod

配方：阻止在特定節點上排程 Pod

概觀

相較於標準機器上的節點，搭載專用處理器的節點執行成本可能更高昂，例如 GPU。您可以利用 Kubernetes 污點，來保護這些節點免於處理無須特殊硬體的工作負載。污點可標記節點，以拒絕沒有相符公差的 Pod，從而確保僅排程相容的工作負載。如需詳細資訊，請參閱 Kubernetes 文件中的污點和容差。

Kubernetes 節點污點可以使用 AWS 管理主控台 或透過 Amazon EKS API 套用至新的和現有的受管節點群組。此配方說明如何使用 CLI 將污點套用至 Amazon EKS AWS 受管節點群組。如需使用 建立具有污點的節點群組的資訊 AWS 管理主控台，請參閱 the section called “建立”。

先決條件

- 現有 Amazon EKS 叢集。
- AWS CLI 已安裝並設定適當的許可。

步驟

步驟 1：建立具有污點的節點群組

使用 `aws eks create-nodegroup` 命令來建立具有污點的全新受管節點群組。此範例可套用具有鍵 `dedicated`、值 `gpuGroup` 及效果 `NO_SCHEDULE` 的污點。

```
aws eks create-nodegroup \
 --cli-input-json '
{
  "clusterName": "my-cluster",
  "nodegroupName": "node-taints-example",
  "subnets": [
     "subnet-1234567890abcdef0",
     "subnet-abcdef01234567890",
     "subnet-021345abcdef67890"
   ],
  "nodeRole": "arn:aws: iam::111122223333:role/AmazonEKSNodeRole",
  "taints": [
     {
         "key": "dedicated",
         "value": "gpuGroup",
         "effect": "NO_SCHEDULE"
     }
   ]
}'
```

若要了解詳細資訊及範例，請參閱 Kubernetes 參考文件中的污點。

步驟 2：更新現有節點群組上的污點

使用 aws eks update-nodegroup-config AWS CLI 命令，新增、移除或取代受管節點群組的污點。

```
aws eks update-nodegroup-config \
  --cluster-name my-cluster \
  --nodegroup-name node-taints-example \
  --taints 'removeTaints=[{key=dedicated,value=gpuGroup,effect=NO_SCHEDULE}]'
```

備註

- 污點可以在使用 UpdateNodegroupConfig API 建立節點群組後更新。
- 污點金鑰必須以字母或數字開頭。可包含字母、數字、連字號 (-)、句點 (.) 及底線 (_)。長度上限為 63 個字元。
- 或者，污點金鑰可以使用 DNS 子網域字首和單一 / 開頭。如果其以 DNS 子網域字首開頭，則長度可為 253 個字元。
- 值是選用的，必須以字母或數字開頭。可包含字母、數字、連字號 (-)、句點 (.) 及底線 (_)。長度上限為 63 個字元。
- 直接使用 Kubernetes 或 AWS 管理主控台時，污點效果必須為 NoSchedule、PreferNoSchedule 或 NoExecute。不過，使用 AWS CLI 或 API 時，污點效果必須是 NO_SCHEDULE、 PREFER_NO_SCHEDULE或 NO_EXECUTE。
- 每個節點群組最多允許 50 個污點。
- 如果從節點手動移除使用受管節點群組建立的污點，則 Amazon EKS 不會將污點重新加入節點。即使在受管節點群組組態中指定了污點，也是如此。

開始使用 Amazon EKS 上的 AI/ML 的資源

若要進入 EKS 上的機器學習，請先從這些規範性模式中進行選擇，以快速取得 EKS 叢集和 ML 軟體與硬體，進而準備好開始執行 ML 工作負載。

研討會

Amazon EKS 上的生成式 AI 工作坊

了解如何開始使用 Amazon EKS 上的大型語言模型 (LLM) 應用程式和推論。探索如何部署和管理生產級 LLM 工作負載。透過實作實驗室，您將探索如何利用 Amazon EKS 以及 AWS 服務和開放原始碼工具來建立強大的 LLM 解決方案。工作坊環境會提供所有必要的基礎結構和工具，讓您能夠專注於學習和實作。

Amazon EKS 上的生成式 AI，使用 Neuron

了解如何開始使用 Amazon EKS 上的大型語言模型 (LLM) 應用程式和推論。探索如何部署和管理生產級 LLM 工作負載、使用向量資料庫實作進階 RAG 模式，以及使用開放原始碼架構建置資料後端 LLM 應用程式。透過實作實驗室，您將探索如何利用 Amazon EKS 以及 AWS 服務和開放原始碼工具來建立強大的 LLM 解決方案。工作坊環境會提供所有必要的基礎結構和工具，讓您能夠專注於學習和實作。

最佳實務

Amazon EKS 最佳實務指南中的 AI/ML 重點主題提供下列領域的詳細建議，以最佳化 Amazon EKS 上的 AI/ML 工作負載。

AI/ML 運算和自動擴展

本節概述了在 Amazon EKS 中最佳化 AI/ML 運算和自動擴展的最佳實務，其中重點關注 GPU 資源管理、節點彈性和應用程式擴展。它提供了諸多策略，例如使用已知的標籤和節點親和性來排程工作負載、使用 ML 容量區塊或隨需容量保留，以及使用 EKS 節點監控代理程式等工具實作節點運作狀態檢查。

AI/ML 聯網

本節概述了在 Amazon EKS 中最佳化 AI/ML 聯網以增強效能和可擴展性的最佳實務，包括選取具有較高網路頻寬或 Elastic Fabric Adapter (EFA) 的執行個體以進行分散式訓練、安裝 MPI 和 NCCL 等工具，以及啟用字首委派以增加 IP 位址並改善 Pod 啟動時間等策略。

AI/ML 安全

本節著重於保護資料儲存的安全並確保 Amazon EKS 上 AI/ML 工作負載的合規性，包括使用 Amazon S3 搭配 AWS Key Management Service (KMS) 進行伺服器端加密 (SSE-KMS)、使用區域 KMS 金鑰和 S3 儲存貯體金鑰設定儲存貯體以降低成本、授予 IAM 許可給 EKS Pod 解密等 KMS 動作，以及使用 AWS CloudTrail 日誌稽核。

AI/ML 儲存

本節提供最佳化 Amazon EKS 上 AI/ML 工作負載中儲存的最佳實務，包括使用 CSI 驅動程式部署模型以掛載 S3、FSx for Lustre 或 EFS 等服務作為持久性磁碟區、基於工作負載需求選取儲存 (例如 FSx for Lustre 用於使用 Scratch-SSD 或 Persistent-SSD 等選項進行分散式訓練)，以及啟用資料壓縮和分割等功能。

AI/ML 可觀測性

本節著重於監控和最佳化 Amazon EKS 上 AI/ML 工作負載的 GPU 使用率，以提高效率並降低成本，包括使用 CloudWatch Container Insights 和與 Prometheus 和 Grafana 整合的 NVIDIA DCGM-Exporter 等工具鎖定高 GPU 使用率等策略，以及我們建議您分析 AI/ML 工作負載的指標。

AI/ML 效能

本節著重於透過容器映像管理和啟動最佳化，在 Amazon EKS 上增強 AI/ML 工作負載的應用程式擴展和效能，包括使用小型輕量型基礎映像或具有多階段建置的 AWS 深度學習容器、透過 EBS 快照預先載入映像，或使用 DaemonSets 或 部署預先提取至執行階段快取等實務。

參考架構

探索這些 GitHub 儲存庫的參考架構、範例程式碼和公用程式，以在 Amazon EKS 和其他 AWS 服務上實作 AI/ML 工作負載的分散式訓練和推論。

AWSome 分散式訓練

此儲存庫提供在 AWS上訓練大型模型的最佳實務、參考架構、模型訓練範例和公用程式的集合。它可支援 Amazon EKS 的分散式訓練，包括適用於 EKS 叢集的 CloudFormation 範本、自訂 AMI 和容器建置、適用於 PyTorch (DDP/FSDP、MegatronLM、NeMo) 和 JAX 等架構的測試案例，以及各種驗證、可觀測性和效能監控工具，例如 EFA Prometheus 匯出程式和 Nvidia Nsight Systems。

AWSome 推論

此儲存庫提供參考架構和測試案例，以最佳化 上的推論解決方案 AWS，並著重於 Amazon EKS 和加速的 EC2 執行個體。它包含 VPC 和 EKS 叢集的基礎結構設定、NVIDIA NIM、TensorRT-LLM、Triton 推論伺服器和 RayService 等架構的專案，以及 Llama3-8B 和 Llama 3.1 405B 等模型的範例。功能特色包括：使用 K8s LeaderWorkerSet 進行多節點部署、EKS 自動擴展、多執行個體 GPU (MIG)，以及用於 ASR、推論和 TTS 的音訊機器人等實際使用案例。

教學

如果您有興趣在 EKS 中設定機器學習平台和架構，請探索本節所述的教學課程。這些教學課程涵蓋了各種內容，從充分利用 GPU 處理器的模式到選擇建模工具，再到為專業產業建置架構。

在 EKS 上建置生成式 AI 平台

- 在 Amazon EKS 上部署生成式 AI 模型

- 在 Amazon EKS 上建置多租戶 JupyterHub 平台

在 EKS 上執行專用的生成式 AI 架構

- 使用 Amazon EKS 上的 NVIDIA NeMo Framework 加速生成式 AI 分散式訓練工作負載
- 執行 Amazon Elastic Kubernetes Service 上的 TorchServe

最大化 EKS 上 ML 的 NVIDIA GPU 效能

- 實作 GPU 共用，以便高效地為 EKS 叢集使用 NVIDIA GPU：

  在 Amazon EKS 上，使用 NVIDIA 時間分割和加速的 EC2 執行個體實現 GPU 共用
- 使用多執行個體 GPU (MIG) 和 NIM 微服務，在 EKS 叢集上按 GPU 執行更多 Pod：

  在 Amazon EKS 上，使用 NVIDIA 的多執行個體 GPU (MIG) 最大限度地提高 GPU 使用率：按 GPU 執行更多 Pod 以增強效能
- 在具有 Kubeflow on 的 Kubernetes 上建置和部署可擴展的機器學習系統 AWS

在 EKS 上執行影片編碼工作負載

- 在 Amazon EKS 上的容器中，使用分數 GPU 交付影片內容

加速推論工作負載的影像載入

- H2O.ai 如何使用 Karpenter 和 Bottlerocket 最佳化及保護其 AI/ML 基礎結構

監控 ML 工作負載

- 使用 AWS 受管開放原始碼服務監控 Amazon EKS 上的 GPU 工作負載
- 在 Amazon CloudWatch 中啟用 Pod 型 GPU 指標

Amazon EKS 工具

Amazon EKS 提供一套 AWS 工具，可協助您更有效率地管理、疑難排解 EKS 叢集，以及與叢集互動。這些工具利用 AI 功能和標準化界面來簡化叢集操作、加速故障診斷，以及啟用與您的 Kubernetes 資源的自然語言互動。可用的工具包括：

- Amazon EKS 模型內容通訊協定 (MCP) 伺服器 - 一種全受管服務，可為 EKS 叢集管理啟用 AI 驅動的體驗。EKS MCP Server 與 MCP 相容 AI 編碼助理整合，以提供叢集和 Kubernetes 資源的即時情境知識。使用它來管理叢集、部署應用程式、疑難排解問題，以及透過自然語言互動查詢 EKS 文件。
- Amazon EKS 主控台上的 Amazon Q - 直接在 AWS 管理主控台中提供採用 AI 技術的故障診斷整合。當您在 EKS 主控台中遇到錯誤或警告時，Amazon Q 會自動分析叢集問題，並提供調查結果、根本原因分析和建議的緩解步驟。

這些工具可協同運作，在整個 EKS 叢集生命週期中提供全面的支援，從初始設定和組態到持續操作和故障診斷。無論您是開發應用程式、管理基礎設施或解決生產問題，這些 AWS 工具都可協助您更有效率地使用 EKS 叢集。

主題

- Amazon EKS 模型內容通訊協定 (MCP) 伺服器
- 在 Amazon EKS 主控台上使用 Amazon Q Developer

Amazon EKS 模型內容通訊協定 (MCP) 伺服器

Amazon EKS MCP 伺服器是全受管服務，可為開發和操作提供 AI 驅動的體驗。模型內容通訊協定 (MCP) 提供標準化界面，透過對 EKS 叢集和 Kubernetes 資源的即時、內容相關知識，讓 AI 代理程式和應用程式在整個應用程式生命週期中，從初始設定到生產最佳化和故障診斷，都能更準確、更了解內容的回應和採用 AI 技術的工作流程。

> **Note**
>
> Amazon EKS MCP 伺服器目前為 Amazon EKS 的預覽版本，可能會有所變更。

概觀

EKS MCP 伺服器可以輕鬆與任何 MCP 相容 AI 編碼助理整合，以增強您的開發工作流程，例如 Kiro、Amazon Q Developer CLI 或 Cursor 或 Cline 等第三方工具。開始使用時，EKS MCP 伺服器會引導您完成叢集建立、自動佈建先決條件，以及套用 AWS 最佳實務。在開發期間，它透過為應用程式部署和叢集管理提供高階工作流程，簡化 EKS 和 Kubernetes 操作。針對偵錯和故障診斷，伺服器可透過 EKS 主控台和您最愛的 AI 助理上提供的整合式故障診斷輔助和知識庫存取，加速問題解決。這些功能可透過自然語言互動存取，讓您能夠更直覺且有效率地執行複雜的 Kubernetes 操作。

全受管 EKS MCP 伺服器託管在 AWS 雲端中，無需本機安裝和維護。它提供企業級功能，例如自動更新和修補、透過 IAM AWS 整合的集中式安全性、透過 AWS CloudTrail 的全方位稽核記錄，以及經過驗證的可擴展性、可靠性和支援 AWS。託管在 AWS 雲端的全受管 EKS MCP 伺服器提供下列主要優點：

- 消除安裝和維護。在 AWS 雲端託管 EKS MCP 伺服器時，您不再需要管理版本更新，或疑難排解本機伺服器問題。只要將 AI 助理設定為連線至託管 EKS MCP 伺服器端點，您就可以開始使用 EKS 叢集。
- 集中式存取管理。EKS MCP 伺服器與 IAM 整合，提供集中且安全的方式來控制對伺服器的存取。所有請求都會透過輕量型代理使用 AWS SigV4 簽署，以便與您現有的 AWS 憑證和 IAM 政策無縫整合。
- 增強監控和可見性。 AWS CloudTrail 整合會擷取透過託管服務進行的初始化和完整存取工具呼叫，以啟用詳細的稽核追蹤和合規報告。
- 隨時up-to-date狀態。自動接收新的工具、功能和錯誤修正，而不需要更新本機安裝。託管服務會根據您的意見回饋和 AWS 最佳實務持續改善。

整合範例

EKS MCP 伺服器提供數種工具，您可以用來：

- 管理您的叢集 使用自動化最佳實務建立、設定和管理 EKS 叢集。
- 管理 Kubernetes 資源 部署應用程式、管理 Kubernetes 資源，以及檢查叢集狀態。
- 對叢集進行故障診斷 使用整合式故障診斷工具和 Runbook 的知識庫來診斷問題
- 查詢文件 以情境方式搜尋和擷取相關的 EKS 文件。

探索您的叢集

```
Show me all EKS clusters and their status
What insights does EKS have about my production-cluster?
Show me the VPC configuration for my staging cluster
```

檢查 Kubernetes 資源

```
List all deployments in the production namespace
Show me pods that are not in Running state
Get the logs from the api-server pod in the last 30 minutes
```

針對 問題進行故障診斷

```
Why is my nginx-ingress-controller pod failing to start?
Search the EKS troubleshooting guide for pod networking issues
Show me events related to the failed deployment in the staging namespace
```

建立資源 （ 如果已啟用「寫入」模式 ）

```
Create a new EKS cluster named demo-cluster with VPC and Auto Mode
Deploy my containerized app from ECR to the production namespace with 3 replicas
Generate a Kubernetes deployment YAML for my Node.js app running on port 3000
```

開始使用

若要開始使用，請參閱the section called “開始使用”。

Amazon EKS MCP 伺服器入門

本指南會逐步引導您設定 AI 程式碼助理並使用 EKS MCP 伺服器的步驟。您將了解如何設定環境、連線至伺服器，以及透過自然語言互動開始管理 EKS 叢集。

> **Note**
>
> Amazon EKS MCP 伺服器目前為 Amazon EKS 的預覽版本，可能會有所變更。

先決條件

開始之前，請確定您已執行下列任務：

- 建立可存取 Amazon EKS AWS 的帳戶
- 使用登入資料安裝和設定 AWS CLI
- 安裝 Python 3.10+
- 安裝 uv

設定

1. 確認 先決條件

```
# Check that your Python version is 3.10 or higher
python3 --version

# Check uv installation
uv --version

# Verify CLI configuration
aws configure list
```

2. 設定 IAM 許可

若要連線至 EKS MCP 伺服器，您的 IAM 角色必須連接下列政策： **eks-mcp:InvokeMcp**（初始化和擷取可用工具相關資訊所需的許可）、 **eks-mcp:CallReadOnlyTool**（使用唯讀工具所需的許可）和 **eks-mcp:CallPrivilegedTool**（使用完整存取（寫入）工具所需的許可）。這些eks-mcp許可包含在以下提供的唯讀和完整存取 AWS 受管政策中。

- 開啟 IAM 主控台。
- 在左側導覽窗格中，根據您要連接政策的身分，然後選擇使用者、使用者群組或角色，然後選擇特定使用者、群組或角色的名稱。
- 選擇許可索引標籤。
- 選擇連接政策（如果是第一次新增許可)。
- 在政策清單中，搜尋並選取您要連接的受管政策：
- 唯讀操作：AmazonEKSMCPReadOnlyAccess
- 選擇連接政策（或下一步，然後選擇新增許可以確認）。

這會連接政策，且許可會立即生效。您可以將多個政策連接到相同的身分，而且每個政策都可以包含各種許可。若要進一步了解這些政策，請參閱 AWS Amazon Elastic Kubernetes Service 的 受管政策。

3. 選擇 AI 助理

選擇下列其中一個 MCP 相容 AI 助理或任何 MCP 相容工具：

- 安裝 Amazon Q 開發人員 CLI
- 安裝 Kiro
- 安裝游標
- 安裝 Cline VS 程式碼延伸模組

步驟 1：設定您的 AI 助理

從下列任一選項中選擇，以設定您的 AI 程式碼助理。完成此步驟會將您的 AI 程式碼助理設定為使用 MCP Proxy AWS，這是安全、已驗證存取 Amazon EKS MCP 伺服器的必要項目。這包括新增或編輯 MCP 組態檔案（例如，`~/.aws/amazonq/mcp.json`適用於 Amazon Q Developer CLI)。代理充當用戶端橋接器，使用本機 AWS 登入資料處理 AWS SigV4 身分驗證，並啟用動態工具探索，以便與後端 AWS MCP 伺服器互動，例如 EKS MCP 伺服器。若要進一步了解，請參閱儲存 AWS 庫的 MCP Proxy。

選項 A：Amazon Q 開發人員 CLI

Q Developer CLI 提供與 EKS MCP 伺服器最整合的體驗。

1. 尋找 MCP 組態檔案

- macOS/Linux： `~/.aws/q/mcp.json`
- Windows: `%USERPROFILE%\.aws\q\mcp.json`

2. 新增 MCP 伺服器組態

如果組態檔案不存在，請建立組態檔案。請務必將區域 (`{region}`) 預留位置取代為您想要的區域。

對於 Mac/Linux：

```
{
  "mcpServers": {
```

```
    "eks-mcp": {
      "disabled": false,
      "type": "stdio",
      "command": "uvx",
      "args": [
        "mcp-proxy-for-aws@latest",
        "https://eks-mcp.{region}.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
        "default",
        "--region",
        "{region}"
      ]
    }
  }
}
```

用於 Windows：

```
{
  "mcpServers": {
    "eks-mcp": {
      "disabled": false,
      "type": "stdio",
      "command": "uvx",
      "args": [
        "--from",
        "mcp-proxy-for-aws@latest",
        "mcp-proxy-for-aws.exe",
        "https://eks-mcp.{region}.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
        "default",
        "--region",
        "{region}"
      ]
    }
  }
}
```

安全備註： `--read-only` 只能用於允許唯讀工具操作。

3. 驗證組態

重新啟動 Q 開發人員 CLI，然後檢查可用的工具：

```
q /tools
```

選項 B：Kiro IDE

Kiro 是具有內建 MCP 支援的 AI 優先編碼工作區。

1. 開啟 Kiro 設定

- 開啟 Kiro
- 前往 Kiro → 設定並搜尋 "MCP Config"
- 或按 Cmd+Shift+P(Mac) 或 Ctrl+Shift+P(Windows/Linux) 搜尋 "MCP Config"

2. 新增 MCP 伺服器組態

- 按一下「開啟工作區 MCP Config」或「開啟使用者 MCP Config」以直接編輯 MCP 組態檔案。

請務必將區域 ({region}) 預留位置取代為您想要的區域。

對於 Mac/Linux：

```
{
  "mcpServers": {
    "eks-mcp": {
      "disabled": false,
      "type": "stdio",
      "command": "uvx",
      "args": [
        "mcp-proxy-for-aws@latest",
        "https://eks-mcp.{region}.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
        "default",
        "--region",
        "{region}"
      ]
    }
```

```
  }
}
```

用於 Windows：

```
{
  "mcpServers": {
    "eks-mcp": {
      "disabled": false,
      "type": "stdio",
      "command": "uvx",
      "args": [
        "--from",
        "mcp-proxy-for-aws@latest",
        "mcp-proxy-for-aws.exe",
        "https://eks-mcp.{region}.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
        "default",
        "--region",
        "{region}"
      ]
    }
  }
}
```

安全備註： `--read-only` 只能用於允許唯讀工具操作。

選項 C：游標 IDE

游標透過圖形組態界面提供內建的 MCP 支援。

1. 開啟游標設定

- 開啟游標
- 前往設定 → 游標設定 → 工具和 MCP
- 或按 `Cmd+Shift+P`(Mac) / `Ctrl+Shift+P`(Windows) 搜尋 "MCP"

2. 新增 MCP 伺服器組態

- 按一下「新的 MCP 伺服器」

如果組態檔案不存在，請建立組態檔案。請務必將區域 ({region}) 預留位置取代為您想要的區域。

對於 Mac/Linux：

```
{
  "mcpServers": {
    "eks-mcp": {
      "disabled": false,
      "type": "stdio",
      "command": "uvx",
      "args": [
        "mcp-proxy-for-aws@latest",
        "https://eks-mcp.{region}.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
        "default",
        "--region",
        "{region}"
      ]
    }
  }
}
```

用於 Windows：

```
{
  "mcpServers": {
    "eks-mcp": {
      "disabled": false,
      "type": "stdio",
      "command": "uvx",
      "args": [
        "--from",
        "mcp-proxy-for-aws@latest",
        "mcp-proxy-for-aws.exe",
        "https://eks-mcp.{region}.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
        "default",
        "--region",
        "{region}"
```

```
      ]
    }
  }
}
```

安全備註： `--read-only` 只能用於允許唯讀工具操作。

3. 重新啟動游標

關閉並重新開啟游標，以使變更生效。

4. 在游標聊天中驗證

開啟聊天面板並嘗試：

```
What EKS MCP tools are available?
```

您應該會看到可用的 EKS 管理工具清單。

選項 D：Cline (VS 程式碼延伸）

Cline 是熱門的 VS 程式碼延伸模組，可將 AI 協助直接帶入編輯器。

1. 開啟曲線設定

- 開啟曲線
- 按 `Cmd+Shift+P`(Mac) / `Ctrl+Shift+P`(Windows) 搜尋 "MCP"

2. 新增 MCP 伺服器組態

- 按一下「新增伺服器」
- 按一下「開啟使用者組態」

如果組態檔案不存在，請建立組態檔案。請務必將區域 (`{region}`) 預留位置取代為您想要的區域。

對於 Mac/Linux：

```
{
  "mcpServers": {
```

```
    "eks-mcp": {
      "disabled": false,
      "type": "stdio",
      "command": "uvx",
      "args": [
        "mcp-proxy-for-aws@latest",
        "https://eks-mcp.{region}.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
        "default",
        "--region",
        "{region}"
      ]
    }
  }
}
```

用於 Windows：

```
{
  "mcpServers": {
    "eks-mcp": {
      "disabled": false,
      "type": "stdio",
      "command": "uvx",
      "args": [
        "--from",
        "mcp-proxy-for-aws@latest",
        "mcp-proxy-for-aws.exe",
        "https://eks-mcp.{region}.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
        "default",
        "--region",
        "{region}"
      ]
    }
  }
}
```

安全備註： `--read-only` 只能用於允許唯讀工具操作。

3. 重新載入 VS 程式碼

按 Cmd+Shift+P / Ctrl+Shift+P然後選取「開發人員：重新載入視窗」

4. 驗證組態

開啟 Cline 並詢問：

```
List the available MCP tools for EKS
```

步驟 2：（選用）建立「寫入」政策

或者，您可以建立客戶受管 IAM 政策，提供 Amazon EKS MCP 伺服器的完整存取權。此政策授予許可，以使用 EKS MCP 伺服器中的所有工具，包括可能涉及寫入操作和唯讀工具的特權工具。請注意，此政策包含高風險許可（具備 Delete* 的任何項目，或不受限制的 IAM 資源），因為在 manage_eks_stacks 工具中設定/縮減叢集資源時需要這些許可。

```
aws iam create-policy \
 --policy-name EKSMcpWriteManagementPolicy \
 --policy-document "{\"Version\": \"2012-10-17\", \"Statement\": [{\"Effect
\": \"Allow\", \"Action\": [\"eks:DescribeCluster\", \"eks:ListClusters\",
 \"eks:DescribeNodegroup\", \"eks:ListNodegroups\", \"eks:DescribeAddon\",
 \"eks:ListAddons\", \"eks:DescribeAccessEntry\", \"eks:ListAccessEntries\",
 \"eks:DescribeInsight\", \"eks:ListInsights\", \"eks:AccessKubernetesApi\"],
 \"Resource\": \"*\"}, {\"Effect\": \"Allow\", \"Action\": [\"eks:CreateCluster
\", \"eks:DeleteCluster\", \"eks:CreateAccessEntry\", \"eks:TagResource\"],
 \"Resource\": \"*\"}, {\"Effect\": \"Allow\", \"Action\": [\"iam:GetRole\",
 \"iam:ListRolePolicies\", \"iam:ListAttachedRolePolicies\", \"iam:GetRolePolicy\",
 \"iam:GetPolicy\", \"iam:GetPolicyVersion\"], \"Resource\": \"*\"}, {\"Effect\":
 \"Allow\", \"Action\": [\"iam:TagRole\", \"iam:CreateRole\", \"iam:AttachRolePolicy
\", \"iam:PutRolePolicy\", \"iam:DetachRolePolicy\", \"iam:DeleteRole\"],
 \"Resource\": \"*\"}, {\"Effect\": \"Allow\", \"Action\": [\"iam:PassRole\"],
 \"Resource\": \"*\", \"Condition\": {\"StringEquals\": {\"iam:PassedToService
\": [\"eks.amazonaws.com\", \"ec2.amazonaws.com\"]}}}, {\"Effect\": \"Allow\",
 \"Action\": [\"ec2:CreateVpc\", \"ec2:CreateSubnet\", \"ec2:CreateRouteTable\",
 \"ec2:CreateRoute\", \"ec2:CreateInternetGateway\", \"ec2:CreateNatGateway\",
 \"ec2:CreateSecurityGroup\", \"ec2:AttachInternetGateway\", \"ec2:AssociateRouteTable
\", \"ec2:ModifyVpcAttribute\", \"ec2:ModifySubnetAttribute\", \"ec2:AllocateAddress
\", \"ec2:CreateTags\"], \"Resource\": \"*\"}, {\"Effect\": \"Allow\", \"Action
\": [\"ec2:DeleteVpc\", \"ec2:DeleteSubnet\", \"ec2:DisassociateRouteTable\",
 \"ec2:DeleteRouteTable\", \"ec2:DeleteRoute\", \"ec2:DetachInternetGateway\",
 \"ec2:DeleteInternetGateway\", \"ec2:DeleteNatGateway\", \"ec2:ReleaseAddress\",
```

```
 \"ec2:DeleteSecurityGroup\"], \"Resource\": \"*\"}, {\"Effect\": \"Allow\", \"Action
\": [\"ec2:DescribeVpcs\", \"ec2:DescribeSubnets\", \"ec2:DescribeRouteTables\",
 \"ec2:DescribeInternetGateways\", \"ec2:DescribeNatGateways\", \"ec2:DescribeAddresses
\", \"ec2:DescribeSecurityGroups\", \"ec2:DescribeAvailabilityZones\"], \"Resource
\": \"*\"}, {\"Effect\": \"Allow\", \"Action\": [\"cloudformation:CreateStack
\", \"cloudformation:UpdateStack\", \"cloudformation:DeleteStack\",
 \"cloudformation:DescribeStacks\", \"cloudformation:TagResource\"], \"Resource
\": \"*\"}, {\"Effect\": \"Allow\", \"Action\": [\"sts:GetCallerIdentity\"],
 \"Resource\": \"*\"}, {\"Effect\": \"Allow\", \"Action\": [\"logs:StartQuery\",
 \"logs:GetQueryResults\"], \"Resource\": \"*\"}, {\"Effect\": \"Allow\", \"Action\":
 [\"cloudwatch:GetMetricData\"], \"Resource\": \"*\"}, {\"Effect\": \"Allow\", \"Action
\": [\"eks-mcp:*\"], \"Resource\": \"*\"}]}"
```

步驟 3：驗證您的設定

測試連線

向您的 AI 助理詢問簡單的問題，以驗證連線：

```
List all EKS clusters in my {aws} account
```

您應該會看到 EKS 叢集的清單。

步驟 4：執行您的第一個任務

範例 1：探索您的叢集

```
Show me all EKS clusters and their status
What insights does EKS have about my production-cluster?
Show me the VPC configuration for my staging cluster
```

範例 2：檢查 Kubernetes 資源

```
Get the details of all the kubernetes resources deployed in my EKS cluster
Show me pods that are not in Running state or pods with any restarts
Get the logs from the aws-node daemonset in the last 30 minutes
```

範例 3：故障診斷問題

```
Why is my nginx-ingress-controller pod failing to start?
Search the EKS troubleshooting guide for pod networking issues
```

```
Show me events related to the failed deployment in the staging namespace
```

範例 4：建立資源（如果已啟用「寫入」模式）

```
Create a new EKS cluster named demo-cluster with VPC and Auto Mode
Deploy my containerized app from ECR to the production namespace with 3 replicas
Generate a Kubernetes deployment YAML for my Node.js app running on port 3000
```

常見組態

案例 1：多個 AWS 設定檔

如果您使用多個 AWS 帳戶，請建立個別的 MCP 伺服器組態。

對於 Mac/Linux：

```
{
  "mcpServers": {
    "eks-mcp-prod": {
      "disabled": false,
      "type": "stdio",
      "command": "uvx",
      "args": [
        "mcp-proxy-for-aws@latest",
        "https://eks-mcp.us-west-2.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
        "production",
        "--region",
        "us-west-2"
      ]
    },
    "eks-mcp-dev": {
      "disabled": false,
      "type": "stdio",
      "command": "uvx",
      "args": [
        "mcp-proxy-for-aws@latest",
        "https://eks-mcp.us-east-1.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
```

```
        "development",
        "--region",
        "us-east-1"
      ]
    }
  }
}
```

用於 Windows：

```
{
  "mcpServers": {
    "eks-mcp-prod": {
      "disabled": false,
      "type": "stdio",
      "command": "uvx",
      "args": [
        "--from",
        "mcp-proxy-for-aws@latest",
        "mcp-proxy-for-aws.exe",
        "https://eks-mcp.us-west-2.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
        "production",
        "--region",
        "us-west-2"
      ]
    },
    "eks-mcp-dev": {
      "disabled": false,
      "type": "stdio",
      "command": "uvx",
      "args": [
        "--from",
        "mcp-proxy-for-aws@latest",
        "mcp-proxy-for-aws.exe",
        "https://eks-mcp.us-east-1.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
        "development",
        "--region",
```

```
        "us-east-1"
      ]
    }
  }
}
```

案例 2：生產唯讀

建立生產環境的唯讀組態。

對於 Mac/Linux：

```
{
  "mcpServers": {
    "eks-mcp-prod-readonly": {
      "command": "uvx",
      "args": [
        "mcp-proxy-for-aws@latest",
        "https://eks-mcp.us-west-2.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
        "production",
        "--region",
        "us-west-2",
        "--read-only"
      ],
      "autoApprove": [
        "list_k8s_resources",
        "get_pod_logs",
        "get_k8s_events"
      ]
    }
  }
}
```

用於 Windows：

```
{
  "mcpServers": {
    "eks-mcp-prod-readonly": {
      "command": "uvx",
```

```
      "args": [
        "--from",
        "mcp-proxy-for-aws@latest",
        "mcp-proxy-for-aws.exe",
        "https://eks-mcp.us-west-2.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
        "production",
        "--region",
        "us-west-2",
        "--read-only"
      ],
      "autoApprove": [
        "list_k8s_resources",
        "get_pod_logs",
        "get_k8s_events"
      ]
    }
  }
}
```

案例 3：具有完整存取權的開發

適用於具有完整寫入存取權的開發環境。

對於 Mac/Linux：

```
{
  "mcpServers": {
    "eks-mcp-dev-full": {
      "command": "uvx",
      "args": [
        "mcp-proxy-for-aws@latest",
        "https://eks-mcp.us-east-1.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
        "development",
        "--region",
        "us-east-1"
      ]
    }
  }
```

```
}
```

用於 Windows：

```
{
  "mcpServers": {
    "eks-mcp-dev-full": {
      "command": "uvx",
      "args": [
        "--from",
        "mcp-proxy-for-aws@latest",
        "mcp-proxy-for-aws.exe",
        "https://eks-mcp.us-east-1.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
        "development",
        "--region",
        "us-east-1"
      ]
    }
  }
}
```

考量事項

安全

請勿透過允許的輸入機制傳遞秘密或敏感資訊：

- 請勿在搭配 apply_yaml 套用的 YAML 檔案中包含秘密或登入資料。
- 請勿直接在提示中將敏感資訊傳遞給模型。
- 請勿在 CloudFormation 範本或應用程式資訊清單中包含秘密。
- 避免使用 MCP 工具建立 Kubernetes 秘密，因為這需要將秘密資料提供給模型。
- 避免在 Kubernetes Pod 內的應用程式日誌中記錄敏感資訊。

YAML 內容安全性：

- 僅使用來自可信任來源的 YAML 檔案。

- 伺服器依賴 Kubernetes API 驗證 YAML 內容，並且不執行自己的驗證。
- 將 YAML 檔案套用至叢集之前，請對其進行稽核。

而不是透過 MCP 傳遞秘密：

- 使用 AWS Secrets Manager 或 參數存放區來存放敏感資訊。
- 為服務帳戶設定適當的 Kubernetes RBAC。
- 將服務帳戶 (IRSA) 的 IAM 角色用於 Pod AWS 的服務存取。

敏感資料的編輯：

- EKS MCP 伺服器會自動修訂工具回應中安全字符、憑證和其他敏感資訊的常見模式。
- 修訂後的值會取代為 `HIDDEN_FOR_SECURITY_REASONS`，以避免不小心將資料暴露至模型。
- 此修訂適用於所有工具回應，包括日誌、資源描述和組態資料。

下一步

如需組態選項，請參閱 the section called “組態”。如需工具的完整清單，請參閱 the section called “工具”。

Amazon EKS MCP 伺服器組態參考

本指南顯示 mcp-proxy-for-aws 用戶端工具可用的所有組態，可讓您從 IDE 連線至全受管 Amazon EKS MCP 伺服器。

> **Note**
>
> Amazon EKS MCP 伺服器目前為 Amazon EKS 的預覽版本，可能會有所變更。

範例

```
{
  "mcpServers": {
    "eks-mcp": {
      "disabled": false,
```

```
      "type": "stdio",
      "command": "uvx",
      "args": [
        "mcp-proxy-for-aws@latest",
        "https://eks-mcp.us-west-2.api.aws/mcp",
        "--service",
        "eks-mcp",
        "--profile",
        "default",
        "--region",
        "us-east-1",
        "--read-only"
      ]
    }
  }
}
```

IAM 許可

用於連線至 MCP 伺服器的角色需要**eks-mcp:InvokeMcp**許可，才能初始化和擷取可用工具的相關資訊。使用唯讀工具**eks-mcp:CallReadOnlyTool**時需要 ，使用完整存取 （ 寫入 ） 工具**eks-mcp:CallPrivilegedTool**時需要 。

環境變數

AWS_PROFILE （選用）要使用的 AWS 憑證設定檔名稱；可由`--profile`命令列引數覆寫。

- 範例：`export AWS_PROFILE=production`

AWS_REGION （選用）SigV4 簽署 AWS 的區域；`us-west-2`如果未設定，則預設為 。

- 範例：`export AWS_REGION=us-east-1`

引數

SigV4 MCP 端點 URL （ 必要 ） 要連線的 MCP 端點 URL。

- 範例：`https://eks-mcp.us-west-2.api.aws/mcp`

--service （選用）SigV4 簽署 AWS 的服務名稱；若未提供，會自動從端點主機名稱偵測。

- 範例：`--service eks-mcp`

`--profile` （選用）要使用的 AWS 憑證設定檔。如果未指定，則預設為 `AWS_PROFILE`環境變數。

- 範例：`--profile production`

`--region` AWS 要使用的區域。如果未設定 ，則使用`AWS_REGION`環境變數，預設為 `us-east-1`。

- 範例：`--region us-west-2`

`--read-only` （選用）停用可能需要寫入許可的工具（不需要寫入許可的工具會以 readOnlyHint=true 標註）。預設會啟用所有工具。

- 範例：`--read-only`

如需更多組態選項，請參閱組態參數。

工具

如需所有可用工具的詳細資訊，包括參數和使用範例，請參閱 the section called “工具”。

Amazon EKS MCP 伺服器工具參考

伺服器會公開下列 MCP 工具。

> **Note**
>
> Amazon EKS MCP 伺服器目前為 Amazon EKS 的預覽版本，可能會有所變更。

唯讀工具

本節說明 EKS MCP 伺服器可用的唯讀工具。請注意，所有唯讀 Kubernetes API 操作都可以存取兩者：

- 私有叢集（請參閱叢集私有端點)
- 公有叢集

search_eks_documentation

搜尋 EKS 文件以取得up-to-date和指引。此工具可讓您存取最新的 EKS 文件，包括客服人員可能不知道的新功能和最近的增強功能。

參數：

- 查詢 （必要）：與 EKS 文件、功能或最佳實務相關的特定問題或搜尋查詢。
- limit （選用）：要傳回的文件結果數目上限 (1-10)。預設：5。

search_eks_troubleshooting_guide

搜尋 EKS 故障診斷指南，以根據查詢對資訊進行故障診斷。它有助於識別常見問題並提供step-by-step解決方案。

參數：

- query （必要）：與 EKS 疑難排解相關的特定問題或問題描述。

describe_eks_resource

擷取特定 EKS 叢集資源的詳細資訊，包括組態、狀態和中繼資料。

參數：

- cluster_name （必要）：EKS 叢集的名稱 （叢集範圍的資源必要）。
- resource_type （必要）：要描述的 EKS 資源類型。有效值：
- accessentry （需要 cluster_name 和 resource_name 作為 principalArn)
- 附加元件 （需要 cluster_name 和 resource_name 作為附加元件名稱）
- 叢集 （需要 cluster_name)、節點群組 （需要 cluster_name 和 resource_name 作為節點群組名稱）。
- resource_name （選用）：要描述的特定資源名稱 （大多數資源類型需要）。

list_eks_resources

列出特定類型的 EKS 資源，傳回可存取之指定類型的所有資源摘要。

參數：

- resource_type （必要）：要列出的 EKS 資源類型。有效值：
- accessentry （需要 cluster_name)
- 附加元件 （需要 cluster_name)
- 叢集 （不需要其他參數）
- nodegroup （需要 cluster_name)。
- cluster_name （選用）：EKS 叢集的名稱 （叢集範圍的資源需要）。

get_eks_insights

擷取 EKS 叢集洞察和最佳化建議。根據 AWS 最佳實務和叢集分析，提供安全性、效能和成本最佳化的可行洞見。

參數：

- cluster_name （必要）：EKS 叢集的名稱。
- 類別 （選用）：選用類別來篩選洞見 （例如，「MISCONFIGURATION」或「UPGRADE_READINESS」)。
- insight_id （選用）：特定洞見的選用 ID，以取得詳細資訊。
- next_token （選用）：選用分頁權杖，以取得下一組結果。

get_eks_vpc_config

擷取 EKS 叢集的 VPC 組態，包括子網路、路由表和網路連線。

參數：

- cluster_name （必要）：要取得 VPC 組態的 EKS 叢集名稱。
- vpc_id （選用）：要查詢的特定 VPC ID （選用，若未指定，將使用叢集 VPC)。

get_k8s_events

擷取與特定資源相關的 Kubernetes 事件，以進行疑難排解和監控。

參數：

- cluster_name （必要）：資源所在的 EKS 叢集名稱。

- kind （必要）：所涉及物件的類型（例如 "Pod"、"Deployment"、"Service")。必須與資源類型完全相符。
- name （必要）：要取得事件的涉及物件名稱。
- 命名空間 （選用）：所涉及物件的命名空間。命名空間資源 （例如 Pod、部署） 的必要項目。叢集範圍的資源 （例如 Nodes、PersistentVolumes) 不需要。

get_pod_logs

使用篩選選項從 EKS 叢集中的 Pod 擷取日誌。

參數：

- cluster_name （必要）：執行 Pod 的 EKS 叢集名稱。
- 命名空間 （必要）：Pod 所在的 Kubernetes 命名空間。
- pod_name （必要）：要從中擷取日誌的 Pod 名稱。
- container_name （選用）：要從中取得日誌的特定容器名稱。只有在 Pod 包含多個容器時才需要。
- limit_bytes （選用）：傳回的位元組數上限。預設：10KB (10240 位元組）。
- 上一個 （選用）：傳回先前終止的容器日誌 （預設為 false)。用於取得正在重新啟動之 Pod 的日誌。
- since_seconds （選用）：僅傳回比此數秒更新的日誌。有助於在不擷取整個歷史記錄的情況下取得最近的日誌。
- tail_lines （選用）：從日誌結尾傳回的行數。預設：100。

list_api_versions

列出指定 Kubernetes 叢集中所有可用的 API 版本。

參數：

- cluster_name （必要）：EKS 叢集的名稱。

list_k8s_resources

列出 EKS 叢集中特定類型的 Kubernetes 資源。

參數：

- cluster_name （必要）：資源所在的 EKS 叢集名稱。
- kind （必要）：要列出的 Kubernetes 資源類型 （例如 'Pod'、'Service'、'Deployment')。使用 `list_api_versions`工具尋找可用的資源類型。
- api_version （必要）：Kubernetes 資源的 API 版本 （例如 'v1'、'apps/v1'、'networking.k8s.io/v1')。使用 `list_api_versions`工具尋找可用的 API 版本。
- field_selector （選用）：篩選資源的欄位選擇器 （例如，'metadata.name=my-pod、status.phase=Running')。使用與 kubectl 的 --field-selector 旗標相同的語法。
- label_selector （選用）：用於篩選資源的標籤選擇器 （例如，'app=nginx，tier=frontend')。使用與 kubectl `--selector`旗標相同的語法。
- 命名空間 （選用）：要列出的 Kubernetes 資源命名空間。如果未提供，則會在所有命名空間中列出資源 （適用於命名空間資源）。

read_k8s_resource

擷取 EKS 叢集中特定 Kubernetes 資源的詳細資訊。

參數：

- api_version （必要）：Kubernetes 資源的 API 版本 （例如 "v1"、"apps/v1"、"networking.k8s.io/v1")。
- cluster_name （必要）：資源所在的 EKS 叢集名稱。
- kind （必要）：Kubernetes 資源的類型 （例如 "Pod"、"Service"、"Deployment")。
- name （必要）：要讀取的 Kubernetes 資源名稱。
- 命名空間 （選用）：Kubernetes 資源的命名空間。命名空間資源的必要項目。叢集範圍的資源 （例如 Nodes、PersistentVolumes) 不需要。

generate_app_manifest

產生容器化應用程式的標準化 Kubernetes 部署和服務資訊清單。

參數：

- app_name （必要）：應用程式的名稱。用於部署和服務名稱，以及用於標籤。
- image_uri （必要）：帶有標籤的完整 ECR 映像 URI （例如 123456789012.dkr.ecr.region.amazonaws.com/repo:tag)。必須包含完整的儲存庫路徑和標籤。
- load_balancer_scheme （選用）： AWS load balancer scheme。有效值：

- 「內部」（僅限私有 VPC)
- 「面向網際網路」（公有存取）。
- 預設：「內部」。
- cpu （選用）：每個容器的 CPU 請求 （例如，0.1 CPU 核心為 "100m"，半核心為 "500m")。預設："100m"。
- memory （選用）：每個容器的記憶體請求 （例如，128 MiB 為 "128Mi"，1 GiB 為 "1Gi")。預設："128Mi"
- 命名空間 （選用）：要部署應用程式的 Kubernetes 命名空間。預設：「預設」。
- 連接埠 （選用）：應用程式接聽的容器連接埠。預設：80
- 複本 （選用）：要部署的複本數量。預設：2

get_cloudwatch_logs

查詢 CloudWatch 日誌，根據輸入參數進行篩選，並支援用於 EKS 叢集可觀測性的標準日誌群組。

參數：

- cluster_name （必要）：資源所在的 EKS 叢集名稱。用來建構 CloudWatch 日誌群組名稱。
- resource_type （必要）：要搜尋日誌的資源類型。有效值：+"pod"、"node"、"container"、"cluster"。這會決定如何篩選日誌。
- log_type （必要）：要查詢的日誌類型。有效值：
- 「應用程式」：容器/應用程式日誌
- "host"：節點層級系統日誌
- "performance"：績效指標日誌
- "control-plane"：EKS 控制平面日誌
- "your-log-group-name"：直接提供自訂 CloudWatch 日誌群組名稱。
- resource_name （選用）：在日誌訊息中搜尋的資源名稱 （例如 Pod 名稱、節點名稱、容器名稱）。用來篩選特定資源的日誌。
- 分鐘 （選用）：要回顧日誌的分鐘數。預設：15。如果提供 start_time，則忽略。針對最近的問題使用較小的值，針對歷史分析使用較大的值。
- start_time （選用）：ISO 格式的開始時間 （例如 "2023-01-01T00：00：00Z")。如果提供， 會覆寫分鐘參數。
- end_time （選用）：ISO 格式的結束時間 （例如 "2023-01-01T01：00：00Z")。如果未提供，則 預設為目前時間。

- 欄位（選用）：要包含在查詢結果中的自訂欄位（預設為 "@timestamp， @message"）。使用 CloudWatch Logs Insights 欄位語法。
- filter_pattern（選用）：要套用的其他 CloudWatch Logs 篩選條件模式。使用 CloudWatch Logs Insights 語法（例如 "ERROR"、"field=value"）。
- limit（選用）：要傳回的日誌項目數目上限。對於更快的查詢，請使用較低的值 (10-50)，對於更全面的結果，請使用較高的值 (100-1000)。數值越大，越可能影響效能。

get_cloudwatch_metrics

擷取 EKS 叢集監控和效能分析的 CloudWatch 指標和資料點。處理 Container Insights 指標、自訂指標，以及可設定的時段和維度。

參數：

- cluster_name（必要）：要取得指標的 EKS 叢集名稱。
- 維度（必要）：用於 CloudWatch 指標查詢的維度做為 JSON 字串。必須包含資源類型和指標的適當維度（例如 '{"ClusterName"： "my-cluster"， "PodName"： "my-pod"， "Namespace"： "default"}'）。
- metric_name（必要）：要擷取的指標名稱。常見範例：
- `cpu_usage_total`：CPU 總用量
- `memory_rss`：常駐設定大小記憶體用量
- `network_rx_bytes`：已接收網路位元組
- `network_tx_bytes`：傳輸的網路位元組
- 命名空間（必要）：存放指標的 CloudWatch 命名空間。常見值：
- `"ContainerInsights"`：適用於容器指標
- `"AWS/EC2"`：適用於 EC2 執行個體指標
- `"AWS/EKS"`：適用於 EKS 控制平面指標
- 分鐘（選用）：要回顧指標的分鐘數。預設：15。如果提供 start_time，則忽略。
- start_time（選用）：ISO 格式的開始時間（例如 "2023-01-01T00：00：00Z"）。如果提供， 會覆寫分鐘參數。
- end_time（選用）：ISO 格式的結束時間（例如 "2023-01-01T01：00：00Z"）。如果未提供，則 預設為目前時間。
- limit（選用）：要傳回的資料點數目上限。較高的值 (100-1000) 提供更精細的資料，但可能會影響效能。預設：50。

- period （選用）：指標資料點的期間，以秒為單位。預設：60 (1 分鐘）。較低的值 (1-60) 提供更高的解析度，但可能較少。
- stat （選用）：用於指標彙總的統計資料。預設：「平均」。有效值：
- `Average`：期間內的平均值
- `Sum`：期間內的總值
- `Maximum`：在此期間的最大值
- `Minimum`：期間內的最小值
- `SampleCount`：期間內的樣本數量。

get_eks_metrics_guidance

取得 EKS 叢集中特定資源類型的 CloudWatch 指標指引。在判斷要與 get_cloudwatch_metrics 工具搭配使用的正確維度時，對代理程式很有用。

參數：

- resource_type （必要）：取得 指標的資源類型 （叢集、節點、Pod、命名空間、服務）。

get_policies_for_role

擷取連接至指定 IAM 角色的所有政策，包括擔任角色政策、受管政策和內嵌政策。

參數：

- role_name （必要）：要取得政策的 IAM 角色名稱。角色必須存在於您的帳戶中 AWS 。

完整存取 （寫入） 工具

本節說明 EKS MCP 伺服器可用的完整存取 （寫入） 工具。請注意，（截至「今天」) 所有寫入 Kubernetes API 操作只能存取：

- 公有叢集 (endpointPublicAccess=true)

manage_k8s_resource

使用寫入操作 （建立、更新、修補或刪除） 管理單一 Kubernetes 資源。

參數：

- 操作（必要）：在資源上執行的操作。有效值：
- `create`: 建立新的資源
- `replace`：取代現有的資源
- `patch`：更新現有資源的特定欄位
- `delete`：刪除現有資源
- 注意：使用 read_k8s_resource 讀取資源，並使用 list_k8s_resources 列出多個資源。
- cluster_name（必要）：資源所在或將要建立的 EKS 叢集名稱。
- kind（必要）：Kubernetes 資源的類型（例如 "Pod"、"Service"、"Deployment")。
- api_version（必要）：Kubernetes 資源的 API 版本（例如 "v1"、"apps/v1"、"networking.k8s.io/v1")。
- 內文（選用）：資源定義為 JSON 字串。建立、取代和修補操作時需要。對於建立和取代，這應該是完整的資源定義。對於修補程式，這應該只包含要更新的欄位。
- name（選用）：Kubernetes 資源的名稱。除了建立（可在內文中指定）之外的所有操作都需要。
- 命名空間（選用）：Kubernetes 資源的命名空間。命名空間資源的必要項目。叢集範圍的資源（例如 Nodes、PersistentVolumes) 不需要。

apply_yaml

將 Kubernetes YAML 資訊清單套用至 EKS 叢集。

參數：

- cluster_name（必要）：建立或更新資源的 EKS 叢集名稱。
- 命名空間（必要）：要套用資源的 Kubernetes 命名空間。將用於未指定命名空間的命名空間資源。
- yaml_content（必要）：要套用至叢集的 YAML 內容。可以包含多個以 '---' 分隔的文件。
- force（選用）：是否更新已存在的資源（類似套用 kubectl)。設定為 false 僅建立新資源。

manage_eks_stacks

使用產生範本、部署、描述和刪除 EKS 叢集及其基礎基礎設施的操作來管理 EKS CloudFormation 堆疊。叢集建立通常需要 15-20 分鐘才能完成。對於部署和刪除操作，堆疊必須由此工具建立（即使用 CreatedBy=EksMcpServer 標記）。

參數：

- cluster_name （必要）：EKS 叢集的名稱（用於產生、部署、描述和刪除操作）。此名稱將用於衍生 CloudFormation 堆疊名稱，並將內嵌在叢集資源中。
- 操作（必要）：要執行的操作。有效值：
- `generate`：產生 CloudFormation 範本
- `deploy`：部署 CloudFormation 堆疊（需要 `template_content`)
- `describe`：描述/讀取 CloudFormation 堆疊（唯讀）
- `delete`：刪除 CloudFormation 堆疊
- template_content （選用）：CloudFormation 範本內容（用於部署操作）。這應該是完整的 YAML 或 JSON 範本內容。支援以 '---' 分隔的單一資源和多文件 YAML 內容。

add_inline_policy

將新的內嵌政策新增至 IAM 角色。

參數：

- 許可（必要）：將政策中包含為代表 IAM 政策陳述式之 JSON 字串的許可。可以是單一 JSON 字串或 JSON 字串陣列。
- policy_name （必要）：要建立的內嵌政策名稱。在角色中必須是唯一的。
- role_name （必要）：要新增政策的 IAM 角色名稱。角色必須存在。

在 Amazon EKS 主控台上使用 Amazon Q Developer

Amazon Elastic Kubernetes Service (EKS) 與 Amazon Q 整合，可直接在 AWS 管理主控台中提供採用 AI 技術的故障診斷。此整合可協助您透過 Amazon Q 的 AI 協助，快速調查和解決叢集、控制平面、節點和工作負載問題。

運作方式

Amazon EKS 主控台會以情境方式顯示 Inspect with Amazon Q 按鈕，以及整個主控台的錯誤或問題。當您按一下此按鈕時，Amazon Q 會自動分析問題，並在主控台右側開啟聊天面板，其中包含調查結果、根本原因分析和建議的緩解步驟。

整合會顯示在 Amazon EKS 主控台中的下列位置：

- 叢集運作狀態 - 在可觀測性儀表板的叢集運作狀態索引標籤中調查叢集運作狀態問題和狀態訊息
- 控制平面 - 對可觀測性儀表板中控制平面監控中的控制平面元件錯誤和警告進行故障診斷

- 升級洞見 - 在可觀測性儀表板的升級洞見中分析潛在的升級封鎖程式和相容性問題
- 節點運作狀態 - 在可觀測性儀表板中調查會影響節點運作狀態問題中叢集容量的節點層級問題
- 工作負載 - 分析 Pod 上的 Kubernetes 事件，指出失敗或問題

使用 Amazon Q 進行疑難排解

調查 Amazon Q 的問題

1. 開啟 Amazon EKS 主控台。
2. 選擇要調查的叢集名稱。
3. 當您遇到錯誤訊息或問題指標時，請尋找使用 Amazon Q 檢查按鈕。按鈕會在問題旁或在錯誤狀態詳細資訊檢視中以情境方式顯示。
4. 選擇使用 Amazon Q 檢查。
5. Amazon Q 會自動調查問題，並在主控台右側的聊天面板中顯示分析。
6. 檢閱調查結果，包括根本原因分析和建議的緩解步驟。
7. 您可以詢問 Amazon Q 後續問題來繼續對話。

注意 Amazon Q 整合只會在出現需要調查的錯誤、警告或問題時顯示。當資源運作狀態良好時不會顯示。

考量事項

搭配使用 Amazon Q 與 Amazon EKS 時，請考慮下列事項：

- 唯讀操作 - Amazon Q 整合只會對叢集資源執行讀取操作。它不會對叢集組態或工作負載進行任何變動或寫入動作。
- 跨區域處理 - Amazon Q 可能會跨 AWS 區域處理資料，以提供 AI 驅動的分析。如需跨區域處理的詳細資訊，請參閱《Amazon Q 開發人員使用者指南》中的跨區域處理。
- AWS 僅限 管理主控台 - 此整合只能透過 AWS 管理主控台使用。它無法透過 AWS CLI、 AWS APIs 或基礎設施做為程式碼工具使用。

進一步了解

如需使用 Amazon Q 的詳細資訊，請參閱《Amazon Q 開發人員使用者指南》中的與 Amazon Q 聊天。

Amazon EKS 上的版本控制

這部分內容旨在協助您了解 Kubernetes 版本控制在 Amazon Elastic Kubernetes Service (EKS) 的運作方式。您可尋找支援版本、我們的棄用政策及支援政策、升級程序，及管理叢集版本的重要考量等相關詳細資訊。

主題

- 了解 EKS 上的 Kubernetes 版本生命週期
- 檢視每個 Kubernetes 版本的 Amazon EKS 平台版本

了解 EKS 上的 Kubernetes 版本生命週期

> **Tip**
>
> 註冊即將舉行的 Amazon EKS 研討會。

透過新功能，設計更新和錯誤修復，Kubernetes 得以快速進化。社群平均每四個月發行一次新的 Kubernetes 次要版本 (例如 `1.36`)。Amazon EKS 遵循次要版本的上游發行和停用週期。當 Amazon EKS 中提供新的 Kubernetes 版本時，建議您主動更新叢集，以便使用最新的可用版本。

在發布後的前 14 個月內，次要版本在 Amazon EKS 受到標準支援。一旦過了標準支援日期，後續 12 個月版本就會進入延長支援。延長支援可讓您持續使用特定 Kubernetes 版本更長時間，並依叢集小時支付額外費用。如果您在延長支援期間結束之前尚未更新叢集，您的叢集會自動升級至最舊的目前支援的延伸版本。

根據預設，已啟用延長支援。若要停用，請參閱停用 EKS 延長支援。

建議使用 Amazon EKS 支援的最新推出 Kubernetes 版本建立您的叢集。如果您的應用程式需要特定 Kubernetes 版本，您可以選取較舊的版本。您可以在標準或延長支援中提供的任何版本建立新的 Amazon EKS 叢集。

標準支援的可用版本

Amazon EKS 標準支援目前提供下列 Kubernetes 版本：

- 1.36
- 1.35
- 1.34

有關標準支援每個版本需要注意的重要變更，請參閱 Kubernetes 版本標準支援。

延長支援的可用版本

Amazon EKS 延長支援目前提供下列 Kubernetes 版本：

- 1.33
- 1.32
- 1.31

有關延長支援每個版本需要注意的重要變更，請參閱 Kubernetes 版本延長支援。

版本轉返

透過就地升級升級升級的叢集可以在升級完成的 7 天內復原至先前的次要版本。若要復原至延伸支援中的版本，您必須先將叢集的升級政策變更為 EXTENDED。在延伸支援結束時自動升級的叢集無法復原。如需詳細資訊，請參閱the section called “復原 Kubernetes 版本”。

Amazon EKS Kubernetes 發佈日曆

下表顯示每個 Kubernetes 版本需要考慮的重要發行和支援日期。在 UTC+0 時區中，延長支援的計費從版本達到標準支援結束當日開始計算。下表中的日期使用 UTC+0 時區。

> **Note**
>
> 只有月份和年份的日期是近似值，並會在已知確切日期時進行更新。

如需接收有關此特定文件頁面所有來源檔案變更的通知，您可透過 RSS 閱讀器來訂閱以下 URL：

```
https://github.com/awsdocs/amazon-eks-user-guide/commits/mainline/latest/ug/versioning/kubernetes-versions.adoc.atom
```

Kubernetes 版本	上游發佈	Amazon EKS 發佈	標準支援結束	延長支援結束
1.36	2026 年 4 月 22 日	2026 年 6 月 2 日	2027 年 8 月 2 日	2028 年 8 月 2 日
1.35	2025 年 12 月 17 日	2026 年 1 月 27 日	2027 年 3 月 27 日	2028 年 3 月 27 日
1.34	2025 年 8 月 27 日	2025 年 10 月 2 日	2026 年 12 月 2 日	2027 年 12 月 2 日
1.33	2025 年 4 月 23 日	2025 年 5 月 29 日	2026 年 7 月 29 日	2027 年 7 月 29 日

Kubernetes 版本	上游發佈	Amazon EKS 發佈	標準支援結束	延長支援結束
1.32	2024 年 12 月 11 日	2025 年 1 月 23 日	2026 年 3 月 23 日	2027 年 3 月 23 日
1.31	2024 年 8 月 13 日	2024 年 9 月 26 日	2025 年 11 月 26 日	2026 年 11 月 26 日

使用 CLI AWS 取得版本資訊

您可以使用 AWS CLI 來取得 EKS 上可用 Kubernetes 版本的相關資訊，例如標準支援的結束日期。

使用 CLI 擷取 EKS 上可用 Kubernetes AWS 版本的相關資訊

1. 開啟終端。
2. 確保您已安裝並設定 AWS CLI。如需詳細資訊，請參閱安裝或更新至最新版本的 CLI。
3. 執行以下命令：

```
aws eks describe-cluster-versions
```

4. 該命令會傳回 JSON 輸出，其中包含可用叢集版本的詳細資訊。以下是輸出的範例：

```
{
    "clusterVersions": [
        {
            "clusterVersion": "1.31",
            "clusterType": "eks",
            "defaultPlatformVersion": "eks.21",
            "defaultVersion": true,
            "releaseDate": "2024-09-25T17:00:00-07:00",
            "endOfStandardSupportDate": "2025-11-25T16:00:00-08:00",
            "endOfExtendedSupportDate": "2026-11-25T16:00:00-08:00",
            "status": "STANDARD_SUPPORT",
            "kubernetesPatchVersion": "1.31.3"
        }
    ]
}
```

輸出會為每個叢集版本提供下列資訊：

- `clusterVersion`：EKS 叢集的 Kubernetes 版本
- `clusterType`：叢集的類型 (例如 "eks")
- `defaultPlatformVersion`：預設的 EKS 平台版本
- `defaultVersion`：這是否是預設版本
- `releaseDate`：此版本的發布日期
- `endOfStandardSupportDate`：標準支援的結束日期
- `endOfExtendedSupportDate`：延長支援的結束日期
- `status`：該版本的目前支援狀態，例如 `STANDARD_SUPPORT` 或 `EXTENDED_SUPPORT`
- `kubernetesPatchVersion`：特定的 Kubernetes 修補程式版本

Amazon EKS 版常見問題

標準支援有多少個 Kubernetes 版本？

在符合適用於 Kubernetes 版本的 Kubernetes 社群支援方面，Amazon EKS 致力於在任何指定時間都能為至少三種 Kubernetes 版本提供支援。我們將至少提前 60 天宣布特定 Kubernetes 次要版本的標準支援結束日期。基於 Amazon EKS 資格授予和新 Kubernetes 版本的發布程序，Kubernetes 版本的標準支援結束日期將為 Amazon EKS 上將在 Kubernetes 專案不再支援上游版本當天或之後。

Kubernetes 版本獲得 Amazon EKS 標準支援的時間有多長？

Kubernetes 版本在首次在 Amazon EKS 上提供 14 個月後獲得標準支援。即使上游 Kubernetes 不再支援 Amazon EKS 上可用的版本，也是如此。我們向後移植適用於 Amazon EKS 所支援的 Kubernetes 版本的安全修補程式。

Amazon EKS 上 Kubernetes 版本的標準支援即將結束時，是否會通知我？

是。如果您帳戶中的任何叢集正在執行即將終止支援的版本，Amazon EKS 會在 Kubernetes 版本在 Amazon EKS 上發佈約 12 個月後，透過 AWS 運作狀態儀表板傳送通知。此通知包括支援終止的日期。此日期距通知發出日期起至少 60 天。

Amazon EKS 支援哪些 Kubernetes 功能？

Amazon EKS 支援所有 Kubernetes API 一般可用 (GA) 功能。依預設，不會在叢集中啟用新的 Beta API。然而，依預設，仍會繼續啟用先前的 Beta API 和新版本的現有 Beta API。Alpha 功能不受支援。

Amazon EKS 受管節點群組是否與叢集控制平面版本一起自動更新？

不受管理節點群組會在您的帳戶建立 Amazon EC2 執行個體。當您或 Amazon EKS 更新控制平面時，這些執行個體不會自動升級。如需詳細資訊，請參閱the section called “更新”。建議您在控制平面和節點上維持相同的 Kubernetes 版本。

自我管理節點群組是否與叢集控制平面版本一起自動更新？

不會。自我管理的節點群組包含您帳戶的 Amazon EC2 執行個體。當您或 Amazon EKS 代您更新控制平面版本時，這些執行個體不會自動升級。自我管理節點群組在主控台中沒有任何需要更新的指示。您可以檢視安裝在節點上的 `kubelet` 版本，方法是選取您叢集的 Overview (概觀) 標籤上的 Nodes (節點) 清單上的節點，以判斷哪些節點需要更新。您必須手動更新節點。如需詳細資訊，請參閱the section called “更新方法”。

Kubernetes 專案可測試控制平面和節點之間的相容性，最多三個次要版本。例如，協調工作是由 `1.36` 控制平面進行時，`1.33` 節點將可繼續操作。不過，不建議執行節點持續位於控制平面後面三個次要版本的叢集。如需詳細資訊，請參閱 Kubernetes 文件中的 Kubernetes 版本和版本偏移支援政策。建議您在控制平面和節點上維持相同的 Kubernetes 版本。

在 Fargate 上執行的 Pod 是否會透過自動叢集控制平面版本升級而自動升級？

不。我們強烈建議執行 Fargate Pod 作為複製控制器的一部分，例如 Kubernetes 部署。然後對所有 Fargate Pod 進行滾動式重新啟動。新版本的 Fargate Pod 部署了 `kubelet` 版本，其版本與您更新的叢集控制平面版本相同。如需詳細資訊，請參閱 Kubernetes 文件中的部署。

> **Important**
>
> 如果您更新控制平面，您仍必須自行更新 Fargate 節點。若要更新 Fargate 節點，請刪除節點所代表的 Fargate Pod，然後重新部署 Pod。新的 Pod 部署了 `kubelet` 版本，其版本與您叢集的版本相同。

混合節點會支援哪些 Kubernetes 版本？

Amazon EKS 混合節點支援與具有其他節點運算類型的 Amazon EKS 叢集相同的 Kubernetes 版本，包括標準和延長 Kubernetes 版本支援。升級控制平面版本時，混合節點不會自動升級，並且您要負責升級混合節點。如需詳細資訊，請參閱the section called “升級混合節點”。

Amazon EKS 延長支援常見問答集

標準支援和延伸支援術語對我來說是新的。這些術語是什麼意思？

Amazon EKS 中對 Kubernetes 版本的標準支援會在 Amazon EKS 發行 Kubernetes 版本時開始，並且在發布日期後 14 個月結束。對 Kubernetes 版本的延長支援將在標準支援結束後立即開始，並在接下來的 12 個月後結束。例如，Amazon EKS 版本 `1.23` 的標準支援已於 2023 年 10 月 11 日結束。對版本 `1.23` 的延長支援已於 2023 年 10 月 12 日開始，並於 2024 年 10 月 11 日結束。

我需要做什麼才能取得 Amazon EKS 叢集的延伸支援？

您需要將叢集升級政策變更為「延長」，以啟用叢集的延長支援 (請參閱 EKS 延長支援)。根據預設，除非另有指定，否則對於所有新的和現有的叢集，升級政策都已設定為「延長」。請參閱叢集升級政策，以檢視叢集的升級政策。標準支援將在 Amazon EKS 發行 Kubernetes 版本時開始，並且在發布日期後 14 個月結束。對 Kubernetes 版本的延長支援將在標準支援結束後立即開始，並在接下來的 12 個月後結束。

我可以針對哪些 Kubernetes 版本獲得延長支援？

在該版本的標準支援結束後，您可以在任何版本執行叢集，長達 12 個月。這代表每個版本將在 Amazon EKS 支援 26 個月（14 個月的標準支援加 12 個月的延長支援）。

如果我不想使用延伸支援該怎麼辦？

如果您不想自動註冊延長支援，您可以將叢集升級到標準 Amazon EKS 支援的 Kubernetes 版本。若要停用延長支援，請參閱停用 EKS 延長支援。注意：如果您停用延長支援，則在標準支援結束時，您的叢集將會自動升級。

延長支援 12 個月結束時會發生什麼情況？

對於已完成 26 個月生命週期的 Kubernetes 版本，在其中執行的叢集 (14 個月的標準支援加 12 個月的延長支援) 將自動升級到下一版本。自動升級僅包含 Kubernetes 控制平面。如果您有 EKS 自動模式節點，它們可能會自動更新。自我管理節點和 EKS 受管節點群組將保留在先前的版本中。

在延長支援日期結束時，您就無法再使用不支援的版本來建立新的 Amazon EKS 叢集。現有的控制平面會在終止支援日期後由 Amazon EKS 透過逐步部署程序，自動將控制平面更新為最舊的支援版本。自動更新控制平面後，請務必手動更新叢集附加元件和 Amazon EC2 節點。如需詳細資訊，請參閱the section called “更新 Kubernetes 版本”。

延長支援日期結束後，我的控制平面何時會自動更新？

Amazon EKS 無法提供特定的時間範圍。於延長支援日期結束後，可隨時進行自動更新。更新之前，您不會收到任何通知。我們建議您主動更新控制平面，而不需依賴 Amazon EKS 自動更新程序。如需詳細資訊，請參閱the section called “更新 Kubernetes 版本”。

我可以無限期地將控制平面留在 Kubernetes 版本上嗎？

否。 的雲端安全性 AWS 是最高優先順序。在過去一定時間點 (通常為 1 年) 後，Kubernetes 社群就會停止發布常見的漏洞和風險修補程式 (CVE)，並且不鼓勵針對不支援版本提交 CVE。這表示可能甚至不會報告舊版 Kubernetes 特有的弱點。讓叢集暴露於漏洞之下，不會發出通知，也沒有修復選項。鑑於這一點，Amazon EKS 不允許控制平面保持在延長支援已結束的版本。

取得延長支援是否需要額外費用？

是的，在延長支援期間執行的 Amazon EKS 叢集須支付額外費用。如需定價詳細資訊，請參閱 AWS 部落格或我們的定價頁面上 Kubernetes 版本定價的 Amazon EKS 延伸支援。

延伸支援包含哪些內容？

延長支援的 Amazon EKS 叢集會持續接收 Kubernetes 控制平面的安全性修補程式。此外，Amazon EKS 將針對延長支援版本的 Amazon VPC CNI、`kube-proxy` 和 CoreDNS 附加元件發布修補程式。Amazon EKS 也會針對 AWS Amazon Linux、 Bottlerocket 和 Windows 以及這些版本的 Amazon EKS Fargate 節點，發行已發佈 Amazon EKS 最佳化 AMIs 的修補程式。延伸支援中的所有叢集將繼續從 存取技術支援 AWS。

延長支援的非 Kubernetes 元件修補程式是否有任何限制？

雖然延伸支援涵蓋來自 的所有 Kubernetes 特定元件 AWS，但它將始終僅支援 AWS針對 Amazon Linux、 Bottlerocket 和 Windows 發佈的 Amazon EKS 最佳化 AMIs。這代表您在使用延長支援時，您可能會在 Amazon EKS 最佳化 AMI 擁有更新的元件 (例如作業系統或核心)。例如，一旦 Amazon Linux 2 在 2025 年達到生命週期結束，Amazon EKS 最佳化的 Amazon Linux AMI 將使用更新的 Amazon Linux 作業系統構建。Amazon EKS 將針對每個 Kubernetes 版本宣布並記錄重要的支援生命週期差異，例如此版本。

我可以使用延長支援的版本建立新的叢集嗎？

是。

檢閱標準支援 Kubernetes 版本的版本備註

> **Tip**
>
> 註冊即將舉行的 Amazon EKS 研討會。

本主題提供標準支援的每個 Kubernetes 版本需要注意的重要變更。於升級時，請仔細檢閱叢集舊版和新版本之間發生的變更。

Kubernetes 1.36

Kubernetes 1.36 現已可在 Amazon EKS 中使用。如需 Kubernetes 1.36 的詳細資訊，請參閱官方版本公告。

Important

- gitRepo 磁碟區移除：在 Kubernetes 1.36 中永久停用磁碟`gitRepo`區類型，無法重新啟用。Kubernetes API 仍然接受具有`gitRepo`磁碟區的 Pod，但 kubelet 會拒絕執行它們並傳回錯誤。
  - 所需動作：在升級至 1.36 之前遷移至初始化容器或 git-sync 附屬容器。如需詳細資訊，請參閱 KEP-5040。
- SELinux 磁碟區標籤變更 (GA)：SELinux 磁碟區標籤現在預設為 Kubernetes 1.36 中的所有磁碟區，使用 `mount -o context` 而不是遞迴檔案重新標籤。在相同節點上的特殊權限和非特殊權限 Pod 之間共用磁碟區可能會導致問題。未來的 Kubernetes 版本可能會帶來與此功能相關的其他重大變更。
  - 必要動作：執行 SELinux 強制執行系統的客戶應稽核叢集，並確保 `seLinuxChangePolicy` 欄位和 SELinux 磁碟區標籤在 Pod 上正確設定，再升級。如需詳細資訊，請參閱 SELinux 磁碟區標籤變更進入 GA（以及 v1.37 中可能的影響）。
- 依預設啟用嚴格 IP/CIDR 驗證：功能`StrictIPCIDRValidation`閘道現在預設為針對內建 API 類型啟用。API 欄位不再接受具有外部前導零（例如，`010.000.000.005`而非 `10.0.0.5`) 的 IP 或 CIDR 值，或具有不明確語意（例如，`192.168.0.5/24`而非 `192.168.0.0/24`) 的 CIDR 值。現有的儲存物件會透過驗證追逐保留，但新建立和更新將被拒絕。這不適用於自訂資源類型。
  - 必要動作：檢閱資訊清單、Helm Chart 和包含前導零或非正式 CIDR 表示法的 IP 地址自動化。升級之前，請更新它們以使用正式格式。如需詳細資訊，請參閱 KEP-4858。

- User Namespaces (Stable)：User Namespaces 透過將容器的根使用者映射到主機上的非特殊權限使用者來提供defense-in-depth，確保容器細分不會授予節點的管理能力。
  - 如需詳細資訊，請參閱 Kubernetes 部落格上的 Kubernetes 中的使用者命名空間最終是 GA。
- 資源運作狀態 (Beta)：以 Pod 狀態報告每個裝置的運作狀態，讓操作員判斷當機迴圈是由於運作狀態不佳或不明的裝置狀態，而非應用程式問題。適用於裝置外掛程式和動態資源配置。
  - 如需詳細資訊，請參閱 KEP-4680。

- 動態資源配置功能 (Beta)：數個 DRA 功能現在預設為啟用：可分割裝置和消耗容量，用於更精細地共用 GPUs 等硬體，以及裝置繫結條件，用於在排程之前進行徹底的裝置準備度檢查。
  - 如需詳細資訊，請參閱 Kubernetes 部落格上的 Kubernetes 1.36 版：更多驅動程式、新功能和 DRA 的下一個時代。
- 棄用通知 — 服務 externalIPs：服務中的 `externalIPs` 欄位`.spec`已在 Kubernetes 1.36 中棄用。建立或更新使用此欄位的 服務時，您會看到棄用警告。Kubernetes 1.43 計劃完全移除。使用 的客戶`externalIPs`應遷移至 LoadBalancer Services、NodePort 或 Gateway API。如需詳細資訊，請參閱 KEP-5707。

如需完整的 Kubernetes `1.36`變更日誌，請參閱 https：//https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.36.md

Kubernetes 1.35

Kubernetes `1.35` 現已可在 Amazon EKS 中使用。如需 Kubernetes `1.35` 的詳細資訊，請參閱官方版本公告。

> **Important**
>
> - Cgroup v1 支援已移除：Kubernetes 1.35 取代 cgroup v1 支援，這表示 kubelet 預設會拒絕在使用 cgroup v1 的節點上啟動。
>   - AL2023：AL2023 預設使用 cgroup v2，並與 Kubernetes 上游行為保持一致。
>   - 必要動作：手動設定 AL2023 使用 cgroup v1 的客戶必須遷移至 cgroups v2 或手動`failCgroupV1: false`設定 kubelet 組態。
>   - Bottlerocket：根據預設， Bottlerocket 1.35 會使用 cgroup v2，但 kubelet 組態`failCgroupV1: false`中會設定 ，以維持回溯相容性。
>   - Fargate：Fargate 繼續使用 cgroup v1。
> - Containerd 1.x 終止支援：Kubernetes 1.35 是支援 Containerd 1.x 的最後一個版本。您必須先切換至容器化 2.0 或更新版本，才能升級至下一個 Kubernetes 版本。
> - 2026 年 3 月，上游 Kubernetes 專案將淘汰 Ingress NGINX，這是許多 Kubernetes 環境的關鍵基礎設施元件。
>   - 必要動作：EKS 客戶應評估他們是否依賴輸入 NGINX，並開始規劃遷移到閘道 API 或第三方輸入控制器等替代方案，因為淘汰後不會再發佈錯誤修正、安全修補程式或更新。現有的部署將繼續運作，但在淘汰後繼續使用 Ingress NGINX 會讓您的環境面臨安全風險，因為沒有任何可用的替代方案是直接插入取代，而且需要規劃和工程時間。如需此

Kubernetes 公告的詳細資訊，請參閱 Kubernetes 指導和安全回應委員會傳入 NGINX 淘汰的官方陳述式。

- 就地 Pod 資源更新 （穩定）：就地 Pod 資源更新可讓使用者調整 CPU 和記憶體資源，而無需重新啟動 Pod 或容器。先前，這類修改需要重新建立 Pod，這可能會中斷工作負載，特別是對於有狀態或批次的應用程式。新的就地功能允許更順暢、不中斷的垂直擴展、提高效率，也可以簡化開發。
  - 如需詳細資訊，請參閱 Kubernetes 部落格上的 Kubernetes 1.35：就地 Pod 調整大小至穩定。
- PreferSameNode 流量分佈 （穩定）：已更新 服務`trafficDistribution`的欄位，以更明確地控制流量路由。已推出新選項 `PreferSameNode`，以允許服務在可用時嚴格排定本機節點上端點的優先順序，否則會回到遠端端點。此變更可讓 API 更明確地偏好目前節點內的流量。
- StatefulSet MaxUnavailable (Beta)：此功能可透過設定 `maxUnavailable` (例如 3 或 10%) 來啟用平行 Pod 更新，讓資料庫叢集等具狀態應用程式比循序one-at-a-time更新最多快 60%，大幅減少維護時段。
- Windows Server 2025 支援：EKS 1.35 新增對 Windows Server 2025 的支援。
- Kubelet 標記移除：已從 kubelet 移除`--pod-infra-container-image`標記。自訂 AMI 使用者必須先從 kubelet 組態中移除此旗標，才能升級至 1.35。
- 棄用通知 - IPVS 模式：kube-proxy 中的 IPVS 模式已棄用，並將在 Kubernetes 1.36 中移除。

如需完整的 Kubernetes `1.35`變更日誌，請參閱 https：//https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.35.md

Kubernetes 1.34

Kubernetes `1.34` 現已可在 Amazon EKS 中使用。如需 Kubernetes `1.34` 的詳細資訊，請參閱官方版本公告。

Important

- 容器已更新至 1.34 版中的 2.1 以啟動。
  - 如果您在升級後遇到任何問題，請檢查容器化 2.1 版本備註。
- AWS 未發佈適用於 Kubernetes 1.34 的 EKS 最佳化 Amazon Linux 2 AMI。
  - AWS 鼓勵您遷移至 Amazon Linux 2023。了解如何 the section called “升級至 AL2023”。
  - 如需詳細資訊，請參閱the section called “Amazon Linux 2 AMI 棄用”。
- AppArmor 在 Kubernetes 1.34 中已棄用。

    - 我們建議您遷移到替代的容器安全解決方案，例如 seccomp 或 Pod 安全標準。
  - VolumeAttributesClass (VAC) 在 Kubernetes 1.34 中畢業至 GA，從 Beta API (`storage.k8s.io/v1beta1`) 遷移至穩定的 API (`storage.k8s.io/v1`)。
    - 如果您使用 EBS CSI 驅動程式搭配受管附屬容器 AWS（來自 ECR Gallery 上的 CSI 元件），磁碟區修改將繼續在 EKS 1.31-1.33 叢集上順暢運作。 AWS 將修補附屬項目以支援 Beta VAC APIs，直到 EKS 1.33 標準支援結束 (2026 年 7 月 29 日）。
    - 如果您自行管理 CSI 附屬容器，您可能需要鎖定至 1.34 版之前的叢集上的較舊附屬版本，以維護 VAC 功能。
    - 若要使用 GA VolumeAttributesClass 功能（例如修改轉返），請升級至 EKS 1.34 或更新版本。
  - 外部 JWT Signer for Service 帳戶權杖已提升為 Beta 版。使用外部簽署者時，不再完全遵守 --service-account-extend-token-expiration 旗標。API 伺服器會強制執行所需延伸項目 (1 年） 與外部簽署者限制 (24 小時） 之間的最短過期時間。
    - 建議使用繫結服務帳戶字符，由 Kubernetes 自動掛載和輪換。

- 動態資源配置 (DRA) 核心 APIs (GA)：動態資源配置已進入穩定狀態，可透過標準化配置界面有效率地管理 GPUs 等特殊硬體 - 簡化硬體加速器的資源管理並改善特殊資源的使用率。
- 適用於 Kubelet (Beta) 的投影 ServiceAccount 權杖：此增強功能透過使用容器映像提取的短期憑證而非長期秘密來提高安全性 - 降低憑證暴露風險並增強叢集的整體安全狀態。
- Pod 層級的資源請求和限制 (Beta)：此功能允許多容器 Pod 的共用資源集區來簡化資源管理，為具有多個容器的複雜應用程式提供更有效率的資源配置和使用率。
- 可變 CSI 節點可配置計數 (Beta)：`MutableCSINodeAllocatableCount`特徵閘道預設為在 EKS 1.34 中啟用，使 CSINode 最大可連接磁碟區計數屬性可變，並引入根據 CSI 驅動程式層級的使用者組態動態更新的機制。這些更新可以透過定期間隔或故障偵測觸發，透過解決節點上報告和實際連接容量之間的不相符問題，增強具狀態 Pod 排程的可靠性。
  - 如需詳細資訊，請參閱 Kubernetes 部落格上的 Kubernetes v1.34：Mutable CSI Node Allocatable Count。
- 棄用通知 - cgroup 驅動程式組態：手動 cgroup 驅動程式組態已棄用，有利於自動偵測。
  - 客戶影響：如果您目前在 kubelet 組態中手動設定`--cgroup-driver`旗標，您應該準備移除此組態。
  - 必要動作：計劃更新節點引導指令碼和自訂 AMI 組態，以在功能在未來 Kubernetes 版本中移除之前移除手動 cgroup 驅動程式設定。
  - 如需詳細資訊，請參閱 cgroup 驅動程式文件。

如需完整的 Kubernetes 1.34變更日誌，請參閱 https：//https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.34.md

檢閱延長支援 Kubernetes 版本的版本備註

Amazon EKS 對 Kubernetes 版本的支援時間比對上游版本的支援時間更長，對 Kubernetes 次要版本的標準支援時間為自其在 Amazon EKS 發布之日起 14 個月，而對 Kubernetes 次要版本的延長支援則額外增加 12 個月 (每個版本總計 26 個月)。

本主題提供延長支援的每個 Kubernetes 版本需要注意的重要變更。於升級時，請仔細檢閱叢集舊版和新版本之間發生的變更。

> **Note**
>
> 如果您從標準支援下的版本轉返至延伸支援下的版本，則該叢集的延伸支援費用會繼續。

Kubernetes 1.33

Kubernetes 1.33 現已可在 Amazon EKS 中使用。如需 Kubernetes 1.33 的詳細資訊，請參閱官方版本公告。

> **Important**
>
> - 動態資源配置 Beta 版 Kubernetes API 現已啟用。
>   - 此 Beta 版 API 可改善排程和監控需要 GPU 等資源的工作負載的體驗。
>   - Beta 版 API 由 Kubernetes 社群所定義，且 Kubernetes 的未來版本可能會有所變更。
>   - 仔細檢閱 Kubernetes 文件中的功能階段，了解使用 Beta 版 API 的影響。
> - AWS 未發佈適用於 Kubernetes 1.33 的 EKS 最佳化 Amazon Linux 2 AMI。
>   - AWS 鼓勵您遷移至 Amazon Linux 2023。了解如何 the section called “升級至 AL2023”。
>   - 如需詳細資訊，請參閱the section called “Amazon Linux 2 AMI 棄用”。

- 就地 Pod 資源調整大小 (Beta)：Kubernetes 1.33 將就地資源調整大小提升為 Beta。這允許在不重新啟動的情況下動態更新現有 Pod 的 CPU 和記憶體資源，以零停機時間的方式垂直擴展具狀態的工作負載。

- 附屬容器現在穩定：附屬容器已漸進到穩定。它們會實作為特殊初始化容器`restartPolicy: Always`，其會在應用程式容器之前開始，在整個 Pod 生命週期中執行，並支援用於操作狀態訊號的探查。
  - 如需詳細資訊，請參閱 Kubernetes 文件中的附屬容器。
- 端點 API 棄用：端點 API 現在已正式棄用，並在存取時傳回警告 - 改為遷移工作負載和指令碼以使用 EndpointSlices API，以支援雙堆疊聯網等現代功能，並處理每個服務的多個 EndpointSlices。
  - 如需詳細資訊，請參閱 Kubernetes 部落格上的 Kubernetes v1.33：繼續從端點轉換到 EndpointSlice。
- Elastic Fabric Adapter 支援：Amazon EKS 叢集的預設安全群組現在支援 Elastic Fabric Adapter (EFA) 流量。預設的安全群組具有新的傳出規則，該規則允許具有相同安全群組目的地的 EFA 流量。這允許叢集內的 EFA 流量。
  - 如需詳細資訊，請參閱《Amazon Elastic Compute Cloud 使用者指南》中的 Amazon EC2 上適用於 AI/ML 與 HPC 工作負載的 Elastic Fabric Adapter。

如需完整的 Kubernetes `1.33`變更日誌，請參閱 https：//https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.33.md

Kubernetes 1.32

Kubernetes `1.32` 現已可在 Amazon EKS 中使用。如需 Kubernetes `1.32` 的詳細資訊，請參閱官方版本公告。

> **Important**
>
> - FlowSchema 和 PriorityLevelConfiguration 的 `flowcontrol.apiserver.k8s.io/v1beta3` API 版本已在版本 `1.32` 中移除。如果您正在使用這些 API，則必須在升級之前更新組態以便使用最新的支援版本。
> - 版本 `1.32` 中已棄用 ServiceAccount `metadata.annotations[kubernetes.io/enforce-mountable-secrets]`，且未來的 Kubernetes 次要版本也會將其移除。建議使用不同的命名空間來隔離對已掛載秘密的存取。
> - Kubernetes 版本 `1.32` 是 Amazon EKS 將發行的 Amazon Linux 2 (AL2) AMI 的最新版本。從版本 `1.33` 開始，Amazon EKS 將繼續發行 Amazon Linux 2023 (AL2023) 和 Bottlerocket 型 AMI。

- 在 Kubernetes 版本 1.32 中，Memory Manager 功能已逐步升級為正式上市 (GA) 狀態。此增強功能可為容器化應用程式提供更高效且可預測的記憶體配置，特別有益於具有特定記憶體需求的工作負載。
- StatefulSets 建立的 PersistentVolumeClaims (PVC) 現在包含自動清除功能。當不再需要 PVC 時，系統會自動將其刪除，同時在 StatefulSet 更新和節點維護操作期間，維護資料持續性。此功能可簡化儲存管理，並有助於防止叢集中的孤立 PVC。
- 自訂資源欄位選取器功能現已推出，讓開發人員能夠將欄位選取器新增至自訂資源。此功能為自訂資源提供與內建 Kubernetes 物件相同的篩選功能，從而實現更精確且更高效的資源篩選，並促進更優質的 API 設計實務。

如需完整的 Kubernetes 1.32變更日誌，請參閱 https：//https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.32.md

匿名身分驗證變更

從 Amazon EKS 1.32 開始，匿名身分驗證僅限於下列 API 伺服器運作狀態檢查端點：

- /healthz
- /livez
- /readyz

使用 system:unauthenticated 使用者而向任何其他端點提出的請求將會收到 401 Unauthorized HTTP 回應。此安全增強功能有助於避免因設定錯誤的 RBAC 政策而可能發生的意外叢集存取。

> **Note**
>
> public-info-viewer RBAC 角色會繼續套用至上文所列的運作狀態檢查端點。

Amazon Linux 2 AMI 棄用

Kubernetes 版本1.32是 Amazon EKS 發行 AL2 AMIs最後一個版本。從版本 1.33 開始，Amazon EKS 將繼續發行 AL2023 和 Bottlerocket 型 AMI。如需詳細資訊，請參閱the section called “AL2 AMI 棄用”。

Kubernetes 1.31

Kubernetes 1.31 現已可在 Amazon EKS 中使用。如需 Kubernetes 1.31 的詳細資訊，請參閱官方版本公告。

> **Important**
>
> - 自 2017 年起棄用的 kubelet 旗標 `--keep-terminated-pod-volumes` 已作為版本 1.31 發行的一部分予以移除。此變更會影響 kubelet 處理終止的 Pod 磁碟區的方式。如果您在節點組態中使用此旗標，則必須更新引導指令碼和啟動範本，以便在升級之前將其移除。

- Amazon EKS 版本 1.31 中啟用了 Beta 版 `VolumeAttributesClass` 功能閘道和 API 資源。此功能允許叢集運算子修改由相容 CSI 驅動程式管理的持久性磁碟區 (PV) 的可變屬性，包括 Amazon EBS CSI 驅動程式。若要利用此功能，請確保您的 CSI 驅動程式支援 `VolumeAttributesClass` 功能 (對於 Amazon EBS CSI 驅動程式，請升級至版本 `1.35.0` 或更新版本，以自動啟用此功能)。您將能夠建立 `VolumeAttributesClass` 物件來定義所需的磁碟區屬性，例如磁碟區類型和輸送量，並將其與您的持久性磁碟區宣告 (PVC) 建立關聯。如需詳細資訊，請參閱官方 Kubernetes 文件以及 CSI 驅動程式的文件。
  - 如需有關 Amazon EBS CSI 驅動程式的詳細資訊，請參閱 the section called “Amazon EBS”。
- AppArmor 的 Kubernetes 支援已逐步進入穩定狀態，並且現已正式推出，可供大眾使用。此功能可讓您在容器的 `securityContext` 中設定 `appArmorProfile.type` 欄位，以使用 AppArmor 保護您的容器。Kubernetes 版本 `1.30` 之前的版本，AppArmor 由註釋所控制。從版本 `1.30` 開始，其使用欄位進行控制。若要利用此功能，建議您從註釋移轉並使用 `appArmorProfile.type` 欄位，以確保您的工作負載相容。
- PersistentVolume 上一階段轉換時間功能已逐步進入穩定狀態，並且現已在 Kubernetes 版本 `1.31` 中正式推出，可供大眾使用。此功能會在 PersistentVolumeStatus 中引入新欄位 `.status.lastTransitionTime`，而該欄位可提供 PersistentVolume 上次轉換至不同階段的時間戳記。此增強功能可以更好地追蹤和管理 PersistentVolumes，尤其是在了解磁碟區的生命週期至關重要的情況下。

如需完整的 Kubernetes `1.31`變更日誌，請參閱 https：//https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.31.md

檢視目前的叢集支援時段

AWS 主控台的叢集支援時段區段指示叢集目前為標準支援還是延長支援。若叢集支援時段為延長支援，則需要向您收取 EKS 延長支援費用。

若要了解標準支援與延長支援的相關詳細資訊，請參閱了解 EKS 上的 Kubernetes 版本生命週期。

1. 請導覽至 AWS 主控台的 EKS 區段中的叢集頁面。請確認主控台的 AWS 區域設定與您想要檢閱的叢集相同。
2. 檢閱支援時段欄。若值為...之前為標準支援，則目前不會向您收取延長支援費用。目前為標準支援時段。若值為延長支援...，則目前需要對此叢集收取延長支援費用。

> **Note**
>
> 不能透過 AWS API 或 CLI 擷取支援時段。

檢視目前的叢集升級政策

叢集升級政策確定叢集標準支援時段過後會發生的情況。若升級政策為 `EXTENDED`，叢集不會自動升級，並且會進入延長支援。若升級政策為 `STANDARD`，叢集會自動升級。

Kubernetes 版本政策的 Amazon EKS 控制項可讓您選擇 EKS 叢集的標準支援行為結束。藉由這些控制項，您可決定哪些叢集應進入延長支援，哪些叢集應在適用於 Kubernetes 版本的標準支援結束時自動升級。

在發布後的前 14 個月內，次要版本在 Amazon EKS 受到標準支援。一旦過了標準支援日期，後續 12 個月版本就會進入延長支援。延長支援可讓您持續使用特定 Kubernetes 版本更長時間，並依叢集小時支付額外費用。您可以啟用或停用 EKS 叢集的延伸支援。如果您停用延伸支援， AWS 會在標準支援結束時自動將您的叢集升級至下一個版本。若您啟用延長支援，可在有限時段內支付額外付費來保持目前的版本。即使您使用延長支援，也請規劃定期升級 Kubernetes 叢集。

您可利用 `supportType` 屬性，為新的和現有的叢集設定版本政策。可使用兩個選項來設定版本支援政策：

- **`STANDARD`** — 您的 EKS 叢集有資格在標準支援結束時自動升級。使用此設定不會產生延長支援費用，但您的 EKS 叢集會在標準支援中自動升級至下一個支援的 Kubernetes 版本。

- **EXTENDED** ：Kubernetes 版本到達標準支援結束時間後，EKS 叢集將進入延長支援。使用此設定會產生延長支援費用。您可將叢集升級至標準支援的 Kubernetes 版本，以免產生延長支援費用。以延長支援執行的叢集，在延長支援結束時符合自動升級資格。

新叢集與現有叢集預設會啟用延長支援。您可以檢視 中的叢集是否已啟用延伸支援 AWS 管理主控台，或使用 AWS CLI。

> **Important**
>
> 如果您希望叢集保持其目前的 Kubernetes 版本，以利用延長的支援期間，強烈建議在叢集 Kubernetes 版本的標準支援期間結束前啟用延伸支援升級政策。您可以在標準支援期間結束後啟用延伸支援。不過，如果叢集已啟動自動升級，Amazon EKS 無法保證變更會生效。

叢集進入延伸支援後，您就無法停用它。若要變更版本支援政策設定，您的叢集必須在標準支援的 Kubernetes 版本上執行。

檢視叢集升級政策AWS （主控台）

1. 導覽至 AWS 主控台 EKS 區段中的叢集頁面。確認主控台已設定為與您要檢閱的叢集相同的 AWS 區域。
2. 檢閱升級政策資料欄。若值為標準支援，叢集不會進入延長支援。若值為延長支援，叢集將進入延長支援。

檢視叢集升級政策 (AWS CLI)

1. 確認已安裝 AWS CLI 且您已登入。了解如何更新和安裝 AWS CLI。
2. 確定 EKS 叢集的名稱。將 CLI 設定為與 EKS 叢集相同的 AWS 區域。
3. 執行以下命令：

```
aws eks describe-cluster \
--name <cluster-name> \
--query "cluster.upgradePolicy.supportType"
```

4. 若值為 STANDARD，叢集不會進入延長支援。若值為 EXTENDED，叢集將進入延長支援。

啟用 EKS 延伸支援以增加規劃 Kubernetes 版本升級的彈性

本主題會說明如何設定 EKS 叢集的升級政策，以啟用延長支援。EKS 叢集的升級政策會決定叢集的標準支援期間結束時所發生的情況。如果叢集升級政策已啟用延長支援，則其會在標準支援期間結束時進入延長支援期間。標準支援期間結束時，叢集不會自動升級。

實際上，在延長支援期間，叢集會產生更高的成本。如果叢集僅將升級政策設定為啟用延長支援，並且處於標準支援期間，則會產生標準成本。

如果您在 AWS 主控台中建立叢集，則會將升級政策設定為停用延伸支援。如果您以另一種方式建立叢集，則會將升級政策設定為啟用延長支援。例如，使用 AWS API 建立的叢集已啟用延伸支援。

如需有關升級政策的詳細資訊，請參閱叢集升級政策。

> **Important**
>
> 如果您希望叢集保持其目前的 Kubernetes 版本，以利用延長的支援期間，強烈建議在叢集 Kubernetes 版本的標準支援期間結束前啟用延伸支援升級政策。您可以在標準支援期間結束後啟用延伸支援。不過，如果叢集已啟動自動升級，Amazon EKS 無法保證變更會生效。
> 叢集進入延伸支援後，您就無法停用它。您只能針對在標準支援中執行 Kubernetes 版本的叢集停用延伸支援。
> 如果您未啟用延伸支援，您的叢集將在標準支援期間結束後自動升級。

啟用 EKS 延伸支援AWS （主控台）

1. 在 AWS 主控台中導覽至您的 EKS 叢集。選取叢集資訊頁面上的概觀索引標籤。
2. 在 Kubernetes 版本設定區段中，選取管理。
3. 選取延長支援，然後選取儲存變更。

啟用 EKS 延伸支援 (AWS CLI)

1. 確認已安裝 AWS CLI 且您已登入。了解如何更新和安裝 AWS CLI。
2. 確定 EKS 叢集的名稱。
3. 執行以下命令：

```
aws eks update-cluster-config \
```

```
--name <cluster-name> \
--upgrade-policy supportType=EXTENDED
```

透過停用 EKS 延伸支援來防止叢集成本增加

本主題會說明如何設定 EKS 叢集的升級政策，以停用延長支援。EKS 叢集的升級政策會決定叢集的標準支援期間結束時所發生的情況。如果叢集升級政策已停用延長支援，則會自動升級至下一個 Kubernetes 版本。

如需有關升級政策的詳細資訊，請參閱叢集升級政策。

> **Important**
>
> 一旦您的叢集進入延伸支援，就無法停用延伸支援。您只能為處於標準支援期的叢集停用延伸支援。
> AWS 建議在標準支援期間將您的叢集升級至版本。

停用 EKS 延伸支援AWS （主控台）

1. 在 AWS 主控台中導覽至您的 EKS 叢集。選取叢集資訊頁面上的概觀索引標籤。
2. 在 Kubernetes 版本設定區段中，選取管理。
3. 選取標準支援，然後選取儲存變更。

停用 EKS 延伸支援 (AWS CLI)

1. 確認已安裝 AWS CLI 且您已登入。了解如何更新和安裝 AWS CLI。
2. 確定 EKS 叢集的名稱。
3. 執行以下命令：

```
aws eks update-cluster-config \
--name <cluster-name> \
--upgrade-policy supportType=STANDARD
```

檢視每個 Kubernetes 版本的 Amazon EKS 平台版本

Amazon EKS 平台版本代表 Amazon EKS 叢集控制平面的功能，例如啟用了哪些 Kubernetes API 伺服器旗標以及目前的 Kubernetes 修補程式版本。每個 Kubernetes 次要版本皆有一或多個相關的 Amazon EKS 平台版本。適用於不同 Kubernetes 次要版本的平台版本都是獨立的。您可以使用 CLI 或擷取叢集目前的平台版本 AWS 管理主控台。 AWS 如果您在 AWS Outposts 上有本機叢集，請參閱 the section called “EKS 平台版本”，而非本主題。

當 Amazon EKS 中提供新的 Kubernetes 次要版本時，例如 1.36，該 Kubernetes 次要版本的初始 Amazon EKS 平台版本會從 開始`eks.1`。但是，Amazon EKS 會定期發佈新平台版本，以啟用新的 Kubernetes 控制平面設定，以及提供安全性問題修正。

當有新的 Amazon EKS 平台版本可供次要版本使用時：

- Amazon EKS 平台版本編號將會遞增 (`eks.<n+1>`)。
- Amazon EKS 會自動將所有現有的叢集升級到與其對應之 Kubernetes 次要版本的最新 Amazon EKS 平台版本。現有 Amazon EKS 平台版本的自動升級將會逐步發佈。發行程序可能需要一些時間。如果您立刻需要最新 Amazon EKS 平台版本的功能，您應該建立新的 Amazon EKS 叢集。

  如果您的叢集落後於目前平台版本超過兩個平台版本，則 Amazon EKS 可能無法自動更新叢集。如需可能導致此問題的詳細資訊，請參閱 the section called “Amazon EKS 平台版本比目前平台版本落後兩個版本以上”。
- Amazon EKS 可能會發佈具有對應之修補程式版本的新節點 AMI。但是，對於指定的 Kubernetes 次要版本而言，所有修補程式版本在 EKS 控制平面和節點 AMI 之間都是相容的。

新的 Amazon EKS 平台版本不會帶來重大改變或導致服務中斷。

叢集一律會透過指定 Kubernetes 版本的最新可用 Amazon EKS 平台版本 (`eks.<n>`) 建立。如果您將叢集更新為新的 Kubernetes 次要版本，您的叢集會收到適用於您要更新之 Kubernetes 次要版本的目前 Amazon EKS 平台版本。

Amazon EKS 的目前及最新平台版本如下表所示。

> **Note**
>
> AWS 最近停用了 2024 年 6 月發佈的一些平台版本。平台版本有穩定性問題。不需採取任何動作。

如需接收有關此特定文件頁面所有來源檔案變更的通知，您可透過 RSS 閱讀器來訂閱以下 URL：

```
https://github.com/awsdocs/amazon-eks-user-guide/commits/mainline/latest/ug/versioning/platform-versions.adoc.atom
```

Kubernetes 版本 **1.36**

下列許可控制器已為所有 1.36 平台版本啟用：NodeRestriction、ExtendedResourceToleration、NamespaceLifecycle、LimitRanger、

Kubernetes 版本	EKS 平台版本	版本備註	版本日期
1.36.2	eks.10	具有安全性修正和增強功能的新平台版本。	2026 年 8 月 17 日
1.36.2	eks.9	具有安全性修正和增強功能的新平台版本。	2026 年 8 月 7 日
1.36.2	eks.8	具有安全性修正和增強功能的新平台版本。	2026 年 7 月 23 日
1.36.2	eks.7	具有安全性修正和增強功能的新平台版本。	2026 年 7 月 14 日
1.36.2	eks.6	具有安全性修正和增強功能的新平台版本。 1.36 eks.5 已在內部捨棄，從未發佈。	2026 年 6 月 22 日
1.36.1	eks.4	具有安全性修正和增強功能的新平台版本。	2026 年 6 月 3 日
1.36.1	eks.3	適用於 EKS 的 Kubernetes 初始版本	2026 年 6 月 2 日

Kubernetes 版本	EKS 平台版本	版本備註	版本日期
		1.36。如需詳細資訊，請參閱the section called “Kubernetes 1.36”。	

Kubernetes 版本 1.35

下列許可控制器已為所有 1.35 平台版本啟用：NodeRestriction、ExtendedResourceToleration、NamespaceLifecycle、LimitRanger、

Kubernetes 版本	EKS 平台版本	版本備註	版本日期
1.35.6	eks.21	具有安全性修正和增強功能的新平台版本。	2026 年 8 月 17 日
1.35.6	eks.20	具有安全性修正和增強功能的新平台版本。	2026 年 8 月 7 日
1.35.6	eks.19	具有安全性修正和增強功能的新平台版本。	2026 年 7 月 23 日
1.35.6	eks.18	具有安全性修正和增強功能的新平台版本。	2026 年 7 月 14 日
1.35.6	eks.17	具有安全性修正和增強功能的新平台版本。 1.35 eks.16 已在內部捨棄，從未發佈。	2026 年 6 月 22 日

Kubernetes 版本	EKS 平台版本	版本備註	版本日期
1.35.5	eks.15	具有安全性修正和增強功能的新平台版本。	2026 年 6 月 3 日
1.35.5	eks.14	具有安全性修正和增強功能的新平台版本。	2026 年 5 月 26 日
1.35.4	eks.13	具有安全性修正和增強功能的新平台版本。 1.35 eks.12 已在內部捨棄，從未發佈。	2026 年 5 月 19 日
1.35.4	eks.11	具有安全性修正和增強功能的新平台版本。 1.35 eks.10 已在內部捨棄，從未發佈。	2026 年 5 月 7 日
1.35.2	eks.9	具有安全性修正和增強功能的新平台版本。	2026 年 4 月 3 日
1.35.2	eks.8	具有安全性修正和增強功能的新平台版本。 1.35 eks.6和 1.35 eks.7 已在內部捨棄，且從未發佈。	2026 年 3 月 18 日
1.35.0	eks.5	具有安全性修正和增強功能的新平台版本。	2026 年 3 月 5 日

Kubernetes 版本	EKS 平台版本	版本備註	版本日期
1.35.0	eks.4	具有安全性修正和增強功能的新平台版本。	2026 年 2 月 24 日
1.35.0	eks.3	適用於 EKS 的 Kubernetes 初始版本 1.35。如需詳細資訊，請參閱the section called “Kubernetes 1.35”。	2026 年 1 月 27 日

Kubernetes 版本 **1.34**

下列許可控制器已為所有 1.34 平台版本啟用：NodeRestriction、ExtendedResourceToleration、NamespaceLifecycle、LimitRanger、

Kubernetes 版本	EKS 平台版本	版本備註	版本日期
1.34.9	eks.31	具有安全性修正和增強功能的新平台版本。	2026 年 8 月 17 日
1.34.9	eks.30	具有安全性修正和增強功能的新平台版本。	2026 年 8 月 7 日
1.34.9	eks.29	具有安全性修正和增強功能的新平台版本。	2026 年 7 月 23 日
1.34.9	eks.28	具有安全性修正和增強功能的新平台版本。	2026 年 7 月 14 日
1.34.9	eks.27	具有安全性修正和增強功能的新平台版本	2026 年 6 月 22 日

Kubernetes 版本	EKS 平台版本	版本備註	版本日期
		。 1.34 eks.26 已在內部捨棄，從未發佈。	
1.34.8	eks.25	具有安全性修正和增強功能的新平台版本。	2026 年 6 月 3 日
1.34.8	eks.24	具有安全性修正和增強功能的新平台版本。	2026 年 5 月 26 日
1.34.7	eks.23	具有安全性修正和增強功能的新平台版本。 1.34 eks.22 已在內部捨棄，從未發佈。	2026 年 5 月 19 日
1.34.7	eks.21	具有安全性修正和增強功能的新平台版本。 1.34 eks.20 已在內部捨棄，從未發佈。	2026 年 5 月 7 日
1.34.4	eks.19	具有安全性修正和增強功能的新平台版本。	2026 年 4 月 3 日
1.34.4	eks.18	具有安全性修正和增強功能的新平台版本。 1.34 eks.16和 1.34 eks.17 已在內部捨棄，且從未發佈。	2026 年 3 月 18 日

Kubernetes 版本	EKS 平台版本	版本備註	版本日期
1.34.3	eks.15	具有安全性修正和增強功能的新平台版本。	2026 年 3 月 5 日
1.34.3	eks.14	具有安全性修正和增強功能的新平台版本。	2026 年 2 月 24 日
1.34.3	eks.13	具有安全性修正和增強功能的新平台版本。	2026 年 1 月 28 日
1.34.3	eks.12	具有安全性修正和增強功能的新平台版本。 1.34 eks.11 已在內部捨棄，從未發佈。	2026 年 1 月 16 日
1.34.2	eks.10	具有安全性修正和增強功能的新平台版本。	2025 年 12 月 19 日
1.34.1	eks.9	具有安全性修正和增強功能的新平台版本。	2025 年 11 月 18 日
1.34.1	eks.8	具有安全性修正和增強功能的新平台版本。	2025 年 11 月 17 日
1.34.1	eks.7	具有安全性修正和增強功能的新平台版本。	2025 年 11 月 6 日
1.34.1	eks.6	具有安全性修正和增強功能的新平台版本。	2025 年 10 月 30 日

Kubernetes 版本	EKS 平台版本	版本備註	版本日期
1.34.1	eks.5	具有安全性修正和增強功能的新平台版本。	2025 年 10 月 15 日
1.34.1	eks.4	適用於 EKS 的 Kubernetes 初始版本 1.34。如需詳細資訊，請參閱the section called “Kubernetes 1.34”。	2025 年 10 月 2 日

Kubernetes 版本 **1.33**

下列許可控制器已為所有 1.33 平台版本啟用：NodeRestriction、ExtendedResourceToleration、NamespaceLifecycle、LimitRanger、

Kubernetes 版本	EKS 平台版本	版本備註	版本日期
1.33.13	eks.45	具有安全性修正和增強功能的新平台版本。	2026 年 8 月 17 日
1.33.13	eks.44	具有安全性修正和增強功能的新平台版本。	2026 年 8 月 7 日
1.33.13	eks.43	具有安全性修正和增強功能的新平台版本。	2026 年 7 月 23 日
1.33.13	eks.42	具有安全性修正和增強功能的新平台版本。	2026 年 7 月 14 日
1.33.13	eks.41	具有安全性修正和增強功能的新平台版本	2026 年 6 月 22 日

Kubernetes 版本	EKS 平台版本	版本備註	版本日期
		。 1.33 eks.40 已在內部捨棄，從未發佈。	
1.33.12	eks.39	具有安全性修正和增強功能的新平台版本。	2026 年 6 月 3 日
1.33.12	eks.38	具有安全性修正和增強功能的新平台版本。	2026 年 5 月 26 日
1.33.11	eks.37	具有安全性修正和增強功能的新平台版本。 1.33 eks.36 已在內部捨棄，從未發佈。	2026 年 5 月 19 日
1.33.11	eks.35	具有安全性修正和增強功能的新平台版本。 1.33 eks.34 已在內部捨棄，從未發佈。	2026 年 5 月 7 日
1.33.8	eks.33	具有安全性修正和增強功能的新平台版本。	2026 年 4 月 3 日
1.33.8	eks.32	包含安全修正與增強功能的全新平台版本。1.33 eks.30 及 1.33 eks.31 已在內部捨棄且從未發布。	2026 年 3 月 18 日
1.33.7	eks.29	具有安全性修正和增強功能的新平台版本。	2026 年 3 月 5 日

Kubernetes 版本	EKS 平台版本	版本備註	版本日期
1.33.7	eks.28	具有安全性修正和增強功能的新平台版本。	2026 年 2 月 24 日
1.33.7	eks.27	具有安全性修正和增強功能的新平台版本。	2026 年 1 月 28 日
1.33.7	eks.26	具有安全性修正和增強功能的新平台版本。 1.33 eks.25 已在內部捨棄，從未發佈。	2026 年 1 月 16 日
1.33.6	eks.24	具有安全性修正和增強功能的新平台版本。	2025 年 12 月 19 日
1.33.5	eks.23	具有安全性修正和增強功能的新平台版本。	2025 年 11 月 18 日
1.33.5	eks.22	具有安全性修正和增強功能的新平台版本。	2025 年 11 月 17 日
1.33.5	eks.21	具有安全性修正和增強功能的新平台版本。	2025 年 11 月 6 日
1.33.5	eks.20	具有安全性修正和增強功能的新平台版本。	2025 年 10 月 30 日
1.33.5	eks.19	具有安全性修正和增強功能的新平台版本。	2025 年 10 月 28 日

Kubernetes 版本	EKS 平台版本	版本備註	版本日期
1.33.5	eks.18	具有安全性修正和增強功能的新平台版本。	2025 年 10 月 15 日
1.33.5	eks.17	具有安全性修正和增強功能的新平台版本。	2025 年 10 月 15 日
1.33.5	eks.16	具有安全性修正和增強功能的新平台版本。	2025 年 10 月 2 日
1.33.4	eks.15	具有安全性修正和增強功能的新平台版本。	2025 年 9 月 29 日
1.33.4	eks.14	具有安全性修正和增強功能的新平台版本。	2025 年 9 月 29 日
1.33.4	eks.13	具有安全性修正和增強功能的新平台版本。	2025 年 9 月 16 日
1.33.4	eks.12	具有安全性修正和增強功能的新平台版本。	2025 年 9 月 5 日
1.33.3	eks.11	具有安全性修正和增強功能的新平台版本。	2025 年 8 月 25 日
1.33.3	eks.10	具有安全性修正和增強功能的新平台版本。	2025 年 8 月 17 日

Kubernetes 版本	EKS 平台版本	版本備註	版本日期
1.33.2	eks.9	包含安全修正與增強功能的全新平台版本。1.33 eks.7 及 1.33 eks.8 已在內部捨棄且從未發布。	2025 年 7 月 30 日
1.33.1	eks.6	具有安全性修正和增強功能的新平台版本。	2025 年 6 月 26 日
1.33.1	eks.5	具有安全性修正和增強功能的新平台版本。	2025 年 6 月 11 日
1.33.1	eks.4	適用於 EKS 的 Kubernetes 初始版本 1.33。如需詳細資訊，請參閱the section called “Kubernetes 1.33”。	2025 年 5 月 28 日

Kubernetes 版本 **1.32**

下列許可控制器已為所有 1.32 平台版本啟用：NodeRestriction、ExtendedResourceToleration、NamespaceLifecycle、LimitRanger、

Kubernetes 版本	EKS 平台版本	版本備註	版本日期
1.32.13	eks.52	具有安全性修正和增強功能的新平台版本。	2026 年 8 月 17 日
1.32.13	eks.51	具有安全性修正和增強功能的新平台版本。	2026 年 8 月 7 日

Kubernetes 版本	EKS 平台版本	版本備註	版本日期
1.32.13	eks.50	具有安全性修正和增強功能的新平台版本。	2026 年 7 月 23 日
1.32.13	eks.49	具有安全性修正和增強功能的新平台版本。	2026 年 7 月 14 日
1.32.13	eks.48	具有安全性修正和增強功能的新平台版本。 1.32 eks.47 已在內部捨棄，且從未發佈。	2026 年 6 月 22 日
1.32.13	eks.46	具有安全性修正和增強功能的新平台版本。	2026 年 6 月 3 日
1.32.13	eks.45	具有安全性修正和增強功能的新平台版本。	2026 年 5 月 26 日
1.32.13	eks.44	具有安全性修正和增強功能的新平台版本。 1.32 eks.43 已在內部捨棄，且從未發佈。	2026 年 5 月 19 日
1.32.13	eks.42	具有安全性修正和增強功能的新平台版本。 1.32 eks.41 已在內部捨棄，且從未發佈。	2026 年 5 月 7 日

Kubernetes 版本	EKS 平台版本	版本備註	版本日期
1.32.12	eks.40	具有安全性修正和增強功能的新平台版本。	2026 年 4 月 3 日
1.32.12	eks.39	包含安全修正與增強功能的全新平台版本。1.32 eks.37 及 1.32 eks.38 已在內部捨棄且從未發布。	2026 年 3 月 18 日
1.32.11	eks.36	具有安全性修正和增強功能的新平台版本。	2026 年 3 月 5 日
1.32.11	eks.35	具有安全性修正和增強功能的新平台版本。	2026 年 2 月 24 日
1.32.11	eks.34	具有安全性修正和增強功能的新平台版本。	2026 年 1 月 28 日
1.32.11	eks.33	具有安全性修正和增強功能的新平台版本。 1.32 eks.32 已在內部捨棄，且從未發佈。	2026 年 1 月 16 日
1.32.10	eks.31	具有安全性修正和增強功能的新平台版本。	2025 年 12 月 19 日
1.32.9	eks.30	具有安全性修正和增強功能的新平台版本。	2025 年 11 月 18 日

Kubernetes 版本	EKS 平台版本	版本備註	版本日期
1.32.9	eks.29	具有安全性修正和增強功能的新平台版本。	2025 年 11 月 17 日
1.32.9	eks.28	具有安全性修正和增強功能的新平台版本。	2025 年 11 月 6 日
1.32.9	eks.27	具有安全性修正和增強功能的新平台版本。	2025 年 10 月 30 日
1.32.9	eks.26	具有安全性修正和增強功能的新平台版本。	2025 年 10 月 28 日
1.32.9	eks.25	具有安全性修正和增強功能的新平台版本。	2025 年 10 月 15 日
1.32.9	eks.24	具有安全性修正和增強功能的新平台版本。	2025 年 10 月 15 日
1.32.9	eks.23	具有安全性修正和增強功能的新平台版本。	2025 年 10 月 2 日
1.32.8	eks.22	具有安全性修正和增強功能的新平台版本。	2025 年 9 月 29 日
1.32.8	eks.21	具有安全性修正和增強功能的新平台版本。	2025 年 9 月 29 日

Kubernetes 版本	EKS 平台版本	版本備註	版本日期
1.32.8	eks.20	具有安全性修正和增強功能的新平台版本。	2025 年 9 月 16 日
1.32.8	eks.19	具有安全性修正和增強功能的新平台版本。	2025 年 9 月 5 日
1.32.7	eks.18	具有安全性修正和增強功能的新平台版本。	2025 年 8 月 25 日
1.32.7	eks.17	具有安全性修正和增強功能的新平台版本。	2025 年 8 月 17 日
1.32.6	eks.16	包含安全修正與增強功能的全新平台版本。1.32 eks.14 及 1.32 eks.15 已在內部捨棄且從未發布。	2025 年 7 月 30 日
1.32.5	eks.13	具有安全性修正和增強功能的新平台版本。	2025 年 6 月 26 日
1.32.5	eks.12	具有安全性修正和增強功能的新平台版本。	2025 年 6 月 11 日
1.32.5	eks.11	具有安全性修正和增強功能的新平台版本。	2025 年 5 月 30 日
1.32.3	eks.10	具有安全性修正和增強功能的新平台版本。	2025 年 5 月 16 日

Kubernetes 版本	EKS 平台版本	版本備註	版本日期
1.32.3	eks.9	具有安全性修正和增強功能的新平台版本。	2025 年 4 月 29 日
1.32.3	eks.8	具有安全性修正和增強功能的新平台版本。	2025 年 4 月 18 日
1.32.3	eks.7	具有安全性修正和增強功能的新平台版本。	2025 年 4 月 18 日
1.32.3	eks.6	具有安全性修正和增強功能的新平台版本。	2025 年 4 月 2 日
1.32.2	eks.5	具有安全性修正和增強功能的新平台版本。	2025 年 3 月 17 日
1.32.2	eks.4	具有安全性修正和增強功能的新平台版本。	2025 年 3 月 4 日
1.32.1	eks.3	具有安全性修正和增強功能的新平台版本。	2025 年 2 月 24 日
1.32.0	eks.2	適用於 EKS 的 Kubernetes 初始版本 1.32。如需詳細資訊，請參閱the section called “Kubernetes 1.32”。	2025 年 1 月

Kubernetes 版本 **1.31**

下列許可控制器已為所有 1.31 平台版本啟用：NodeRestriction、ExtendedResourceToleration、NamespaceLifecycle、LimitRanger、

Kubernetes 版本	EKS 平台版本	版本備註	版本日期
1.31.14	eks.68	具有安全性修正和增強功能的新平台版本。	2026 年 8 月 17 日
1.31.14	eks.67	具有安全性修正和增強功能的新平台版本。	2026 年 8 月 7 日
1.31.14	eks.66	具有安全性修正和增強功能的新平台版本。	2026 年 7 月 23 日
1.31.14	eks.65	具有安全性修正和增強功能的新平台版本。	2026 年 7 月 14 日
1.31.14	eks.64	具有安全性修正和增強功能的新平台版本。 1.31 eks.63 已在內部捨棄，且從未發佈。	2026 年 6 月 22 日
1.31.14	eks.62	具有安全性修正和增強功能的新平台版本。	2026 年 6 月 3 日
1.31.14	eks.61	具有安全性修正和增強功能的新平台版本。	2026 年 5 月 26 日
1.31.14	eks.60	具有安全性修正和增強功能的新平台版本	2026 年 5 月 19 日

Kubernetes 版本	EKS 平台版本	版本備註	版本日期
		。 1.31 eks.59 已在內部捨棄，且從未發佈。	
1.31.14	eks.58	具有安全性修正和增強功能的新平台版本。 1.31 eks.57 已在內部捨棄，且從未發佈。	2026 年 5 月 7 日
1.31.14	eks.56	具有安全性修正和增強功能的新平台版本。	2026 年 4 月 3 日
1.31.14	eks.55	包含安全修正與增強功能的全新平台版本。1.31 eks.53 及 1.31 eks.54 已在內部捨棄且從未發布。	2026 年 3 月 18 日
1.31.14	eks.52	具有安全性修正和增強功能的新平台版本。	2026 年 3 月 5 日
1.31.14	eks.51	具有安全性修正和增強功能的新平台版本。	2026 年 2 月 24 日
1.31.14	eks.50	具有安全性修正和增強功能的新平台版本。	2026 年 1 月 28 日
1.31.14	eks.49	具有安全性修正和增強功能的新平台版本。 1.31 eks.48 已在內部捨棄，且從未發佈。	2026 年 1 月 16 日

Kubernetes 版本	EKS 平台版本	版本備註	版本日期
1.31.14	eks.47	具有安全性修正和增強功能的新平台版本。	2025 年 12 月 19 日
1.31.13	eks.46	具有安全性修正和增強功能的新平台版本。	2025 年 11 月 18 日
1.31.13	eks.45	具有安全性修正和增強功能的新平台版本。	2025 年 11 月 17 日
1.31.13	eks.44	具有安全性修正和增強功能的新平台版本。	2025 年 11 月 6 日
1.31.13	eks.43	具有安全性修正和增強功能的新平台版本。	2025 年 10 月 30 日
1.31.13	eks.42	具有安全性修正和增強功能的新平台版本。	2025 年 10 月 28 日
1.31.13	eks.41	具有安全性修正和增強功能的新平台版本。	2025 年 10 月 15 日
1.31.13	eks.40	具有安全性修正和增強功能的新平台版本。	2025 年 10 月 15 日
1.31.13	eks.39	具有安全性修正和增強功能的新平台版本。	2025 年 10 月 2 日

Kubernetes 版本	EKS 平台版本	版本備註	版本日期
1.31.12	eks.38	具有安全性修正和增強功能的新平台版本。	2025 年 9 月 29 日
1.31.12	eks.37	具有安全性修正和增強功能的新平台版本。	2025 年 9 月 29 日
1.31.12	eks.36	具有安全性修正和增強功能的新平台版本。	2025 年 9 月 16 日
1.31.12	eks.35	具有安全性修正和增強功能的新平台版本。	2025 年 9 月 5 日
1.31.11	eks.34	具有安全性修正和增強功能的新平台版本。	2025 年 8 月 25 日
1.31.11	eks.33	具有安全性修正和增強功能的新平台版本。	2025 年 8 月 17 日
1.31.10	eks.32	包含安全修正與增強功能的全新平台版本。1.31 eks.30 及 1.31 eks.31 已在內部捨棄且從未發布。	2025 年 7 月 30 日
1.31.9	eks.29	具有安全性修正和增強功能的新平台版本。	2025 年 6 月 26 日
1.31.9	eks.28	具有安全性修正和增強功能的新平台版本。	2025 年 6 月 11 日

Kubernetes 版本	EKS 平台版本	版本備註	版本日期
1.31.9	eks.27	具有安全性修正和增強功能的新平台版本。	2025 年 5 月 30 日
1.31.7	eks.26	具有安全性修正和增強功能的新平台版本。	2025 年 5 月 16 日
1.31.7	eks.25	具有安全性修正和增強功能的新平台版本。	2025 年 4 月 29 日
1.31.7	eks.24	具有安全性修正和增強功能的新平台版本。	2025 年 4 月 18 日
1.31.7	eks.23	具有安全性修正和增強功能的新平台版本。	2025 年 4 月 18 日
1.31.7	eks.22	具有安全性修正和增強功能的新平台版本。	2025 年 4 月 2 日
1.31.6	eks.21	具有安全性修正和增強功能的新平台版本。	2025 年 3 月 17 日
1.31.6	eks.20	具有安全性修正和增強功能的新平台版本。	2025 年 3 月 4 日
1.31.5	eks.19	具有安全性修正和增強功能的新平台版本。	2025 年 2 月 24 日

Kubernetes 版本	EKS 平台版本	版本備註	版本日期
1.31.5	eks.18	具有安全性修正和增強功能的新平台版本。	2025 年 2 月 24 日
1.31.4	eks.17	具有安全性修正和增強功能的新平台版本。	2025 年 1 月 17 日
1.31.2	eks.12	包含 Amazon EKS 混合節點支援與控制平面可觀測性增強功能的全新平台版本。請分別參閱 the section called “混合節點” 及 Amazon EKS 效能可觀測性增強功能。	2024 年 11 月 15 日
1.31.1	eks.6	具有安全性修正和增強功能的新平台版本。	2024 年 10 月 21 日
1.31.0	eks.4	適用於 EKS 的 Kubernetes 初始版本 1.31。如需詳細資訊，請參閱the section called “Kubernetes 1.31”。	2024 年 9 月 26 日

Kubernetes 版本 **1.30**

下列許可控制器已為所有 1.30 平台版本啟用：NodeRestriction、ExtendedResourceToleration、NamespaceLifecycle、LimitRanger、

Kubernetes 版本	EKS 平台版本	版本備註	版本日期
1.30.14	eks.76	具有安全性修正和增強功能的新平台版本。	2026 年 8 月 17 日
1.30.14	eks.75	具有安全性修正和增強功能的新平台版本。	2026 年 8 月 7 日
1.30.14	eks.74	具有安全性修正和增強功能的新平台版本。	2026 年 7 月 23 日
1.30.14	eks.73	具有安全性修正和增強功能的新平台版本。	2026 年 7 月 14 日
1.30.14	eks.72	具有安全性修正和增強功能的新平台版本。 1.30 eks.71 已在內部捨棄，從未發佈。	2026 年 6 月 22 日
1.30.14	eks.70	具有安全性修正和增強功能的新平台版本。	2026 年 6 月 3 日
1.30.14	eks.69	具有安全性修正和增強功能的新平台版本。	2026 年 5 月 26 日
1.30.14	eks.68	具有安全性修正和增強功能的新平台版本。 1.30 eks.67 已在內部捨棄，從未發佈。	2026 年 5 月 19 日

Kubernetes 版本	EKS 平台版本	版本備註	版本日期
1.30.14	eks.66	具有安全性修正和增強功能的新平台版本。 1.30 eks.65 已在內部捨棄，從未發佈。	2026 年 5 月 7 日
1.30.14	eks.64	具有安全性修正和增強功能的新平台版本。	2026 年 4 月 3 日
1.30.14	eks.63	包含安全修正與增強功能的全新平台版本。1.30 eks.61 及 1.30 eks.62 已在內部捨棄且從未發布。	2026 年 3 月 18 日
1.30.14	eks.60	具有安全性修正和增強功能的新平台版本。	2026 年 3 月 5 日
1.30.14	eks.59	具有安全性修正和增強功能的新平台版本。	2026 年 2 月 24 日
1.30.14	eks.58	具有安全性修正和增強功能的新平台版本。	2026 年 1 月 28 日
1.30.14	eks.57	具有安全性修正和增強功能的新平台版本。 1.30 eks.56 已在內部捨棄，從未發佈。	2026 年 1 月 16 日

Kubernetes 版本	EKS 平台版本	版本備註	版本日期
1.30.14	eks.55	具有安全性修正和增強功能的新平台版本。	2025 年 12 月 19 日
1.30.14	eks.54	具有安全性修正和增強功能的新平台版本。	2025 年 11 月 18 日
1.30.14	eks.53	具有安全性修正和增強功能的新平台版本。	2025 年 11 月 17 日
1.30.14	eks.52	具有安全性修正和增強功能的新平台版本。	2025 年 11 月 6 日
1.30.14	eks.51	具有安全性修正和增強功能的新平台版本。	2025 年 10 月 30 日
1.30.14	eks.50	具有安全性修正和增強功能的新平台版本。	2025 年 10 月 28 日
1.30.14	eks.49	具有安全性修正和增強功能的新平台版本。	2025 年 10 月 15 日
1.30.14	eks.48	具有安全性修正和增強功能的新平台版本。	2025 年 10 月 15 日
1.30.14	eks.47	具有安全性修正和增強功能的新平台版本。	2025 年 10 月 2 日

Kubernetes 版本	EKS 平台版本	版本備註	版本日期
1.30.14	eks.46	具有安全性修正和增強功能的新平台版本。	2025 年 9 月 29 日
1.30.14	eks.45	具有安全性修正和增強功能的新平台版本。	2025 年 9 月 29 日
1.30.14	eks.44	具有安全性修正和增強功能的新平台版本。	2025 年 9 月 16 日
1.30.14	eks.43	具有安全性修正和增強功能的新平台版本。	2025 年 9 月 5 日
1.30.14	eks.42	具有安全性修正和增強功能的新平台版本。	2025 年 8 月 25 日
1.30.14	eks.41	具有安全性修正和增強功能的新平台版本。	2025 年 8 月 17 日
1.30.14	eks.40	包含安全修正與增強功能的全新平台版本。1.30 eks.38 及 1.30 eks.39 已在內部捨棄且從未發布。	2025 年 7 月 30 日
1.30.13	eks.37	具有安全性修正和增強功能的新平台版本。	2025 年 6 月 26 日
1.30.13	eks.36	具有安全性修正和增強功能的新平台版本。	2025 年 6 月 11 日

Kubernetes 版本	EKS 平台版本	版本備註	版本日期
1.30.13	eks.35	具有安全性修正和增強功能的新平台版本。	2025 年 5 月 30 日
1.30.11	eks.34	具有安全性修正和增強功能的新平台版本。	2025 年 5 月 16 日
1.30.11	eks.33	具有安全性修正和增強功能的新平台版本。	2025 年 4 月 29 日
1.30.11	eks.32	具有安全性修正和增強功能的新平台版本。	2025 年 4 月 18 日
1.30.11	eks.31	具有安全性修正和增強功能的新平台版本。	2025 年 4 月 18 日
1.30.11	eks.30	具有安全性修正和增強功能的新平台版本。	2025 年 4 月 2 日
1.30.10	eks.29	具有安全性修正和增強功能的新平台版本。	2025 年 3 月 17 日
1.30.10	eks.28	具有安全性修正和增強功能的新平台版本。	2025 年 3 月 4 日
1.30.9	eks.27	具有安全性修正和增強功能的新平台版本。	2025 年 2 月 24 日

Kubernetes 版本	EKS 平台版本	版本備註	版本日期
1.30.8	eks.25	具有安全性修正和增強功能的新平台版本。	2025 年 1 月 17 日
1.30.8	eks.24	具有安全性修正和增強功能的新平台版本。	2025 年 1 月 3 日
1.30.7	eks.23	具有安全性修正和增強功能的新平台版本。	2024 年 12 月 13 日
1.30.6	eks.22	具有安全性修正和增強功能的新平台版本。	2024 年 12 月 13 日
1.30.6	eks.21	具有安全性修正和增強功能的新平台版本。	2024 年 12 月 13 日
1.30.6	eks.20	包含 Amazon EKS 混合節點支援與控制平面可觀測性增強功能的全新平台版本。請分別參閱 the section called “混合節點” 及 Amazon EKS 效能可觀測性增強功能。	2024 年 11 月 15 日
1.30.5	eks.12	具有安全性修正和增強功能的新平台版本。	2024 年 10 月 21 日
1.30.4	eks.8	具有安全性修正和增強功能的新平台版本。	2024 年 9 月 3 日

Kubernetes 版本	EKS 平台版本	版本備註	版本日期
1.30.3	eks.7	具有安全性修正和增強功能的新平台版本。	2024 年 8 月 28 日
1.30.3	eks.6	具有安全性修正和增強功能的新平台版本。	2024 年 8 月 9 日
1.30.2	eks.5	具有安全性修正和增強功能的新平台版本。	2024 年 7 月 2 日
1.30.0	eks.2	適用於 EKS 的 Kubernetes 初始版本 1.30。	2024 年 5 月 23 日

了解目前的平台版本

取得目前的平台版本AWS （主控台）

1. 開啟 Amazon EKS 主控台。
2. 在導覽窗格中，選擇叢集。
3. 在叢集清單中，選擇需要查看平台版本的叢集名稱。
4. 選擇 Overview (概觀) 索引標籤。
5. 平台版本可在詳細資訊區段中取得。

取得目前的平台版本 (AWS CLI)

1. 決定您要檢查平台版本的叢集名稱。
2. 執行以下命令：

```
aws eks describe-cluster --name my-cluster --query cluster.platformVersion
```

範例輸出如下。

```
"eks.10"
```

變更平台版本

您不能變更 EKS 叢集的平台版本。若有新的 Amazon EKS 平台版本適用於 Kubernetes 版本，EKS 會自動將所有現有叢集升級至適用於其對應 Kubernetes 版本的最新 Amazon EKS 平台版本。現有 Amazon EKS 平台版本的自動升級將會逐步發布。您無法使用 AWS 主控台或 CLI 來變更平台版本。

若升級您的 Kubernetes 版本，您的叢集會移至適用於 Kubernetes 版本的最新平台版本。

排解 Amazon EKS 叢集和節點問題

此章節涵蓋一些您在使用 Amazon EKS 時可能遇到的常見錯誤，並提供解決方法。如果您需要針對特定 Amazon EKS 區域進行疑難排解，請參閱個別 the section called “疑難排解”、the section called “對 EKS 連接器進行疑難排解” 和針對使用 EKS 附加元件的 ADOT 進行疑難排解主題。

如需其他疑難排解資訊，請參閱 AWS re：Post 上的 Amazon Elastic Kubernetes Service 知識中心內容。

容量不足

如果您在嘗試建立 Amazon EKS 叢集時收到下列錯誤，則代表其中一個您指定的可用區域沒有足夠的容量來支援叢集。

```
Cannot create cluster 'example-cluster' because region-1d, the targeted
Availability Zone, does not currently have sufficient capacity to support
the cluster. Retry and choose from these Availability Zones: region-1a,
region-1b, region-1c
```

重新嘗試使用叢集 VPC 中的子網路來建立您的叢集，叢集 VPC 託管於此錯誤訊息傳回之可用區域之中。

有些可用區域是叢集無法駐留的。將子網路所在的可用區域與子網路要求與考量中的可用區域清單進行比較。

節點無法加入叢集

有幾個常見的原因會阻擋節點加入叢集：

- 如果節點是受管節點，則 Amazon EKS 會在您建立節點群組時會將項目新增至 `aws-auth ConfigMap`。如果該項目被移除或修改，則您需要重新新增項目。如需詳細資訊，請在您的終端機中輸入 `eksctl create iamidentitymapping --help`。您可使用您的叢集名稱取代以下命令中的 *`my-cluster`* 部分，然後執行修改後的命令來檢視目前的 `aws-auth ConfigMap` 項目：`eksctl get iamidentitymapping --cluster` *`my-cluster`* 。您指定之角色的 ARN 不能包含 `/` 以外的路徑。例如，如果您的角色名稱是 `development/apps/my-role`，則您需要在指定角色的 ARN 時將其變更為 `my-role`。請確認您指定的是節點 IAM 角色 ARN (而非執行個體設定檔 ARN)。

如果節點是自我管理的，且您尚未為節點的 IAM 角色的 ARN 建立存取項目，則執行所列出的針對受管節點的命令。如果您已為節點 IAM 角色 ARN 建立存取項目，則可能無法在存取項目中正確進行設定。請確認將節點 IAM 角色 ARN (而非執行個體設定檔 ARN) 指定為 `aws-auth ConfigMap` 項目或存取項目中的主體 ARN。如需存取項目的詳細資訊，請參閱 the section called “存取項目”。

- node AWS CloudFormation 範本中的 ClusterName 與您希望節點加入的叢集名稱不完全相符。傳遞不正確的值到節點的 `/var/lib/kubelet/kubeconfig` 檔案之不正確的組態內的此欄位中，而節點不會加入叢集。
- 節點未標記為叢集所擁有。您的節點必須套用下列標籤，其中 *my-cluster* 會以叢集的名稱取代。

金鑰	值
kubernetes.io/cluster/*my-cluster*	owned

- 節點可能無法使用公有 IP 地址存取叢集。確定已將公有子網路中部署的節點指派給公有 IP 地址。如果沒有，您可以在啟動節點後將彈性 IP 位址關聯至節點。如需詳細資訊，請參閱將彈性 IP 地址與執行中的執行個體或網路介面建立關聯。如果公有子網路未設定為自動將公有 IP 地址指派給部署到該子網路的執行個體，則建議您啟用該設定。如需詳細資訊，請參閱修改子網的公有 IPv4 定址屬性。如果節點部署到私有子網路，則子網路必須具有指派給它的公有 IP 地址的 NAT 閘道路由。
- 您的帳戶未啟用您要部署節點之 AWS 區域的 AWS STS 端點。若要啟用區域，請參閱在 區域中啟用和停用 AWS STS AWS。
- 節點沒有私有 DNS 項目，導致 `kubelet` 日誌包含 `node "" not found` 錯誤。確保建立節點所在的 VPC 具有為 `domain-name` 設定的值以及在 `DHCP options set` 中將 `domain-name-servers` 設定為 `Options`。預設值為 `domain-name:<region>.compute.internal` 和 `domain-name-servers:AmazonProvidedDNS`。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的 DHCP 選項集。
- 若受管節點群組中的節點在 15 分鐘內沒有連線至叢集，則會發出 "NodeCreationFailure" 運作狀態問題，且主控台狀態會設定為 `Create failed`。若是 AMI 啟動時間較慢，可使用快速啟動來解決此問題。

若要識別和疑難排解導致 Worker 節點無法加入叢集的常見原因，您可以使用 `AWSSupport-TroubleshootEKSWorkerNode` Runbook。如需詳細資訊，請參閱 AWS Systems Manager Automation Runbook 參考 `AWSSupport-TroubleshootEKSWorkerNode` 中的 。

未經授權或存取遭拒 (kubectl)

如果您在執行 kubectl 命令時，收到以下其中一個錯誤，則表示未為 Amazon EKS 正確設定 kubectl，或者表示您使用的 IAM 主體 (角色或使用者) 憑證並未映射到對 Amazon EKS 叢集上的 Kubernetes 物件具有足夠許可的 Kubernetes 使用者名稱。

- `could not get token: AccessDenied: Access denied`
- `error: You must be logged in to the server (Unauthorized)`
- `error: the server doesn't have a resource type "svc"`

這種情況可能由下列原因之一造成：

- 該叢集使用一個 IAM 主體的憑證建立，而 kubectl 設定為使用另一個 IAM 主體的憑證。若要解決此問題，請更新 kube config 檔案以使用建立叢集時使用的憑證。如需詳細資訊，請參閱the section called "透過 kubectl 存取叢集"。
- 如果叢集符合透過 EKS 存取項目授予 IAM 使用者對 Kubernetes 的存取權的「先決條件」部分的最低平台要求，則 IAM 主體不存在存取項目。如果存在，則沒有為其定義的必要的 Kubernetes 群組名稱，或沒有與其關聯的正確存取政策。如需詳細資訊，請參閱the section called "存取項目"。
- 如果叢集符合透過 EKS 存取項目授予 IAM 使用者對 Kubernetes 的存取權中的最低平台要求，則 IAM 主體的項目在 aws-auth ConfigMap 中不存在。如果存在，則不會映射到繫結到具有必要許可的 Kubernetes Role 或 ClusterRole 的 Kubernetes 群組名稱。若要了解 Kubernetes 角色型授權 (RBAC) 物件的相關詳細資訊，請參閱 Kubernetes 文件中的使用 RBAC 授權。您可使用您的叢集名稱取代以下命令中的 *my-cluster* 部分，然後執行修改後的命令來檢視目前的 aws-auth ConfigMap 項目：`eksctl get iamidentitymapping --cluster my-cluster` 。如果具有 IAM 主體 ARN 的項目不在 中ConfigMap，請在終端機`eksctl create iamidentitymapping --help`中輸入 以了解如何建立項目。

如果您安裝並設定 AWS CLI，則可以設定您使用的 IAM 登入資料。如需詳細資訊，請參閱《 AWS 命令列界面使用者指南》中的設定 AWS CLI。如果您透過擔任 IAM 角色來存取叢集上的 Kubernetes 物件，則也可以將 kubectl 設定為使用 IAM 角色。如需詳細資訊，請參閱the section called "透過 kubectl 存取叢集"。

hostname doesn't match

您的系統 Python 版本必須為 2.7.9 或更新版本。否則，您會收到 CLI AWS 呼叫 Amazon EKS 的hostname doesn't match錯誤。如需詳細資訊，請參閱 Python 請求常見問答集中的 什麼是「主機名稱不相符」錯誤？。

getsockopt: no route to host

Docker 會在 Amazon EKS 叢集的 172.17.0.0/16 CIDR 範圍中執行。我們建議不要將叢集的 VPC 子網路與這個範圍重疊。否則，您會收到以下錯誤：

```
Error: : error upgrading connection: error dialing backend: dial tcp
 172.17.<nn>.<nn>:10250: getsockopt: no route to host
```

Instances failed to join the Kubernetes cluster

如果您在 Instances failed to join the Kubernetes cluster中收到錯誤 AWS 管理主控台，請確定叢集的私有端點存取已啟用，或您已正確設定 CIDR 區塊以進行公有端點存取。如需詳細資訊，請參閱the section called "叢集端點存取"。

受管節點群組錯誤代碼

如果受管節點群組遇到硬體運作狀態問題，Amazon EKS 會傳回錯誤代碼，以協助您診斷問題。這些運作狀態檢查不會偵測軟體問題，因為檢查是以 Amazon EC2 運作狀態檢查為基礎。以下清單描述了這些錯誤代碼。

AccessDenied

Amazon EKS 或一或多個受管節點無法透過 Kubernetes 叢集 API 伺服器進行身分驗證或授權。如需有關解決常見原因的詳細資訊，請參閱 the section called "修正受管節點群組的 AccessDenied 錯誤的常見原因"。私有 Windows AMI 也可能導致此錯誤代碼隨 Not authorized for images 錯誤訊息同時出現。如需詳細資訊，請參閱the section called "Not authorized for images"。

AmiIdNotFound

我們找不到與您的啟動範本相關聯的 AMI ID。請確認 AMI 已存在且已與您的帳戶共用。

AutoScalingGroupNotFound

我們找不到與受管節點群組相關聯的 Auto Scaling 群組。您可以使用相同設定來重新建立 Auto Scaling 群組以復原。

ClusterUnreachable

Amazon EKS 或一或多個受管節點無法與您的 Kubernetes 叢集 API 伺服器通訊。如果發生網路中斷或 API 伺服器處理請求逾時，就會發生這種情況。

Ec2SecurityGroupNotFound

我們找不到叢集的叢集安全群組。您必須重新建立叢集。

Ec2SecurityGroupDeletionFailure

無法刪除受管節點群組的遠端存取安全群組。從安全群組移除任何相依項目。

Ec2LaunchTemplateNotFound

我們找不到受管節點群組的 Amazon EC2 啟動範本。您必須重新建立節點群組以復原。

Ec2LaunchTemplateVersionMismatch

受管節點群組的 Amazon EC2 啟動範本版本與 Amazon EKS 建立的版本不相符。您可以還原為 Amazon EKS 所建立的版本以復原。

IamInstanceProfileNotFound

我們找不到受管節點群組的 IAM 執行個體設定檔。您可以使用相同設定來重新建立執行個體描述檔以復原。

IamNodeRoleNotFound

我們找不到受管節點群組的 IAM 角色。您可以使用相同設定來重新建立 IAM 角色以復原。

AsgInstanceLaunchFailures

您的 Auto Scaling 群組嘗試啟動執行個體時失敗。

NodeCreationFailure

您啟動的執行個體無法向 Amazon EKS 叢集註冊。這項失敗的常見原因是節點 IAM 角色許可不足或節點缺少對外網際網路存取。您的節點必須符合下列任何一項要求：

- 能夠使用公有 IP 地址存取網際網路。與節點所在的子網路相關聯的安全群組必須允許通訊。如需詳細資訊，請參閱the section called “子網需求和注意事項”及the section called “安全群組要求”。
- 您的節點與 VPC 必須滿足部署網際網路存取受到限制的私有叢集中的要求。

InstanceLimitExceeded

AWS 您的帳戶無法啟動任何其他指定執行個體類型的執行個體。您可以請求提升 Amazon EC2 執行個體限制來復原。

InsufficientFreeAddresses

與受管節點群組相關聯的一個或多個子網沒有足夠的可用 IP 位址供新節點使用。

InternalFailure

這些錯誤通常是由 Amazon EKS 伺服器端問題所造成。

修正受管節點群組的 **AccessDenied** 錯誤的常見原因

在受管節點群組上執行操作時發生 AccessDenied 錯誤的最常見原因，是缺少 eks:node-manager ClusterRole 或 ClusterRoleBinding。Amazon EKS 會在您的叢集中設定這些資源，作為與受管節點群組上線的一部分，這些資源是管理節點群組所必需的。

ClusterRole 可能隨著時間而改變，但看起來應與以下範例相似：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: eks:node-manager
rules:
- apiGroups:
  - ''
  resources:
  - pods
  verbs:
  - get
  - list
  - watch
  - delete
- apiGroups:
  - ''
  resources:
  - nodes
  verbs:
  - get
  - list
  - watch
  - patch
```

```
- apiGroups:
  - ''
  resources:
  - pods/eviction
  verbs:
  - create
```

ClusterRoleBinding 可能隨著時間而改變，但看起來應與以下範例相似：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: eks:node-manager
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: eks:node-manager
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: eks:node-manager
```

驗證 eks:node-manager ClusterRole 是否存在。

```
kubectl describe clusterrole eks:node-manager
```

如果存在，請將輸出與前一個 ClusterRole 範例進行比較。

驗證 eks:node-manager ClusterRoleBinding 是否存在。

```
kubectl describe clusterrolebinding eks:node-manager
```

如果存在，請將輸出與前一個 ClusterRoleBinding 範例進行比較。

如果在請求受管節點群組操作時您發現缺少或損壞 ClusterRole 或 ClusterRoleBinding 是 AccessDenied 錯誤的原因，您可以將其還原。儲存以下內容至名稱為 *eks-node-manager-role.yaml* 的檔案。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
```

```
  name: eks:node-manager
rules:
- apiGroups:
  - ''
  resources:
  - pods
  verbs:
  - get
  - list
  - watch
  - delete
- apiGroups:
  - ''
  resources:
  - nodes
  verbs:
  - get
  - list
  - watch
  - patch
- apiGroups:
  - ''
  resources:
  - pods/eviction
  verbs:
  - create
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: eks:node-manager
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: eks:node-manager
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: eks:node-manager
```

套用檔案。

```
kubectl apply -f eks-node-manager-role.yaml
```

重試節點群組操作，看看是否可以解決您的問題。

Not authorized for images

導致 Not authorized for images 錯誤訊息的一個潛在原因是使用私有 Amazon EKS Windows AMI 啟動 Windows 受管節點群組。發佈新的 Windows AMIs 後， AWS 會將超過 4 個月的 AMIs 設為私有，使其不再可存取。如果您的受管節點群組使用私有 Windows AMI，請考慮更新 Windows 受管節點群組。雖然我們無法保證我們可以提供已設為私有AMIs 存取權，但您可以透過向 AWS Support 提交票證來請求存取權。如需詳細資訊，請參閱Amazon EC2 使用者指南中的修補程式。

節點進入 NotReady 狀態

若節點進入 NotReady 狀態，這可能表明節點狀況不佳，並且無法排程新的 Pod。發生這種情況可能出於各種原因，例如，節點缺少 CPU、記憶體或可用磁碟空間等充足的資源。

若是 Amazon EKS 最佳化 Windows AMI，依預設，kubelet 組態中不會保留運算資源。為協助避免資源問題，可向 kubelet 提供 kube-reserved 及/或 system-reserved 的組態值，藉此來為系統程序預留運算資源。您可使用引導指令碼中的 -KubeletExtraArgs 命令列參數來執行此動作。如需詳細資訊，請參閱《Kubernetes 文件》中的為系統常駐程式預留運算資源，以及本《使用者指南》中的 the section called “引導指令碼組態參數”。

EKS 日誌收集器

若要疑難排解 Amazon EKS 節點的問題，位於 的節點上提供了預先建置的指令碼/etc/eks/log-collector-script/eks-log-collector.sh。您可以使用指令碼來收集支援案例和一般故障診斷的診斷日誌。

在您的節點使用以下命令即可執行指令碼：

```
sudo bash /etc/eks/log-collector-script/eks-log-collector.sh
```

> **Note**
>
> 如果指令碼在該位置不存在，您可以使用下列命令手動下載並執行指令碼：
>
> ```
> curl -O https://amazon-eks.s3.amazonaws.com/support/log-collector-script/linux/
> eks-log-collector.sh
> ```

```
sudo bash eks-log-collector.sh
```

該指令碼收集的診斷資訊如下。

```
$ sudo bash /etc/eks/log-collector-script/eks-log-collector.sh

      This is version 0.7.8. New versions can be found at https://github.com/awslabs/
amazon-eks-ami/blob/main/log-collector-script/

Trying to collect common operating system logs...
Trying to collect kernel logs...
Trying to collect mount points and volume information...
...
...

 Done... your bundled logs are located in /var/log/eks_i-EXAMPLE_2025-03-25_0000-
UTC_0.7.8.tar.gz
```

診斷資訊收集後將存放於：

```
/var/log/eks_i-EXAMPLE_2025-03-25_0000-UTC_0.7.8.tar.gz
```

若要擷取 Bottlerocket 節點的日誌套件，請參閱 Bottlerocket Logs 以取得更多詳細資訊。

容器執行階段網路尚未就緒

您可能會收到類似下列的 `Container runtime network not ready` 錯誤和授權錯誤：

```
4191 kubelet.go:2130] Container runtime network not ready: NetworkReady=false
 reason:NetworkPluginNotReady message:docker: network plugin is not ready: cni config
 uninitialized
4191 reflector.go:205] k8s.io/kubernetes/pkg/kubelet/kubelet.go:452: Failed to list
 *v1.Service: Unauthorized
4191 kubelet_node_status.go:106] Unable to register node
 "ip-10-40-175-122.ec2.internal" with API server: Unauthorized
4191 reflector.go:205] k8s.io/kubernetes/pkg/kubelet/kubelet.go:452: Failed to list
 *v1.Service: Unauthorized
```

這種情況可能由下列原因之一造成：

1. 叢集上沒有 aws-auth ConfigMap，或者它不包含針對您設定節點時所使用 IAM 角色的項目。

   若要解決此問題，您可使用您的叢集名稱取代以下命令中的 *my-cluster* 部分，然後執行修改後的命令來檢視目前 ConfigMap 中的項目：eksctl get iamidentitymapping --cluster *my-cluster* 。如果執行該命令時遇到錯誤訊息，則可能是因為叢集沒有 aws-auth ConfigMap。以下命令會將項目新增至 ConfigMap。如果 ConfigMap 不存在，則該命令會建立它。將 *111122223333* 取代為 IAM 角色 AWS 的帳戶 ID，並將 *myAmazonEKSNodeRole* 取代為節點角色的名稱。

   ```
   eksctl create iamidentitymapping --cluster my-cluster \
       --arn arn:aws: iam::111122223333:role/myAmazonEKSNodeRole --group
    system:bootstrappers,system:nodes \
       --username system:node:{{EC2PrivateDNSName}}
   ```

   您指定之角色的 ARN 不能包含 / 以外的路徑。例如，如果您的角色名稱是 development/apps/my-role，則您需要在指定角色的 ARN 時將其變更為 my-role。請確認您指定的是節點 IAM 角色 ARN (而非執行個體設定檔 ARN)。
2. 您的自我管理節點所在的叢集的平台版本為透過 EKS 存取項目授予 IAM 使用者對 Kubernetes 的存取權主題「先決條件」中列出的最低版本，但 aws-auth ConfigMap (請參閱上一項) 中沒有針對該節點之 IAM 角色的項目或不存在針對該角色的存取項目。若要解決此問題，您可使用您的叢集名稱取代以下命令中的 *my-cluster* 部分，然後執行修改後的命令來檢視目前的存取項目：aws eks list-access-entries --cluster-name *my-cluster* 。以下命令會為相應節點的 IAM 角色新增一個存取項目。將 *111122223333* 取代為 IAM 角色 AWS 的帳戶 ID，並將 *myAmazonEKSNodeRole* 取代為節點角色的名稱。如果節點為 Windows 節點，請使用 EC2_Windows 取代 *EC2_LINUX*。請確認您指定的是節點 IAM 角色 ARN (而非執行個體設定檔 ARN)。

   ```
   aws eks create-access-entry --cluster-name my-cluster --principal-arn arn:aws:
    iam::111122223333:role/myAmazonEKSNodeRole --type EC2_LINUX
   ```

TLS 交握逾時

當節點無法建立與公有 API 伺服器端點的連接時，您可能會收到類似下列的錯誤。

```
server.go:233] failed to run Kubelet: could not init cloud provider "aws": error
 finding instance i-1111f2222f333e44c: "error listing AWS instances: \"RequestError:
 send request failed\\ncaused by: Post  net/http: TLS handshake timeout\""
```

kubelet 程序將持續重新產生並測試 API 伺服器端點。在控制平面中執行叢集滾動更新 (例如組態變更或版本更新) 的任何程序期間，也可能會暫時發生錯誤。

若要解決此問題，請檢查路由表和安全群組，以確保來自節點的流量可以到達公有端點。

InvalidClientTokenId

如果您針對部署至中國 AWS 區域中叢集的 Pod 或 DaemonSet 的服務帳戶使用 IAM 角色，且尚未在規格中設定AWS_DEFAULT_REGION環境變數，Pod 或 DaemonSet 可能會收到下列錯誤：

```
An error occurred (InvalidClientTokenId) when calling the GetCallerIdentity operation:
 The security token included in the request is invalid
```

若要解決此問題，您需要將 AWS_DEFAULT_REGION 環境變數新增至您的 Pod 或 Daemonset 規格，如下列範例 Pod 規格所示。

```
apiVersion: v1
kind: Pod
metadata:
  name: envar-demo
  labels:
    purpose: demonstrate-envars
spec:
  containers:
  - name: envar-demo-container
    image: gcr.io/google-samples/node-hello:1.0
    env:
    - name: AWS_DEFAULT_REGION
      value: "region-code"
```

升級控制平面之前，節點群組必須符合 Kubernetes 版本

將控制平面升級至新的 Kubernetes 版本之前，叢集中受管節點和 Fargate 節點的次要版本必須與控制平面的目前版本相同。在將所有 Amazon EKS 受管節點升級為目前的叢集版本前，Amazon EKS update-cluster-version API 都會拒絕請求。Amazon EKS 提供 API 來升級受管節點。如需升級受管節點群組的 Kubernetes 版本的資訊，請參閱 the section called “更新”。若要升級 Fargate 節點的版本，請刪除節點所代表的 Pod，並在升級控制平面後重新部署 Pod。如需詳細資訊，請參閱the section called “更新 Kubernetes 版本”。

啟動許多節點時，會出現 Too Many Requests 錯誤

如果同時啟動許多節點，您可能會 Amazon EC2 使用者資料執行日誌看見錯誤訊息，其顯示 Too Many Requests。這可能是因為控制平面因 describeCluster 呼叫超載。超載會導致調節、節點無法執行引導指令碼，以及節點無法完全加入叢集。

確認將 --apiserver-endpoint、--b64-cluster-ca 和 --dns-cluster-ip 引數傳遞至節點的引導指令碼。包含這些引數時，不需要引導指令碼進行 describeCluster 呼叫，這有助於防止控制平面超載。如需詳細資訊，請參閱the section called “指定 AMI”。

針對 Kubernetes API 伺服器請求回應的 HTTP 401 未經授權的錯誤

如果叢集上的 Pod 服務帳戶字符已過期，您會看到這些錯誤。

您的 Amazon EKS 叢集的 Kubernetes API 伺服器拒絕使用超過 90 天的字符的請求。在 Kubernetes 舊版本中，字符沒有過期。這意味著倚賴這些字符的客戶端必須在一小時內進行重新整理。為了防止 Kubernetes API 伺服器因無效字符而拒絕您的請求，您的工作負載使用的 Kubernetes 用戶端 SDK 版本必須與以下版本相同或為更新版本：

- Go 版本 0.15.7 和更新版本
- Python 版本 12.0.0 和更新版本
- Java 版本 9.0.0 和更新版本
- JavaScript 版本 0.10.3 和更新版本
- Ruby master 分支
- Haskell 版本 0.3.0.0
- C# 7.0.5 版及更新版本

您可以識別叢集中所有使用過時字符的現有 Pod。如需詳細資訊，請參閱the section called “服務帳戶字符”。

Amazon EKS 平台版本比目前平台版本落後兩個版本以上

當 Amazon EKS 無法自動更新叢集的 platform-version 時，可能會發生這種情況。儘管造成這種情況的原因很多，然而一些常見的原因如下。如果這些問題中有任何一個適用於您的叢集，其可能仍然可以運作，但 Amazon EKS 不會更新其平台版本。

問題

該叢集 IAM 角色已刪除，此角色是在建立叢集時指定。您可以使用以下命令，查看指定了哪個角色。使用您叢集的名稱取代 *my-cluster*。

```
aws eks describe-cluster --name my-cluster --query cluster.roleArn --output text | cut
 -d / -f 2
```

範例輸出如下。

```
eksClusterRole
```

解決方案

建立具有相同名稱的新叢集 IAM 角色。

問題

已刪除叢集建立期間指定的子網路 – 即叢集建立期間指定要與叢集搭配使用的子網路。您可以使用以下命令，查看指定了哪些子網路。使用您叢集的名稱取代 *my-cluster*。

```
aws eks describe-cluster --name my-cluster --query cluster.resourcesVpcConfig.subnetIds
```

範例輸出如下。

```
[
"subnet-EXAMPLE1",
"subnet-EXAMPLE2"
]
```

解決方案

確認您的帳戶中是否存在子網路 ID。

```
vpc_id=$(aws eks describe-cluster --name my-cluster --query
 cluster.resourcesVpcConfig.vpcId --output text)
aws ec2 describe-subnets --filters "Name=vpc-id,Values=$vpc_id" --query
 "Subnets[*].SubnetId"
```

範例輸出如下。

```
[
"subnet-EXAMPLE3",
```

```
"subnet-EXAMPLE4"
]
```

如果輸出中傳回的子網路 ID 與建立叢集時指定的子網路 ID 不符，假如您希望 Amazon EKS 更新叢集，則需要變更叢集使用的子網路。這是因為如果您在建立叢集時指定了兩個以上的子網路，Amazon EKS 會隨機選取您指定在其中建立新彈性網路介面的子網路。這些網路介面可讓控制平面與節點進行通訊。如果叢集選取的子網路不存在，Amazon EKS 將不會更新叢集。您無法控制 Amazon EKS 會從您於建立叢集時指定的子網路中選擇哪些在其中建立新網路介面。

當您啟動叢集的 Kubernetes 版本更新，該更新可能會因相同的原因而失敗。

問題

叢集建立期間指定的安全群組已刪除 – 如果您在叢集建立期間指定了安全群組，您可以使用下列命令查看其 ID。使用您叢集的名稱取代 *my-cluster*。

```
aws eks describe-cluster --name my-cluster --query
 cluster.resourcesVpcConfig.securityGroupIds
```

範例輸出如下。

```
[
    "sg-EXAMPLE1"
]
```

如果傳回 []，若在建立叢集時未指定安全群組，而缺少安全群組並不是問題所在。如果傳回安全群組，則請確認安全群組位於您的帳戶中。

解決方案

請確認這些安全性群組是否位於您的帳戶中。

```
vpc_id=$(aws eks describe-cluster --name my-cluster --query
 cluster.resourcesVpcConfig.vpcId --output text)
aws ec2 describe-security-groups --filters "Name=vpc-id,Values=$vpc_id" --query
 "SecurityGroups[*].GroupId"
```

範例輸出如下。

```
[
"sg-EXAMPLE2"
```

```
]
```

如果輸出中傳回的安全群組 ID 與建立叢集時指定的安全群組 ID 不符，若您希望 Amazon EKS 更新該叢集，則需要變更叢集使用的安全群組。如果在建立叢集時指定的安全群組 ID 不存在，Amazon EKS 將不會更新叢集。

當您啟動叢集的 Kubernetes 版本更新，該更新可能會因相同的原因而失敗。

- 您在建立叢集時指定的每個子網路中，沒有至少六個 (雖然我們建議使用 16 個) 可用的 IP 位址。如果子網路中可用的 IP 位址不足，則需要釋出子網路中的 IP 位址，或是需要變更叢集使用的子網路，使用具有足夠 IP 位址的子網路。
- 您在建立叢集時啟用秘密加密，且您指定的 AWS KMS 金鑰已刪除。假如您希望 Amazon EKS 更新叢集，您必須建立新的叢集

叢集運作狀態常見問答集與包含解析路徑的錯誤代碼

Amazon EKS 可偵測 Amazon EKS 叢集以及叢集基礎結構的問題，並將問題存放在 EKS 叢集資源的運作狀態物件中。藉助叢集運作狀態資訊，您可以更快速地偵測、診斷並解決叢集問題，從而建立更安全且及時更新的應用程式環境。此外，由於必要的基礎結構或叢集組態出現問題，您可能無法升級至較新版本的 Kubernetes，也可能無法讓 Amazon EKS 在降級的叢集上安裝安全更新。Amazon EKS 可能需要 3 小時才能偵測到問題或偵測某個問題是否已解決。

維護 Amazon EKS 叢集的運作狀態是 Amazon EKS 及其使用者共同的責任。使用者負責 IAM 角色和 Amazon VPC 子網路的必備基礎設施，以及其他必須事先提供的必要基礎設施。Amazon EKS 偵測此基礎設施和叢集的組態變更。

若要在 Amazon EKS 主控台中存取叢集的運作狀態，請在透過 Amazon EKS 叢集詳細資訊頁面存取的可觀測性儀表板的叢集運作狀態問題索引標籤中，尋找名稱為運作狀態問題的資料表。也可以透過在 EKS API 中呼叫 `DescribeCluster`動作來使用此資料，例如從 AWS 命令列界面內。

這項功能有什麼用？

> 您可以提高對 Amazon EKS 叢集運作狀態的可見性、快速診斷和修正任何問題，而無需花時間偵錯或開啟 AWS 支援案例。例如：您意外刪除了 Amazon EKS 叢集的子網路，Amazon EKS 將無法建立跨帳戶網路介面和 Kubernetes AWS CLI 命令，例如 `kubectl` exec 或 `kubectl` log。這些操作將會失敗並顯示錯誤：`"Error from server: error dialing backend: remote error: tls: internal error."`。這時會出現內容為此的 Amazon EKS 運作狀態問題：`subnet-da60e280 was deleted: could not create network interface`。

此功能與其他 AWS 服務有何關聯或搭配？

IAM 角色和 Amazon VPC 子網路是叢集運作狀態對其進行問題偵測的其中兩項必備基礎設施。如果這些資源設定不正確，則此功能將傳回詳細錯誤資訊。

具有運作狀態問題的叢集會產生費用嗎？

是的。每個 Amazon EKS 叢集均以標準 Amazon EKS 定價計費。但叢集運作狀態功能是免費提供的。

此功能是否適用於 AWS Outposts 上的 Amazon EKS 叢集？

是，在 AWS 雲端中偵測到 EKS 叢集的叢集問題，包括 AWS Outpost 上的擴充叢集和 AWS Outpost 上的本機叢集。叢集運作狀態功能不會偵測 Amazon EKS Anywhere 或 Amazon EKS Distro (EKS-D) 的問題。

偵測到新問題時我是否會收到通知？

是。偵測到新的叢集運作狀態問題時， AWS 會傳送電子郵件和個人運作狀態儀表板通知。

主控台是否針對運作狀態問題警告我？

是。任何存在運作狀態問題的叢集都會在主控台頂部包含一則橫幅。

前兩欄是 API 回應值所需的內容。Health ClusterIssue 物件的第三個欄位是 Health ClusterIssue 物件，其傳回的值取決於問題類型。

代碼	訊息	ResourceIds	叢集可復原？
SUBNET_NOT_FOUND	我們找不到目前與叢集關聯的一個或多個子網路。請呼叫 Amazon EKS update-cluster-config API 以更新子網路。	子網路 ID	是
SECURITY_GROUP_NOT_FOUND	我們找不到目前與叢集關聯的一個或多個安全群組。呼叫 Amazon EKS update-cluster-config API 來更新安全群組。	安全群組 IDs	是

代碼	訊息	ResourceIds	叢集可復原？
IP_NOT_AVAILABLE	與叢集關聯的一個或多個子網路沒有足夠的可用 IP 地址供 Amazon EKS 執行叢集管理操作。請使用 Amazon EKS update-cluster-config API 釋放子網路中的地址或將其他子網路關聯到叢集。	子網路 ID	是
VPC_NOT_FOUND	我們找不到與叢集關聯的 VPC。您必須刪除並重新建立叢集。	VPC ID	否
ASSUME_ROLE_ACCESS_DENIED	叢集未使用 Amazon EKS 服務連結角色。我們無法擔任與叢集關聯的角色，從而無法執行所需的 Amazon EKS 管理操作。請檢查相應角色是否存在並具有所需的信任政策。	叢集 IAM 角色	是
PERMISSION_ACCESS_DENIED	叢集未使用 Amazon EKS 服務連結角色。與叢集關聯的角色未授予 Amazon EKS 執行所需管理操作需要的足夠許可。請檢查連接到叢集角色的政策以及是否套用了任何單獨的拒絕政策。	叢集 IAM 角色	是

代碼	訊息	ResourceIds	叢集可復原？
ASSUME_ROLE_ACCESS_DENIED_USING_SLR	我們無法擔任 Amazon EKS 叢集管理服務連結角色。請檢查相應角色是否存在並具有所需的信任政策。	Amazon EKS 服務連結角色	是
PERMISSION_ACCESS_DENIED_USING_SLR	Amazon EKS 叢集管理服務連結角色未授予 Amazon EKS 執行所需管理操作需要的足夠許可。請檢查連接到叢集角色的政策以及是否套用了任何單獨的拒絕政策。	Amazon EKS 服務連結角色	是
OPT_IN_REQUIRED	您的帳戶沒有 Amazon EC2 服務訂閱。請在帳戶設定頁面中更新帳戶訂閱。	N/A	是
STS_REGIONAL_ENDPOINT_DISABLED	相應 STS 區域端點已停用。請啟用相應端點，以讓 Amazon EKS 能夠執行所需的叢集管理操作。	N/A	是
KMS_KEY_DISABLED	與叢集相關聯的 AWS KMS 金鑰已停用。請重新啟用該金鑰以復原叢集。	KMS 金鑰 ARN	是
KMS_KEY_NOT_FOUND	我們找不到與您的叢集相關聯的 AWS KMS 金鑰。您必須刪除並重新建立叢集。	KMS 金鑰 ARN	否

代碼	訊息	ResourceIds	叢集可復原？
KMS_GRANT_REVOKED	與您的叢集相關聯的 AWS KMS 金鑰授權會遭到撤銷。您必須刪除並重新建立叢集。	KMS 金鑰 ARN	否
ETCD_DB_SIZE_EXCEEDED	您的 Amazon EKS 叢集已超過刻印資料庫大小限制。 etcd 是在 Kubernetes 控制平面中執行和維護叢集組態和狀態的鍵值資料存放區。為了防止您的叢集進入降級狀態，請移除不必要的 Kubernetes 物件，以縮減已壓縮的資料庫大小。如需識別和清除造成資料庫大小之物件的指導方針，請參閱管理 Amazon EKS 叢集上的壓縮資料庫大小。如果您在清除後持續遇到問題，請聯絡 AWS Support。	叢集 ARN	是

藉助開放原始碼專案，來延伸 Amazon EKS 功能

這些開放原始碼專案可延伸在 或外部執行的 Kubernetes 叢集功能 AWS，包括由 Amazon EKS 管理的叢集。

支援部署至 EKS 的軟體

檢閱 Amazon EKS 文件時，您將在整個程序及範例中看到各種開放原始碼工具與軟體的參考。這些工具包括 Kubernetes 指標伺服器與憑證管理器。

請注意，您選擇部署的任何第三方或開放原始碼軟體都超出 AWS 支援協議的範圍。使用 Kubernetes 的優勢包括活躍的開放原始碼社群。建議您與相關的開放原始碼社群及專案維護者直接合作，針對此類元件建立適當的支援通道。如需詳細資訊，請參閱與雲端原生運算基金會 (CNCF) 關聯的漸進式專案與孵化專案。

Kubernetes 生態系統包含眾多專案與元件，隨附不同層級的社群支援、回應時間，以及預期的使用案例。與 EKS 一起實作這些技術時，確認您了解每個元件的支援矩陣。

AWS 會維護我們整合到 EKS 控制平面的開放原始碼元件。這包括全面的安全通道，涵蓋針對我們分發的所有容器映像與二進位檔的建置驗證、漏洞掃描、驗證測試，以及修補程式管理。例如， AWS 負責 Kubernetes API 伺服器。Amazon EKS 服務水準協議涵蓋 Kubernetes API 伺服器。您可藉助 Amazon Web Services 支援方案，來解決 Kubernetes API 伺服器問題，或者獲取一般指引。

您需要仔細檢閱各種 Amazon EKS 附加元件提供的支援。 AWS 附加元件是唯一完全受 支援的 Amazon EKS 附加元件類型 AWS。 AWS Marketplace 附加元件主要受 AWS 合作夥伴支援。社群附加元件從 獲得基本生命週期支援 AWS。如需詳細資訊，請參閱附加元件支援。

無論類型為何，每個 EKS 附加元件都會從 EKS 獲得基本生命週期支援，包括 Marketplace 附加元件。基本的生命週期支援包括安裝與解除安裝附加元件。若要了解可用的 Amazon EKS 附加元件類型及關聯的支援層級相關詳細資訊，請參閱 Amazon EKS 附加元件的支援範圍。若要檢視 完全支援的附加元件 AWS，請參閱 Amazon Web Services 附加元件。

- 若要了解我們的安全實務與支援界限的相關詳細資訊，請參閱 Amazon EKS 中的安全性。
- 如需透過 Amazon EKS 附加元件提供的社群和 AWS 市場附加元件的詳細資訊，請參閱 EKS 附加元件支援。

管理工具

Amazon EKS 和 Kubernetes 叢集的相關管理工具。

eksctl

`eksctl` 是一種簡單的 CLI 工具，可在 Amazon EKS 上建立叢集。

- 專案 URL
- 專案文件
- AWS 開放原始碼部落格：eksctl：使用一個命令的 Amazon EKS 叢集

AWS Kubernetes 的控制器

使用 Kubernetes 的 AWS 控制器，您可以直接從 Kubernetes 叢集建立和管理 AWS 資源。

提供 EKS 功能。

- 專案 URL
- AWS 開放原始碼部落格：適用於 AWS Kubernetes 的 service Operator 現已推出

kro (Kube Resource Orchestrator)

kro 可讓您建立自訂 Kubernetes APIs，將多個資源組成更高階的抽象概念。平台團隊可以使用護欄定義可重複使用的模式，而應用程式團隊則使用簡單的高階 APIs 來佈建和管理資源。

提供 EKS 功能。

- 專案 URL
- 專案文件

Argo CD

Argo CD 是一種適用於 Kubernetes 的宣告式 GitOps 持續交付工具。它會持續監控您的 Git 儲存庫，並自動同步叢集的變更。

提供 EKS 功能。

- 專案 URL
- 專案文件

Flux CD

Flux 是一個工具，您可以用於使用 Git 來管理您的叢集組態。其使用叢集中的運算子來觸發 Kubernetes 內部的部署。若要了解運算子相關詳細資訊，請參閱 GitHub 上的 OperatorHub.io。

- 專案 URL
- 專案文件

Kubernetes 專用 CDK

使用適用於 Kubernetes 的 CDK (cdk8s)，您可以使用熟悉的程式設計語言定義 Kubernetes 應用程式和元件。cdk8s 應用程式會合成為標準的 Kubernetes 資訊清單，可套用至任何 Kubernetes 叢集。

- 專案 URL
- 專案文件
- AWS 容器部落格：介紹 cdk8s+：Kubernetes 物件的意圖驅動 APIs

聯網

Amazon EKS 和 Kubernetes 叢集的相關聯網專案。

Kubernetes 專用 Amazon VPC CNI 外掛程式

Amazon EKS 透過 Kubernetes 專用 Amazon VPC CNI 外掛程式支援原生 VPC 聯網。外掛程式會將 VPC 中的 IP 位址指派給每個 Pod。

- 專案 URL
- 專案文件

Kubernetes 的AWS Load Balancer控制器

The AWS Load Balancer 控制器可協助管理 Kubernetes 叢集的 AWS Elastic Load Balancer。它透過佈建 AWS Application Load Balancer 來滿足 Kubernetes Ingress 資源。它透過佈建 AWS Network Load Balancer 來滿足 Kubernetes 服務資源。

- 專案 URL
- 專案文件

ExternalDNS

ExternalDNS 會將公開的 Kubernetes 服務和輸入與 DNS 供應商同步，包括 Amazon Route 53 和 AWS Service Discovery。

- 專案 URL
- 專案文件

機器學習

Amazon EKS 和 Kubernetes 叢集的相關機器學習專案。

Kubeflow

適用於 Kubernetes 的機器學習工具組。

- 專案 URL
- 專案文件
- AWS 開放原始碼部落格：Amazon EKS 上的 Kubeflow

Auto Scaling

Amazon EKS 和 Kubernetes 叢集的相關自動調整規模專案。

Cluster Autoscaler

Cluster Autoscaler 是一種工具，可根據 CPU 和記憶體壓力，自動調整 Kubernetes 叢集的大小。

- 專案 URL
- 專案文件
- Amazon EKS 工作坊：Cluster Autoscaler

Karpenter

Karpenter 是專為實現靈活性、效能與簡單性而打造的 Kubernetes 節點自動擴展器。

- 專案 URL
- 專案文件
- Amazon EKS 工作坊：Karpenter

Escalator

Escalator 是適用於 Kubernetes 的批次或任務最佳化水平自動擴展工具。

- 專案 URL
- 專案文件

監控

Amazon EKS 和 Kubernetes 叢集的相關監控專案。

Prometheus

Prometheus 是一種開放原始碼系統監控和警示工具組。

- 專案 URL
- 專案文件
- Amazon EKS 研討會： https://eksworkshop.com/intermediate/240_monitoring/

持續整合 / 持續部署

Amazon EKS 和 Kubernetes 叢集的相關必要 CI/CD 專案。

Jenkins X

Amazon EKS 和 Kubernetes 叢集上適用於現代雲端應用程式的 CI/CD 解決方案。

- 專案 URL
- 專案文件

了解 Amazon EKS 新功能與藍圖

若要了解 Amazon EKS 新功能，請捲動至 AWS 最新消息頁面上的最新消息摘要。您也可以在 GitHub 上檢視藍圖，讓您了解即將推出的功能和優先順序，以便規劃未來使用 Amazon EKS 的方式。您可以向我們提供有關藍圖優先順序的直接意見回饋。

文件歷史紀錄

下表說明部分主要更新及《Amazon EKS 使用者指南》的新內容。若要在此表格有新條目時接收通知，您可以使用 RSS 閱讀器訂閱以下 URL：

```
https://docs.aws.amazon.com/eks/latest/userguide/doc-history.rss
```

變更	描述	日期
AWS 受管政策更新	在 中新增`wafv2:ListWebACLs` 了許可`AmazonEKSLoadBalancingPolicy` ，允許 Amazon EKS 自動模式控制器在協調 Application Load Balancer 關聯時列出 AWS WAF Web ACLs。已將 `gateway.eks.amazonaws.com.alb/stack` 、`gateway.eks.amazonaws.com.nlb/stack` 、 `gateway.eks.amazonaws.com.alb/resource` 和 `gateway.eks.amazonaws.com.nlb/resource` 標籤金鑰新增至 中的資源標記允許清單`AmazonEKSLoadBalancingPolicy` 。這可讓 Amazon EKS Auto Mode Controller 管理 Kubernetes Gateway API (ALB 和 NLB) 工作負載 AWS 的資源。	2026 年 9 月 2 日

確定性 Argo CD 功能端點	記錄 Argo CD 端點的 EKS 功能格式。端點主機名稱是確定性的，因此您可以從功能名稱、叢集名稱、 AWS 帳戶 ID 和 AWS 區域建構它，或從 的 `serverUrl` 欄位擷取它`describe-capability` 。	2026 年 8 月 24 日
AWS 叢集 OIDC 端點的 PrivateLink	Amazon EKS 現在支援叢集 OIDC 探索/JWKS 端點 (`com.amazonaws.region-code.oidc-eks`) 的 AWS PrivateLink，可讓您從 VPC 私下存取服務帳戶 (IRSA) 的 IAM 角色所使用的端點，而無需網際網路輸出。	2026 年 7 月 28 日
EFA 支援 EKS Auto 模式的靜態容量	新增在 EKS Auto Mode NodeClass 資源上設定 EFA 網路介面的支援。您現在可以在執行個體啟動時靜態定義高效能分散式訓練和推論工作負載的網路介面。	2026 年 7 月 21 日
服務連結角色更新	`AmazonEKSPodIdentityPolicy` 已新增至 中的存取政策清單`AmazonEKSServiceRolePolicy` ，服務連結角色可以與叢集存取項目建立關聯和取消關聯。此外，Amazon EKS 可以為在 EKS 叢集上執行的工作負載管理 Pod 身分憑證。	2026 年 6 月 26 日

控制平面輸出路由	新增客戶路由控制平面輸出 (controlPlaneEgressMode=CUSTOMER_ROUTED) 的支援，可讓您管理從叢集控制平面到 VPC 資源的網路路徑。	2026 年 6 月 18 日
服務連結角色更新	在 中新增elasticloadbalancing:SetRulePriorities 許可AmazonEKSLoadBalancingPolicy ，以允許 Amazon EKS Auto Mode Controller 為 ALB 設定指定接聽程式規則的優先順序。在 中新增AmazonEKSLoadBalancingPolicy 描述 ipam 集區的ec2:DescribeIpamPools 許可	2026 年 6 月 15 日
Kubernetes 版本 1.36	已將 Kubernetes 版本1.36新增至 Amazon EKS。	2026 年 6 月 2 日
AWS 受管政策更新	除了現有 ec2:AttachVolume 和 eks:eks-cluster-name 標籤之外，更新AmazonEBSCSIDriverEKSClusterScopedPolicy 以授權 eks:cluster-name 和 ec2:DetachVolume 執行個體上加上 ebs.csi.aws.com/cluster-name 標籤。這增加了與遵循 EKS 控制器標記標準的控制器的相容性，包括開放原始碼 Karpenter 和 Amazon EKS Auto 模式。	2026 年 5 月 28 日

AWS 受管政策更新	新增許可至 AmazonEKSComputePolicy ，以允許 Amazon EKS Auto 模式從您帳戶中的可中斷容量保留啟動執行個體。	2026 年 5 月 18 日
AWS 受管政策更新	新增了新的 AWS 受管政策：AmazonEKSBlockStoragePolicyV2	2026 年 5 月 12 日
AWS 受管政策更新	新增 API 動作ec2:DescribeInstanceTypes 至 AmazonEBSCSIDriverPolicy ，讓 EBS CSI 驅動程式在執行時間動態擷取磁碟區限制/卡片資訊。	2026 年 5 月 12 日
服務連結角色更新	在 中新增elasticloadbalancing:DescribeLoadBalancers 許可AmazonEKSLoadBalancingPolicy ，以允許 Amazon EKS 自動模式控制器描述 LoadBalancer	2026 年 4 月 27 日
AWS Outposts 上本機叢集1.32的 Kubernetes 版本	新增對 AWS Outposts 上本機 Amazon EKS 叢集的 Kubernetes 版本1.32支援。	2026 年 4 月 23 日
服務連結角色更新	在 中新增elasticloadbalancing:DescribeCapacityReservation 許可AmazonEKSLoadBalancingPolicy ，以允許 Amazon EKS Auto Mode 控制器描述 LoadBalancer 容量單位	2026 年 4 月 20 日

服務連結角色更新	在 中新增 `ec2:DescribeInstanceStatus` 和 `iam:ListInstanceProfiles` 許可AmazonEKS `ServiceRolePolicy` 。這些許可允許 Amazon EKS Auto Mode 探索受管執行個體的 EC2 運作狀態事件，並清除受管執行個體設定檔	2026 年 4 月 20 日
AWS 受管政策更新	新增許可至 AmazonEKS `ComputePolicy` ，以允許 Amazon EKS Auto 模式在帳戶中的 EC2 置放群組中啟動執行個體。	2026 年 4 月 17 日
新的 AWS 受管政策	引進 AmazonEBSCSIDriverPolicyV2 和 AmazonEBSCSIDriverEKSClusterScopedPolicy 作為 更嚴格的替代方案AmazonEBSCSIDriverPolicy 。會將 EBS CSI 驅動程式AmazonEBSCSIDriverPolicyV2 範圍限定為標記為 `ebs.csi.aws.com/cluster` 的磁碟區和快照true。會使用 `ebs.csi.aws.com/cluster-name` 標籤將驅動程式AmazonEBSCSIDriverEKSClusterScopedPolicy 範圍限定為屬於特定 EKS 叢集的磁碟區和快照。	2026 年 4 月 16 日

AWS 受管政策更新	新增許可至 AmazonEKS ServiceRolePolicy ，以允許 Amazon EKS Auto Mode 在您的帳戶中的 EC2 置放群組中啟動執行個體。	2026 年 4 月 15 日
AWS 受管政策更新	已更新 中的 shield:CreateProtection 和 shield:DeleteProtection 許可AmazonEKSLoadBalancingPolicy 。這可讓 Amazon EKS 自動模式控制器新增防護。	2026 年 4 月 14 日
受管節點群組的暖集區支援	Amazon EKS 受管節點群組現在支援 Amazon EC2 Auto Scaling 暖集區。暖集區會維護預先初始化的 EC2 執行個體集區，這些執行個體已準備好在向外擴展時快速加入您的叢集，減少開機時間較長之應用程式的延遲。	2026 年 4 月 7 日
AWS 受管政策更新	在 中新增elasticloadbalancing:ModifyIpPools 和更新elasticloadbalancing:RegisterTargets 許可AmazonEKSLoadBalancingPolicy 。這可讓 Amazon EKS 自動模式控制器支援設定 ALB 的 IPAM 集區，並分別支援多叢集和自訂目標群組繫結關係	2026 年 3 月 20 日

AWS 受管政策更新	已將 autoscaling:PutWarmPool 、 autoscaling:DeleteWarmPool 和 autoscaling:DescribeWarmPool 許可新增至AWSServiceRoleForAmazonEKSNodegroup 政策。這可讓 Amazon EKS 受管節點群組在整個節點群組生命週期中管理基礎 ASG 暖集區資源。	2026 年 2 月 17 日
AWS 受管政策更新	在 中新增ec2:ModifyNetworkInterfaceAttribute 許可AmazonEKSNetworkingPolicy 。這可讓 Amazon EKS 自動模式控制器修改與 EC2 執行個體相關的網路介面屬性	2026 年 2 月 3 日
AWS 受管政策更新	針對 中的iam:GetInstanceProfile 許可，移除目標執行個體描述檔名稱中的「eks」字首需求 AmazonEKSServiceRolePolicy 。這可讓 Amazon EKS Auto Mode 驗證和使用 NodeClasses 中的自訂執行個體描述檔，而不需要「eks」命名字首。	2026 年 2 月 2 日
Kubernetes 版本 1.35	新增 Kubernetes 版本 1.35 支援新的叢集和版本升級。	2026 年 1 月 27 日

AWS 受管政策更新	新增AmazonEBSCSIDriverPolicy 允許 EBS CSI 驅動程式直接鎖定 EBS 快照的ec2:LockSnapshot 許可。	2026 年 1 月 15 日
Amazon EKS 功能	Amazon EKS 現在提供具有 EKS 功能的全受管叢集功能，支援 Argo CD 的宣告式持續部署、支援 Kubernetes 的 AWS 控制器的 AWS 資源管理，以及支援 Kube Resource Orchestrator 的 Kubernetes 資源合成和協同運作。	2025 年 11 月 30 日
新的 AWS 受管政策	Amazon EKS 已發佈新的受管政策AmazonEKSMCPReadOnlyAccess ，以在 Amazon EKS MCP Server 中啟用唯讀工具，以實現可觀測性和故障診斷。如需詳細資訊，請參閱 AWS 受管政策的 Amazon EKS 更新。	2025 年 11 月 21 日
網路可觀測性	Amazon EKS 現在透過效能監控和工作負載流量可見性，提供增強的容器網路可觀測性。	2025 年 11 月 19 日
AWS 受管政策更新	新增AmazonEBSCSIDriverPolicy 允許 EBS CSI 驅動程式直接複製 EBS 磁碟區的ec2:CopyVolumes 許可。	2025 年 11 月 17 日

AWS 受管政策更新	新增 ec2:DescribeRouteTables 和 ec2:DescribeNetworkAcls 許可至 AmazonEKSServiceRolePolicy 。這可讓 Amazon EKS 在叢集洞察中偵測混合節點的 VPC 路由表和網路 ACLs 的組態問題。	2025 年 10 月 22 日
服務連結角色更新	新增ssmmessages:OpenDataChannel 許可至 AmazonEKSConnectorServiceRolePolicy	2025 年 10 月 15 日
Kubernetes 版本 1.34	新增 Kubernetes 版本 1.34 支援新的叢集和版本升級。	2025 年 10 月 2 日
可設定的節點自動修復	Amazon EKS 現在可以更精細地控制節點自動修復行為。	2025 年 9 月 22 日
重新整理叢集洞察	您現在可以手動重新整理叢集洞察。	2025 年 8 月 27 日
AWS 受管政策更新	新增了許可至 AmazonEKSServiceRolePolicy 。此角色可連接全新存取政策 AmazonEKSEventPolicy 。ec2:DeleteLaunchTemplate 與 ec2:TerminateInstances 的限制許可。	2025 年 8 月 26 日

預防跨服務混淆代理人	新增了包含範例信任政策的主題，您可以套用於預防跨服務混淆客服人員。Amazon EKS 接受 EKS 叢集角色信任政策中的 `aws:SourceArn` 和 `aws:SourceAccount` 條件。	2025 年 8 月 19 日
Amazon EKS 平台版本更新	這是具有安全性修正和增強功能的新平台版本。這包括 Kubernetes `1.33.2`、`1.32.6`、`1.31.10` 和 `1.30.14` 的新修補程式版本。	2025 年 7 月 30 日
VPC CNI 多 NIC 功能，適用於多主目錄 Pod	Amazon EKS 將多主目錄 Pod 加入 VPC CNI。現在您可設定工作負載，並透過 VPC CNI 將 EC2 執行個體上每個 NIC 的 IP 位址指派給每個 Pod。應用程式可同時連線，以便利用每個 NIC 的頻寬。在與節點相同的子網路與安全群組中，設定每一個網路介面。先前，您需要使用 Multus CNI 來執行多個其他 CNI 以建立多主目錄 Pod。文件和執行此作業的步驟已移至 https://docs.aws.amazon.com/eks/latest/userguide/pod-multus.html。	2025 年 7 月 15 日
VPC CNI 疑難排解內容更新	已擴展 VPC CNI 中 Kubernetes 網路政策的疑難排解頁面。新增了 CRD 和 RBAC 許可，以及 12 個已知問題。	2025 年 6 月 30 日

Amazon EKS 平台版本更新	這是具有安全性修正和增強功能的新平台版本。這些平台版本中沒有新的 Kubernetes 修補程式版本。	2025 年 6 月 26 日
AWS 受管政策更新	新增了 ssmmessages:OpenDataChannel 許可至 AmazonEKSLocalOutpostServiceRolePolicy 。	2025 年 6 月 26 日
AWS 受管政策更新	新增對 AmazonEKSServiceRolePolicy 和 AmazonEKSComputePolicy 的許可，以允許 Amazon EKS 自動模式使用您帳戶中的 EC2 隨需容量保留來啟動執行個體。此外，已將許可新增至 AmazonEKSComputePolicy ，讓 Amazon EKS 代表您建立 EC2 Spot 服務連結角色。	2025 年 6 月 20 日
受管政策更新	新增 AmazonEKSDashboardConsoleReadOnly 政策。	2025 年 6 月 19 日
Amazon EKS 自動模式更新至 NodeClass	自動模式節點的 NodeClass 範本新增了個別 Pod 子網路的組態。此功能新增了選用金鑰 podSubnetSelectorTerms 和 podSecurityGroupSelectorTerms ，可用於設定 Pod 的子網路與安全群組。	2025 年 6 月 13 日

使用 EKS Pod 身分識別來定位次要和跨帳戶角色	Amazon EKS 將目標 IAM 角色新增至 EKS Pod 身分識別，以實現自動角色鏈結。您可以使用此功能自動擔任另一個帳戶中的角色，且 EKS Pod 身分識別會輪替暫時憑證。每個 Pod 身分識別關聯必須在同一個帳戶中有一個 IAM 角色來先擔任，然後使用該角色來擔任目標角色。	2025 年 6 月 11 日
Amazon EKS 平台版本更新	這是具有安全性修正和增強功能的新平台版本。這些平台版本中沒有新的 Kubernetes 修補程式版本。	2025 年 6 月 11 日
Amazon EKS AWS 區域擴展	Amazon EKS 現已在亞太區域（台北）(ap-east-2) AWS 區域提供。	2025 年 6 月 6 日
新 IPv6 叢集的雙重堆疊公有端點的 IPv6 存取控制	Amazon EKS 將 IPv6 CIDR 區塊新增至新 IPv6 叢集的公有叢集端點存取控制。先前，即使對於雙重堆疊叢集端點，您也只能新增 IPv4 CIDR 區塊以允許流量到公有叢集端點。	2025 年 6 月 5 日
Amazon EKS 平台版本更新	這是具有安全性修正和增強功能的新平台版本。這包括 Kubernetes 1.32.5、1.31.9 和 1.30.13 的新修補程式版本。	2025 年 5 月 30 日

EKS 混合節點的新叢集洞察	Amazon EKS 新增了新的叢集洞察，用於檢查混合節點的組態。這些洞見檢查將就內部部署節點和 Pod 的問題以及叢集的遠端網路組態向您發出警告。	2025 年 5 月 29 日
Kubernetes 版本 1.33	新增 Kubernetes 版本 1.33 支援新的叢集和版本升級。	2025 年 5 月 29 日
Amazon FSx CSI 驅動程式的附加元件支援	您現在可以使用 AWS 管理主控台、 AWS CLI 和 API 來管理 Amazon FSx CSI 驅動程式。	2025 年 5 月 23 日
在主控台中編輯 Prometheus 湊集器	您現在可在 Amazon EKS 主控台中編輯 Amazon Managed Service for Prometheus 湊集器。	2025 年 5 月 22 日
受管政策更新	新增 AmazonEKSDashboardServiceRolePolicy 政策。	2025 年 5 月 21 日
Amazon EKS 平台版本更新	這是具有安全性修正和增強功能的新平台版本。這些平台版本中沒有新的 Kubernetes 修補程式版本。	2025 年 5 月 16 日
Amazon FSx for Lustre 效能的新頁面	新增了有關最佳化 Amazon FSx for Lustre 效能詳細資訊的新主題。	2025 年 5 月 2 日

Amazon EKS 自動模式更新至 NodeClass	自動模式節點的 NodeClass 範本新增了正向網路代理的組態。此功能新增選用金鑰 advancedNetworking，可用於設定您的 HTTPS 代理。	2025 年 4 月 30 日
適用於混合節點的 Bottlerocket	Bottlerocket 現已適用於 EKS 混合節點。	2025 年 4 月 29 日
Amazon EKS 平台版本更新	這是具有安全性修正和增強功能的新平台版本。這些平台版本中沒有新的 Kubernetes 修補程式版本。	2025 年 4 月 29 日
混合聯網的新概念頁面	新增了 EKS 混合節點概念的頁面。這些頁面透過圖表詳細介紹了內部部署和雲端聯網。	2025 年 4 月 18 日
AWS 受管政策更新	新增許可至 AmazonEKS ClusterPolicy ，以允許 Amazon EKS 清除 VPC CNI 建立的彈性網路介面。此動作很必要，這樣一來 EKS 就能在 VPC CNI 非預期退出的情況下，清理留下的彈性網路介面。	2025 年 4 月 16 日
AWS 受管政策更新	在 AmazonEKSServiceRolePolicy 中新增了許可，允許 EKS AI/ML 客戶將輸出規則新增至預設的 EKS 叢集安全群組。	2025 年 4 月 14 日
Outposts 上的本機叢集現在可使用 Kubernetes AWS 1.31 版	您現在可以使用 Kubernetes 1.31 版在 AWS Outposts 上建立 Amazon EKS 本機叢集。	2025 年 4 月 8 日

EKS 混合節點的節點運作狀態	您可以從版本 1.2.0-eks-build.1 開始在混合節點上使用 eks-node-monitoring-agent 。將 eks-node-monitoring-agent 作為 Amazon EKS 附加元件執行，以偵測並顯示運作狀態問題。	2025 年 3 月 31 日
現有叢集的 EKS 混合節點	您現在可以新增、變更或移除現有叢集的混合節點組態。先前，您只能在建立新叢集時將混合節點組態新增至新叢集。透過 Amazon EKS 混合節點，您可以使用內部部署和邊緣基礎設施做為 Amazon EKS 叢集中的節點。 AWS 管理 Amazon EKS 叢集的 AWS 託管 Kubernetes 控制平面，並管理在內部部署或邊緣環境中執行的混合節點。	2025 年 3 月 31 日
復原：透過叢集洞察防止意外升級	Amazon EKS 已臨時復原一項功能，若發生特定叢集洞察問題，會要求您使用 --force 標誌來升級叢集。如需詳細資訊，請參閱 GitHub 上對更新叢集版本強制執行升級洞察的臨時復原。	2025 年 3 月 28 日
Bottlerocket FIPS AMI	Bottlerocket FIPS AMI 現在可在標準受管節點群組中使用。	2025 年 3 月 27 日
AWS 受管政策更新	在 AmazonEKSServiceRolePolicy 中新增了許可，以允許 Amazon EKS 終止由自動模式建立的 EC2 執行個體。	2025 年 2 月 28 日

受管節點群組的更新策略	您現在可以使用更新策略來設定受管節點群組的版本更新程序。這引入了在建立新的節點之前終止節點的最低更新策略，這在容量受限的環境中非常有用。預設更新策略會繼續現有的行為。	2025 年 1 月 27 日
Kubernetes 版本 1.32	新增 Kubernetes 版本 1.32 支援新的叢集和版本升級。	2025 年 1 月 23 日
Amazon EKS AWS 區域擴展	Amazon EKS 現已在亞太區域 (泰國) 區域 (ap-southeast-7) 和墨西哥 (中部) (mx-central-1) AWS 區域提供。這兩個區域均不提供 EKS API 的 EKS 自動模式和 VPC 端點。	2025 年 1 月 14 日
AWS 受管政策更新	在 AmazonEBSCSIDriverPolicy 中新增了多個許可，允許 Amazon EBS CSI 驅動程式還原所有快照、在 EBS 磁碟區上啟用快速快照還原 (FSR)，以及修改磁碟區上的標籤。	2025 年 1 月 13 日
AWS 受管政策更新	新增了許可至 AmazonEKSLoadBalancingPolicy 。	2024 年 12 月 26 日

更新了叢集洞察	Amazon EKS 升級洞見現在將就更多叢集運作狀態和版本相容性問題發出警告。它可偵測不同 Kubernetes 和 Amazon EKS 元件之間的問題，例如 kubelet、kube-proxy 和 Amazon EKS 附加元件。	2024 年 12 月 20 日
節點監控代理程式與自動修復	您可以使用新的 eks-node-monitoring-agent 作為 Amazon EKS 附加元件執行，以偵測並顯示運作狀態問題。您也可啟用節點自動修復，以便在偵測到問題時自動更換節點。	2024 年 12 月 16 日
Amazon EKS 混合節點	您現在可以在內部部署節點，並連接到 Amazon EKS 叢集。透過 Amazon EKS 混合節點，您可以使用內部部署和邊緣基礎設施做為 Amazon EKS 叢集中的節點。 AWS 管理 Amazon EKS 叢集的 AWS託管 Kubernetes 控制平面，並管理在內部部署或邊緣環境中執行的混合節點。	2024 年 12 月 1 日
Amazon EKS 自動模式	Amazon EKS 自動模式在 AWS上全面自動化 Kubernetes 叢集基礎結構的管理，包括運算、儲存和聯網。它透過自動佈建基礎設施、選取最佳運算執行個體、動態擴展資源、持續最佳化成本、修補作業系統，以及與 AWS 安全服務整合，來簡化 Kubernetes 管理。	2024 年 12 月 1 日

Amazon EKS 平台版本更新	這是具有安全性修正和增強功能的新平台版本。這包括 Kubernetes 1.31.2、1.30.6、1.29.10 和 1.28.15 的新修補程式版本。	2024 年 11 月 22 日
AWS 受管政策更新	更新了 AWSServiceRoleForAmazonEKSNodegroup ，以適用於中國區域。	2024 年 11 月 22 日
Kubernetes 1.30 版現已適用於 AWS Outposts 上的本機叢集	您現在可以使用 Kubernetes 1.30 版在 AWS Outposts 上建立 Amazon EKS 本機叢集。	2024 年 11 月 21 日
AWS 受管政策更新	EKS 已更新 AWS 受管政策 AmazonEKSLocalOutpostClusterPolicy 。新增ec2:DescribeAvailabilityZones 許可，讓叢集控制平面上的 AWS Cloud Controller Manager 可以識別每個節點所在的可用區域。	2024 年 11 月 21 日
使用 FIPS 140-3 的 Bottlerocket AMI	可使用預先設定為利用 FIPS 140-3 驗證加密模組的 Bottlerocket AMI。這包括 Amazon Linux 2023 核心加密 API 密碼編譯模組和 AWS-LC 密碼編譯模組。	2024 年 11 月 20 日

AWS 受管政策更新	更新了 AWSServiceRoleForAmazonEKSNodegroup 政策，以允許透過 Amazon EKS 受管節點群組建立的執行個體的 ec2:RebootInstances 。限制了針對 Amazon EC2 資源的 ec2:CreateTags 許可。	2024 年 11 月 20 日
可觀測性儀表板	可觀測性儀表板有助於您快速進行偵測、故障診斷及修復問題。在 AWS/EKS 命名空間中還有新的 CloudWatch 代售指標可用。	2024 年 11 月 18 日
AWS 受管政策更新	EKS 已更新 AWS 受管政策 AmazonEKSServiceRolePolicy 。新增了針對下列各項的許可：EKS 存取政策、負載平衡器管理，以及自動化叢集資源清理。	2024 年 11 月 16 日
在主控台中為支援 EKS Pod 身分識別的附加元件建立新角色	使用主控台建立或更新支援 EKS Pod 身分識別的附加元件時，有新的步驟，您可以在其中自動產生具有適當名稱、角色政策和附加元件信任政策的 IAM 角色。	2024 年 11 月 15 日
AWS Local Zones 中的受管節點群組	受管節點群組現在可以在 AWS Local Zones 中建立。	2024 年 11 月 15 日
提供新指標	在 API 群組 metrics.eks.amazonaws.com 下有新指標可用。	2024 年 11 月 11 日

AWS 受管政策更新	EKS 已更新 AWS 受管政策 AmazonEKSComputePolicy 。更新了針對 iam:AddRoleToInstanceProfile 動作的資源許可。	2024 年 11 月 7 日
AWS 受管政策更新	EKS 新增了新的 AWS 受管政策： AmazonEKSComputePolicy	2024 年 11 月 1 日
AWS 受管政策更新	新增了許可至 AmazonEKSClusterPolicy 。新增了 ec2:DescribeInstanceTopology 許可，以允許 Amazon EKS 將拓撲資訊作為標籤連接至節點。	2024 年 11 月 1 日
AWS 受管政策更新	EKS 新增了新的 AWS 受管政策： AmazonEKSBlockStoragePolicy	2024 年 10 月 30 日
AWS 受管政策更新	EKS 新增了新的 AWS 受管政策： AmazonEKSLoadBalancingPolicy	2024 年 10 月 30 日
AWS 受管政策更新	新增了 cloudwatch:PutMetricData 許可至 AmazonEKSServiceRolePolicy ，以允許 Amazon EKS 發布指標至 Amazon CloudWatch。	2024 年 10 月 29 日
AWS 受管政策更新	EKS 新增了新的 AWS 受管政策： AmazonEKSNetworkingPolicy	2024 年 10 月 28 日

新 IPv6 叢集的雙堆疊端點	使用雙堆疊的 eks-cluster. *region*.api.aws 端點連線至新的 IPv6 叢集。當您描述這些叢集時會傳回此端點。在 IPv4、IPv6 或雙重堆疊環境中的 kubectl 和其他 Kubernetes API 用戶端可以解析並連接到這些公有或私有叢集的端點。	2024 年 10 月 21 日
AWS 受管政策更新	在中國區域的 AWSServiceRoleForAmazonEKSNodegroup 中新增了 autoscaling:ResumeProcesses 、autoscaling:SuspendProcesses 及相關許可，以與 EKS 的 Amazon 應用程式復原控制器整合。對其他區域沒有變更。	2024 年 10 月 21 日
AL2023 加速 AMI	您現在可以針對以 AL2023 為基礎的 AMIs 使用加速NVIDIA和 AWS Neuron 執行個體。	2024 年 10 月 11 日
新來源格式	我們已切換到具有一些版面配置變更的新來源格式。我們正在解決暫時性的次要格式化問題。	2024 年 10 月 10 日

AWS 受管政策更新	新增了許可至 AmazonEKS ServicePolicy 和 AmazonEKSServiceRo lePolicy 。新增了 ec2:GetSecurityGro upsForVpc 及關聯的標籤許可，以允許 EKS 讀取安全群組資訊及更新相關標籤。	2024 年 10 月 10 日
AWS 受管政策更新 - 新政策	EKS 新增了新的 AWS 受管政策。	2024 年 10 月 3 日
Kubernetes 版本 1.31	新增 Kubernetes 版本 1.31 支援新的叢集和版本升級。	2024 年 9 月 24 日
AWS 受管政策更新 - 更新現有政策	Amazon EKS 已更新現有的 AWS 受管政策。	2024 年 8 月 21 日
Kubernetes 1.29 版現已可用於 AWS Outposts 上的本機叢集	您現在可以使用 Kubernetes 1.29 版在 AWS Outposts 上建立 Amazon EKS 本機叢集。	2024 年 8 月 20 日
EKS Pod Identity in AWS GovCloud （美國）	Amazon EKS Pod 身分識別會將 IAM 角色和 Kubernetes 服務帳戶建立關聯。透過此功能，您不再需要提供延伸許可給節點 IAM 角色。如此一來，該節點上的 Pod 就可以呼叫 AWS APIs。與服務帳戶的 IAM 角色不同，EKS Pod 身分識別完全位於 EKS 內部；您不需要 OIDC 身分識別提供者。	2024 年 8 月 14 日
案例驅動的內容更新	我們重新命名並更新了整個指南中的主題，使其更具案例驅動性。	2024 年 8 月 9 日

Amazon EKS 的雙重堆疊 VPC 介面端點	您現在可以為 Amazon EKS 建立具有 IPv4 和 IPv6 IP 位址和 DNS 名稱的雙重堆疊 VPC 介面端點。	2024 年 8 月 7 日
具有 IPv6 位址的 Amazon EKS API 的新雙重堆疊端點點	用於建立和管理叢集的 EKS API，以及叢集的 OIDC 發行者 URL 具有新的雙重堆疊端點。Amazon EKS API 的新 DNS 名稱是 `eks.region.api.aws`，它會解析為 IPv4 位址和 IPv6 位址。新叢集具有新的雙重堆疊 OIDC 發行者 URL (`oidc-eks. region.api.aws`)。	2024 年 8 月 1 日
受管節點群組的容量區塊	您現在可將容量區塊用於受管節點群組。	2024 年 7 月 1 日
預設啟用 Auto Scaling 群組指標集合	Amazon EKS 受管節點群組現在已預設啟用 Amazon EC2 Auto Scaling 群組指標，且無需任何額外費用。先前，您必須執行幾個步驟才能啟用此功能。	2024 年 6 月 28 日
AWS 受管政策更新 - 更新現有政策	Amazon EKS 已更新現有的 AWS 受管政策。	2024 年 6 月 27 日
Kubernetes 版本 1.26	Kubernetes 版本 1.26 現在支援擴充。	2024 年 6 月 12 日
改進 AMI 資訊參考	我們改進了 AMI 資訊參考，特別是針對 Bottlerocket。	2024 年 6 月 12 日
Kubernetes 版本 1.30	新增 Kubernetes 版本 1.30 支援新的叢集和版本升級。	2024 年 5 月 23 日

CoreDNS Autoscaling	CoreDNS 自動擴展器將根據節點數和 CPU 核心數動態調整 EKS 叢集中 CoreDNS 部署的複本數量。此功能適用於 CoreDNS v1.9 及更高版本，以及 EKS 發行版本 1.25 及更高版本的最新平台版本。	2024 年 5 月 14 日
Amazon EKS 平台版本更新	這是具有安全性修正和增強功能的新平台版本。這包括 Kubernetes 1.29.4、1.28.9 和 1.27.13 的新修補程式版本。	2024 年 5 月 14 日
CloudWatch 容器洞見對 Windows 的支援	Amazon CloudWatch 可觀測性運算子附加元件現在也允許叢集中 Windows 工作節點上的容器洞見。	2024 年 4 月 10 日
Kubernetes 概念	新增了 Kubernetes 概念主題。	2024 年 4 月 5 日
重組存取和 IAM 內容	將與存取和 IAM 主題相關的現有頁面移至新區段，例如身分驗證組態映射、存取項目、Pod 身分識別和 IRSA。修訂概觀內容。	2024 年 4 月 2 日
Bottlerocket 作業系統對 Amazon S3 CSI 驅動程式的支援	適用於 Amazon S3 的掛載點 CSI 驅動程式現在與 Bottlerocket 相容。	2024 年 3 月 13 日
AWS 受管政策更新 - 更新現有政策	Amazon EKS 已更新現有的 AWS 受管政策。	2024 年 3 月 4 日
Amazon Linux 2023	Amazon Linux 2023 (AL2023) 是一款新的 Linux 作業系統，旨在為您的雲端應用提供安全、穩定且高效的運行環境。	2024 年 2 月 29 日

EKS Pod 身分識別和 IRSA 在 Kubernetes 1.29 中支援附屬	在 Kubernetes 1.29 中，附屬可在 Amazon EKS 叢集中使用。附屬容器支援使用服務帳戶的 IAM 角色或 EKS Pod 身分識別。有關附屬的更多資訊，請參閱 Kubernetes 文件中的附屬容器。	2024 年 2 月 26 日
Kubernetes 版本 1.29	新增 Kubernetes 版本 1.29 支援新的叢集和版本升級。	2024 年 1 月 23 日
完整版本：Amazon EKS Kubernetes 版本的延伸支援	Kubernetes 版本的延伸支援可讓您使用特定 Kubernetes 版本超過 14 個月。	2024 年 1 月 16 日
AWS 雲端中的 Amazon EKS 叢集運作狀態偵測	Amazon EKS 在叢集運作狀態中偵測 Amazon EKS 叢集以及叢集必備基礎結構的問題。您可以在 中檢視 EKS 叢集的問題， AWS 管理主控台 並在 EKS API 中檢視叢集health的 。這些問題是主控台偵測到且由主控台顯示之問題之外的問題。先前，叢集運作狀態僅適用於 AWS Outposts 上的本機叢集。	2023 年 12 月 20 日
叢集洞察	您現在可以從定期檢查取得有關您的叢集的建議。	2023 年 12 月 20 日
Amazon EKS AWS 區域擴展	Amazon EKS 現已在加拿大西部（卡加利）(ca-west-1) AWS 區域提供。	2023 年 12 月 20 日

您現在可以使用存取項目向 IAM 角色和使用者授予對叢集的存取權	在引進存取項目之前，您透過在 aws-auth ConfigMap 中新增項目來授予 IAM 角色和使用者對叢集的存取權。現在每個叢集都有一種存取模式，您可以根據排程轉變為使用存取項目。切換模式之後，您可以在 CLI、 AWS CloudFormation 和 AWS SDKs AWS 中新增存取項目來新增使用者。	2023 年 12 月 18 日
Amazon EKS 平台版本更新	這是具有安全性修正和增強功能的新平台版本。這包括 Kubernetes 1.28.4、1.27.8、1.26.11 和 1.25.16 的新修補程式版本。	2023 年 12 月 12 日
Amazon S3 CSI 驅動程式掛載點	您現在可以在 Amazon EKS 叢集上安裝適用於 Amazon S3 的掛載點 CSI 驅動程式。	2023 年 11 月 27 日
建立叢集時開啟 Prometheus 指標	在 中 AWS 管理主控台，您現在可以在建立叢集時開啟 Prometheus 指標。您還可以在可觀測性索引標籤中檢視 Prometheus 湊集器的詳細資訊。	2023 年 11 月 26 日

Amazon EKS Pod 身分識別	Amazon EKS Pod 身分識別會將 IAM 角色和 Kubernetes 服務帳戶建立關聯。透過此功能，您不再需要提供延伸許可給節點 IAM 角色。如此一來，該節點上的 Pod 就可以呼叫 AWS APIs。與服務帳戶的 IAM 角色不同，EKS Pod 身分識別完全位於 EKS 內部；您不需要 OIDC 身分識別提供者。	2023 年 11 月 26 日
AWS 受管政策更新 - 更新現有政策	Amazon EKS 已更新現有的 AWS 受管政策。	2023 年 11 月 26 日
CSI 快照控制器	您現在可以安裝 CSI 快照控制器，以便與相容的 CSI 驅動程式 (例如 Amazon EBS CSI 驅動程式) 搭配使用。	2023 年 11 月 17 日
ADOT 運算子主題重寫	Amazon EKS 附加元件支援 ADOT Operator 區段與 AWS Distro for OpenTelemetry 文件備援。我們將剩餘的重要資訊遷移到該資源中，以減少過時和不一致的資訊。	2023 年 11 月 14 日
適用於 Prometheus 的 CoreDNS EKS 附加元件支援指標	適用於 CoreDNS 的 EKS 附加元件的 `v1.10.1-eksbuild.5` 、`v1.9.3-eksbuild.9` 和 `v1.8.7-eksbuild.8` 版本，會在 `kube-dns` 服務中公開 CoreDNS 發布指標的連接埠。這樣可以更輕鬆地在監控系統中包含 CoreDNS 指標。	2023 年 11 月 10 日

Amazon EKS CloudWatch 可觀測性運算子附加元件	新增的 Amazon EKS CloudWatch 可觀測性運算子頁面。	2023 年 11 月 6 日
美國東部 (俄亥俄) 推出適用於自我管理 P5 執行個體的容量區塊	在美國東部 (俄亥俄)，您現在可以將容量區塊用於自我管理 P5 執行個體。	2023 年 10 月 31 日
叢集支援修改子網路和安全群組	您可以更新叢集，變更叢集使用的子網路和安全群組。您可以從 AWS 管理主控台、最新版本的 AWS CLI、 AWS CloudFormation 和`eksctl`版本 `v0.164.0-rc.0` 或更新版本進行更新。這麼做才能為子網路提供更多可用 IP 地址，以成功升級叢集版本。	2023 年 10 月 24 日
叢集角色和受管節點群組角色支援客戶受管 AWS 身分和存取管理政策	您可以在叢集角色上使用自訂 IAM 政策，而不使用 AmazonEKSClusterPolicy AWS 受管政策。此外，您也可以在受管節點群組中的節點角色使用自訂 IAM 政策，而不使用 AmazonEKSWorkerNodePolicy AWS 受管政策。為符合嚴格的合規要求，請建立具有最低權限的政策。	2023 年 10 月 23 日
修復 eksctl 的安裝連結	在頁面移動後修復 eksctl 的安裝連結。	2023 年 10 月 6 日
預覽版本：Amazon EKS Kubernetes 版本的延伸支援	Kubernetes 版本的延伸支援可讓您使用特定 Kubernetes 版本超過 14 個月。	2023 年 10 月 4 日

移除對 AWS App Mesh 整合的參考	Amazon EKS 與 AWS App Mesh 的整合僅保留給 App Mesh 的現有客戶。	2023 年 9 月 29 日
Kubernetes 版本 1.28	新增 Kubernetes 版本 1.28 支援新的叢集和版本升級。	2023 年 9 月 26 日
現有叢集支援在 Kubernetes 專用 Amazon VPC CNI 外掛程式中強制執行 Kubernetes 網路政策	您可搭配 Kubernetes 專用 Amazon VPC CNI 外掛程式在現有叢集使用 Kubernetes 網路政策，而不需要第三方解決方案。	2023 年 9 月 15 日
CoreDNS Amazon EKS 附加元件支援修改 PDB	您可在版本 v1.9.3-eksbuild.7 與更新版本，以及版本 v1.10.1-eksbuild.4 與更新版本，針對 CoreDNS 的 EKS 附加元件修改 PodDisruptionBudget 。	2023 年 9 月 15 日
Amazon EKS 支援共用子網路	用來在共用子網路中製作 Amazon EKS 叢集的新共用子網路要求與考量。	2023 年 9 月 7 日
「什麼是Amazon EKS？」的更新內容	加入了新的常見使用案例和架構主題。重新整理其他主題。	2023 年 9 月 6 日
在 Kubernetes 專用 Amazon VPC CNI 外掛程式中強制執行 Kubernetes 網路政策	您可搭配 Kubernetes 專用 Amazon VPC CNI 外掛程式使用 Kubernetes 網路政策，而不需要第三方解決方案。	2023 年 8 月 29 日
Amazon EKS AWS 區域擴展	Amazon EKS 現已在以色列（特拉維夫）(il-central-1) AWS 區域提供。	2023 年 8 月 1 日

適用於 Fargate 的可設定暫時性儲存	您可以增加在 Amazon EKS Fargate 上執行的每個 Pod 之暫時性儲存的總量。	2023 年 7 月 31 日
Amazon EFS CSI 驅動程式的附加元件支援	您現在可以使用 AWS 管理主控台、 AWS CLI 和 API 來管理 Amazon EFS CSI 驅動程式。	2023 年 7 月 26 日
AWS 受管政策更新 - 新政策	Amazon EKS 新增了新的 AWS 受管政策。	2023 年 7 月 26 日
Outposts 上的本機叢集現在可使用 1.27、1.26、1.25 和 1.24 的 Kubernetes AWS 版本更新	AWS Outposts 上的本機叢集現在可使用 Kubernetes 版本更新至 1.27.3、1.26.6、1.25.11 和 1.24.15	2023 年 7 月 20 日
IP 字首支援 Windows 節點	相較於將個別的次要 IP 位址指派給節點，將 IP 字首指派給節點可讓您在節點上託管更多數量的 Pod。	2023 年 7 月 6 日
Amazon FSx for OpenZFS CSI 驅動程式	您現在可以在 Amazon EKS 叢集上安裝 Amazon FSx for OpenZFS CSI 驅動程式。	2023 年 6 月 30 日
IPv4 叢集中 Linux 節點上的 Pod 現在可與 IPv6 端點進行通訊。	將 IPv6 位址指派給節點後，您 Pod 的 IPv4 位址就是將網路位址轉換為其執行所在節點的 IPv6 位址。	2023 年 6 月 19 日
AWS GovCloud (US) 區域中的 Windows 受管節點群組	在 AWS GovCloud (US) 區域中，Amazon EKS 受管節點群組現在可以執行 Windows 容器。	2023 年 5 月 30 日
Kubernetes 版本 1.27	新增 Kubernetes 版本 1.27 支援新的叢集和版本升級。	2023 年 5 月 24 日

Kubernetes 版本 1.26	新增 Kubernetes 版本 1.26 支援新的叢集和版本升級。	2023 年 4 月 11 日
無網域 gMSA	您現在可以將無網域 gMSA 與 Windows Pod 搭配使用。	2023 年 3 月 27 日
Amazon EKS AWS 區域擴展	Amazon EKS 現已在亞太區域（墨爾本）(ap-southeast-4) AWS 區域提供。	2023 年 3 月 10 日
Amazon File Cache CSI 驅動程式	您現在可以在 Amazon EKS 叢集上安裝 Amazon File Cache CSI 驅動程式。	2023 年 3 月 3 日
Kubernetes 1.25 版現已適用於 AWS Outposts 上的本機叢集	現在使用 Kubernetes 版本 1.22 – 1.25，可在 Outpost 上建立 Amazon EKS 本機叢集。	2023 年 3 月 1 日
Kubernetes 版本 1.25	新增 Kubernetes 版本 1.25 支援新的叢集和版本升級。	2023 年 2 月 22 日
AWS 受管政策更新 - 更新現有政策	Amazon EKS 已更新現有的 AWS 受管政策。	2023 年 2 月 7 日
Amazon EKS AWS 區域擴展	Amazon EKS 現已在亞太區域（海德拉巴）()ap-south-2 、歐洲（蘇黎世）(eu-central-2) 和歐洲（西班牙）(eu-south-2) AWS 區域提供。	2023 年 2 月 6 日
Kubernetes 版本 1.21 – 1.24 現在可供 AWS Outposts 上的本機叢集使用。	現在使用 Kubernetes 版本 1.21 – 1.24，可在 Outpost 上建立 Amazon EKS 本機叢集。以前，只有 1.21 版本可用。	2023 年 1 月 17 日

Amazon EKS 現在支援 AWS PrivateLink	您可以使用 an AWS PrivateLink 在 VPC 和 Amazon EKS 之間建立私有連線。	2022 年 12 月 16 日
受管節點群組 Windows 支援	您現在可以將 Windows 用於 Amazon EKS 受管節點群組。	2022 年 12 月 15 日
來自獨立軟體廠商的 Amazon EKS 附加元件現在可在 AWS Marketplace 中使用	您現在可以透過 AWS Marketplace 從獨立軟體供應商瀏覽和訂閱 Amazon EKS 附加元件。	2022 年 11 月 28 日
AWS 受管政策更新 - 更新現有政策	Amazon EKS 已更新現有的 AWS 受管政策。	2022 年 11 月 17 日
Kubernetes 版本 1.24	新增 Kubernetes 版本 1.24 支援新的叢集和版本升級。	2022 年 11 月 15 日
Amazon EKS AWS 區域擴展	Amazon EKS 現已在中東（阿拉伯聯合大公國）(me-central-1) AWS 區域提供。	2022 年 11 月 3 日
AWS 受管政策更新 - 更新現有政策	Amazon EKS 已更新現有的 AWS 受管政策。	2022 年 10 月 24 日
AWS 受管政策更新 - 更新現有政策	Amazon EKS 已更新現有的 AWS 受管政策。	2022 年 10 月 20 日
AWS Outpost 上的本機叢集現已推出	您現在可以在 Outpost 上建立 Amazon EKS 本機叢集。	2022 年 9 月 19 日
Fargate vCPU 型配額	Fargate 正在從 Pod 型配額轉換為 vCPU 型配額。	2022 年 9 月 8 日
AWS 受管政策更新 - 更新現有政策	Amazon EKS 已更新現有的 AWS 受管政策。	2022 年 8 月 31 日

成本監控	Amazon EKS 現在支援 Kubernetes，可讓您監控按 Kubernetes 資源 (包括 、節點、命名空間和標籤) 細分的成本。	2022 年 8 月 24 日
AWS 受管政策更新 - 新政策	Amazon EKS 新增了新的 AWS 受管政策。	2022 年 8 月 24 日
AWS 受管政策更新 - 新政策	Amazon EKS 新增了新的 AWS 受管政策。	2022 年 8 月 23 日
帳單的標籤資源	對所有叢集新增了 `aws:eks:cluster-name` 產生的成本分配標籤支援。	2022 年 8 月 16 日
Fargate 設定檔萬用字元	在命名空間、標籤索引鍵和標籤值的選擇器條件中，新增了對 Fargate 設定檔萬用字元的支援。	2022 年 8 月 16 日
Kubernetes 版本 1.23	新增 Kubernetes 版本 1.23 支援新的叢集和版本升級。	2022 年 8 月 11 日
在 中檢視 Kubernetes 資源 AWS 管理主控台	您現在可以使用 AWS 管理主控台來檢視有關部署到叢集的 Kubernetes 資源的資訊。	2022 年 5 月 3 日
Amazon EKS AWS 區域擴展	Amazon EKS 現已在亞太區域 （ 雅加達 ） (`ap-southeast-3`) AWS 區域提供。	2022 年 5 月 2 日
可觀測性頁面和 ADOT 附加元件支援	新增可觀測性頁面和 AWS Distro for OpenTelemetry (ADOT)。	2022 年 4 月 21 日
Kubernetes 版本 1.22	新增 Kubernetes 版本 1.22 支援新的叢集和版本升級。	2022 年 4 月 4 日

AWS 受管政策更新 - 新政策	Amazon EKS 新增了新的 AWS 受管政策。	2022 年 4 月 4 日
新增 Fargate Pod 修補詳細資訊	升級 Fargate Pod 時，Amazon EKS 會先嘗試根據您的 Pod 中斷預算移出 Pod。您可以建立事件規則，以便在刪除 Pod 之前對失敗的移出事件做出反應。	2022 年 4 月 1 日
完整版本：Amazon EBS CSI 驅動程式的附加元件支援	您現在可以使用 、 AWS 管理主控台 AWS CLI 和 API 來管理 Amazon EBS CSI 驅動程式。	2022 年 3 月 31 日
AWS Outposts 內容更新	在 AWS Outposts 上部署 Amazon EKS 叢集的說明。	2022 年 3 月 22 日
AWS 受管政策更新 - 更新現有政策	Amazon EKS 已更新現有的 AWS 受管政策。	2022 年 3 月 21 日
Windows `containerd` 支援	您現在可以選擇 Windows 節點的 `containerd` 執行時間。	2022 年 3 月 14 日
在安全文件中新增 Amazon EKS 連接器考量事項	描述與連接叢集相關的共同責任模型。	2022 年 2 月 25 日
指派 IPv6 位址至您的 Pod 和服務	您現在可以建立 `1.21` 或更新版本的叢集，其會將 IPv6 位址指派給您的 Pod 和服務。	2022 年 1 月 6 日
AWS 受管政策更新 - 更新現有政策	Amazon EKS 已更新現有的 AWS 受管政策。	2021 年 12 月 13 日
預覽版本：Amazon EBS CSI 驅動程式的附加元件支援	您現在可以使用 AWS 管理主控台、 AWS CLI 和 API 預覽來管理 Amazon EBS CSI 驅動程式。	2021 年 12 月 9 日

Karpenter Autoscaler 支援	您現在可以使用 Karpenter 開放原始碼專案來自動調整節點。	2021 年 11 月 29 日
Fargate 記錄支援 Fluent Bit Kubernetes 篩選條件	您現在可以將 Fargate 記錄與 Fluent Bit Kubernetes 篩選條件搭配使用。	2021 年 11 月 10 日
控制平面中提供的 Windows 支援	您現在可在控制平面中使用 Windows 支援。您不再需要於資料平面中將其啟用。	2021 年 11 月 9 日
新增 Bottlerocket 作為受管節點群組的 AMI 類型	Bottlerocket 以往僅能作為自我管理節點選項使用。現在可將其設定為受管節點群組，進而減少為滿足節點合規性要求要耗費的精力。	2021 年 10 月 28 日
DL1 驅動程式支援	自訂 Amazon Linux AMI 現在支援 Amazon Linux 2 深度學習工作負載。此支援操作允許一般內部部署或雲端基準組態設定。	2021 年 10 月 25 日
VT1 視訊支援	自訂 Amazon Linux AMI 現在支援某些發行版本的 VT1。此支援功能會在您的 Amazon EKS 叢集上公告 Xilinx U30 裝置。	2021 年 9 月 13 日
Amazon EKS Connector 現已推出	您可以使用 Amazon EKS 連接器來註冊任何符合標準的 Kubernetes 叢集，並將其連接到 ， AWS 並在 Amazon EKS 主控台中將其視覺化。	2021 年 9 月 8 日

Amazon EKS Anywhere 現已推出	Amazon EKS Anywhere 是 Amazon EKS 的新部署選項，您可輕鬆地在內部部署建立和操作 Kubernetes 叢集。	2021 年 9 月 8 日
Amazon FSx for NetApp ONTAP CSI 驅動程式	已新增主題，其中摘要說明 Amazon FSx for NetApp ONTAP CSI 驅動程式，並提供其他參考的連結。	2021 年 9 月 2 日
受管節點群組現在會自動計算 Amazon EKS 建議的節點 Pod 數量上限	受管節點群組現在會針對您在沒有啟動範本的情況下部署的節點，或使用未在其中指定 AMI ID 的啟動範本所部署的節點，自動計算其 Amazon EKS 最大 Pod 數。	2021 年 8 月 30 日
移除附加元件設定的 Amazon EKS 管理，而不移除 Amazon EKS 附加元件軟體	您現在可以移除 Amazon EKS 附加元件，而無需從叢集移除附加元件軟體。	2021 年 8 月 20 日
使用 Multus 建立多主目錄 Pod	您現在可以使用 Multus 將多個網路介面新增至 Pod。	2021 年 8 月 2 日
將更多 IP 地址新增至您的 Linux Amazon EC2 節點	您現在可以將更多的 IP 地址新增至您的 Linux Amazon EC2 節點。這表示您可以在每個節點上執行更多數量的 Pod。	2021 年 7 月 27 日
`containerd` 執行階段啟動程序	Amazon EKS 最佳化加速 Amazon Linux Amazon Machine Image (AMI) 現在包含啟動程序旗標，您可啟用 Amazon EKS 最佳化和 Bottlerocket AMI 中的 `containerd` 執行時期。此旗標可用於所有支援 Kubernetes 版本的 AMI 中。	2021 年 7 月 19 日

Kubernetes 版本 1.21	新增 Kubernetes 1.21 版支援。	2021 年 7 月 19 日
新增受管政策主題	自 2021 年 6 月 17 日起所有 Amazon EKS IAM 受管政策以及已對其進行變更的清單。	2021 年 6 月 17 日
將 Pod 安全群組搭配 Fargate 使用	除了搭配 Amazon EC2 節點使用外，您現在可以將 Pod 安全群組與 Fargate 搭配使用。	2021 年 6 月 1 日
新增 CoreDNS 和 kube-proxy Amazon EKS 附加元件	Amazon EKS 現在可以幫助您管理叢集的 CoreDNS 和 kube-proxy Amazon EKS 附加元件。	2021 年 5 月 19 日
Kubernetes 版本 1.20	新增 Kubernetes 版本 1.20 支援新的叢集和版本升級。	2021 年 5 月 18 日
AWS2.2.0已發行Load Balancer控制器	您現在可以使用 AWS Load Balancer控制器，使用執行個體或 IP 目標建立 Elastic Load Balancer。	2021 年 5 月 14 日
受管節點群組的節點污點	Amazon EKS 現在支援將節點污點新增至受管節點群組。	2021 年 5 月 11 日
現有叢集的秘密加密	Amazon EKS 現在支援為現有叢集新增秘密加密。	2021 年 2 月 26 日
Kubernetes 版本 1.19	新增 Kubernetes 版本 1.19 支援新的叢集和版本升級。	2021 年 2 月 16 日
Amazon EKS 現在支援 OpenID Connect (OIDC) 身分提供者，作為對 1.16 或更新版本的叢集驗證使用者身分的方法。	OIDC 身分提供者可與 AWS Identity and Access Management (IAM) 搭配使用，或做為替代方式。	2021 年 2 月 12 日

在 中檢視節點和工作負載資源 AWS 管理主控台	您現在可以在 中檢視受管、自我管理和 Fargate 節點的詳細資訊，以及部署的 Kubernetes 工作負載 AWS 管理主控台。	2020 年 12 月 1 日
在受管節點群組中部署 Spot 執行個體類型	您現在可以將多個 Spot 或隨需執行個體類型部署到受管節點群組。	2020 年 12 月 1 日
Amazon EKS 現在可以管理叢集的特定附加元件	您可以自行管理附加元件，或允許 Amazon EKS 透過 Amazon EKS API 控制附加元件的啟動和版本。	2020 年 12 月 1 日
在多個輸入之間共用 ALB	您現在可以跨多個 Kubernetes 輸入共用 an AWS Application Load Balancer (ALB)。在過去，您必須為每個輸入部署個別的 ALB。	2020 年 10 月 23 日
NLB IP 目標支援	您現在可以部署具有 IP 目標的 Network Load Balancer。這意味您可使用 NLB 將網路流量負載平衡到 Fargate Pod，並直接負載平衡到 Amazon EC2 節點上執行的 Pod。	2020 年 10 月 23 日
Kubernetes 版本 1.18	新增 Kubernetes 版本 1.18 支援新的叢集和版本升級。	2020 年 10 月 13 日
為 Kubernetes 服務 IP 地址指派指定自訂 CIDR 區塊。	您現在可以指定 Kubernetes 從中指派服務 IP 地址的自訂 CIDR 區塊。	2020 年 9 月 29 日
將安全群組指派至個別 Pod	您現在可以將不同的安全群組與在許多 Amazon EC2 執行個體類型上執行的一些個別 Pod 建立關聯。	2020 年 9 月 9 日

在節點上部署 Bottlerocket	您現在可以部署執行 Bottlerocket 的節點。	2020 年 8 月 31 日
啟動 Arm 節點的功能已正式可用	您現在可以在受管節點群組和自我管理節點群組中啟動 Arm 節點。	2020 年 8 月 17 日
受管節點群組啟動範本和自訂 AMI	現在可以使用 Amazon EC2 啟動範本部署受管節點群組。您可以自行選擇是否為啟動範本指定自訂 AMI。	2020 年 8 月 17 日
AWS Fargate 的 EFS 支援	您現在可以搭配 AWS Fargate 使用 Amazon EFS。	2020 年 8 月 17 日
Amazon EKS 平台版本更新	這是具有安全性修正和增強功能的新平台版本。這包括在使用具有 Kubernetes 版本 `1.15` 或更高版本的 Network Load Balancer 時對 `LoadBalancer` 類型服務的 UDP 支援。如需詳細資訊，請參閱 GitHub 上的允許 UDP for AWS Network Load Balancer 問題。	2020 年 8 月 12 日
Amazon EKS AWS 區域擴展	非洲 (開普敦) (`af-south-1`) 和歐洲 (米蘭) (`eu-south-1`) AWS 區域現可使用 Amazon EKS。	2020 年 8 月 6 日
Fargate 用量指標	AWS Fargate 提供 CloudWatch 用量指標，可讓您查看帳戶對 Fargate 隨需資源的使用情形。	2020 年 8 月 3 日
Kubernetes 版本 `1.17`	新增 Kubernetes 版本 `1.17` 支援新的叢集和版本升級。	2020 年 7 月 10 日

使用適用於 Kubernetes 的 App Mesh 控制器，從 Kubernetes 內建立和管理 App Mesh 資源	您可以從 Kubernetes 內部建立和管理 App Mesh 資源。控制器也會自動將 Envoy 代理和初始化容器注入到您部署的 Pod 中。	2020 年 6 月 18 日
Amazon EKS 現在支援 Amazon EC2 Inf1 節點	您可以將 Amazon EC2 Inf1 節點新增至叢集。	2020 年 6 月 4 日
Amazon EKS AWS 區域擴展	Amazon EKS 現已在 AWS GovCloud （美國東部）(`us-gov-east-1`) 和 AWS GovCloud （美國西部）(`us-gov-west-1`) AWS 區域提供。	2020 年 5 月 13 日
Amazon EKS 已不再支援 Kubernetes `1.12` 版	Amazon EKS 已不再支援 Kubernetes `1.12` 版。請將任何 `1.12` 版叢集更新至 `1.13` 或更新版本，避免服務中斷。	2020 年 5 月 12 日
Kubernetes 版本 `1.16`	新增 Kubernetes 版本 `1.16` 支援新的叢集和版本升級。	2020 年 4 月 30 日
新增 AWSServiceRoleForAmazonEKS 服務連結角色	新增 AWSServiceRoleForAmazonEKS 服務連結角色。	2020 年 4 月 16 日
Kubernetes 版本 `1.15`	新增 Kubernetes 版本 `1.15` 支援新的叢集和版本升級。	2020 年 3 月 10 日
Amazon EKS AWS 區域擴展	北京 (`cn-north-1`) 和寧夏 (`cn-northwest-1`) AWS 區域現在可以使用 Amazon EKS。	2020 年 2 月 26 日

FSx for Lustre CSI 驅動程式	新增在 Kubernetes `1.14` Amazon EKS 叢集上安裝 FSx for Lustre CSI 驅動程式的相關主題。	2019 年 12 月 23 日
限制對叢集公有存取端點的網路存取	透過此更新，您可以使用 Amazon EKS 來限制與 Kubernetes API 伺服器之公有存取端點通訊的 CIDR 範圍。	2019 年 12 月 20 日
從 VPC 外部解析叢集的私有存取端點地址	透過此更新，您可以使用 Amazon EKS 從 VPC 外部解析 Kubernetes API 伺服器的私有存取端點。	2019 年 12 月 13 日
(Beta 版) Amazon EC2 A1 Amazon EC2 執行個體節點	啟動向您的 Amazon EKS 叢集註冊的 Amazon EC2 A1 Amazon EC2 執行個體節點。	2019 年 12 月 4 日
在 AWS Outposts 上建立叢集	Amazon EKS 現在支援在 AWS Outposts 上建立叢集。	2019 年 12 月 3 日
AWS Amazon EKS 上的 Fargate	Amazon EKS Kubernetes 叢集現在支援在 Fargate 上執行 Pod。	2019 年 12 月 3 日
Amazon EKS AWS 區域擴展	Amazon EKS 現已在加拿大（中部）(`ca-central-1`) AWS 區域提供。	2019 年 11 月 21 日
受管節點群組	Amazon EKS 受管節點群組會自動化 Amazon EKS Kubernetes 叢集節點 (Amazon EC2 執行個體) 的佈建和生命週期管理。	2019 年 11 月 18 日
Amazon EKS 平台版本更新	可解決 CVE-2019-11253 問題的新平台版本。	2019 年 11 月 6 日

Amazon EKS 已不再支援 Kubernetes 1.11 版	Amazon EKS 已不再支援 Kubernetes 1.11 版。請將任何 1.11 版叢集更新至 1.12 或更新版本，避免服務中斷。	2019 年 11 月 4 日
Amazon EKS AWS 區域擴展	Amazon EKS 現已在南美洲（聖保羅）(sa-east-1) AWS 區域提供。	2019 年 10 月 16 日
Windows 支援	執行 Kubernetes 1.14 版的 Amazon EKS 叢集現在支援 Windows 工作負載。	2019 年 10 月 7 日
自動擴展	新增章節以涵蓋 Amazon EKS 叢集上支援的一些不同類型的 Kubernetes 自動擴展。	2019 年 9 月 30 日
Kubernetes Dashboard 更新	已更新在 Amazon EKS 叢集上安裝 Kubernetes 儀表板以使用 Beta 2.0 版的相關主題。	2019 年 9 月 28 日
Amazon EFS CSI 驅動程式	已新增在 Kubernetes 1.14 Amazon EFS 叢集上安裝 Amazon EFS CSI 驅動程式的相關主題。	2019 年 9 月 19 日
Amazon EKS 最佳化 AMI ID 的 Amazon EC2 Systems Manager 參數	已新增使用 Amazon EC2 Systems Manager 參數擷取 Amazon EKS 最佳化 AMI ID 的相關主題。此參數讓您無需查詢 AMI ID。	2019 年 9 月 18 日
Amazon EKS 資源標記	您可管理 Amazon EKS 叢集的標記。	2019 年 9 月 16 日

Amazon EBS CSI 驅動程式	已新增在 Kubernetes 1.14 Amazon EKS 叢集上安裝 Amazon EBS CSI 驅動程式的相關主題。	2019 年 9 月 9 日
全新 Amazon EKS 最佳化 AMI 為 CVE-2019-9512 和 CVE-2019-9514 進行修補	Amazon EKS 已更新 Amazon EKS 最佳化 AMI，可解決 CVE-2019-9512 和 CVE-2019-9514 問題。	2019 年 9 月 6 日
宣布 Amazon EKS 廢除 Kubernetes 1.11	Amazon EKS 已在 2019 年 11 月 4 日停止支援 Kubernetes 1.11 版。	2019 年 9 月 4 日
Kubernetes 版本 1.14	新增 Kubernetes 版本 1.14 支援新的叢集和版本升級。	2019 年 9 月 3 日
服務帳戶的 IAM 角色	透過 Amazon EKS 叢集上服務帳戶的 IAM 角色，您可以建立 IAM 角色與 Kubernetes 服務帳戶的關聯。透過此功能，您不再需要提供延伸許可給節點 IAM 角色。如此一來，該節點上的 Pod 就可以呼叫 AWS APIs。	2019 年 9 月 3 日
Amazon EKS AWS 區域擴展	Amazon EKS 現已在中東（巴林）(me-south-1) AWS 區域提供。	2019 年 8 月 29 日
Amazon EKS 平台版本更新	可解決 CVE-2019-9512 和 CVE-2019-9514 問題的新平台版本。	2019 年 8 月 28 日
Amazon EKS 平台版本更新	可解決 CVE-2019-11247 和 CVE-2019-11249 問題的新平台版本。	2019 年 8 月 5 日

Amazon EKS 區域擴展	Amazon EKS 現已在亞太區域（香港）(ap-east-1) AWS 區域提供。	2019 年 7 月 31 日
Amazon EKS 已不再支援 Kubernetes 1.10 版	Amazon EKS 已不再支援 Kubernetes 1.10 版。請將任何 1.10 版叢集更新至 1.11 或更新版本，避免服務中斷。	2019 年 7 月 30 日
在 ALB 傳入控制器上新增主題	適用於 Kubernetes 的 AWS ALB 傳入控制器是一種控制器，可在建立傳入資源時建立 ALB。	2019 年 7 月 11 日
全新 Amazon EKS 最佳化 AMI	從 AMI 移除不必要的 kubectl 二進位檔。	2019 年 7 月 3 日
Kubernetes 版本 1.13	新增 Kubernetes 版本 1.13 支援新的叢集和版本升級。	2019 年 6 月 18 日
全新 Amazon EKS 最佳化 AMI 為 AWS-2019-005 進行修補	Amazon EKS 已更新 Amazon EKS 最佳化 AMI，以解決 link:security/security-bulletins/AWS-2019-005/[AWS-2019-005,type="marketing"] 中描述的漏洞。	2019 年 6 月 17 日
宣布 Amazon EKS 停止支援 Kubernetes 1.10	Amazon EKS 已在 2019 年 7 月 22 日停止支援 Kubernetes 1.10 版。	2019 年 5 月 21 日
Amazon EKS 平台版本更新	全新推出適用於 Kubernetes 1.11 和 1.10 叢集的平台版本，可供支援 kubelet 憑證中的自訂 DNS 名稱和提升 etcd 效能。	2019 年 5 月 21 日

eksctl 入門	本入門指南說明如何使用 eksctl 來安裝 Amazon EKS 入門所有所需的資源。這是簡單的命令列公用程式，可用於在 Amazon EKS 上建立和管理 Kubernetes 叢集。	2019 年 5 月 10 日
AWS CLI get-token 命令	aws eks get-token 命令已新增至 AWS CLI。您不再需要安裝 AWS IAM Authenticator for Kubernetes，即可建立叢集 API 伺服器通訊的用戶端安全字符。將您的 AWS CLI 安裝升級至最新版本，以使用此新功能。如需詳細資訊，請參閱《AWS 命令列界面使用者指南》中的安裝 AWS 命令列界面。	2019 年 5 月 10 日
Amazon EKS 平台版本更新	全新推出適用於 Kubernetes 1.12 叢集的平台版本，可供支援 kubelet 憑證中的自訂 DNS 名稱和提升 etcd 效能。這個版本修正了造成節點 kubelet 協助程式每隔幾秒便發出請求新憑證的錯誤。	2019 年 5 月 8 日
Prometheus 教學課程	已新增有關將 Prometheus 部署至 Amazon EKS 叢集的主題。	2019 年 4 月 5 日
Amazon EKS 控制平面記錄	透過此更新，您可以直接從 Amazon EKS 控制平面取得稽核和診斷日誌。您可以在帳戶中使用這些 CloudWatch 日誌作為保護和執行叢集的參考。	2019 年 4 月 4 日

Kubernetes 版本 1.12	新增 Kubernetes 版本 1.12 支援新的叢集和版本升級。	2019 年 3 月 28 日
新增 App Mesh 入門指南	新增 App Mesh 和 Kubernetes 入門文件。	2019 年 3 月 27 日
Amazon EKS API 伺服器端點私有存取	新增有關停用公有存取 Amazon EKS 叢集的 Kubernetes API 伺服器端點的文件。	2019 年 3 月 19 日
新增安裝 Kubernetes 指標伺服器的主題	Kubernetes 指標伺服器是叢集中資源使用狀況資料的彙總工具。	2019 年 3 月 18 日
新增相關的開放原始碼專案清單	這些開放原始碼專案會擴展 Kubernetes 叢集在 上執行的功能 AWS，包括由 Amazon EKS 管理的叢集。	2019 年 3 月 15 日
新增有關本機安裝 Helm 的主題	Kubernetes 的 `helm` 套件管理工具可協助您安裝和管理 Kubernetes 叢集上的應用程式。本主題展示如何本機安裝和執行 `helm` 和 `tiller` 二進位檔案。這樣，您可以在本機系統上使用 Helm CLI 安裝和管理圖表。	2019 年 3 月 11 日
Amazon EKS 平台版本更新	新平台版本會將 Amazon EKS Kubernetes `1.11` 叢集更新為修補程式等級 `1.11.8`，可解決 CVE-2019-1002100 問題。	2019 年 3 月 8 日
已提高叢集限制	Amazon EKS 已將您可以在 AWS 區域中建立的叢集數量從 3 增加到 50。	2019 年 2 月 13 日

Amazon EKS AWS 區域擴展	Amazon EKS 現已在歐洲（倫敦）()eu-west-2 、歐洲（巴黎）(eu-west-3 ）和亞太區域（孟買）(ap-south-1) AWS 區域提供。	2019 年 2 月 13 日
全新 Amazon EKS 最佳化 AMI 為 ALAS-2019-1156 進行修補	Amazon EKS 已更新 Amazon EKS 最佳化 AMI，可解決 ALAS-2019-1156 中所述的安全漏洞。	2019 年 2 月 11 日
全新 Amazon EKS 最佳化 AMI 為 ALAS2-2019-1141 進行修補	Amazon EKS 已更新 Amazon EKS 最佳化 AMI，可解決 ALAS2-2019-1141 所述的 CVE 問題。	2019 年 1 月 9 日
Amazon EKS AWS 區域擴展	Amazon EKS 現已在亞太區域（首爾）(ap-northeast-2) AWS 區域提供。	2019 年 1 月 9 日
Amazon EKS 區域擴展	Amazon EKS 現已在以下其他 AWS 區域提供：歐洲（法蘭克福）(eu-central-1)、亞太區域（東京）(ap-northeast-1)、亞太區域（新加坡）(ap-southeast-1) 和亞太區域（雪梨）()ap-southeast-2 。	2018 年 12 月 19 日
Amazon EKS 叢集更新	已針對 Amazon EKS 叢集 Kubernetes 版本更新和節點取代。	2018 年 12 月 12 日
Amazon EKS AWS 區域擴展	Amazon EKS 現已在歐洲（斯德哥爾摩）(eu-north-1) AWS 區域提供。	2018 年 12 月 11 日

Amazon EKS 平台版本更新	將 Kubernetes 更新至修補程式層級 1.10.11 以定址連結的新平台版本：link:security/security-bulletins/AWS-2018-020/[CVE-2018-1002105,type="marketing"]。	2018 年 12 月 4 日
新增支援 ALB 傳入控制器的 1.0.0 版	ALB 輸入控制器會從 發行1.0.0具有正式支援的版本 AWS。	2018 年 11 月 20 日
新增對 CNI 網路組態的支援	Kubernetes 專用 Amazon VPC CNI 外掛程式 1.2.1 版現已針對輔助 Pod 網路介面支援自訂網路組態。	2018 年 10 月 16 日
新增對 MutatingAdmissionWebhook 和 ValidatingAdmissionWebhook 的支援	Amazon EKS 平台版本 1.10-eks.2 現已支援 MutatingAdmissionWebhook 和 ValidatingAdmissionWebhook 許可控制器。	2018 年 10 月 10 日
新增合作夥伴 AMI 資訊	Canonical 已與 Amazon EKS 合作，建立了可供您用於叢集的節點 AMI。	2018 年 10 月 3 日
新增 CLI AWSupdate-kubeconfig 命令的指示	Amazon EKS 已將 update-kubeconfig 新增至 AWS CLI，以簡化建立kubeconfig 檔案以存取叢集的程序。	2018 年 9 月 21 日
全新 Amazon EKS 最佳化 AMI	Amazon EKS 已更新 Amazon EKS 最佳化 AMI (支援及未支援 GPU)，提供了各項安全性修正和 AMI 最佳化。	2018 年 9 月 13 日

Amazon EKS AWS 區域擴展	歐洲 (愛爾蘭) (eu-west-1) 區域現可使用 Amazon EKS。	2018 年 9 月 5 日
Amazon EKS 平台版本更新	新的平台版本支援 Kubernetes 彙整層和 Horizontal Pod Autoscaler (HPA)。	2018 年 8 月 31 日
全新 Amazon EKS 最佳化 AMI 與 GPU 支援	Amazon EKS 已更新 Amazon EKS 最佳化 AMI，以使用新的 AWS CloudFormation 節點範本和引導指令碼。此外，還推出了全新的支援 GPU 的 Amazon EKS 最佳化 AMI。	2018 年 8 月 22 日
全新 Amazon EKS 最佳化 AMI 為 ALAS2-2018-1058 進行修補	Amazon EKS 已更新 Amazon EKS 最佳化 AMI，可解決 ALAS2-2018-1058 所述的 CVE 問題。	2018 年 8 月 14 日
Amazon EKS 最佳化之 AMI 建置指令碼	Amazon EKS 已採用開放原始碼的形式提供建置指令碼，可用於建置 Amazon EKS 最佳化 AMI。這些建置指令碼現可於 GitHub 取得。	2018 年 7 月 10 日
Amazon EKS 初始版本	服務啟動的初始文件	2018 年 6 月 5 日

為 EKS 使用者指南做出貢獻

AWS 已為 EKS 使用者指南推出了改進的貢獻體驗。

您現在可以直接在 GitHub 上編輯 EKS 使用者指南的原始檔案。

文件現已採用 AsciiDoc 格式，這是一種類似 Markdown 的強大撰寫語言。AsciiDoc 結合了簡單的語法與企業級文件功能，例如進階格式化、交叉參考和安全控制。

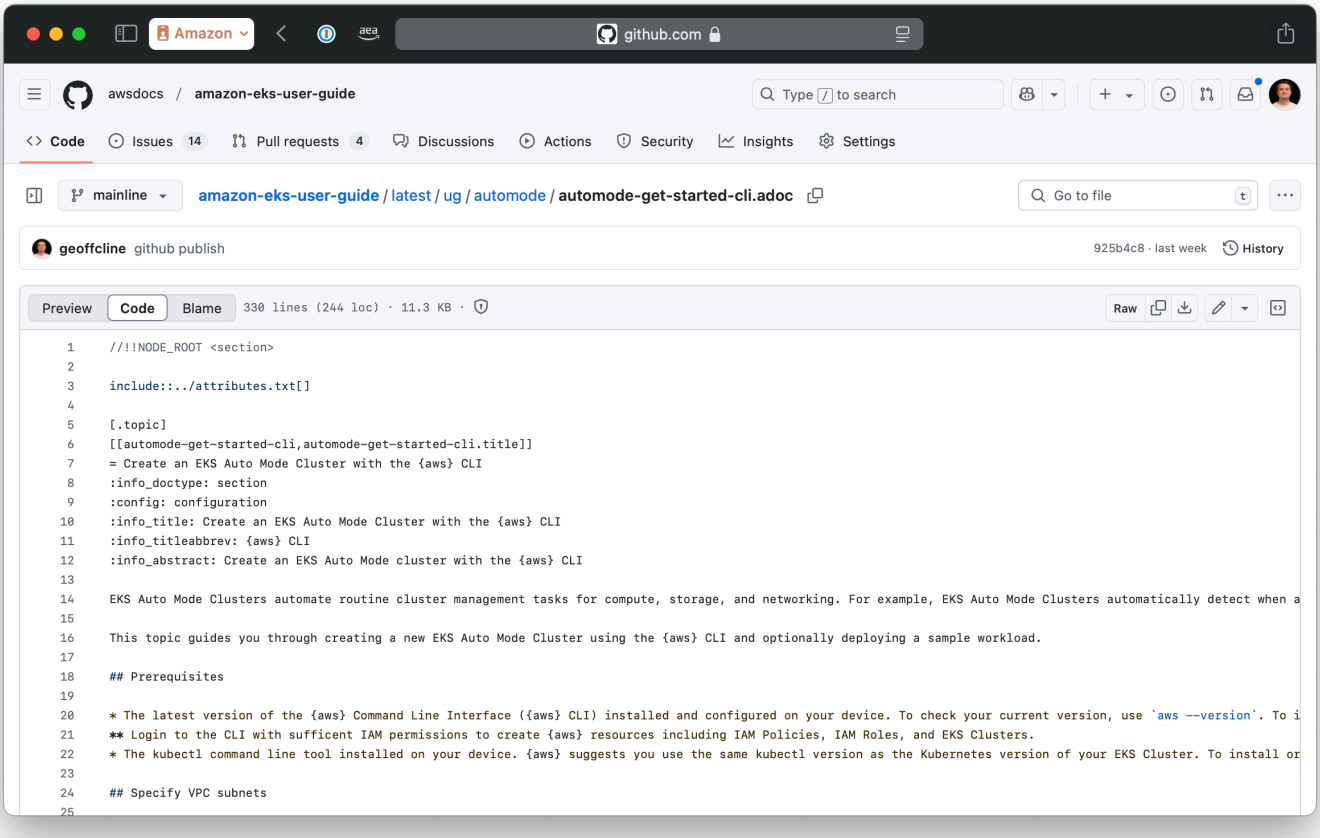
現在您可以直接在 GitHub 上編輯 EKS 文件。我們的簡化程序包括：

- 更快的提取請求處理
- 減少手動操作步驟
- 自動化的內容品質檢查

我們期待您的貢獻。

從 Web 瀏覽器編輯單一頁面

您可透過 Web 瀏覽器，在 EKS 使用者指南中輕鬆編輯單一頁面。



如果您想要從 Web 瀏覽器編輯多個頁面，請參閱 the section called “使用 GitHub 編輯檔案”。

先決條件

- 在 Web 瀏覽器中開啟變更的文件頁面。
- 已登入 GitHub。

程序

1. 導覽至您想要在《Amazon EKS 使用者指南》中編輯的頁面。
2. 在右側窗格中，選擇在 GitHub 上編輯此頁面連結。
3. 到達 GitHub 後，透過以下任一方式開啟編輯器：
   - 按下鍵盤上的 e 鍵。
   - 按一下鉛筆圖示，並從下拉式清單中選取就地編輯。

- 如果您沒有編輯的選項，則需要登入 GitHub。您的 GitHub 帳戶不需要任何特殊許可即可建議變更。但是，內部 Amazon 貢獻者應該連結其 GitHub 設定檔。

4. 在 GitHub 編輯器中對內容進行所需的更改。
   - 編輯器提供語法突顯和預覽功能。
   - 您可使用 AsciiDoc 標記來格式化變更。
   - 您可使用 `ctrl-f` 開啟尋找/取代介面。
5. (選用) 預覽您的變更。
   - 使用 `preview` 索引標籤以預覽具有豐富格式的變更。
   - 使用 `show diff` 選項反白顯示已變更的區段。移除的區段在左邊界中有一個紅色指標。新區段在左邊界有一個綠色指標。
6. 完成編輯後，按一下編輯器頂端的 遞交變更...按鈕
7. 在遞交對話方塊中：
   - 確認您的電子郵件位址是否正確。
   - 新增簡短但具描述性的遞交訊息，解釋您的更改。
   - 如果需要，可以選擇新增更長的描述。
   - 選取以建立新的分支並且提取請求。

您已建立一個包含建議更改的拉取請求。

提取請求概觀

當您建立 PR 時：

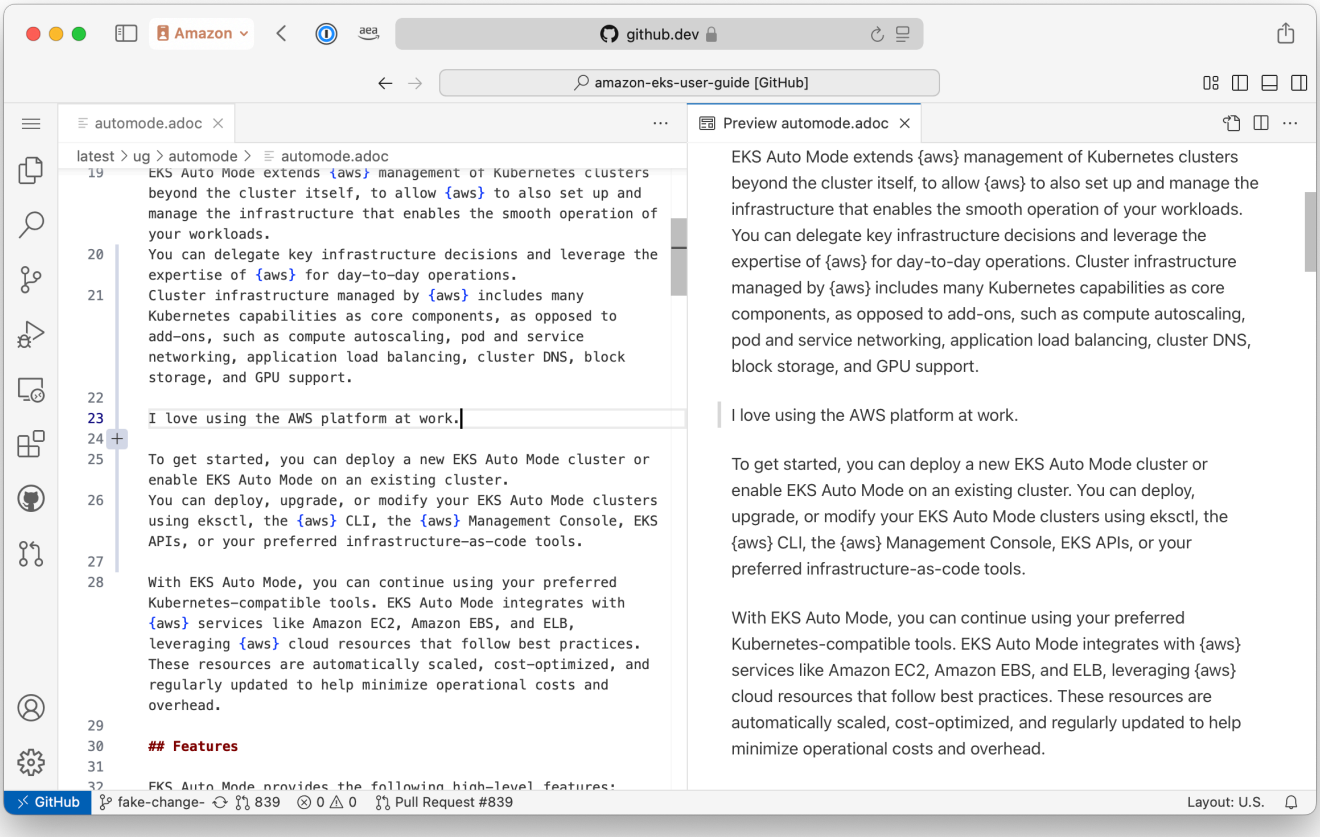
- 您的變更會提交，以供儲存庫維護者檢閱。
- 檢閱者可評論您的變更並請求修改。
- 可能會執行自動化測試來驗證您的變更。
- 核准後，您可將變更合併至主要儲存庫。

提取請求有助於確保品質，並提供在變更整合之前進行討論的方法。

了解如何在 GitHub 文件中檢閱與核准提取請求。

使用 GitHub Web Editor 從 Web 瀏覽器編輯多個檔案

如果您想要提議變更多個頁面，或建立新的文件頁面，請使用 GitHub.dev Web 編輯器。此 Web 編輯器基於流行的 Visual Studio Code 文字編輯器。



先決條件

- 登入 GitHub
- 熟悉 Visual Studio Code 程式碼編輯器
- 熟悉 Git

程序

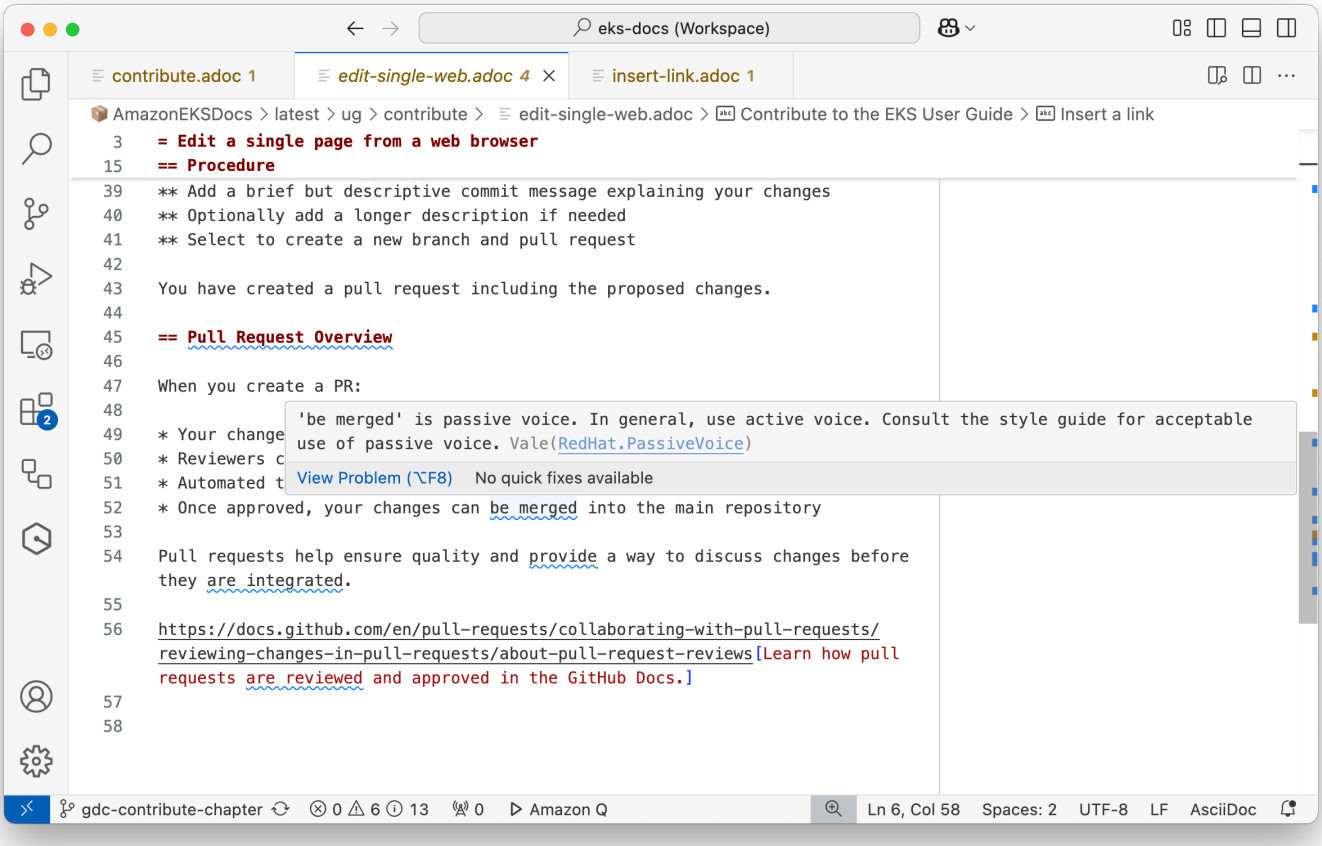
> **Note**
>
> EKS 文件團隊已建立工作區檔案，其中包含編輯器的建議組態，例如文字包裝和 AsciiDoc 語法反白。建議您載入此工作區檔案。

1. 在 GitHub.dev 上開啟工作區。
   - 您可將 URL 加入書籤 `https://github.dev/awsdocs/amazon-eks-user-guide/blob/mainline/eks-docs.code-workspace?workspace=true`
2. (僅限第一次設定) 系統可能會提示您在自己的 GitHub 帳戶中建立儲存庫的分支。接受此提示。如需詳細資訊，請參閱 GitHub 文件中的關於延伸。
3. (僅限第一次設定) 接受右下角的提示來安裝 AsciiDoc 延伸模組。
4. 導覽至 `latest/ug` 的文件內容。
   - 文件檔案會依其最上層區段進行組織。例如，「安全性」章節中的頁面在 "security/" 目錄下具有來源檔案。
5. 要檢視文件頁面的預覽，請使用右上角的開啟預覽到側邊按鈕。圖示包含一個小型放大鏡。
6. 使用左側的來源控制索引標籤遞交您的變更。如需詳細資訊，請參閱 Visual Studio Code 文件：
   - 遞交變更
   - 建立提取請求

建立提取請求後，文件團隊會檢閱該請求。

在本機安裝 Vale 以便在輸入時檢視樣式意見回饋

您可在輸入時查看樣式意見回饋。這可協助辨識不良的寫入及錯別字。



概觀：

- Vale CLI 可載入樣式指南，並針對來源檔案執行這些樣式指南。
- EKS 文件儲存庫包括載入樣式指南及本機規則的 Vale 組態檔案。
- 適用於 Visual Studio (VS) 的 Vale 延伸可在編輯器內顯示 Vale 意見回饋。

安裝 Vale

請遵循 Vale CLI 文件中的說明，以便透過 Package Manager 安裝 Vale。

安裝 VS Code Vale 延伸

1. 開啟 VS Code。
2. 點選活動列中的延伸圖示 (或按下 Ctrl+Shift+X)。
3. 搜尋 "Vale"。

4. 在 Chris Chinchilla 的 "Vale VSCode" 延伸上點選按一下「安裝」。
5. 在提示時重新載入 VS Code。

同步 Vale

Vale 使用專案根目錄中的 `.vale.ini` 組態檔案，來確定要套用的樣式規則。

1. 開啟 VS Code。
2. 點選檢視 > 終端 (或按 Ctrl+`)。
3. 在需要時導覽至專案根目錄。
4. 執行 命令：

```
vale sync
```

5. 等待 Vale 完成下載和同步樣式規則。

在 VS Code 中檢視樣式意見回饋

1. 在 VS Code 中開啟 Markdown 或 AsciiDoc 檔案。
2. Vale 延伸將依據樣式規則來自動檢查您的文字。
3. 樣式問題將會在編輯器中加上底線。
4. 將滑鼠游標懸停於加底線的文字上，可查看特定的樣式建議。
5. 遵循建議或參閱樣式指南來修正問題。

建立新頁面

了解如何建立新的文件頁面。本主題包含建立初始頁面中繼資料，以及將頁面新增至指南目錄的步驟說明。

建立頁面

1. 導覽至章節目錄。例如，如果要在「安全性」區段中建立新頁面，請導航至 `latest/ug/security` 目錄。
2. 確定頁面 ID。依照慣例，頁面 ID 應全部使用小寫並以 - 分隔。此頁面的 ID 為 `create-page`。

3. 使用頁面 ID 和 adoc 副檔名建立新檔案。例如 `create-page.adoc`。
4. 使用此範本插入頁面中繼資料：

```
include::../attributes.txt[]

[.topic]
[#unique-page-id]
= Page title
:info_titleabbrev: Short title

[abstract]
--
Brief summary of the page's content.
--

Introduction paragraph goes here.
```

將頁面新增至導覽

1. 導覽至父頁面。頂層區段的父頁面是 `book.adoc`。
2. 在父頁面的底部，包含子頁面。

   ```
   include::${filename}[leveloffset=+1]
   ```

   例如：

   ```
   include::create-page.adoc[leveloffset=+1]
   ```

插入連結

AsciiDoc 支援多種連結類型。使用正確的連結類型非常重要，這樣連結才可在不同的環境中正常運作。

《EKS 使用者指南》中的頁面或章節的連結

使用交叉參考 (xref) 連結相同文件網站中的頁面或章節，例如《EKS 使用者指南》。如果目標章節移動或重新命名，則它們會自動更新。

使用頁面標題作為連結文字

對於大多數連結至本使用者指南中另一個 ID 的情況，請使用下列方法，以便連結文字視需要自動更新為最新的標題。

```
For more information, see <<page-or-section-id>>.
```

定義自訂連結文字

對於必須使用自訂連結文字的情況，請使用下列格式。

```
Here's an example of a <<page-or-section-id,link with custom text>>.
```

連結至 AWS 文件中的其他指南

1. 尋找 AWS 文件頁面的連結。
2. 移除 `https://docs.aws.amazon.com/` 字首，僅保留路徑。路徑應以字母開頭。
3. 建立連結，如下所示：

```
link:AmazonS3/latest/userguide/create-bucket-overview.html[Create a bucket,
 type="documentation"]
```

外部網頁的連結

此格式會建立指向非 Amazon 託管頁面的標準連結。例如，將其用於 GitHub 連結。

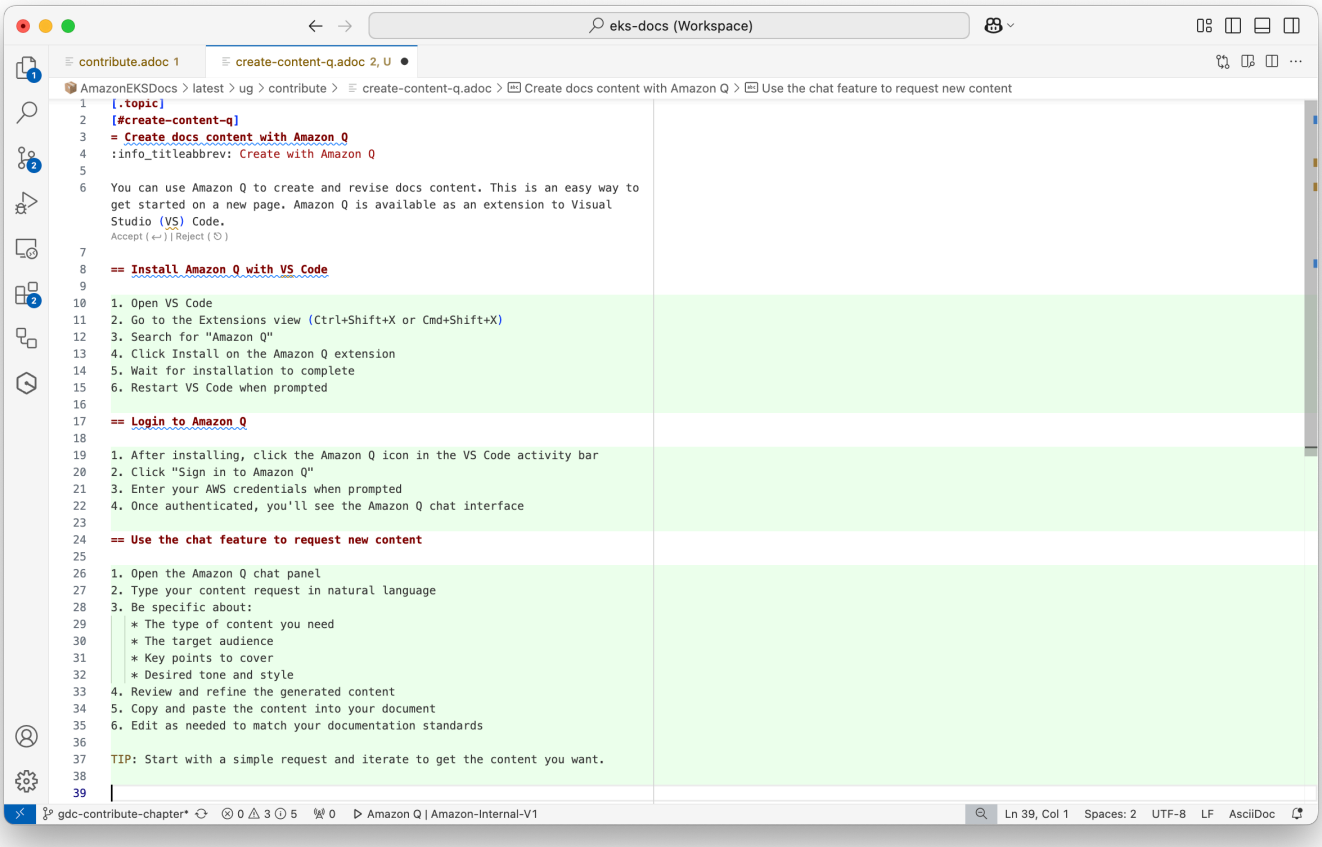
```
https://example.com[Link text]
```

> **Note**
>
> 我們設有外部網域的允許清單。允許清單位於 `vale/styles/EksDocs/ExternalDomains.yml`

使用 Amazon Q 建立文件內容

您可使用 Amazon Q 來建立與修訂文件內容。這是在新頁面上開始的簡便方法。Amazon Q 可作為 Visual Studio (VS) Code 的擴充功能使用。

在下圖中，Amazon Q 產生了以綠色標記的行。



透過 VS Code 安裝 Amazon Q

1. 開啟 VS Code。
2. 前往延伸模組檢視 (Ctrl+Shift+X 或 Cmd+Shift+X)。
3. 搜尋「Amazon Q」。
4. 選擇在 Amazon Q 擴充功能上安裝 。
5. 等待安裝完成。
6. 出現提示時重新啟動 VS 程式碼。

登入 Amazon Q

1. 安裝完成後，選擇 VS Code 活動列中的 Amazon Q 圖示。
2. 選擇登入 Amazon Q。
3. 出現提示時輸入您的 AWS 登入資料。
4. 驗證之後，Amazon Q 聊天介面隨即顯示。

透過 Amazon Q 建立內容

1. 在 VS Code 中開啟要編輯的檔案。
2. 選取您要修訂的文字，或新內容的放置位置。
3. 按下 Ctrl+I 或 Cmd+I。
4. 在提示中，明確指定下列事項：
   - 您需要的內容類型。
   - 目標受眾。
   - 需涵蓋的要點。
   - 所需的語氣和風格。
5. 在內嵌預覽中檢視產生的內容。
6. 接下 Enter 接受變更，或按下 esc 拒絕變更。
7. 根據需要進一步編輯。

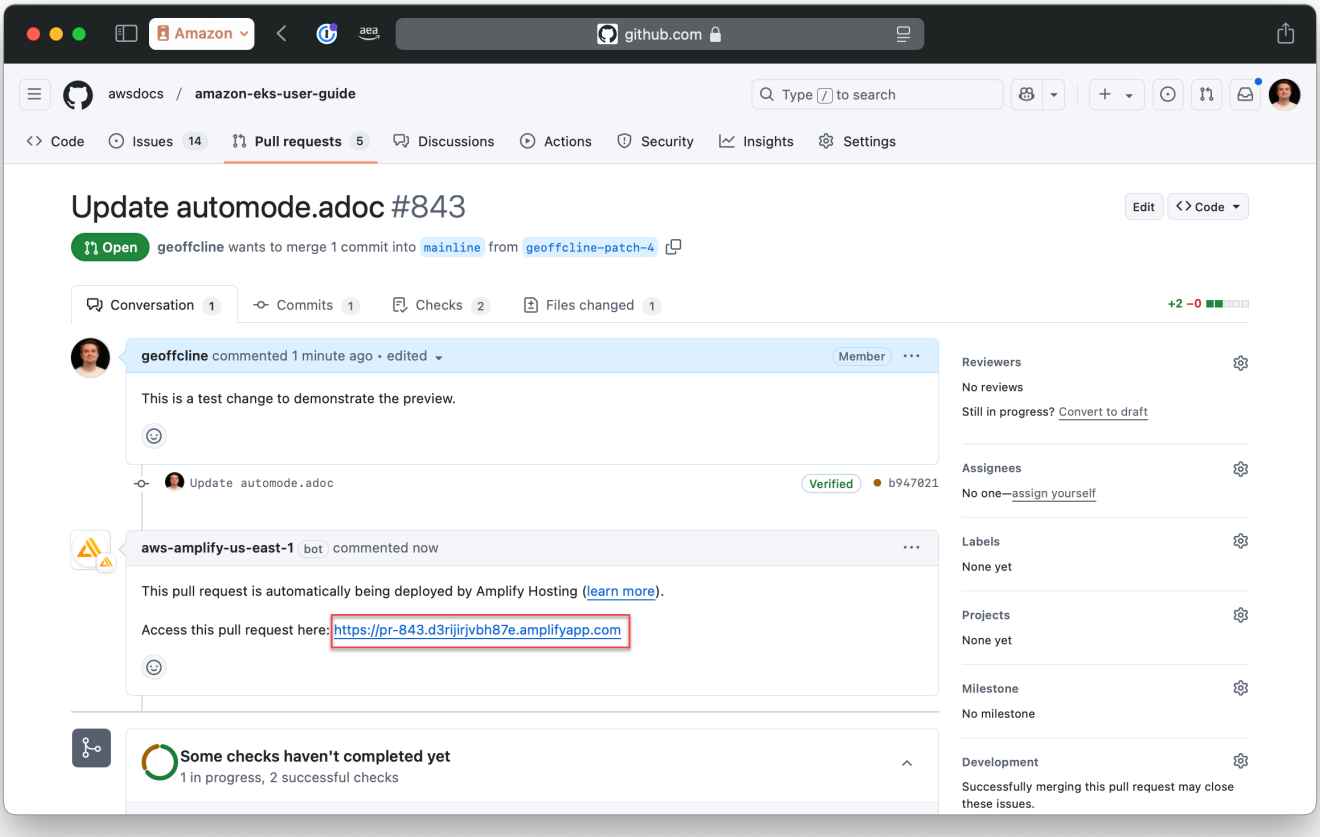
提示

- 從簡單的要求開始，並反覆調整，以取得您想要的內容。
- 先建立頁面標題的初稿，再要求 Q 填入內容。
- Amazon Q 可能會輸出 Markdown 格式。這是可接受的。AsciiDoc 工具可辨識大部分的 Markdown 語法。

若要進一步了解 Amazon Q Developer，請參閱在 IDE 中使用 Amazon Q Developer。

檢視提取請求內容預覽

設定《Amazon EKS 使用者指南》GitHub 以建置及生成文件站點預覽。此預覽沒有完整的 AWS 佈景主題，但確實會正確檢查內容建置和連結。



此預覽由 AWS Amplify 託管於暫時 URL。

檢視預覽

當您提交提取請求時， AWS Amplify 會嘗試建置和部署內容預覽。

若建置成功，aws-amplify-us-east-1 將新增註解至包含預覽連結的提取請求。請點選「在這裡存取此提取請求」右側的連結 (如帶有紅色外框的螢幕擷取畫面所示)。

若建置失敗，儲存庫管理員可查看日誌及提供意見回饋。

> **Note**
>
> 若您之前未參與，專案維護者可能需要對建置的執行進行核准。

預覽限制

預覽將作為單一大型 HTML 檔案來建置。發布後會顯示為多個頁面。

運作內容：

- 交叉參考 (`xref`)
- 網際網路連結
- 映像
- 內容託管來源為 `samples/`

未運作內容：

- 使用 連結到其他 AWS 內容`type="documentation"`。這是由於此內容在預覽環境中不存在。
- 屬性 `{aws}` 未適當顯示。其值會根據環境而變更。

AsciiDoc 語法參考

AsciiDoc 是 AWS 文件的偏好標記語言。儘管工具對 Markdown 語法 (如標題與清單) 有部分支援，但為確保內容一致性與正確轉譯，我們建議所有內容皆使用 AsciiDoc 語法。

本指南涵蓋您在貢獻 Amazon EKS 文件時所需的常見語法元素。如需更進階的語法與詳細資訊，請參閱 AsciiDoc 文件。

新頁面

每個新的 AsciiDoc 文件均應以 the section called “建立頁面” 中定義的結構開頭。

標題

使用標題以階層方式組織內容：

```
= Page/topic title (level 1)
== Section title (level 2)
=== Level 3 heading
==== Level 4 heading
===== Level 5 heading
====== Level 6 heading
```

注意： AWS 文件標題一律使用句子大小寫。

文字格式

透過文字格式化突出重要資訊：

```
Use *bold text* for UI labels.
Use _italic text_ for introducing terms or light emphasis.
Use `monospace text` for code, file names, and commands.
```

清單

無序清單

為無特定先後順序的項目建立項目符號清單：

```
* First item
* Second item
** Nested item
** Another nested item
* Third item
```

有序清單

為循序步驟或有優先順序的項目建立編號清單：

```
. First step
. Second step
.. Substep 1
.. Substep 2
. Third step
```

連結

如需了解如何正確格式化連結的詳細資訊，請參閱 the section called “插入連結”。不支援 Markdown 樣式的連結。

程式碼範例

內嵌程式碼

針對內嵌程式碼使用反引號：

```
Use the `kubectl get pods` command to list all pods.
```

程式碼區塊

建立具備語法反白功能且支援屬性 (類似實體) 的程式碼區塊：

```
[source,python,subs="verbatim,attributes"]
----
def hello_world():
    print("Hello, World!")
----
```

映像

插入映像時，請加上替代文字以確保可存取性：

```
image::images/image-file.png[Alt text description]
```

表格

建立資料表，以組織資訊：

```
[%header,cols="2"]
|===
|Header 1
|Header 2

|Cell 1,1
```

```
|Cell 1,2

|Cell 2,1
|Cell 2,2
|===
```

若需更複雜的資料表，請參閱 AsciiDoc 資料表文件。

標註

使用標註突出重要資訊與提示內容：

```
NOTE: This is a note callout for general information.

TIP: This is a tip callout for helpful advice.

IMPORTANT: This is an important callout for critical information.
```

預覽版：

> **Note**
>
> 這是一個備註標註。

包含其他檔案

包含了來自其他檔案的內容：

```
include::filename.adoc[]
```

屬性 (類似實體)

使用預先定義的屬性以維持一致性。特別是，您必須使用 AWS 和 的屬性 `arn:aws:` 。

```
{aws} provides Amazon EKS as a managed Kubernetes service.
```

```
[source,bash,subs="verbatim,attributes"]
----
```

```
 aws iam attach-role-policy \
     --role-name AmazonEKSAutoClusterRole \
     --policy-arn {arn-aws}iam::aws:policy/AmazonEKSClusterPolicy
 ----
```

如需屬性清單，請參閱 `../attributes.txt` 檔案。

程序

格式化逐步程序：

```
To create an Amazon EKS cluster, do the following steps.

. Sign in to the {aws} Management Console.
. Open the Amazon EKS console.
. Choose *Create cluster*.

...
```