使用者指南

AWS 私有憑證授權單位

版本 latest

AWS 私有憑證授權單位: 使用者指南

Table of Contents

什麼是 AWS 私有 CA？ 1
區域可用性 1
整合服務 2
支援的演算法 2
RFC 5280 合規 3
定價 4
的術語和概念 AWS 私有 CA 4
信任 5
TLS 伺服器憑證 5
憑證簽章 5
憑證授權單位 6
根 CA 6
CA 憑證 6
根 CA 憑證 7
結束實體憑證 7
自我簽署的憑證 7
私有憑證 8
憑證路徑 9
路徑長度限制 9
什麼是最適合我需求的憑證服務？ 10
最佳實務 11
記錄 CA 結構和政策 11
盡可能將根 CA 的使用降至最低 11
為根 CA 提供自己的 AWS 帳戶 12
個別管理員和發行者角色 12
實作憑證的受管撤銷 12
開啟 AWS CloudTrail 12
輪換 CA 私有金鑰 13
刪除未使用的 CAs 13
封鎖公開存取 CRLs 13
Amazon EKS 應用程式最佳實務 13
AWS 私有 CA 搭配 使用 適用於 Java 的 AWS SDK 14
API 範例 14
以程式設計方式建立和啟用根 CA 15

以程式設計方式建立和啟用次級 CA 23
CreateCertificateAuthority 33
使用 CreateCertificateAuthority 支援 Active Directory 37
CreateCertificateAuthorityAuditReport 45
CreatePermission 48
DeleteCertificateAuthority 50
DeletePermission 52
DeletePolicy 54
DescribeCertificateAuthority 56
DescribeCertificateAuthorityAuditReport 59
GetCertificate 61
GetCertificateAuthorityCertificate 64
GetCertificateAuthorityCsr 66
GetPolicy 69
ImportCertificateAuthorityCertificate 71
IssueCertificate 73
ListCertificateAuthorities 77
ListPermissions 81
ListTags 83
PutPolicy 85
RestoreCertificateAuthority 87
RevokeCertificate 89
TagCertificateAuthorities 91
UntagCertificateAuthority 93
UpdateCertificateAuthority 95
使用自訂主旨名稱建立 CAs和憑證 98
建立具有自訂擴充功能的憑證 106
事項範例 121
啟用產品鑑證授權機構 (PAA) 121
啟用產品鑑證中級 (PAI) 132
建立裝置證明憑證 (DAC) 143
為節點操作憑證 (NOC) 啟用根 CA。 147
為節點操作憑證 (NOC) 啟用次級 CA 157
建立節點操作憑證 (NOC) 167
mDL 範例 171
啟用發行授權機構憑證授權機構 (IACA) 憑證 172

建立文件簽署者憑證 181
建構您的 解決方案 AWS 私有 CA 186
設計 CA 階層 186
驗證終端實體憑證 188
規劃 CA 階層的結構 189
設定認證路徑的長度限制 192
管理 CA 生命週期 194
選擇有效期間 194
管理 CA 接續 195
撤銷 CA 196
計劃憑證撤銷 196
要求 198
設定 CRL 199
自訂 OCSP URL 205
CA 模式 207
一般用途 （預設） 208
短期憑證 208
規劃彈性 208
備援和災難復原 209
憑證授權單位 210
設定 211
註冊 AWS 帳戶 211
建立具有管理存取權的使用者 211
安裝 AWS Command Line Interface 212
建立私有 CA 213
CLI 範例 219
安裝 CA 憑證 231
相容簽署演算法 231
安裝根 CA 憑證 233
安裝由 託管的次級 CA 憑證 AWS 私有 CA 240
安裝由外部父 CA 簽署的次級 CA 憑證 241
控制存取 242
為 IAM 使用者建立單一帳戶許可 242
連接跨帳戶存取的政策 245
列出私有 CAs 247
檢視私有 CA 248

新增標籤 251
CA 狀態 253
CA 狀態與 CA 生命週期之間的關係 255
更新 CA 256
更新 CA （主控台） 256
更新 CA (CLI) 259
刪除 CA 267
還原 CA 268
還原私有 CA （主控台） 269
還原私有 CA (AWS CLI) 269
外部簽署的 CA 憑證 270
發行和管理憑證 274
發行私有終端實體憑證 274
發行標準憑證 (AWS CLI) 275
使用 APIPassthrough 範本發行具有自訂主旨名稱的憑證 277
使用 APIPassthrough 範本發行具有自訂擴充功能的憑證 280
擷取私有憑證 281
列出私有憑證 282
匯出憑證 287
撤銷私有憑證 287
撤銷的憑證和 OCSP 288
CRL 中的已撤銷憑證 288
稽核報告中的已撤銷憑證 289
自動化匯出 290
憑證範本 291
範本變體 291
範本操作順序 302
範本定義 303
安全 342
IAM 342
API 許可 343
AWS 受管政策 348
客戶管理政策 350
內嵌政策 352
跨帳戶存取權 357
資源型政策 358

資料保護 361
AWS 私有 CA 私有金鑰的儲存和安全性合規 362
AWS 私有 CA Connector for Active Directory 中的資料加密 362
法規遵循驗證 362
建立稽核報告 363
基礎設施安全性 369
VPC 端點 (AWS PrivateLink) 370
雙堆疊端點支援 374
在 IAM 和 中使用 IPv6 地址 AWS 私有 CA 374
CP/CPS 375
CP/CPS 要求和責任 376
監控 資源 383
AWS 私有 CA CloudWatch 指標 383
AWS 私有 CA 使用 CloudWatch Events 監控 384
建立私有 CA 時成功或失敗 384
發出憑證時成功或失敗 385
撤銷憑證時成功 387
產生 CRL 時成功或失敗 387
建立 CA 稽核報告時成功或失敗 389
CloudTrail 日誌 391
AWS 私有 CA CloudTrail 中的資訊 391
AWS 私有 CA 管理事件 392
範例 AWS 私有 CA 事件 393
疑難排解 396
憑證撤銷問題 396
OCSP 回應延遲 396
撤銷自我簽署憑證 396
例外狀況訊息 396
符合事項的憑證錯誤 398
使用 保護 Kubernetes AWS 私有 CA 401
概念 401
考量 403
cert-manager 的跨帳戶使用 403
開始使用 404
安裝 cert-manager 407
設定 IAM 許可 407

安裝和設定 AWS 私有 CA 叢集發行者 410
使用 cert-manager 管理 AWS 私有 CA 用戶端憑證 414
發行您的第一個 TLS 憑證 415
範例 416
監控 416
疑難排解 417
適用於 Active Directory 的連接器 362
您是 AD 使用者的第一次連接器嗎？ 419
適用於 AD 的 Access Connector 419
定價 419
設定 420
步驟 1：使用 建立私有 CA AWS 私有 CA 420
步驟 2：設定 Active Directory 420
（僅限 Active Directory Connector) 步驟 3：將許可委派給服務帳戶 420
步驟 4：建立 IAM 政策 421
步驟 5：與 Connector for AD 共用您的私有 CA 424
步驟 6：建立目錄註冊 424
步驟 7：設定安全群組 425
步驟 8：設定目錄物件的網路存取 425
開始使用 426
開始之前 426
步驟 1：建立連接器 426
步驟 2：設定 Microsoft Active Directory 政策 426
步驟 3：建立範本 428
步驟 4：設定 Microsoft 群組許可 428
Active Directory 的連接器 428
建立連接器 428
建立範本 430
更新範本 434
列出連接器 435
清單範本 436
檢視連接器 437
檢視範本 438
目錄註冊 440
範本存取控制項目 442
服務委託人名稱 443

Tags (標籤) 444
與 EventBridge 整合 445
EventBridge 如何路由 Connector for AD 事件 446
適用於 AD 事件的連接器 446
建立事件模式 447
接收事件 447
故障診斷適用於 Active Directory 的 Connector 447
Connector for AD 錯誤代碼 448
連接器建立失敗 452
SPN 建立失敗 455
範本更新問題 456
適用於 SCEP 的連接器 458
功能 458
如何開始使用 Connector for SCEP 459
相關服務 459
適用於 SCEP 的 Access Connector 459
定價 459
概念 460
考量和限制 461
考量事項 461
限制 462
設定 462
步驟 1：建立 AWS Identity and Access Management 政策 462
步驟 2：建立私有 CA 464
步驟 3：建立資源共享 464
開始使用 465
開始之前 465
步驟 1：建立連接器 466
步驟 2：將連接器詳細資訊複製到 MDM 系統 467
設定 MDM 系統 468
一般用途連接器 468
AWS 私有 CA Connector for SCEP for Microsoft Intune 469
設定 Jamf Pro 470
設定 Microsoft Intune 476
設定 Omnissa Workspace ONE 478
監控 483

使用 EventBridge 進行自動化 484
CloudTrail 日誌 489
疑難排解 497
HTTP 錯誤 497
客戶端錯誤 511
Service Quotas 512
文件歷史記錄 513
舊版更新 519
..... dxx

什麼是 AWS 私有 CA？

AWS 私有 CA 可建立私有憑證授權機構 (CA) 階層，包括根 CA 和次級 CAs，而不需要操作內部部署 CA 的投資和維護成本。您的私有 CA 可以發行終端實體 X.509 憑證，在下列案例中很實用：

- 建立加密的 TLS 通訊通道
- 對使用者、電腦、API 端點和 IoT 裝置進行身分驗證
- 透過加密技術簽署程式碼
- 實作線上憑證狀態通訊協定 (OCSP) 以取得憑證撤銷狀態

AWS 私有 CA 操作可從 AWS 管理主控台、使用 AWS 私有 CA API 或使用 存取 AWS CLI。

主題

- 的區域可用性 AWS 私有憑證授權單位
- 與 整合的服務 AWS 私有憑證授權單位
- 中的支援密碼編譯演算法 AWS 私有憑證授權單位
- 中的 RFC 5280 合規 AWS 私有憑證授權單位
- 的定價 AWS 私有憑證授權單位
- 的術語和概念 AWS 私有 CA

的區域可用性 AWS 私有憑證授權單位

與大多數 AWS 資源一樣，私有憑證授權機構 CAs) 是區域資源。若要在多個區域使用私有 CA，您必須在這些區域建立 CA。您無法在區域間複製私有 CA。請造訪 AWS 中的區域和端點AWS 一般參考或AWS 區域資料表，以查看 的區域可用性 AWS 私有 CA。

> **Note**
>
> ACM 目前可在部分 AWS 私有 CA 區域使用。

與 整合的服務 AWS 私有憑證授權單位

如果您使用 AWS Certificate Manager 請求私有憑證，您可以將該憑證與任何與 ACM 整合的服務建立關聯。這同時適用於鏈結至 AWS 私有 CA 根的憑證，以及鏈結至外部根的憑證。如需詳細資訊，請參閱 AWS Certificate Manager 《使用者指南》中的整合式服務。

您也可以將私有 CAs 整合到 Amazon Elastic Kubernetes Service，以在 Kubernetes 叢集內提供憑證發行。如需詳細資訊，請參閱使用 保護 Kubernetes AWS 私有憑證授權單位。

> **Note**
>
> Amazon Elastic Kubernetes Service 不是 ACM 整合服務。

如果您使用 AWS 私有 CA API 或 AWS CLI 來發行憑證，或從 ACM 匯出私有憑證，您可以隨心所欲地安裝憑證。

中的支援密碼編譯演算法 AWS 私有憑證授權單位

AWS 私有 CA 支援下列用於產生私有金鑰和憑證簽署的密碼編譯演算法。

支援的演算法

私有金鑰演算法	簽署演算法
ML_DSA_44	ML_DSA_44
ML_DSA_65	ML_DSA_65
ML_DSA_87	ML_DSA_87
RSA_2048	SHA256WITHRSA
RSA_3072	SHA384WITHRSA
RSA_4096	SHA512WITHRSA
EC_prime256v1	SHA256WITHECDSA
EC_secp384r1	SHA384WITHECDSA
	SHA512WITHECDSA

私有金鑰演算法	簽署演算法
EC_secp521r1 SM2 (僅限中國區域)	SM3WITHSM2

此清單僅適用於 AWS 私有 CA 直接透過其主控台、API 或命令列發行的憑證。當 從 使用 CA AWS Certificate Manager 發行憑證時 AWS 私有 CA，它支援部分但並非所有的演算法。如需詳細資訊，請參閱 AWS Certificate Manager 《 使用者指南》中的請求私有憑證。

> **Note**
>
> 對於 RSA 或 ECDSA，指定的簽署演算法系列必須符合 CA 私有金鑰的金鑰演算法系列。
> 對於 ML-DSA，雜湊函數定義為演算法本身的一部分。沒有使用 ML-DSA 選取不同雜湊函數的選項。為了維持與 APIs的回溯相容性，金鑰演算法和簽署演算法會使用相同的值。

中的 RFC 5280 合規 AWS 私有憑證授權單位

AWS 私有 CA 不會強制執行 RFC 5280 中定義的某些限制條件。相反的情況也是如此：適用於私有 CA 的某些額外限制會強制執行。

強制執行

- 不在日期後。為了符合 RFC 5280 的規範，在發行 CA 憑證的 `Not After` 日期後， AWS 私有 CA 會防止發行具有 `Not After` 的憑證。
- 基本限制條件。 AWS 私有 CA 強制執行匯入 CA 憑證中的基本限制條件和路徑長度。

  基本限制條件會指出憑證所識別的資源是否為 CA 且可以發行憑證。匯入 AWS 私有 CA 的 CA 憑證必須包含基本限制延伸，且延伸必須標記為 `critical`。除了 `critical`旗標之外，還`CA=true`必須設定 。由於下列原因，驗證例外狀況失敗，以 AWS 私有 CA 強制執行基本限制條件：

  - 延伸不包含在 CA 憑證中。
  - 延伸未標記為 `critical`。

  路徑長度 (pathLenConstraint) 會判斷在匯入 CAs憑證下游可能有多少次級 CA。由於下列原因，驗證例外狀況失敗，因此 AWS 私有 CA 強制執行路徑長度：

  - 匯入 CA 憑證會違反 CA 憑證中 (或鏈結中的任何 CA 憑證中) 的路徑長度限制條件。

  - 發行憑證將違反路徑長度限制條件。
- 名稱限制條件表示名稱空間，認證路徑中後續憑證中的所有主體名稱都必須位於其中。限制適用於主體辨別名稱和主體替代名稱。

未強制執行

- 憑證政策。憑證政策會規範 CA 發行憑證的條件。
- 抑制 anyPolicy。用於發行給 CAs憑證。
- 發行者替代名稱。允許其他身分與 CA 憑證的發行者相關聯。
- 政策限制。這些限制條件會限制 CA 發行次級 CA 憑證的容量。
- 政策映射。用於 CA 憑證。列出一或多個 OIDs 對；每一對都包含一個 issuerDomainPolicy 和一個 subjectDomainPolicy。
- 主旨目錄屬性。用來傳達主體的識別屬性。
- 主體資訊存取。如何存取延伸項目出現之憑證主體的資訊和服務。
- 主體金鑰識別符 (SKI) 和授權機構金鑰識別符 (AKI)。RFC 需要 CA 憑證，才能包含 SKI 延伸。CA 發行的憑證必須包含符合 CA 憑證 SKI 的 AKI 延伸。 AWS 不會強制執行這些要求。如果您的 CA 憑證不包含 SKI，則發行的終端實體或次級 CA 憑證 AKI，將會改為發行者公有金鑰的 SHA-1 雜湊。
- SubjectPublicKeyInfo 和 主體替代名稱 (SAN)。發行憑證時， AWS 私有 CA 會從提供的 CSR 複製 SubjectPublicKeyInfo 和 SAN 延伸，而不會執行驗證。

的定價 AWS 私有憑證授權單位

從您帳戶的建立時間開始，會針對每個私有 CA 每月向您收取費用。也會向您收取您發行的每個憑證費用。此費用包含您從 ACM 匯出的憑證，以及您從 AWS 私有 CA API 或 CLI AWS 私有 CA 建立的憑證。私有 CA 刪除後，您不需付費。不過，如果您還原私有 CA，您需支付刪除到還原這段期間的費用。您無權存取其私密金鑰的私有憑證是免費的。其中包括與 Elastic Load Balancing、CloudFront 和 API Gateway 等整合式服務搭配使用的憑證。

如需最新的 AWS 私有 CA 定價資訊，請參閱 AWS 私有憑證授權單位 定價。您也可以使用AWS 定價計算器來估計成本。

的術語和概念 AWS 私有 CA

下列術語和概念可協助您使用 AWS 私有憑證授權單位。

主題

- 信任
- TLS 伺服器憑證
- 憑證簽章
- 憑證授權單位
- 根 CA
- CA 憑證
- 根 CA 憑證
- 結束實體憑證
- 自我簽署的憑證
- 私有憑證
- 憑證路徑
- 路徑長度限制

信任

為了讓 Web 瀏覽器信任網站的身分，瀏覽器必須能驗證網站的憑證。不過，瀏覽器只信任少數稱為 CA 根憑證的憑證。稱為憑證授權機構 (CA) 的信任第三方會驗證網站的身分，並將已簽署的數位憑證發給網站的營運商。然後，瀏覽器便可檢查數位簽章，以驗證網站的身分。如果驗證成功，瀏覽器會在網址列中顯示鎖定圖示。

TLS 伺服器憑證

HTTPS 交易需要伺服器憑證，才能對伺服器進行身分驗證。伺服器憑證是 X.509 v3 資料結構，將憑證中的公有金鑰繫結至憑證主體。TLS 憑證由憑證授權機構 (CA) 簽署。它包含伺服器的名稱、有效期、公有金鑰、簽章演算法等。

憑證簽章

數位簽章是透過憑證的加密雜湊。簽章是用於確認憑證資料的完整性。私有 CA 會在變數大小的憑證內容上使用加密雜湊函數 (例如 SHA256) 來建立簽章。這個雜湊函數會產生有效不可偽造的固定大小資料字串。此字串稱為雜湊。然後，CA 會使用其私密金鑰來加密雜湊值，並使用憑證串連加密雜湊。

憑證授權單位

憑證授權機構 (CA) 發行及 (在必要時) 撤銷數位憑證。最常見的憑證類型是根據 ISO X.509 標準。X.509 憑證會確認憑證主體的身分，並將該身分繫結至公開金鑰。主體可以是使用者、應用程式、電腦或其他裝置。CA 透過雜湊內容來簽署憑證，然後使用憑證中與公有金鑰相關的私有金鑰來加密雜湊。Web 瀏覽器等需要確認主體身分的用戶端應用程式使用公開金鑰來解密憑證簽章。然後，用戶端應用程式會雜湊憑證內容，並將雜湊值與解密簽章進行比較，以判斷是否相符。如需憑證簽署的相關資訊，請參閱 憑證簽章。

您可以使用 來 AWS 私有 CA 建立私有 CA，並使用私有 CA 來發行憑證。私有 CA 只發行私有 SSL/TLS 憑證，以供組織內使用。如需詳細資訊，請參閱私有憑證。您的私有 CA 還需要憑證才能使用。如需詳細資訊，請參閱CA 憑證。

根 CA

是一種加密建置區塊和信任根，可做為發行憑證的根據。其包含私有金鑰，用於簽署 (發行) 憑證和根憑證，這些憑證可識別根 CA 並將私有金鑰繫結至 CA 名稱。根憑證會分佈到環境中每個實體的信任存放區。管理員會建構信任存放區以僅包含其信任的 CA。管理員會更新或建置信任存放區到其環境中實體的作業系統、執行個體和主機映像。當資源嘗試互相連線時，會檢查每個實體呈現的憑證狀態。用戶端會檢查憑證是否有效，以及從憑證到信任存放區中安裝的根憑證是否存在鏈結。如果符合這些條件，則資源之間會完成「握手」。此交握會以加密方式來互相證實各實體的身分，然後在這些實體之間建立加密的通訊通道 (TLS/SSL)。

CA 憑證

憑證授權機構 (CA) 憑證確認 CA 的身分，並將該身分繫結至憑證所含的公開金鑰。

您可以使用 來 AWS 私有 CA 建立私有根 CA 或私有次級 CA，每個 CA 都由 CA 憑證支援。次級 CA 憑證由信任鏈結中較高層的另一個 CA 憑證所簽署。但是在根 CA 的情況下，憑證會自我簽署。您也可以建立外部根授權機構 (例如裝載在內部部署)。然後您就可以使用根授權機構來簽署由 AWS 私有 CA 裝載的次級根 CA 憑證。

下列範例顯示 AWS 私有 CA X.509 CA 憑證中包含的典型欄位。請注意，針對 CA 憑證，`CA:` 欄位中的 `Basic Constraints` 值設為 `TRUE`。

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 4121 (0x1019)
```

```
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=US, ST=Washington, L=Seattle, O=Example Company Root CA, OU=Corp,
CN=www.example.com/emailAddress=corp@www.example.com
        Validity
            Not Before: Feb 26 20:27:56 2018 GMT
            Not After : Feb 24 20:27:56 2028 GMT
        Subject: C=US, ST=WA, L=Seattle, O=Examples Company Subordinate CA,
OU=Corporate Office, CN=www.example.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:c0: ... a3:4a:51
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                F8:84:EE:37:21:F2:5E:0B:6C:40:C2:9D:C6:FE:7E:49:53:67:34:D9
            X509v3 Authority Key Identifier:
                keyid:0D:CE:76:F2:E3:3B:93:2D:36:05:41:41:16:36:C8:82:BC:CB:F8:A0

            X509v3 Basic Constraints: critical
                CA:TRUE
            X509v3 Key Usage: critical
                Digital Signature, CRL Sign
    Signature Algorithm: sha256WithRSAEncryption
         6:bb:94: ... 80:d8
```

根 CA 憑證

憑證授權機構 (CA) 通常存在於階層結構中，其中包含多個其他 CAs並在兩者之間明確定義父子關係。下層 CA 或次級 CA 是由上層 CA 認證，並形成憑證鏈。階層頂端的 CA 稱為根 CA，其憑證稱為根憑證。此憑證通常為自我簽署。

結束實體憑證

終端實體憑證可識別資源，例如伺服器、執行個體、容器或裝置。與 CA 憑證不同，終端實體憑證無法用來發行憑證。終端實體憑證的其他常用術語是「用戶端」或「葉子」憑證。

自我簽署的憑證

由發行者 (而不是較高的 CA) 簽署的憑證。與 CA 維護的安全根發行的憑證不同，自我簽署的憑證會做為自己的根，因此具有重大限制：它們可用來在線路加密上提供，但無法驗證身分，而且無法撤銷。

從安全角度來看，它們是不可接受的。但組織仍會使用這種憑證，因為易於產生、不需要專門技術或基礎設施，而且許多應用程式都能接受。自我簽署憑證的發行無法控管。由於無法追蹤這種憑證的過期日期，所以使用自我簽署憑證的組織，因憑證過期導致服務中斷時間的風險較高。

私有憑證

AWS 私有 CA 憑證是私有 SSL/TLS 憑證，您可以在組織中使用，但在公有網際網路上不受信任。您可以使用這些憑證來識別資源，例如用戶端、伺服器、應用程式、服務、裝置和使用者。建立安全加密通訊通道時，每個資源皆會使用如下所示的憑證和加密技術來證明其對另一個資源的身分。內部 API 端點、Web 伺服器、VPN 使用者、IoT 裝置和其他多種類型的應用程式皆使用私有憑證來建立安全操作需要的加密通訊通道。在預設情況下，私有憑證不受公開信任。內部管理員必須明確設定應用程式，以信任私有憑證及分配憑證。

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            e8:cb:d2:be:db:12:23:29:f9:77:06:bc:fe:c9:90:f8
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=US, ST=WA, L=Seattle, O=Example Company CA, OU=Corporate,
 CN=www.example.com
        Validity
            Not Before: Feb 26 18:39:57 2018 GMT
            Not After : Feb 26 19:39:57 2019 GMT
        Subject: C=US, ST=Washington, L=Seattle, O=Example Company, OU=Sales,
 CN=www.example.com/emailAddress=sales@example.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00...c7
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Authority Key Identifier:
                keyid:AA:6E:C1:8A:EC:2F:8F:21:BC:BE:80:3D:C5:65:93:79:99:E7:71:65

            X509v3 Subject Key Identifier:
                C6:6B:3C:6F:0A:49:9E:CC:4B:80:B2:8A:AB:81:22:AB:89:A8:DA:19
            X509v3 Key Usage: critical
                Digital Signature, Key Encipherment
```

```
            X509v3 Extended Key Usage:
                TLS Web Server Authentication, TLS Web Client Authentication
            X509v3 CRL Distribution Points:

                Full Name:
                  URI:http://NA/crl/12345678-1234-1234-1234-123456789012.crl

    Signature Algorithm: sha256WithRSAEncryption
         58:32:...:53
```

憑證路徑

依賴憑證的用戶端會驗證路徑是否存在於最終實體憑證，可能透過中繼憑證鏈，到達信任的根目錄。用戶端會檢查路徑上的每個憑證是否有效 (未遭撤銷)。它也會檢查最終實體憑證是否尚未過期、完整性（未遭到竄改或修改），以及憑證中的限制是否受到強制執行。

路徑長度限制

CA 憑證的基本限制條件 pathLenConstraint 會設定其下方鏈結中可能存在的次級 CA 憑證數量。例如，路徑長度限制為零的 CA 憑證不能有任何次級 CAs。路徑長度限制條件為 1 的 CA 下方，最多可以有一個層級的次級 CA。RFC 5280 將此定義為「在有效憑證路徑中可遵循此憑證non-self-issued中繼憑證數量上限。」 路徑長度值不包括最終實體憑證，但有關驗證鏈的「長度」或「深度」的非正式語言可能包括它...導致混淆。

什麼是最適合我需求的憑證服務？

發行和部署 X.509 憑證有兩種 AWS 服務。選擇最符合您需求的服務。考量包括您是否需要公有或私有憑證、自訂憑證、要部署到其他服務的憑證 AWS ，或是自動憑證管理和續約。

1. AWS 私有 CA — 此服務適用於在 AWS 雲端內建置公有金鑰基礎結構 (PKI)，並預計供組織內私人使用的企業客戶。使用 AWS 私有 CA，您可以建立自己的 CA 階層並發行憑證，以驗證內部使用者、電腦、應用程式、服務、伺服器和其他裝置，以及簽署電腦程式碼。由私有 CA 發行的憑證，只會在您的組織內受信任，而不會在網際網路上受信任。

   建立私有 CA 之後，您可以直接發行憑證 (也就是不需要從第三方 CA 取得驗證)，並進行自訂以符合組織的內部需求。例如，建議您：

   - 使用任何主體名稱建立憑證。
   - 使用任何過期日期建立憑證。
   - 使用任何支援的私密金鑰演算法和金鑰長度。
   - 使用任何支援的簽署演算法。
   - 使用範本來控制憑證發行。

   您正位於此服務的正確位置。若要開始使用，請登入 https://console.aws.amazon.com/acm-pca/ 主控台。
2. AWS Certificate Manager (ACM) — 此服務會使用 TLS 為需要公開信任的安全 Web 存在的企業客戶管理憑證。您可以將 ACM 憑證部署到 AWS Elastic Load Balancing、Amazon CloudFront、Amazon API Gateway 和其他整合服務。最常見的這類應用是具有龐大流量要求的安全公有網站。

   透過此服務，您可以使用 ACM 提供的公有憑證 (&ACM; 憑證) 或匯入 ACM 的憑證。如果您使用 AWS 私有 CA 來建立 CA，ACM 可以管理該私有 CA 的憑證發行，並自動化憑證續約。

   如需詳細資訊，請參閱《AWS Certificate Manager 使用者指南》。

AWS 私有 CA 最佳實務

最佳實務是可協助您 AWS 私有 CA 有效使用的建議。下列最佳實務是以目前 AWS Certificate Manager 和 AWS 私有 CA 客戶的真實體驗為基礎。

記錄 CA 結構和政策

AWS 建議您記錄所有政策和實務，以操作您的 CA。這可能包括：

- CA 結構決策的理由
- 顯示 CA 及其關係的圖表
- CA 有效期間的政策
- CA 繼承規劃
- 路徑長度政策
- 許可目錄
- 管理控制結構的說明
- 安全

您可以在兩份文件中擷取此資訊，稱為認證原則 (CP) 和認證實務聲明 (CPS)。如需了解擷取 CA 操作重要資訊的框架，請參閱 RFC 3647。

盡可能將根 CA 的使用降至最低

一般而言，根 CA 應僅用於發行中繼 CA 的憑證。這可以讓您透過不會造成損害的方式存放根 CA，並讓中繼 CA 執行發行終端實體憑證的日常任務。

不過，如果您組織的目前做法是直接從根 CA 發行最終實體憑證， AWS 私有 CA 可以支援此工作流程，同時改善安全性和操作控制。在此案例中發行終端實體憑證需要 IAM 許可政策，允許您的根 CA 使用終端實體憑證範本。如需 IAM 政策的資訊，請參閱 的 Identity and Access Management (IAM) AWS 私有憑證授權單位。

> **Note**
>
> 此組態會施加可能導致操作挑戰的限制。例如，如果您的根 CA 遭到洩露或遺失，您必須建立新的根 CA，並將其分配到您環境中所有的用戶端。在完成此復原程序前，您將無法發行新的

憑證。直接從根 CA 發行憑證也會讓您無法限制存取及限制從您根發行的憑證數，即使這兩種皆是管理根 CA 的最佳實務。

為根 CA 提供自己的 AWS 帳戶

在兩個不同 AWS 帳戶中建立根 CA 和次級 CA 是建議的最佳實務。如此可以為您提供額外的保護，以及您根 CA 的存取控制。您可以透過將 CSR 從一個帳戶中的次級 CA 匯出，然後在不同帳戶中使用根 CA 進行簽署。這種方法的優點是您可以根據帳戶區別 CA 的控制。缺點是您無法使用 AWS 管理主控台 精靈簡化從根 CA 簽署次級 CA 的 CA 憑證的程序。

Important

強烈建議您隨時存取時使用多重要素驗證 (MFA) AWS 私有 CA。

個別管理員和發行者角色

CA 管理員角色應與只需要發行最終實體憑證的使用者分開。如果您的 CA 管理員和憑證發行者位於相同的 中 AWS 帳戶，您可以建立專門用於該目的的 IAM 使用者，來限制發行者許可。

實作憑證的受管撤銷

受管撤銷會在憑證遭到撤銷時，自動向憑證用戶端提供通知。如果憑證的密碼編譯資訊遭到洩露或錯誤發出，您可能需要撤銷憑證。用戶端通常會拒絕接受已撤銷的憑證。 AWS 私有 CA 提供兩種標準選項來管理撤銷：線上憑證狀態通訊協定 (OCSP) 和憑證撤銷清單 (CRLs)。如需詳細資訊，請參閱規劃您的 AWS 私有 CA 憑證撤銷方法。

開啟 AWS CloudTrail

在建立和開始操作私有 CA 之前，請開啟 CloudTrail 記錄。使用 CloudTrail，您可以擷取帳戶 AWS API 呼叫的歷史記錄，以監控您的 AWS 部署。此歷史記錄包含從 AWS 管理主控台、 AWS SDKs AWS Command Line Interface、 和更高層級 AWS 服務的 API 呼叫。您也可以找出哪些使用者和帳戶呼叫 PCA API 操作、發出呼叫的來源 IP 地址，以及呼叫的發生時間。您可以使用 API 將 CloudTrail 整合至應用程式，以自動建立組織的追蹤記錄、查看追蹤記錄的狀態，並控制管理員開啟和關閉 CloudTrail 記錄功能的方式。如需詳細資訊，請參閱建立追蹤記錄。前往 使用 記錄 AWS 私有憑證授權單位 API 呼叫 AWS CloudTrail 查看 AWS 私有 CA 操作的範例追蹤。

輪換 CA 私有金鑰

這是為您的私有 CA 定期更新私密金鑰的最佳實務。您可以透過匯入新的 CA 憑證，或是將私有 CA 取代為新的 CA，來更新金鑰。

> **Note**
>
> 如果您取代 CA 本身，請注意 CA 的 ARN 會變更。這會導致依賴硬式編碼 ARN 的自動化失敗。

刪除未使用的 CAs

您可以永久刪除私有 CA。如果您不再需要 CA，或想要以具有較新私密金鑰的 CA 取代現有 CA，可執行此動作。若要安全刪除 CA，我們建議您依照 刪除您的私有 CA 中所述的流程操作。

> **Note**
>
> AWS 會向您收取 CA 的費用，直到刪除為止。

封鎖公開存取 CRLs

AWS 私有 CA 建議在包含 CRLs儲存貯體上使用 Amazon S3 Block Public Access (BPA) 功能。這可避免不必要地將私有 PKI 的詳細資訊公開給潛在的對手。BPA 是 S3 最佳實務，並預設會在新的儲存貯體上啟用。在某些情況下，需要額外設定。如需詳細資訊，請參閱使用 CloudFront 啟用 S3 封鎖公開存取 (BPA)。

Amazon EKS 應用程式最佳實務

使用 AWS 私有 CA 以 X.509 憑證佈建 Amazon EKS 時，請遵循 Amazon EKS 最佳實務指南中保護多租戶環境的建議。如需 AWS 私有 CA 整合 Kubernetes 的一般資訊，請參閱 使用 保護 Kubernetes AWS 私有憑證授權單位。

AWS 私有 CA 搭配 使用 適用於 Java 的 AWS SDK

您可以使用 AWS 私有憑證授權單位 API 透過傳送 HTTP 請求，以程式設計方式與服務互動。服務會傳回 HTTP 回應。如需詳細資訊，請參閱 AWS 私有憑證授權單位 API 參考。

除了 HTTP API 之外，您還可以使用 AWS SDKs和命令列工具來互動 AWS 私有 CA。建議透過 HTTP API 進行。如需詳細資訊，請參閱 Amazon Web Services 適用工具。下列主題說明如何使用 適用於 Java 的 AWS SDK 來程式設計 AWS 私有 CA API。

GetCertificateAuthorityCsr、GetCertificate 和 DescribeCertificateAuthorityAuditReport 操作支援等待者。您可以使用等待程式以根據特定資源的存在或狀態來控制程式碼的進度。如需詳細資訊，請參閱 AWS 開發人員部落格中的下列主題以及 中的等待者 適用於 Java 的 AWS SDK。

AWS 私有 CA API 範例

下列程式碼範例示範如何搭配 使用選取的 AWS 私有 CA API 動作和資料類型 適用於 Java 的 AWS SDK。

主題

- 以程式設計方式建立和啟用根 CA
- 以程式設計方式建立和啟用次級 CA
- CreateCertificateAuthority
- 使用 CreateCertificateAuthority 支援 Active Directory
- CreateCertificateAuthorityAuditReport
- CreatePermission
- DeleteCertificateAuthority
- DeletePermission
- DeletePolicy
- DescribeCertificateAuthority
- DescribeCertificateAuthorityAuditReport
- GetCertificate
- GetCertificateAuthorityCertificate
- GetCertificateAuthorityCsr
- GetPolicy

- ImportCertificateAuthorityCertificate
- IssueCertificate
- ListCertificateAuthorities
- ListPermissions
- ListTags
- PutPolicy
- RestoreCertificateAuthority
- RevokeCertificate
- TagCertificateAuthorities
- UntagCertificateAuthority
- UpdateCertificateAuthority
- 使用自訂主旨名稱建立 CAs和憑證
- 建立具有自訂擴充功能的憑證

以程式設計方式建立和啟用根 CA

此 Java 範例示範如何使用下列 AWS 私有 CA API 動作啟用根 CA：

- CreateCertificateAuthority
- GetCertificateAuthorityCsr
- IssueCertificate
- GetCertificate
- ImportCertificateAuthorityCertificate

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.samples.GetCertificateAuthorityCertificate;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
```

```
import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.ASN1Subject;
import com.amazonaws.services.acmpca.model.CertificateAuthorityConfiguration;
import com.amazonaws.services.acmpca.model.CertificateAuthorityType;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityResult;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityRequest;
import com.amazonaws.services.acmpca.model.CrlConfiguration;
import com.amazonaws.services.acmpca.model.KeyAlgorithm;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Tag;

import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.Objects;

import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrRequest;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrResult;
import com.amazonaws.services.acmpca.model.GetCertificateRequest;
import com.amazonaws.services.acmpca.model.GetCertificateResult;
import
 com.amazonaws.services.acmpca.model.ImportCertificateAuthorityCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateResult;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Validity;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.CertificateMismatchException;
import com.amazonaws.services.acmpca.model.ConcurrentModificationException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidPolicyException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.MalformedCertificateException;
import com.amazonaws.services.acmpca.model.MalformedCSRException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.RequestInProgressException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.RevocationConfiguration;
import com.amazonaws.services.acmpca.model.AWSACMPCAException;
```

```
import com.amazonaws.waiters.Waiter;
import com.amazonaws.waiters.WaiterParameters;
import com.amazonaws.waiters.WaiterTimedOutException;
import com.amazonaws.waiters.WaiterUnrecoverableException;

public class RootCAActivation {
    public static void main(String[] args) throws Exception {
        // Define the endpoint region for your sample.
        String endpointRegion = "region"; // Substitute your region here, e.g. "us-
west-2"

        // Define a CA subject.
        ASN1Subject subject = new ASN1Subject();
        subject.setOrganization("Example Organization");
        subject.setOrganizationalUnit("Example");
        subject.setCountry("US");
        subject.setState("Virginia");
        subject.setLocality("Arlington");
        subject.setCommonName("www.example.com");

        // Define the CA configuration.
        CertificateAuthorityConfiguration configCA = new
 CertificateAuthorityConfiguration();
        configCA.withKeyAlgorithm(KeyAlgorithm.RSA_2048);
        configCA.withSigningAlgorithm(SigningAlgorithm.SHA256WITHRSA);
        configCA.withSubject(subject);

        // Define a certificate revocation list configuration.
        CrlConfiguration crlConfigure = new CrlConfiguration();
        crlConfigure.withEnabled(true);
        crlConfigure.withExpirationInDays(365);
        crlConfigure.withCustomCname(null);
        crlConfigure.withS3BucketName("your-bucket-name");

        // Define a certificate authority type
        CertificateAuthorityType CAtype = CertificateAuthorityType.ROOT;

        // ** Execute core code samples for Root CA activation in sequence **
        AWSACMPCA client = ClientBuilder(endpointRegion);
        String rootCAArn = CreateCertificateAuthority(configCA, crlConfigure, CAtype,
 client);
        String csr = GetCertificateAuthorityCsr(rootCAArn, client);
        String rootCertificateArn = IssueCertificate(rootCAArn, csr, client);
```

```
        String rootCertificate = GetCertificate(rootCertificateArn, rootCAArn, client);
        ImportCertificateAuthorityCertificate(rootCertificate, rootCAArn, client);
    }

    private static AWSACMPCA ClientBuilder(String endpointRegion) {
        // Retrieve your credentials from the C:\Users\name\.aws\credentials file
        // in Windows or the .aws/credentials file in Linux.
        AWSCredentials credentials = null;
        try {
            credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException(
                    "Cannot load the credentials from the credential profiles file. " +
                    "Please make sure that your credentials file is at the correct " +
                    "location (C:\\Users\\joneps\\.aws\\credentials), and is in valid
format.",
                    e);
        }

        String endpointProtocol = "https://acm-pca." + endpointRegion +
".amazonaws.com/";
        EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
endpointRegion);

        // Create a client that you can use to make requests.
        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
            .withEndpointConfiguration(endpoint)
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();

        return client;
    }

    private static String CreateCertificateAuthority(CertificateAuthorityConfiguration
configCA, CrlConfiguration crlConfigure, CertificateAuthorityType CAtype, AWSACMPCA
client) {
        RevocationConfiguration revokeConfig = new RevocationConfiguration();
        revokeConfig.setCrlConfiguration(crlConfigure);

        // Create the request object.
        CreateCertificateAuthorityRequest createCARequest = new
CreateCertificateAuthorityRequest();
        createCARequest.withCertificateAuthorityConfiguration(configCA);
```

```
        createCARequest.withRevocationConfiguration(revokeConfig);
        createCARequest.withIdempotencyToken("123987");
        createCARequest.withCertificateAuthorityType(CAtype);

        // Create the private CA.
        CreateCertificateAuthorityResult createCAResult = null;
        try {
            createCAResult = client.createCertificateAuthority(createCARequest);
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (InvalidPolicyException ex) {
            throw ex;
        } catch (LimitExceededException ex) {
            throw ex;
        }

        // Retrieve the ARN of the private CA.
        String rootCAArn = createCAResult.getCertificateAuthorityArn();
        System.out.println("Root CA Arn: " + rootCAArn);

        return rootCAArn;
    }

    private static String GetCertificateAuthorityCsr(String rootCAArn, AWSACMPCA
client) {

        // Create the CSR request object and set the CA ARN.
        GetCertificateAuthorityCsrRequest csrRequest = new
GetCertificateAuthorityCsrRequest();
        csrRequest.withCertificateAuthorityArn(rootCAArn);

        // Create waiter to wait on successful creation of the CSR file.
        Waiter<GetCertificateAuthorityCsrRequest> getCSRWaiter =
client.waiters().certificateAuthorityCSRCreated();
        try {
            getCSRWaiter.run(new WaiterParameters<>(csrRequest));
        } catch (WaiterUnrecoverableException e) {
            //Explicit short circuit when the recourse transitions into
            //an undesired state.
        } catch (WaiterTimedOutException e) {
            //Failed to transition into desired state even after polling.
        } catch (AWSACMPCAException e) {
            //Unexpected service exception.
        }
```

```
        // Retrieve the CSR.
        GetCertificateAuthorityCsrResult csrResult = null;
        try {
            csrResult = client.getCertificateAuthorityCsr(csrRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }

        // Retrieve and display the CSR;
        String csr = csrResult.getCsr();
        System.out.println(csr);

        return csr;
    }

    private static String IssueCertificate(String rootCAArn, String csr, AWSACMPCA
 client) {

        // Create a certificate request:
        IssueCertificateRequest issueRequest = new IssueCertificateRequest();

        // Set the CA ARN.
        issueRequest.withCertificateAuthorityArn(rootCAArn);

        // Set the template ARN.
        issueRequest.withTemplateArn("arn:aws:acm-pca:::template/RootCACertificate/
V1");

        ByteBuffer csrByteBuffer = stringToByteBuffer(csr);
        issueRequest.setCsr(csrByteBuffer);

        // Set the signing algorithm.
        issueRequest.withSigningAlgorithm(SigningAlgorithm.SHA256WITHRSA);

        // Set the validity period for the certificate to be issued.
        Validity validity = new Validity();
        validity.withValue(3650L);
```

```
        validity.withType("DAYS");
        issueRequest.withValidity(validity);

        // Set the idempotency token.
        issueRequest.setIdempotencyToken("1234");

        // Issue the certificate.
        IssueCertificateResult issueResult = null;
        try {
            issueResult = client.issueCertificate(issueRequest);
        } catch (LimitExceededException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (MalformedCSRException ex) {
            throw ex;
        }

        // Retrieve and display the certificate ARN.
        String rootCertificateArn = issueResult.getCertificateArn();
        System.out.println("Root Certificate Arn: " + rootCertificateArn);

        return rootCertificateArn;
    }

    private static String GetCertificate(String rootCertificateArn, String rootCAArn,
AWSACMPCA client) {

        // Create a request object.
        GetCertificateRequest certificateRequest = new GetCertificateRequest();

        // Set the certificate ARN.
        certificateRequest.withCertificateArn(rootCertificateArn);

        // Set the certificate authority ARN.
        certificateRequest.withCertificateAuthorityArn(rootCAArn);

        // Create waiter to wait on successful creation of the certificate file.
```

```
        Waiter<GetCertificateRequest> getCertificateWaiter =
client.waiters().certificateIssued();
        try {
            getCertificateWaiter.run(new WaiterParameters<>(certificateRequest));
        } catch (WaiterUnrecoverableException e) {
            //Explicit short circuit when the recourse transitions into
            //an undesired state.
        } catch (WaiterTimedOutException e) {
            //Failed to transition into desired state even after polling.
        } catch (AWSACMPCAException e) {
            //Unexpected service exception.
        }

        // Retrieve the certificate and certificate chain.
        GetCertificateResult certificateResult = null;
        try {
            certificateResult = client.getCertificate(certificateRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        }

        // Get the certificate and certificate chain and display the result.
        String rootCertificate = certificateResult.getCertificate();
        System.out.println(rootCertificate);

        return rootCertificate;
    }

    private static void ImportCertificateAuthorityCertificate(String rootCertificate,
String rootCAArn, AWSACMPCA client) {

        // Create the request object and set the signed certificate, chain and CA ARN.
        ImportCertificateAuthorityCertificateRequest importRequest =
            new ImportCertificateAuthorityCertificateRequest();

        ByteBuffer certByteBuffer = stringToByteBuffer(rootCertificate);
```

```
        importRequest.setCertificate(certByteBuffer);

        importRequest.setCertificateChain(null);

        // Set the certificate authority ARN.
        importRequest.withCertificateAuthorityArn(rootCAArn);

        // Import the certificate.
        try {
            client.importCertificateAuthorityCertificate(importRequest);
        } catch (CertificateMismatchException ex) {
            throw ex;
        } catch (MalformedCertificateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (ConcurrentModificationException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }

        System.out.println("Root CA certificate successfully imported.");
        System.out.println("Root CA activated successfully.");
    }

    private static ByteBuffer stringToByteBuffer(final String string) {
        if (Objects.isNull(string)) {
            return null;
        }
        byte[] bytes = string.getBytes(StandardCharsets.UTF_8);
        return ByteBuffer.wrap(bytes);
    }
}
```

以程式設計方式建立和啟用次級 CA

此 Java 範例示範如何使用下列 AWS 私有 CA API 動作來啟用次級 CA：

- GetCertificateAuthorityCertificate
- CreateCertificateAuthority
- GetCertificateAuthorityCsr
- IssueCertificate
- GetCertificate
- ImportCertificateAuthorityCertificate

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;


import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.ASN1Subject;
import com.amazonaws.services.acmpca.model.CertificateAuthorityConfiguration;
import com.amazonaws.services.acmpca.model.CertificateAuthorityType;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityResult;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityRequest;
import com.amazonaws.services.acmpca.model.CrlConfiguration;
import com.amazonaws.services.acmpca.model.KeyAlgorithm;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Tag;

import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.Objects;

import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCertificateRequest;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCertificateResult;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrRequest;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrResult;
import com.amazonaws.services.acmpca.model.GetCertificateRequest;
import com.amazonaws.services.acmpca.model.GetCertificateResult;
```

```
import
 com.amazonaws.services.acmpca.model.ImportCertificateAuthorityCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateResult;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Validity;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.CertificateMismatchException;
import com.amazonaws.services.acmpca.model.ConcurrentModificationException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidPolicyException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.MalformedCertificateException;
import com.amazonaws.services.acmpca.model.MalformedCSRException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.RequestInProgressException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.RevocationConfiguration;
import com.amazonaws.services.acmpca.model.AWSACMPCAException;

import com.amazonaws.waiters.Waiter;
import com.amazonaws.waiters.WaiterParameters;
import com.amazonaws.waiters.WaiterTimedOutException;
import com.amazonaws.waiters.WaiterUnrecoverableException;

public class SubordinateCAActivation {

    public static void main(String[] args) throws Exception {
        // Place your own Root CA ARN here.
        String rootCAArn = "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566";

        // Define the endpoint region for your sample.
        String endpointRegion = "region"; // Substitute your region here, e.g. "us-
west-2"

        // Define a CA subject.
        ASN1Subject subject = new ASN1Subject();
        subject.setOrganization("Example Organization");
        subject.setOrganizationalUnit("Example");
        subject.setCountry("US");
```

```
        subject.setState("Virginia");
        subject.setLocality("Arlington");
        subject.setCommonName("www.example.com");

        // Define the CA configuration.
        CertificateAuthorityConfiguration configCA = new
CertificateAuthorityConfiguration();
        configCA.withKeyAlgorithm(KeyAlgorithm.RSA_2048);
        configCA.withSigningAlgorithm(SigningAlgorithm.SHA256WITHRSA);
        configCA.withSubject(subject);

        // Define a certificate revocation list configuration.
        CrlConfiguration crlConfigure = new CrlConfiguration();
        crlConfigure.withEnabled(true);
        crlConfigure.withExpirationInDays(365);
        crlConfigure.withCustomCname(null);
        crlConfigure.withS3BucketName("your-bucket-name");

        // Define a certificate authority type
        CertificateAuthorityType CAtype = CertificateAuthorityType.SUBORDINATE;

        // ** Execute core code samples for Subordinate CA activation in sequence **
        AWSACMPCA client = ClientBuilder(endpointRegion);
        String rootCertificate = GetCertificateAuthorityCertificate(rootCAArn, client);
        String subordinateCAArn = CreateCertificateAuthority(configCA, crlConfigure,
CAtype, client);
        String csr = GetCertificateAuthorityCsr(subordinateCAArn, client);
        String subordinateCertificateArn = IssueCertificate(rootCAArn, csr, client);
        String subordinateCertificate = GetCertificate(subordinateCertificateArn,
rootCAArn, client);
        ImportCertificateAuthorityCertificate(subordinateCertificate, rootCertificate,
subordinateCAArn, client);

    }

    private static AWSACMPCA ClientBuilder(String endpointRegion) {
        // Retrieve your credentials from the C:\Users\name\.aws\credentials file
        // in Windows or the .aws/credentials file in Linux.
        AWSCredentials credentials = null;
        try {
            credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException(
                    "Cannot load the credentials from the credential profiles file. " +
```

```
                    "Please make sure that your credentials file is at the correct " +
                    "location (C:\\Users\\joneps\\.aws\\credentials), and is in valid
format.",
                    e);
        }

        String endpointProtocol = "https://acm-pca." + endpointRegion +
".amazonaws.com/";
        EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
endpointRegion);

        // Create a client that you can use to make requests.
        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
            .withEndpointConfiguration(endpoint)
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();

        return client;
    }

    private static String GetCertificateAuthorityCertificate(String rootCAArn,
AWSACMPCA client) {
        // ** GetCertificateAuthorityCertificate **

        // Create a request object and set the certificate authority ARN,
        GetCertificateAuthorityCertificateRequest getCACertificateRequest =
        new GetCertificateAuthorityCertificateRequest();
        getCACertificateRequest.withCertificateAuthorityArn(rootCAArn);

        // Create a result object.
        GetCertificateAuthorityCertificateResult getCACertificateResult = null;
        try {
            getCACertificateResult =
client.getCertificateAuthorityCertificate(getCACertificateRequest);
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        }

        // Retrieve and display the certificate information.
```

```
        String rootCertificate = getCACertificateResult.getCertificate();
        System.out.println("Root CA Certificate / Certificate Chain:");
        System.out.println(rootCertificate);

        return rootCertificate;
    }

    private static String CreateCertificateAuthority(CertificateAuthorityConfiguration
configCA, CrlConfiguration crlConfigure, CertificateAuthorityType CAtype, AWSACMPCA
client) {
        RevocationConfiguration revokeConfig = new RevocationConfiguration();
        revokeConfig.setCrlConfiguration(crlConfigure);

        // Create the request object.
        CreateCertificateAuthorityRequest createCARequest = new
CreateCertificateAuthorityRequest();
        createCARequest.withCertificateAuthorityConfiguration(configCA);
        createCARequest.withRevocationConfiguration(revokeConfig);
        createCARequest.withIdempotencyToken("123987");
        createCARequest.withCertificateAuthorityType(CAtype);

        // Create the private CA.
        CreateCertificateAuthorityResult createCAResult = null;
        try {
            createCAResult = client.createCertificateAuthority(createCARequest);
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (InvalidPolicyException ex) {
            throw ex;
        } catch (LimitExceededException ex) {
            throw ex;
        }

        // Retrieve the ARN of the private CA.
        String subordinateCAArn = createCAResult.getCertificateAuthorityArn();
        System.out.println("Subordinate CA Arn: " + subordinateCAArn);

        return subordinateCAArn;
    }

    private static String GetCertificateAuthorityCsr(String subordinateCAArn, AWSACMPCA
client) {

        // Create the CSR request object and set the CA ARN.
```

```
        GetCertificateAuthorityCsrRequest csrRequest = new
GetCertificateAuthorityCsrRequest();
        csrRequest.withCertificateAuthorityArn(subordinateCAArn);

        // Create waiter to wait on successful creation of the CSR file.
        Waiter<GetCertificateAuthorityCsrRequest> getCSRWaiter =
client.waiters().certificateAuthorityCSRCreated();
        try {
            getCSRWaiter.run(new WaiterParameters<>(csrRequest));
        } catch (WaiterUnrecoverableException e) {
            //Explicit short circuit when the recourse transitions into
            //an undesired state.
        } catch (WaiterTimedOutException e) {
            //Failed to transition into desired state even after polling.
        } catch(AWSACMPCAException e) {
            //Unexpected service exception.
        }

        // Retrieve the CSR.
        GetCertificateAuthorityCsrResult csrResult = null;
        try {
            csrResult = client.getCertificateAuthorityCsr(csrRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }

        // Retrieve and display the CSR;
        String csr = csrResult.getCsr();
        System.out.println("Subordinate CSR:");
        System.out.println(csr);

        return csr;
    }

    private static String IssueCertificate(String rootCAArn, String csr, AWSACMPCA
client) {

        // Create a certificate request:
```

```
        IssueCertificateRequest issueRequest = new IssueCertificateRequest();

        // Set the issuing CA ARN.
        issueRequest.withCertificateAuthorityArn(rootCAArn);

        // Set the template ARN.
        issueRequest.withTemplateArn("arn:aws:acm-pca:::template/
SubordinateCACertificate_PathLen0/V1");

        ByteBuffer csrByteBuffer = stringToByteBuffer(csr);
        issueRequest.setCsr(csrByteBuffer);

        // Set the signing algorithm.
        issueRequest.withSigningAlgorithm(SigningAlgorithm.SHA256WITHRSA);

        // Set the validity period for the certificate to be issued.
        Validity validity = new Validity();
        validity.withValue(730L); // Approximately two years
        validity.withType("DAYS");
        issueRequest.withValidity(validity);

        // Set the idempotency token.
        issueRequest.setIdempotencyToken("1234");

        // Issue the certificate.
        IssueCertificateResult issueResult = null;
        try {
            issueResult = client.issueCertificate(issueRequest);
        } catch (LimitExceededException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (MalformedCSRException ex) {
            throw ex;
        }

        // Retrieve and display the certificate ARN.
        String subordinateCertificateArn = issueResult.getCertificateArn();
```

```
        System.out.println("Subordinate Certificate Arn: " +
subordinateCertificateArn);

        return subordinateCertificateArn;
    }

    private static String GetCertificate(String subordinateCertificateArn, String
rootCAArn, AWSACMPCA client) {

        // Create a request object.
        GetCertificateRequest certificateRequest = new GetCertificateRequest();

        // Set the certificate ARN.
        certificateRequest.withCertificateArn(subordinateCertificateArn);

        // Set the certificate authority ARN.
        certificateRequest.withCertificateAuthorityArn(rootCAArn);

        // Create waiter to wait on successful creation of the certificate file.
        Waiter<GetCertificateRequest> getCertificateWaiter =
client.waiters().certificateIssued();
        try {
            getCertificateWaiter.run(new WaiterParameters<>(certificateRequest));
        } catch (WaiterUnrecoverableException e) {
            //Explicit short circuit when the recourse transitions into
            //an undesired state.
        } catch (WaiterTimedOutException e) {
            //Failed to transition into desired state even after polling.
        } catch (AWSACMPCAException e) {
            //Unexpected service exception.
        }

        // Retrieve the certificate and certificate chain.
        GetCertificateResult certificateResult = null;
        try {
            certificateResult = client.getCertificate(certificateRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
```

```
        } catch (InvalidStateException ex) {
            throw ex;
        }

        // Get the certificate and certificate chain and display the result.
        String subordinateCertificate = certificateResult.getCertificate();
        System.out.println("Subordinate CA Certificate:");
        System.out.println(subordinateCertificate);

        return subordinateCertificate;
    }

    private static void ImportCertificateAuthorityCertificate(String
subordinateCertificate, String rootCertificate, String subordinateCAArn, AWSACMPCA
client) {

        // Create the request object and set the signed certificate, chain and CA ARN.
        ImportCertificateAuthorityCertificateRequest importRequest =
            new ImportCertificateAuthorityCertificateRequest();

        ByteBuffer certByteBuffer = stringToByteBuffer(subordinateCertificate);
        importRequest.setCertificate(certByteBuffer);

        ByteBuffer rootCACertByteBuffer = stringToByteBuffer(rootCertificate);
        importRequest.setCertificateChain(rootCACertByteBuffer);

        // Set the certificate authority ARN.
        importRequest.withCertificateAuthorityArn(subordinateCAArn);

        // Import the certificate.
        try {
            client.importCertificateAuthorityCertificate(importRequest);
        } catch (CertificateMismatchException ex) {
            throw ex;
        } catch (MalformedCertificateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (ConcurrentModificationException ex) {
            throw ex;
```

```
        } catch (RequestFailedException ex) {
            throw ex;
        }
        System.out.println("Subordinate CA certificate successfully imported.");
        System.out.println("Subordinate CA activated successfully.");
    }

    private static ByteBuffer stringToByteBuffer(final String string) {
        if (Objects.isNull(string)) {
            return null;
        }
        byte[] bytes = string.getBytes(StandardCharsets.UTF_8);
        return ByteBuffer.wrap(bytes);
    }
}
```

CreateCertificateAuthority

下列 Java 範例示範如何使用 CreateCerticateAuthority 操作。

此操作可建立私有次級憑證授權機構 (CA)。您必須指定 CA 組態、撤銷組態、CA 類型和選用的冪等符記。

CA 組態指定以下項目：

- 用來建立 CA 私有金鑰的演算法名稱和金鑰大小
- CA 用來簽署自己的憑證簽署請求、CRLs 和 OCSP 回應的簽署演算法類型
- X.500 主旨資訊

CRL 組態指定以下項目：

- CRL 過期期間天數 (CRL 有效期間)
- 將包含 CRL 的 Amazon S3 儲存貯體
- 包含於 CA 發行的憑證的 S3 儲存貯體 CNAME 別名

如果成功，此函數會傳回 CA 的 Amazon Resource Name (ARN)。

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
```

```
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;


import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.ASN1Subject;
import com.amazonaws.services.acmpca.model.CertificateAuthorityConfiguration;
import com.amazonaws.services.acmpca.model.CertificateAuthorityType;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityResult;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityRequest;
import com.amazonaws.services.acmpca.model.CrlConfiguration;
import com.amazonaws.services.acmpca.model.KeyAlgorithm;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Tag;

import java.util.ArrayList;
import java.util.Objects;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.InvalidPolicyException;
import com.amazonaws.services.acmpca.model.RevocationConfiguration;


public class CreateCertificateAuthority {

    public static void main(String[] args) throws Exception {

        // Retrieve your credentials from the C:\Users\name\.aws\credentials file
        // in Windows or the .aws/credentials file in Linux.
        AWSCredentials credentials = null;
        try {
            credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException(
                   "Cannot load the credentials from the credential profiles file. " +
                   "Please make sure that your credentials file is at the correct " +
                   "location (C:\\Users\\joneps\\.aws\\credentials), and is in valid
 format.",
```

```
                    e);
        }

        // Define the endpoint for your sample.
        String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
        String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
        EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
 endpointRegion);

        // Create a client that you can use to make requests.
        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
            .withEndpointConfiguration(endpoint)
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();

        // Define a CA subject.
        ASN1Subject subject = new ASN1Subject();
        subject.setOrganization("Example Organization");
        subject.setOrganizationalUnit("Example");
        subject.setCountry("US");
        subject.setState("Virginia");
        subject.setLocality("Arlington");
        subject.setCommonName("www.example.com");

        // Define the CA configuration.
        CertificateAuthorityConfiguration configCA = new
 CertificateAuthorityConfiguration();
        configCA.withKeyAlgorithm(KeyAlgorithm.RSA_2048);
        configCA.withSigningAlgorithm(SigningAlgorithm.SHA256WITHRSA);
        configCA.withSubject(subject);

        // Define a certificate revocation list configuration.
        CrlConfiguration crlConfigure = new CrlConfiguration();
        crlConfigure.withEnabled(true);
        crlConfigure.withExpirationInDays(365);
        crlConfigure.withCustomCname(null);
        crlConfigure.withS3BucketName("your-bucket-name");

        RevocationConfiguration revokeConfig = new RevocationConfiguration();
        revokeConfig.setCrlConfiguration(crlConfigure);
```

```
        // Define a certificate authority type: ROOT or SUBORDINATE
        CertificateAuthorityType CAtype = CertificateAuthorityType.<<SUBORDINATE>>;

        // Create a tag - method 1
        Tag tag1 = new Tag();
        tag1.withKey("PrivateCA");
        tag1.withValue("Sample");

        // Create a tag - method 2
        Tag tag2 = new Tag()
            .withKey("Purpose")
            .withValue("WebServices");

        // Add the tags to a collection.
        ArrayList<Tag> tags = new ArrayList<Tag>();
        tags.add(tag1);
        tags.add(tag2);

        // Create the request object.
        CreateCertificateAuthorityRequest req = new
CreateCertificateAuthorityRequest();
        req.withCertificateAuthorityConfiguration(configCA);
        req.withRevocationConfiguration(revokeConfig);
        req.withIdempotencyToken("123987");
        req.withCertificateAuthorityType(CAtype);
        req.withTags(tags);


        // Create the private CA.
        CreateCertificateAuthorityResult result = null;
        try {
            result = client.createCertificateAuthority(req);
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (InvalidPolicyException ex) {
            throw ex;
        } catch (LimitExceededException ex) {
            throw ex;
        }

        // Retrieve the ARN of the private CA.
        String arn = result.getCertificateAuthorityArn();
        System.out.println(arn);
    }
```

```
}
```

您的輸出應類似以下內容：

```
arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566
```

使用 CreateCertificateAuthority 支援 Active Directory

下列 Java 範例示範如何使用 CreateCerticateAuthority 操作來建立 CA，該 CA 可以安裝在 Microsoft Active Directory (AD) 的企業 NTAuth 存放區中。

操作會使用自訂物件識別符 (OIDs) 建立私有根憑證授權機構 (CA)。如需詳細資訊和對等操作 AWS CLI 的範例，請參閱為 Active Directory 登入建立 CA。

如果成功，此函數會傳回 CA 的 Amazon Resource Name (ARN)。

```
package com.amazonaws.samples.appstream;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.samples.GetCertificateAuthorityCertificate;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.ASN1Subject;
import com.amazonaws.services.acmpca.model.ApiPassthrough;
import com.amazonaws.services.acmpca.model.CertificateAuthorityConfiguration;
import com.amazonaws.services.acmpca.model.CertificateAuthorityType;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityResult;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityRequest;
import com.amazonaws.services.acmpca.model.CrlConfiguration;
import com.amazonaws.services.acmpca.model.CustomAttribute;
import com.amazonaws.services.acmpca.model.KeyAlgorithm;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Tag;

import java.io.ByteArrayInputStream;
import java.io.InputStreamReader;
```

```
import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Base64;
import java.util.List;
import java.util.Objects;

import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrRequest;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrResult;
import com.amazonaws.services.acmpca.model.GetCertificateRequest;
import com.amazonaws.services.acmpca.model.GetCertificateResult;
import
 com.amazonaws.services.acmpca.model.ImportCertificateAuthorityCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateResult;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Validity;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.CertificateMismatchException;
import com.amazonaws.services.acmpca.model.ConcurrentModificationException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidPolicyException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.MalformedCertificateException;
import com.amazonaws.services.acmpca.model.MalformedCSRException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.RequestInProgressException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.RevocationConfiguration;
import com.amazonaws.services.acmpca.model.AWSACMPCAException;

import com.amazonaws.waiters.Waiter;
import com.amazonaws.waiters.WaiterParameters;
import com.amazonaws.waiters.WaiterTimedOutException;
import com.amazonaws.waiters.WaiterUnrecoverableException;

import org.bouncycastle.asn1.x509.SubjectPublicKeyInfo;
import org.bouncycastle.cert.jcajce.JcaX509ExtensionUtils;
import org.bouncycastle.openssl.PEMParser;
import org.bouncycastle.pkcs.PKCS10CertificationRequest;
```

```
import org.bouncycastle.util.io.pem.PemReader;

import lombok.SneakyThrows;

public class RootCAActivation {
    public static void main(String[] args) throws Exception {
        // Define the endpoint region for your sample.
        String endpointRegion = "region";  // Substitute your region here, e.g. "ap-
southeast-2"

        // Define custom attributes
        List<CustomAttribute> customAttributes = Arrays.asList(
            new CustomAttribute()
                .withObjectIdentifier("2.5.4.3") // OID for Common Name
                .withValue("root CA"),
            new CustomAttribute()
                .withObjectIdentifier("0.9.2342.19200300.100.1.25") // OID for Domain
 Component
                .withValue("example"),
            new CustomAttribute()
                .withObjectIdentifier("0.9.2342.19200300.100.1.25") // OID for Domain
 Component
                .withValue("com")

        );

        // Define a CA subject.
        ASN1Subject subject = new ASN1Subject();
        subject.setCustomAttributes(customAttributes);

        // Define the CA configuration.
        CertificateAuthorityConfiguration configCA = new
 CertificateAuthorityConfiguration();
        configCA.withKeyAlgorithm(KeyAlgorithm.EC_prime256v1);
        configCA.withSigningAlgorithm(SigningAlgorithm.SHA256WITHECDSA);
        configCA.withSubject(subject);

        // Define a certificate authority type
        CertificateAuthorityType CAtype = CertificateAuthorityType.ROOT;

        // ** Execute core code samples for Root CA activation in sequence **
        AWSACMPCA client = ClientBuilder(endpointRegion);
        String rootCAArn = CreateCertificateAuthority(configCA, CAtype, client);
        String csr = GetCertificateAuthorityCsr(rootCAArn, client);
```

```
        String rootCertificateArn = IssueCertificate(rootCAArn, csr, client);
        String rootCertificate = GetCertificate(rootCertificateArn, rootCAArn, client);
        ImportCertificateAuthorityCertificate(rootCertificate, rootCAArn, client);
    }

    private static AWSACMPCA ClientBuilder(String endpointRegion) {
        // Retrieve your credentials from the C:\Users\name\.aws\credentials file
        // in Windows or the .aws/credentials file in Linux.
        AWSCredentials credentials = null;
        try {
            credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException(
                    "Cannot load the credentials from the credential profiles file. " +
                    "Please make sure that your credentials file is at the correct " +
                    "location (C:\\Users\\joneps\\.aws\\credentials), and is in valid
format.",
                    e);
        }

        String endpointProtocol = "https://acm-pca." + endpointRegion +
".amazonaws.com/";
        EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
endpointRegion);

        // Create a client that you can use to make requests.
        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
            .withEndpointConfiguration(endpoint)
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();

        return client;
    }

    private static String CreateCertificateAuthority(CertificateAuthorityConfiguration
configCA, CertificateAuthorityType CAtype, AWSACMPCA client) {
        // Create the request object.
        CreateCertificateAuthorityRequest createCARequest = new
CreateCertificateAuthorityRequest();
        createCARequest.withCertificateAuthorityConfiguration(configCA);
        createCARequest.withIdempotencyToken("123987");
        createCARequest.withCertificateAuthorityType(CAtype);
```

```
        // Create the private CA.
        CreateCertificateAuthorityResult createCAResult = null;
        try {
            createCAResult = client.createCertificateAuthority(createCARequest);
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (InvalidPolicyException ex) {
            throw ex;
        } catch (LimitExceededException ex) {
            throw ex;
        }

        // Retrieve the ARN of the private CA.
        String rootCAArn = createCAResult.getCertificateAuthorityArn();
        System.out.println("Root CA Arn: " + rootCAArn);

        return rootCAArn;
    }

    private static String GetCertificateAuthorityCsr(String rootCAArn, AWSACMPCA
client) {

        // Create the CSR request object and set the CA ARN.
        GetCertificateAuthorityCsrRequest csrRequest = new
GetCertificateAuthorityCsrRequest();
        csrRequest.withCertificateAuthorityArn(rootCAArn);

        // Create waiter to wait on successful creation of the CSR file.
        Waiter<GetCertificateAuthorityCsrRequest> getCSRWaiter =
client.waiters().certificateAuthorityCSRCreated();
        try {
            getCSRWaiter.run(new WaiterParameters<>(csrRequest));
        } catch (WaiterUnrecoverableException e) {
            //Explicit short circuit when the recourse transitions into
            //an undesired state.
        } catch (WaiterTimedOutException e) {
            //Failed to transition into desired state even after polling.
        } catch (AWSACMPCAException e) {
            //Unexpected service exception.
        }

        // Retrieve the CSR.
        GetCertificateAuthorityCsrResult csrResult = null;
        try {
```

```
            csrResult = client.getCertificateAuthorityCsr(csrRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }

        // Retrieve and display the CSR;
        String csr = csrResult.getCsr();
        System.out.println(csr);

        return csr;
    }

    private static String IssueCertificate(String rootCAArn, String csr, AWSACMPCA
 client) {

        // Create a certificate request:
        IssueCertificateRequest issueRequest = new IssueCertificateRequest();

        // Set the CA ARN.
        issueRequest.withCertificateAuthorityArn(rootCAArn);

        // Set the template ARN.
        issueRequest.withTemplateArn("arn:aws:acm-pca:::template/RootCACertificate/
V1");

        ByteBuffer csrByteBuffer = stringToByteBuffer(csr);
        issueRequest.setCsr(csrByteBuffer);

        // Set the signing algorithm.
        issueRequest.withSigningAlgorithm(SigningAlgorithm.SHA256WITHECDSA);

        // Set the validity period for the certificate to be issued.
        Validity validity = new Validity();
        validity.withValue(3650L);
        validity.withType("DAYS");
        issueRequest.withValidity(validity);

        // Set the idempotency token.
```

```
        issueRequest.setIdempotencyToken("1234");

        // Issue the certificate.
        IssueCertificateResult issueResult = null;
        try {
            issueResult = client.issueCertificate(issueRequest);
        } catch (LimitExceededException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (MalformedCSRException ex) {
            throw ex;
        }

        // Retrieve and display the certificate ARN.
        String rootCertificateArn = issueResult.getCertificateArn();
        System.out.println("Root Certificate Arn: " + rootCertificateArn);

        return rootCertificateArn;
    }

    private static String GetCertificate(String rootCertificateArn, String rootCAArn,
AWSACMPCA client) {

        // Create a request object.
        GetCertificateRequest certificateRequest = new GetCertificateRequest();

        // Set the certificate ARN.
        certificateRequest.withCertificateArn(rootCertificateArn);

        // Set the certificate authority ARN.
        certificateRequest.withCertificateAuthorityArn(rootCAArn);

        // Create waiter to wait on successful creation of the certificate file.
        Waiter<GetCertificateRequest> getCertificateWaiter =
client.waiters().certificateIssued();
        try {
            getCertificateWaiter.run(new WaiterParameters<>(certificateRequest));
```

```
        } catch (WaiterUnrecoverableException e) {
            //Explicit short circuit when the recourse transitions into
            //an undesired state.
        } catch (WaiterTimedOutException e) {
            //Failed to transition into desired state even after polling.
        } catch (AWSACMPCAException e) {
            //Unexpected service exception.
        }

        // Retrieve the certificate and certificate chain.
        GetCertificateResult certificateResult = null;
        try {
            certificateResult = client.getCertificate(certificateRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        }

        // Get the certificate and certificate chain and display the result.
        String rootCertificate = certificateResult.getCertificate();
        System.out.println(rootCertificate);

        return rootCertificate;
    }

    private static void ImportCertificateAuthorityCertificate(String rootCertificate,
String rootCAArn, AWSACMPCA client) {

        // Create the request object and set the signed certificate, chain and CA ARN.
        ImportCertificateAuthorityCertificateRequest importRequest =
            new ImportCertificateAuthorityCertificateRequest();

        ByteBuffer certByteBuffer = stringToByteBuffer(rootCertificate);
        importRequest.setCertificate(certByteBuffer);

        importRequest.setCertificateChain(null);
```

```
        // Set the certificate authority ARN.
        importRequest.withCertificateAuthorityArn(rootCAArn);

        // Import the certificate.
        try {
            client.importCertificateAuthorityCertificate(importRequest);
        } catch (CertificateMismatchException ex) {
            throw ex;
        } catch (MalformedCertificateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (ConcurrentModificationException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }

        System.out.println("Root CA certificate successfully imported.");
        System.out.println("Root CA activated successfully.");
    }

    private static ByteBuffer stringToByteBuffer(final String string) {
        if (Objects.isNull(string)) {
            return null;
        }
        byte[] bytes = string.getBytes(StandardCharsets.UTF_8);
        return ByteBuffer.wrap(bytes);
    }
}
```

您的輸出應類似以下內容：

```
arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566
```

CreateCertificateAuthorityAuditReport

以下 Java 範例示範如何使用 CreateCertificateAuthorityAuditReport 操作。

此操作可在每次發行或撤銷憑證時建立稽核報告。報告會儲存在您在輸入時指定的 Amazon S3 儲存貯體中。您可以每 30 分鐘產生一份新的報告。

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import
 com.amazonaws.services.acmpca.model.CreateCertificateAuthorityAuditReportRequest;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityAuditReportResult;

import com.amazonaws.services.acmpca.model.RequestInProgressException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidStateException;

public class CreateCertificateAuthorityAuditReport {

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
         credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
         throw new AmazonClientException("Cannot load your credentials from file.", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
```

```
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
 endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
         .withEndpointConfiguration(endpoint)
         .withCredentials(new AWSStaticCredentialsProvider(credentials))
         .build();

      // Create a request object and set the certificate authority ARN.
      CreateCertificateAuthorityAuditReportRequest req =
          new CreateCertificateAuthorityAuditReportRequest();

      // Set the certificate authority ARN.
      req.setCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");

      // Specify the S3 bucket name for your report.
      req.setS3BucketName("your-bucket-name");

      // Specify the audit response format.
      req.setAuditReportResponseFormat("JSON");

      // Create a result object.
      CreateCertificateAuthorityAuditReportResult result = null;
      try {
         result = client.createCertificateAuthorityAuditReport(req);
      } catch (RequestInProgressException ex) {
         throw ex;
      } catch (RequestFailedException ex) {
         throw ex;
      } catch (ResourceNotFoundException ex) {
         throw ex;
      } catch (InvalidArnException ex) {
         throw ex;
      } catch (InvalidArgsException ex) {
         throw ex;
      } catch (InvalidStateException ex) {
         throw ex;
      }
```

```
        String ID = result.getAuditReportId();
        String S3Key = result.getS3Key();

        System.out.println(ID);
        System.out.println(S3Key);

    }
}
```

您的輸出應類似以下內容：

```
58904752-7de3-4bdf-ba89-6953e48c3cc7
audit-report/16075838-061c-4f7a-b54b-49bbc111bcff/58904752-7de3-4bdf-
ba89-6953e48c3cc7.json
```

CreatePermission

下列 Java 範例示範如何使用 CreatePermission 操作。

操作會將私有 CA 的存取許可指派給指定的 AWS 服務主體。您可以將建立及擷取私有 CA 憑證的許可授予服務，及列出由私有 CA 授予的作用中許可。若要透過 ACM 自動續約憑證，您必須將所有可能的許可 (IssueCertificate、 GetCertificate和 ListPermissions) 從 CA 指派給 ACM 服務主體 ()acm.amazonaws.com。您可以呼叫 ListCertificateAuthorities 函數來尋找 CA 的 ARN。

建立許可後，您就可以使用 ListPermissions 函數來進行檢查，或使用 DeletePermission 函數將其刪除。

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.CreatePermissionRequest;
import com.amazonaws.services.acmpca.model.CreatePermissionResult;
```

```
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.PermissionAlreadyExistsException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;

import java.util.ArrayList;

public class CreatePermission {

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
         credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
         throw new AmazonClientException("Cannot load your credentials from file.", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
 endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
         .withEndpointConfiguration(endpoint)
         .withCredentials(new AWSStaticCredentialsProvider(credentials))
         .build();

      // Create a request object.
      CreatePermissionRequest req =
          new CreatePermissionRequest();

      //  Set the certificate authority ARN.
      req.setCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");
```

```
        // Set the permissions to give the user.
        ArrayList<String> permissions = new ArrayList<>();
        permissions.add("IssueCertificate");
        permissions.add("GetCertificate");
        permissions.add("ListPermissions");

        req.setActions(permissions);

        // Set the Principal.
        req.setPrincipal("acm.amazonaws.com");

        // Create a result object.
        CreatePermissionResult result = null;
        try {
           result = client.createPermission(req);
        } catch (InvalidArnException ex) {
           throw ex;
        } catch (InvalidStateException ex) {
           throw ex;
        } catch (LimitExceededException ex) {
           throw ex;
        } catch (PermissionAlreadyExistsException ex) {
           throw ex;
        } catch (RequestFailedException ex) {
           throw ex;
        } catch (ResourceNotFoundException ex) {
           throw ex;
        }
    }
}
```

DeleteCertificateAuthority

以下 Java 範例說明如何使用 DeleteCertificateAuthority 操作。

此操作可刪除您使用 CreateCertificateAuthority 操作建立的私有憑證授權機構 (CA)。DeleteCertificateAuthority 操作需要您提供要刪除 CA 的 ARN。您可以呼叫 ListCertificateAuthorities 操作來尋找 ARN。如果私有 CA 的狀態為 CREATING 或 PENDING_CERTIFICATE，您可以立即刪除。不過，如果您已經匯入憑證，則無法立即將其刪除。您必須先呼叫 UpdateCertificateAuthority 操作並將 Status 參數設定為 DISABLED，來停用 CA。您接著可以在 DeleteCertificateAuthority 操作中使用 PermanentDeletionTimeInDays 參數

來指定天數 (7 到 30)。在此期間，私有 CA 皆可以還原至 disabled 狀態。依預設，如果您未設定 PermanentDeletionTimeInDays 參數，還原期間為 30 天。過了此期間，私有 CA 會永久刪除，無法還原。如需詳細資訊，請參閱還原 CA。

如需說明如何使用 RestoreCertificateAuthority 操作的 Java 範例，請參閱RestoreCertificateAuthority。

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.DeleteCertificateAuthorityRequest;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.RequestFailedException;

public class DeleteCertificateAuthority {

   public static void main(String[] args) throws Exception{

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
         credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
         throw new AmazonClientException("Cannot load your credentials from disk", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
```

```
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
 endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
         .withEndpointConfiguration(endpoint)
         .withCredentials(new AWSStaticCredentialsProvider(credentials))
         .build();

      // Create a requrest object and set the ARN of the private CA to delete.
      DeleteCertificateAuthorityRequest req = new DeleteCertificateAuthorityRequest();

      // Set the certificate authority ARN.
      req.withCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");

      // Set the recovery period.
      req.withPermanentDeletionTimeInDays(12);

      // Delete the CA.
      try {
         client.deleteCertificateAuthority(req);
      } catch (ResourceNotFoundException ex) {
         throw ex;
      } catch (InvalidArnException ex) {
         throw ex;
      } catch (InvalidStateException ex) {
         throw ex;
      } catch (RequestFailedException ex) {
         throw ex;
      }
   }
}
```

DeletePermission

下列 Java 範例示範如何使用 DeletePermission 操作。

操作會刪除私有 CA 使用 CreatePermissions 操作委派給 AWS 服務主體的許可。您可以呼叫 ListCertificateAuthorities 函數來尋找 CA 的 ARN。您可以呼叫 ListPermissions 來檢查 CA 授予的許可。

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.DeletePermissionRequest;
import com.amazonaws.services.acmpca.model.DeletePermissionResult;

import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;

public class DeletePermission {

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
         credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
         throw new AmazonClientException("Cannot load your credentials from file.", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
```

```
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
 endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
         .withEndpointConfiguration(endpoint)
         .withCredentials(new AWSStaticCredentialsProvider(credentials))
         .build();

      // Create a request object.
      DeletePermissionRequest req =
          new DeletePermissionRequest();

      //  Set the certificate authority ARN.
      req.setCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");

      // Set the AWS service principal.
      req.setPrincipal("acm.amazonaws.com");

      // Create a result object.
      DeletePermissionResult result = null;
      try {
         result = client.deletePermission(req);
      } catch (InvalidArnException ex) {
         throw ex;
      } catch (InvalidStateException ex) {
         throw ex;
      } catch (RequestFailedException ex) {
         throw ex;
      } catch (ResourceNotFoundException ex) {
         throw ex;
      }
   }
}
```

DeletePolicy

下列 Java 範例示範如何使用 DeletePolicy 操作。

操作會刪除連接到私有 CA 的資源型政策。資源型政策用於啟用跨帳戶 CA 共享。您可以呼叫 ListCertificateAuthorities 動作來尋找私有 CA 的 ARN。

相關 API 動作包括 PutPolicy 和 GetPolicy。

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.DeletePolicyRequest;
import com.amazonaws.services.acmpca.model.DeletePolicyResult;
import com.amazonaws.services.acmpca.model.AWSACMPCAException;
import com.amazonaws.services.acmpca.model.ConcurrentModificationException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.LockoutPreventedException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;

public class DeletePolicy {

    public static void main(String[] args) throws Exception {

        // Retrieve your credentials from the C:\Users\name\.aws\credentials file
        // in Windows or the .aws/credentials file in Linux.
        AWSCredentials credentials = null;
        try {
            credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException("Cannot load your credentials from file.",
 e);
        }

        // Define the endpoint for your sample.
        String endpointRegion = "us-west-2";  // Substitute your Region here, e.g. "us-
west-2"
        String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
        EndpointConfiguration endpoint =
```

```
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
 endpointRegion);

        // Create a client that you can use to make requests.
        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
            .withEndpointConfiguration(endpoint)
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();

        // Create the request object.
        DeletePolicyRequest req = new DeletePolicyRequest();

        // Set the resource ARN.
        req.withResourceArn("arn:aws:acm-pca:us-west-2:111122223333:certificate-
authority/11223344-44ee-aa22-bb33-4cd2d13f1f18");

        // Retrieve a list of your CAs.
        DeletePolicyResult result = null;
        try {
            result = client.deletePolicy(req);
        } catch (ConcurrentModificationException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (LockoutPreventedException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (AWSACMPCAException ex) {
            throw ex;
        }
    }
}
```

DescribeCertificateAuthority

以下 Java 範例說明如何使用 DescribeCertificateAuthority 操作。

此操作可列出私有憑證授權機構 (CA) 的相關資訊。您必須指定私有 CA 的 ARN (Amazon Resource Name)。輸出包含 CA 的狀態。此可為下列任何一項：

- CREATING – AWS 私有 CA 正在建立您的私有憑證授權機構。
- PENDING_CERTIFICATE – 憑證正在等待。您必須使用現場部署的根 CA 或次級 CA 來簽署私有 CA CSR，然後將其匯入 PCA。
- ACTIVE – 您的私有 CA 已啟用。
- DISABLED – 您的私有 CA 已停用。
- EXPIRED – 您的私有 CA 憑證已過期。
- FAILED – 無法建立您的私有 CA。
- DELETED – 您的私有 CA 位於還原期間內，之後會永久刪除。

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.CertificateAuthority;
import com.amazonaws.services.acmpca.model.DescribeCertificateAuthorityRequest;
import com.amazonaws.services.acmpca.model.DescribeCertificateAuthorityResult;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidArnException;

public class DescribeCertificateAuthority {

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
```

```
         credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
         throw new AmazonClientException("Cannot load your credentials from disk", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
 endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
         .withEndpointConfiguration(endpoint)
         .withCredentials(new AWSStaticCredentialsProvider(credentials))
         .build();

      // Create a request object
      DescribeCertificateAuthorityRequest req = new
 DescribeCertificateAuthorityRequest();

      // Set the certificate authority ARN.
      req.withCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");

      // Create a result object.
      DescribeCertificateAuthorityResult result = null;
      try {
         result = client.describeCertificateAuthority(req);
      } catch (ResourceNotFoundException ex) {
         throw ex;
      } catch (InvalidArnException ex) {
         throw ex;
      }

      // Retrieve and display information about the CA.
      CertificateAuthority PCA = result.getCertificateAuthority();
      String strPCA = PCA.toString();
      System.out.println(strPCA);
   }
```

```
}
```

DescribeCertificateAuthorityAuditReport

下列 Java 範例示範如何使用 DescribeCertificateAuthorityAuditReport 操作。

此操作會列出您透過呼叫 CreateCertificateAuthorityAuditReport 操作建立的特定稽核報告相關資訊。稽核資訊會在每次使用憑證授權單位 (CA) 私密金鑰時建立。私密金鑰會在您發行憑證、的私有金鑰、簽署 CRL 或撤銷憑證時使用。

```
package com.amazonaws.samples;

import java.util.Date;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import
 com.amazonaws.services.acmpca.model.DescribeCertificateAuthorityAuditReportRequest;
import
 com.amazonaws.services.acmpca.model.DescribeCertificateAuthorityAuditReportResult;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.AWSACMPCAException;

import com.amazonaws.waiters.Waiter;
import com.amazonaws.waiters.WaiterParameters;
import com.amazonaws.waiters.WaiterTimedOutException;
import com.amazonaws.waiters.WaiterUnrecoverableException;

public class DescribeCertificateAuthorityAuditReport {

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
```

```
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
         credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
         throw new AmazonClientException("Cannot load your credentials from file.", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
 endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
         .withEndpointConfiguration(endpoint)
         .withCredentials(new AWSStaticCredentialsProvider(credentials))
         .build();

      // Create a request object.
      DescribeCertificateAuthorityAuditReportRequest req =
            new DescribeCertificateAuthorityAuditReportRequest();

      // Set the certificate authority ARN.
      req.withCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");

      // Set the audit report ID.
      req.withAuditReportId("11111111-2222-3333-4444-555555555555");

      // Create waiter to wait on successful creation of the audit report file.
      Waiter<DescribeCertificateAuthorityAuditReportRequest> waiter =
 client.waiters().auditReportCreated();
      try {
         waiter.run(new WaiterParameters<>(req));
      } catch (WaiterUnrecoverableException e) {
          //Explicit short circuit when the recourse transitions into
          //an undesired state.
      } catch (WaiterTimedOutException e) {
          //Failed to transition into desired state even after polling.
```

```
        } catch (AWSACMPCAException e) {
            //Unexpected service exception.
        }

        // Create a result object.
        DescribeCertificateAuthorityAuditReportResult result = null;
        try {
           result = client.describeCertificateAuthorityAuditReport(req);
        } catch (ResourceNotFoundException ex) {
           throw ex;
        } catch (InvalidArgsException ex) {
           throw ex;
        }

        String status = result.getAuditReportStatus();
        String S3Bucket = result.getS3BucketName();
        String S3Key = result.getS3Key();
        Date createdAt = result.getCreatedAt();

        System.out.println(status);
        System.out.println(S3Bucket);
        System.out.println(S3Key);
        System.out.println(createdAt);
    }
}
```

您的輸出應類似以下內容：

```
SUCCESS
your-audit-report-bucket-name
audit-report/a4119411-8153-498a-a607-2cb77b858043/25211c3d-f2fe-479f-b437-
fe2b3612bc45.json
Tue Jan 16 13:07:58 PST 2018
```

GetCertificate

下列 Java 範例示範如何使用 GetCertificate 操作。

此操作可從您的私有 CA 擷取憑證。當您呼叫 IssueCertificate 操作時，憑證的 ARN 便會傳回。呼叫 GetCertificate 操作時，您必須指定私有 CA 的 ARN，和發行憑證的 ARN。如果憑證的狀態為 ISSUED，便可擷取憑證。您可以呼叫 CreateCertificateAuthorityAuditReport 操作，建立包含您私有 CA 發行及撤銷之所有憑證相關資訊的報告。

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.GetCertificateRequest;
import com.amazonaws.services.acmpca.model.GetCertificateResult;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.RequestInProgressException;
import com.amazonaws.services.acmpca.model.RequestFailedException ;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidStateException;

import com.amazonaws.waiters.Waiter;
import com.amazonaws.waiters.WaiterParameters;
import com.amazonaws.waiters.WaiterTimedOutException;
import com.amazonaws.waiters.WaiterUnrecoverableException;

import com.amazonaws.services.acmpca.model.AWSACMPCAException;

public class GetCertificate {

   public static void main(String[] args) throws Exception{

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
         credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
         throw new AmazonClientException("Cannot load your credentials from disk", e);
      }

      // Define the endpoint for your sample.
```

```
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
 endpointRegion);

      // Create a client.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
         .withEndpointConfiguration(endpoint)
         .withCredentials(new AWSStaticCredentialsProvider(credentials))
         .build();

      // Create a request object.
      GetCertificateRequest req = new GetCertificateRequest();

      // Set the certificate ARN.
      req.withCertificateArn("arn:aws:acm-pca:region:account:certificate-
authority/CA_ID/certificate/certificate_ID");

      // Set the certificate authority ARN.
      req.withCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");

      // Create waiter to wait on successful creation of the certificate file.
      Waiter<GetCertificateRequest> waiter = client.waiters().certificateIssued();
      try {
         waiter.run(new WaiterParameters<>(req));
      } catch (WaiterUnrecoverableException e) {
          //Explicit short circuit when the recourse transitions into
          //an undesired state.
      } catch (WaiterTimedOutException e) {
          //Failed to transition into desired state even after polling.
      } catch (AWSACMPCAException e) {
          //Unexpected service exception.
      }

      // Retrieve the certificate and certificate chain.
      GetCertificateResult result = null;
      try {
         result = client.getCertificate(req);
      } catch (RequestInProgressException ex) {
         throw ex;
```

```
      } catch (RequestFailedException ex) {
         throw ex;
      } catch (ResourceNotFoundException ex) {
         throw ex;
      } catch (InvalidArnException ex) {
         throw ex;
      } catch (InvalidStateException ex) {
         throw ex;
      }

      // Get the certificate and certificate chain and display the result.
      String strCert = result.getCertificate();
      System.out.println(strCert);
   }
}
```

您的輸出應為類似以下的憑證授權單位 (CA) 憑證鏈和您指定的憑證。

```
-----BEGIN CERTIFICATE----- base64-encoded certificate -----END CERTIFICATE-----

-----BEGIN CERTIFICATE----- base64-encoded certificate -----END CERTIFICATE-----

-----BEGIN CERTIFICATE----- base64-encoded certificate -----END CERTIFICATE-----
```

GetCertificateAuthorityCertificate

以下 Java 範例會示範如何使用 GetCertificateAuthorityCertificate 操作。

此操作能為您的私有憑證授權機構 (CA) 擷取憑證和憑證鏈。憑證和鏈結都是 PEM 格式的 base64 編碼字串。憑證鏈不包含 CA 憑證。憑證鏈中的每個憑證都會依序簽署。

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCertificateRequest;
```

```
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCertificateResult;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.InvalidArnException;

public class GetCertificateAuthorityCertificate {

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
         credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
         throw new AmazonClientException("Cannot load your credentials from disk", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
 endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
         .withEndpointConfiguration(endpoint)
         .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();

      // Create a request object
      GetCertificateAuthorityCertificateRequest req =
            new GetCertificateAuthorityCertificateRequest();

      // Set the certificate authority ARN,
      req.withCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");

      // Create a result object.
```

```
      GetCertificateAuthorityCertificateResult result = null;
      try {
         result = client.getCertificateAuthorityCertificate(req);
      } catch (ResourceNotFoundException ex) {
         throw ex;
      } catch (InvalidStateException ex) {
         throw ex;
      } catch (InvalidArnException ex) {
         throw ex;
      }

      // Retrieve and display the certificate information.
      String strPcaCert = result.getCertificate();
      System.out.println(strPcaCert);
      String strPCACChain = result.getCertificateChain();
      System.out.println(strPCACChain);
   }
}
```

您的輸出應為類似以下您指定的憑證授權單位 (CA) 的憑證和憑證鏈。

```
-----BEGIN CERTIFICATE----- base64-encoded certificate -----END CERTIFICATE-----

-----BEGIN CERTIFICATE----- base64-encoded certificate -----END CERTIFICATE-----
```

GetCertificateAuthorityCsr

下列 Java 範例說明如何使用 GetCertificateAuthorityCsr 操作。

此操作能為您的私有憑證授權機構 (CA) 擷取憑證簽署要求 (CSR)。CSR 會在您呼叫 CreateCertificateAuthority 操作時建立。將 CSR 帶到您的內部部署 X.509 基礎設施，並使用您的根 CA 或次級 CA 簽署。然後呼叫 ImportCertificateAuthorityCertificate 操作，將已簽署的憑證匯入回 ACM PCA。CSR 會以 PEM 格式傳回為 base64 編碼字串。

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
```

```
import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrRequest;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrResult;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.RequestInProgressException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.AWSACMPCAException;

import com.amazonaws.waiters.Waiter;
import com.amazonaws.waiters.WaiterParameters;
import com.amazonaws.waiters.WaiterTimedOutException;
import com.amazonaws.waiters.WaiterUnrecoverableException;

public class GetCertificateAuthorityCsr {

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
         credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
         throw new AmazonClientException("Cannot load your credentials from disk", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
 endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
         .withEndpointConfiguration(endpoint)
         .withCredentials(new AWSStaticCredentialsProvider(credentials))
```

```
         .build();

      // Create the request object and set the CA ARN.
      GetCertificateAuthorityCsrRequest req = new GetCertificateAuthorityCsrRequest();
      req.withCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");

      // Create waiter to wait on successful creation of the CSR file.
      Waiter<GetCertificateAuthorityCsrRequest> waiter =
 client.waiters().certificateAuthorityCSRCreated();
      try {
         waiter.run(new WaiterParameters<>(req));
      } catch (WaiterUnrecoverableException e) {
          //Explicit short circuit when the recourse transitions into
          //an undesired state.
      } catch (WaiterTimedOutException e) {
          //Failed to transition into desired state even after polling.
      } catch (AWSACMPCAException e) {
          //Unexpected service exception.
      }

      // Retrieve the CSR.
      GetCertificateAuthorityCsrResult result = null;
      try {
         result = client.getCertificateAuthorityCsr(req);
      } catch (RequestInProgressException ex) {
         throw ex;
      } catch (ResourceNotFoundException ex) {
         throw ex;
      } catch (InvalidArnException ex) {
         throw ex;
      } catch (RequestFailedException ex) {
         throw ex;
      }

      // Retrieve and display the CSR;
      String Csr = result.getCsr();
      System.out.println(Csr);
   }
}
```

您指定的憑證授權單位 (CA) 的輸出應類似下列。憑證簽署要求 (CSR) 是以 base64 編碼 (PEM 格式)。儲存到本機檔案，並帶到您的內部部署 X.509 基礎設施，然後使用您的根 CA 或次級 CA 簽署。

```
-----BEGIN CERTIFICATE REQUEST----- base64-encoded request -----END CERTIFICATE
 REQUEST-----
```

GetPolicy

下列 Java 範例示範如何使用 GetPolicy 操作。

操作會擷取連接至私有 CA 的資源型政策。資源型政策用於啟用跨帳戶 CA 共享。您可以呼叫 ListCertificateAuthorities 動作來尋找私有 CA 的 ARN。

相關 API 動作包括 PutPolicy 和 DeletePolicy。

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.GetPolicyRequest;
import com.amazonaws.services.acmpca.model.GetPolicyResult;
import com.amazonaws.services.acmpca.model.AWSACMPCAException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;

public class GetPolicy {

    public static void main(String[] args) throws Exception {

        // Retrieve your credentials from the C:\Users\name\.aws\credentials file
        // in Windows or the .aws/credentials file in Linux.
        AWSCredentials credentials = null;
        try {
            credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
```

```
            throw new AmazonClientException("Cannot load your credentials from file.",
 e);
        }

        // Define the endpoint for your sample.
        String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
        String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
        EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
 endpointRegion);

        // Create a client that you can use to make requests.
        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
            .withEndpointConfiguration(endpoint)
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();

        // Create the request object.
        GetPolicyRequest req = new GetPolicyRequest();

        // Set the resource ARN.
        req.withResourceArn("arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566");

        // Retrieve a list of your CAs.
        GetPolicyResult result= null;
        try {
            result = client.getPolicy(req);
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (AWSACMPCAException ex) {
            throw ex;
        }

        // Display the policy.
        System.out.println(result.getPolicy());
```

```
    }
}
```

ImportCertificateAuthorityCertificate

下列 Java 範例示範如何使用 ImportCertificateAuthorityCertificate 操作。

此操作會將您簽署的私有 CA 憑證匯入 AWS 私有 CA。若要呼叫此操作，您必須先呼叫 CreateCertificateAuthority 操作以建立私有憑證授權機構。然後，您必須呼叫 GetCertificateAuthorityCsr 操作來產生憑證簽署要求 (CSR)。將 CSR 帶到現場部署的 CA，並使用您的根憑證或次級憑證簽署。建立憑證鏈，並將簽署的憑證和憑證鏈複製到您的工作目錄。

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import
 com.amazonaws.services.acmpca.model.ImportCertificateAuthorityCertificateRequest;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.RequestInProgressException;
import com.amazonaws.services.acmpca.model.MalformedCertificateException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.ConcurrentModificationException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.CertificateMismatchException;
import com.amazonaws.services.acmpca.model.RequestFailedException;

import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.Objects;

public class ImportCertificateAuthorityCertificate {

   public static ByteBuffer stringToByteBuffer(final String string) {
      if (Objects.isNull(string)) {
```

```
            return null;
        }
        byte[] bytes = string.getBytes(StandardCharsets.UTF_8);
        return ByteBuffer.wrap(bytes);
        }

    public static void main(String[] args) throws Exception {

        // Retrieve your credentials from the C:\Users\name\.aws\credentials file
        // in Windows or the .aws/credentials file in Linux.
        AWSCredentials credentials = null;
        try {
            credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException("Cannot load your credentials from disk", e);
        }

        // Define the endpoint for your sample.
        String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
        String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
        EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol, endpointRegion);

        // Create a client that you can use to make requests.
        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
           .withEndpointConfiguration(endpoint)
           .withCredentials(new AWSStaticCredentialsProvider(credentials))
           .build();

        // Create the request object and set the signed certificate, chain and CA ARN.
        ImportCertificateAuthorityCertificateRequest req =
              new ImportCertificateAuthorityCertificateRequest();

        // Set the signed certificate.
        String strCertificate =
              "-----BEGIN CERTIFICATE-----\n" +
              "base64-encoded certificate\n" +
              "-----END CERTIFICATE-----\n";
        ByteBuffer certByteBuffer = stringToByteBuffer(strCertificate);
        req.setCertificate(certByteBuffer);

        // Set the certificate chain.
```

```
      String strCertificateChain =
            "-----BEGIN CERTIFICATE-----\n" +
            "base64-encoded certificate\n" +
            "-----END CERTIFICATE-----\n";
      ByteBuffer chainByteBuffer = stringToByteBuffer(strCertificateChain);
      req.setCertificateChain(chainByteBuffer);

      // Set the certificate authority ARN.
      req.withCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");

      // Import the certificate.
      try {
         client.importCertificateAuthorityCertificate(req);
      } catch (CertificateMismatchException ex) {
         throw ex;
      } catch (MalformedCertificateException ex) {
         throw ex;
      } catch (InvalidArnException ex) {
         throw ex;
      } catch (ResourceNotFoundException ex) {
         throw ex;
      } catch (RequestInProgressException ex) {
         throw ex;
      } catch (ConcurrentModificationException ex) {
         throw ex;
      } catch (RequestFailedException ex) {
         throw ex;
      }
   }
}
```

IssueCertificate

下列 Java 範例會示範如何使用 IssueCertificate 操作。

此操作使用您的私有憑證授權機構 (CA) 來發行最終實體憑證。此操作會傳回憑證的 Amazon Resource Name (ARN)。您可以透過呼叫 GetCertificate 及指定 ARN 來擷取憑證。

> **Note**
>
> IssueCertificate 操作需要您指定憑證範本。此範例使用 EndEntityCertificate/V1 範本。如需所有可用範本的相關資訊，請參閱 使用 AWS 私有 CA 憑證範本。

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.Objects;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.IssueCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateResult;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Validity;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.MalformedCSRException;

public class IssueCertificate {
   public static ByteBuffer stringToByteBuffer(final String string) {
      if (Objects.isNull(string)) {
          return null;
      }
      byte[] bytes = string.getBytes(StandardCharsets.UTF_8);
      return ByteBuffer.wrap(bytes);
      }
```

```
   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
          credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
          throw new AmazonClientException("Cannot load your credentials from disk", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
          new AwsClientBuilder.EndpointConfiguration(endpointProtocol, endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
          .withEndpointConfiguration(endpoint)
          .withCredentials(new AWSStaticCredentialsProvider(credentials))
          .build();

      // Create a certificate request:
      IssueCertificateRequest req = new IssueCertificateRequest();

      // Set the CA ARN.
      req.withCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");

      // Specify the certificate signing request (CSR) for the certificate to be signed
 and issued.
      String strCSR =
      "-----BEGIN CERTIFICATE REQUEST-----\n" +
      "base64-encoded certificate\n" +
      "-----END CERTIFICATE REQUEST-----\n";
      ByteBuffer csrByteBuffer = stringToByteBuffer(strCSR);
      req.setCsr(csrByteBuffer);

      // Specify the template for the issued certificate.
      req.withTemplateArn("arn:aws:acm-pca:::template/EndEntityCertificate/V1");
```

```
      // Set the signing algorithm.
      req.withSigningAlgorithm(SigningAlgorithm.SHA256WITHRSA);

      // Set the validity period for the certificate to be issued.
      Validity validity = new Validity();
      validity.withValue(<<3650L>>);
      validity.withType("DAYS");
      req.withValidity(validity);

      // Set the idempotency token.
      req.setIdempotencyToken("1234");

      // Issue the certificate.
      IssueCertificateResult result = null;
      try {
         result = client.issueCertificate(req);
      } catch (LimitExceededException ex) {
         throw ex;
      } catch (ResourceNotFoundException ex) {
         throw ex;
      } catch (InvalidStateException ex) {
         throw ex;
      } catch (InvalidArnException ex) {
         throw ex;
      } catch (InvalidArgsException ex) {
         throw ex;
      } catch (MalformedCSRException ex) {
         throw ex;
      }

      // Retrieve and display the certificate ARN.
      String arn = result.getCertificateArn();
      System.out.println(arn);
   }
}
```

您的輸出應類似以下內容：

```
arn:aws:acm-pca:region:account:certificate-authority/CA_ID/certificate/certificate_ID
```

ListCertificateAuthorities

下列 Java 範例示範如何使用 ListCertificateAuthorities 操作。

此操作可列出您使用 CreateCertificateAuthority 操作建立的私有憑證授權機構 (CA)。

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.ListCertificateAuthoritiesRequest;
import com.amazonaws.services.acmpca.model.ListCertificateAuthoritiesResult;
import com.amazonaws.services.acmpca.model.InvalidNextTokenException;

public class ListCertificateAuthorities {

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
          credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
          throw new AmazonClientException("Cannot load your credentials from file.",
 e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
          new AwsClientBuilder.EndpointConfiguration(endpointProtocol, endpointRegion);
```

```
        // Create a client that you can use to make requests.
        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
            .withEndpointConfiguration(endpoint)
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();

        // Create the request object.
        ListCertificateAuthoritiesRequest req = new ListCertificateAuthoritiesRequest();
        req.withMaxResults(10);

        // Retrieve a list of your CAs.
        ListCertificateAuthoritiesResult result= null;
        try {
            result = client.listCertificateAuthorities(req);
        } catch (InvalidNextTokenException ex) {
            throw ex;
        }

        // Display the CA list.
        System.out.println(result.getCertificateAuthorities());
    }
}
```

如果您有任何可列出的憑證授權機構，則您的輸出應類似下列：

```
[{
 Arn: arn: aws: acm-pca: region: account: certificate-
authority/12345678-1234-1234-1234-123456789012,
 CreatedAt: TueNov0712: 05: 39PST2017,
 LastStateChangeAt: WedJan1012: 35: 39PST2018,
 Type: SUBORDINATE,
 Serial: 4109,
 Status: DISABLED,
 NotBefore: TueNov0712: 19: 15PST2017,
 NotAfter: FriNov0513: 19: 15PDT2027,
 CertificateAuthorityConfiguration: {
  KeyType: RSA2048,
  SigningAlgorithm: SHA256WITHRSA,
  Subject: {
   Organization: ExampleCorp,
   OrganizationalUnit: HR,
   State: Washington,
```

```
    CommonName: www.example.com,
    Locality: Seattle,

   }
  },
  RevocationConfiguration: {
   CrlConfiguration: {
    Enabled: true,
    ExpirationInDays: 3650,
    CustomCname: your-custom-name,
    S3BucketName: your-bucket-name
   }
  }
},
{
 Arn: arn: aws: acm-pca: region: account>: certificate-
authority/12345678-1234-1234-1234-123456789012,
 CreatedAt: WedSep1312: 54: 52PDT2017,
 LastStateChangeAt: WedSep1312: 54: 52PDT2017,
 Type: SUBORDINATE,
 Serial: 4100,
 Status: ACTIVE,
 NotBefore: WedSep1314: 11: 19PDT2017,
 NotAfter: SatSep1114: 11: 19PDT2027,
 CertificateAuthorityConfiguration: {
  KeyType: RSA2048,
  SigningAlgorithm: SHA256WITHRSA,
  Subject: {
   Country: US,
   Organization: ExampleCompany,
   OrganizationalUnit: Sales,
   State: Washington,
   CommonName: www.example.com,
   Locality: Seattle,

  }
 },
 RevocationConfiguration: {
  CrlConfiguration: {
   Enabled: false,
   ExpirationInDays: 5,
   CustomCname: your-custom-name,
   S3BucketName: your-bucket-name
  }
```

```
 }
},
{
 Arn: arn: aws: acm-pca: region: account>: certificate-
authority/12345678-1234-1234-1234-123456789012,
 CreatedAt: FriJan1213: 57: 11PST2018,
 LastStateChangeAt: FriJan1213: 57: 11PST2018,
 Type: SUBORDINATE,
 Status: PENDING_CERTIFICATE,
 CertificateAuthorityConfiguration: {
  KeyType: RSA2048,
  SigningAlgorithm: SHA256WITHRSA,
  Subject: {
   Country: US,
   Organization: Examples-R-Us Ltd.,
   OrganizationalUnit: corporate,
   State: WA,
   CommonName: www.examplesrus.com,
   Locality: Seattle,

  }
 },
 RevocationConfiguration: {
  CrlConfiguration: {
   Enabled: true,
   ExpirationInDays: 365,
   CustomCname: your-custom-name,
   S3BucketName: your-bucket-name
  }
 }
},
{
 Arn: arn: aws: acm-pca: region: account>: certificate-
authority/12345678-1234-1234-1234-123456789012,
 CreatedAt: FriJan0511: 14: 21PST2018,
 LastStateChangeAt: FriJan0511: 14: 21PST2018,
 Type: SUBORDINATE,
 Serial: 4116,
 Status: ACTIVE,
 NotBefore: FriJan0512: 12: 56PST2018,
 NotAfter: MonJan0312: 12: 56PST2028,
 CertificateAuthorityConfiguration: {
  KeyType: RSA2048,
  SigningAlgorithm: SHA256WITHRSA,
```

```
  Subject: {
   Country: US,
   Organization: ExamplesLLC,
   OrganizationalUnit: CorporateOffice,
   State: WA,
   CommonName: www.example.com,
   Locality: Seattle,

  }
 },
 RevocationConfiguration: {
  CrlConfiguration: {
   Enabled: true,
   ExpirationInDays: 3650,
   CustomCname: your-custom-name,
   S3BucketName: your-bucket-name
  }
 }
}]
```

ListPermissions

以下 Java 範例示範如何使用 ListPermissions 操作。

此操作會列出您私有 CA 所指派的許可 (若有的話)。許可，包括 IssueCertificate、GetCertificate和 ListPermissions，可以使用 CreatePermission 操作指派給 AWS 服務主體，並使用 DeletePermissions 操作撤銷。

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.ListPermissionsRequest;
import com.amazonaws.services.acmpca.model.ListPermissionsResult;

import com.amazonaws.AmazonClientException;
```

```
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidNextTokenException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.RequestFailedException;

public class ListPermissions {

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
          credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
          throw new AmazonClientException("Cannot load your credentials from disk", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
          new AwsClientBuilder.EndpointConfiguration(endpointProtocol, endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
         .withEndpointConfiguration(endpoint)
         .withCredentials(new AWSStaticCredentialsProvider(credentials))
         .build();

      // Create a request object and set the CA ARN.
      ListPermissionsRequest req = new ListPermissionsRequest();
      req.withCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");

      // List the tags.
      ListPermissionsResult result = null;
      try {
         result = client.listPermissions(req);
      } catch (InvalidArnException ex) {
         throw ex;
```

```
      } catch (InvalidStateException ex) {
         throw ex;
      } catch(RequestFailedException ex) {
         throw ex;
      } catch (ResourceNotFoundException ex) {
         throw ex;
      }

      // Retrieve and display the permissions.
      System.out.println(result);
   }
}
```

若指定的私有 CA 已將許可指派給服務委託人，您的輸出應該會和以下內容相似：

```
[{
       Arn: arn:aws:acm-
pca:region:account:permission/12345678-1234-1234-1234-123456789012,
       CreatedAt: WedFeb0317: 05: 39PST2019,
       Prinicpal: acm.amazonaws.com,
       Permissions: {
            ISSUE_CERTIFICATE,
            GET_CERTIFICATE,
            DELETE,CERTIFICATE
       },
       SourceAccount: account
}]
```

ListTags

下列 Java 範例示範如何使用 ListTags 操作。

此操作可列出與私有 CA 相關的標籤 (如果有)。標籤是您可以用於識別及組織 CA 的標記。每個標籤皆包含索引鍵與選用值。呼叫 TagCertificateAuthority 操作，即可新增一或多個標籤到您的 CA。呼叫 UntagCertificateAuthority 操作可移除標籤。

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
```

```
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.ListTagsRequest;
import com.amazonaws.services.acmpca.model.ListTagsResult;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidArnException;

public class ListTags {

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
          credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
          throw new AmazonClientException("Cannot load your credentials from disk", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
          new AwsClientBuilder.EndpointConfiguration(endpointProtocol, endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
          .withEndpointConfiguration(endpoint)
          .withCredentials(new AWSStaticCredentialsProvider(credentials))
          .build();

      // Create a request object and set the CA ARN.
      ListTagsRequest req = new ListTagsRequest();
      req.withCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");
```

```
        // List the tags
        ListTagsResult result = null;
        try {
            result = client.listTags(req);
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        }

        // Retrieve and display the tags.
        System.out.println(result);
    }
}
```

如果您有任何可列出的標籤，則您的輸出應類似下列：

```
{Tags: [{Key: Admin,Value: Alice}, {Key: Purpose,Value: WebServices}],}
```

PutPolicy

下列 Java 範例示範如何使用 PutPolicy 操作。

操作會將資源型政策連接至私有 CA，以啟用跨帳戶共用。獲得政策授權時，位於另一個 AWS 帳戶的委託人可以使用其未擁有的私有 CA 來發行和續約私有最終實體憑證。您可以呼叫 ListCertificateAuthorities 動作來尋找私有 CA 的 ARN。如需政策的範例，請參閱 AWS 私有 CA 以資源為基礎的政策指南。

一旦政策連接到 CA，您可以使用 GetPolicy 動作來檢查政策，或使用 DeletePolicy 動作來刪除政策。

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.AmazonClientException;
```

```
import com.amazonaws.services.acmpca.model.PutPolicyRequest;
import com.amazonaws.services.acmpca.model.PutPolicyResult;
import com.amazonaws.services.acmpca.model.AWSACMPCAException;
import com.amazonaws.services.acmpca.model.ConcurrentModificationException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.InvalidPolicyException;
import com.amazonaws.services.acmpca.model.LockoutPreventedException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;

public class PutPolicy {

    public static void main(String[] args) throws Exception {

        // Retrieve your credentials from the C:\Users\name\.aws\credentials file
        // in Windows or the .aws/credentials file in Linux.
        AWSCredentials credentials = null;
        try {
            credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException("Cannot load your credentials from file.",
 e);
        }

        // Define the endpoint for your sample.
        String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
        String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
        EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
 endpointRegion);

        // Create a client that you can use to make requests.
        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
            .withEndpointConfiguration(endpoint)
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();
```

```
        // Create the request object.
        PutPolicyRequest req = new PutPolicyRequest();

        // Set the resource ARN.
        req.withResourceArn("arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566");

        // Import and set the policy.
        // Note: This code assumes the file "ShareResourceWithAccountPolicy.json" is in
 a folder titled policy.
        String policy = new String(Files.readAllBytes(Paths.get("policy",
 "ShareResourceWithAccountPolicy.json")));
        req.withPolicy(policy);

        // Retrieve a list of your CAs.
        PutPolicyResult result = null;
        try {
            result = client.putPolicy(req);
        } catch (ConcurrentModificationException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (InvalidPolicyException ex) {
            throw ex;
        } catch (LockoutPreventedException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (AWSACMPCAException ex) {
            throw ex;
        }
    }
}
```

RestoreCertificateAuthority

以下 Java 範例會示範如何使用 RestoreCertificateAuthority 操作。您可以隨時還原仍處於還原期間的私有 CA。目前，這個期間可以從刪除之日起持續 7 到 30 天，並可在刪除 CA 時加以定義。如需詳細資訊，請參閱還原 CA。另請參閱 DeleteCertificateAuthority Java 範例。

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.RestoreCertificateAuthorityRequest;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;

public class RestoreCertificateAuthority {

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
          credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
          throw new AmazonClientException("Cannot load your credentials from file.",
 e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
          new AwsClientBuilder.EndpointConfiguration(endpointProtocol, endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
          .withEndpointConfiguration(endpoint)
```

```
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();

        // Create the request object.
        RestoreCertificateAuthorityRequest req = new
 RestoreCertificateAuthorityRequest();

        // Set the certificate authority ARN.
        req.withCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");

        // Restore the CA.
        try {
            client.restoreCertificateAuthority(req);
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        }
    }
}
```

RevokeCertificate

下列 Java 範例示範如何使用 RevokeCertificate 操作。

此操作可撤銷您透過呼叫 IssueCertificate 操作發行的憑證。如果您在建立或更新私有 CA 時啟用憑證撤銷清單 (CRL)，CRL 中會包含已撤銷憑證的相關資訊。 會將 CRL AWS 私有 CA 寫入您指定的 Amazon S3 儲存貯體。如需詳細資訊，請參閱 CrlConfiguration 結構。

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;
```

```
import com.amazonaws.services.acmpca.model.RevokeCertificateRequest;
import com.amazonaws.services.acmpca.model.RevocationReason;

import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.RequestAlreadyProcessedException;
import com.amazonaws.services.acmpca.model.RequestInProgressException;

public class RevokeCertificate {

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
          credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
          throw new AmazonClientException("Cannot load your credentials from disk", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
          new AwsClientBuilder.EndpointConfiguration(endpointProtocol, endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
          .withEndpointConfiguration(endpoint)
          .withCredentials(new AWSStaticCredentialsProvider(credentials))
          .build();

      // Create a request object.
      RevokeCertificateRequest req = new RevokeCertificateRequest();

      // Set the certificate authority ARN.
      req.setCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");
```

```
        // Set the certificate serial number.
        req.setCertificateSerial("79:3f:0d:5b:6a:04:12:5e:2c:9c:fb:52:37:35:98:fe");

        // Set the RevocationReason.
        req.withRevocationReason(RevocationReason.<<KEY_COMPROMISE>>);

        // Revoke the certificate.
        try {
            client.revokeCertificate(req);
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (RequestAlreadyProcessedException ex) {
            throw ex;
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }
    }
}
```

TagCertificateAuthorities

下列 Java 範例示範如何使用 TagCertificateAuthority 操作。

此操作會將一或多個標籤新增到您的私有 CA。標籤是可用來識別和組織 AWS 資源的標籤。每個標籤皆包含索引鍵與選用值。呼叫此操作時，您可以依據 Amazon Resource Name (ARN) 來指定私有 CA。使用鍵值組指定標籤。若要識別該 CA 的特定特性，您可以將標籤套用至一個私有 CA。或者，若您要篩選這些 CA 之間的共同關係，您可以將相同的標籤套用至多個私有 CA。若要移除一或多個標籤，請使用 UntagCertificateAuthority 操作。呼叫 ListTags 操作以查看與 CA 相關聯的標籤。

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
```

```
import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.TagCertificateAuthorityRequest;
import com.amazonaws.services.acmpca.model.Tag;

import java.util.ArrayList;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidTagException;
import com.amazonaws.services.acmpca.model.TooManyTagsException;

public class TagCertificateAuthorities {

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
          credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
          throw new AmazonClientException("Cannot load your credentials from disk", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
          new AwsClientBuilder.EndpointConfiguration(endpointProtocol, endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
          .withEndpointConfiguration(endpoint)
          .withCredentials(new AWSStaticCredentialsProvider(credentials))
          .build();

      // Create a tag - method 1
      Tag tag1 = new Tag();
```

```
        tag1.withKey("Administrator");
        tag1.withValue("Bob");

        // Create a tag - method 2
        Tag tag2 = new Tag()
            .withKey("Purpose")
            .withValue("WebServices");

        // Add the tags to a collection.
        ArrayList<Tag> tags = new ArrayList<Tag>();
        tags.add(tag1);
        tags.add(tag2);

        // Create a request object and specify the certificate authority ARN.
        TagCertificateAuthorityRequest req = new TagCertificateAuthorityRequest();
        req.setCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");
        req.setTags(tags);

        // Add a tag
        try {
            client.tagCertificateAuthority(req);
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidTagException ex) {
            throw ex;
        } catch (TooManyTagsException ex) {
            throw ex;
        }
    }
}
```

UntagCertificateAuthority

以下 Java 範例會示範如何使用 UntagCertificateAuthority 操作。

此操作會從私有 CA 移除一或多個標籤。一個標籤包含一對索引鍵/值對。如果您在呼叫此操作時沒有指定標籤的值部分，便會移除標籤，不論其值為何。如果您指定一個值，標籤只會在與指定的值相關聯時移除。若要新增標籤至私有 CA，請使用 TagCertificateAuthority 操作。呼叫 ListTags 操作以查看與 CA 相關聯的標籤。

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import java.util.ArrayList;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.UntagCertificateAuthorityRequest;
import com.amazonaws.services.acmpca.model.Tag;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidTagException;

public class UntagCertificateAuthority {

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
          credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
          throw new AmazonClientException("Cannot load your credentials from disk", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
          new AwsClientBuilder.EndpointConfiguration(endpointProtocol, endpointRegion);

      // Create a client that you can use to make requests.
```

```
        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
            .withEndpointConfiguration(endpoint)
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();

        // Create a Tag object with the tag to delete.
        Tag tag = new Tag();
        tag.withKey("Administrator");
        tag.withValue("Bob");

        // Add the tags to a collection.
        ArrayList<Tag> tags = new ArrayList<Tag>();
        tags.add(tag);

        // Create a request object and specify the certificate authority ARN.
        UntagCertificateAuthorityRequest req = new UntagCertificateAuthorityRequest();
        req.withCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");
        req.withTags(tags);

        // Delete the tag
        try {
            client.untagCertificateAuthority(req);
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidTagException ex) {
            throw ex;
        }
    }
}
```

UpdateCertificateAuthority

下列 Java 範例會示範如何使用 UpdateCertificateAuthority 操作。

此操作會更新私有憑證授權機構 (CA) 的狀態或組態。您的私有 CA 必須處於 ACTIVE 或 DISABLED 狀態才能更新。您可以停用 ACTIVE 狀態的私有 CA，或將處於 DISABLED 狀態的 CA 再次啟用。

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
```

```
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.UpdateCertificateAuthorityRequest;
import com.amazonaws.services.acmpca.model.CertificateAuthorityStatus;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.ConcurrentModificationException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.InvalidPolicyException;
import com.amazonaws.services.acmpca.model.CrlConfiguration;
import com.amazonaws.services.acmpca.model.RevocationConfiguration;

public class UpdateCertificateAuthority {

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
          credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
          throw new AmazonClientException("Cannot load your credentials from file.",
 e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
          new AwsClientBuilder.EndpointConfiguration(endpointProtocol, endpointRegion);

      // Create a client that you can use to make requests.
```

```
        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
            .withEndpointConfiguration(endpoint)
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();

        // Create the request object.
        UpdateCertificateAuthorityRequest req = new UpdateCertificateAuthorityRequest();

        // Set the ARN of the private CA that you want to update.
        req.setCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");

        // Define the certificate revocation list configuration. If you do not want to
        // update the CRL configuration, leave the CrlConfiguration structure alone and
        // do not set it on your UpdateCertificateAuthorityRequest object.
        CrlConfiguration crlConfigure = new CrlConfiguration();
        crlConfigure.withEnabled(true);
        crlConfigure.withExpirationInDays(365);
        crlConfigure.withCustomCname("your-custom-name");
        crlConfigure.withS3BucketName("your-bucket-name");

        // Set the CRL configuration onto your UpdateCertificateAuthorityRequest object.
        // If you do not want to change your CRL configuration, do not use the
        // setCrlConfiguration method.
        RevocationConfiguration revokeConfig = new RevocationConfiguration();
        revokeConfig.setCrlConfiguration(crlConfigure);
        req.setRevocationConfiguration(revokeConfig);

        // Set the status.
        req.withStatus(CertificateAuthorityStatus.<<ACTIVE>>);

        // Create the result object.
        try {
            client.updateCertificateAuthority(req);
        } catch (ConcurrentModificationException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
```

```
        } catch (InvalidPolicyException ex) {
            throw ex;
        }
    }
}
```

使用自訂主旨名稱建立 CAs和憑證

CustomAttribute 物件可讓管理員將自訂物件識別符 (OIDs) 傳遞至私有 CAs和憑證。自訂 OIDs 可用來建立特殊的主題名稱階層，以反映組織的結構和需求。自訂憑證必須使用其中一個ApiPassthrough範本建立。如需範本的詳細資訊，請參閱AWS 私有 CA 範本變體。如需使用自訂屬性的詳細資訊，請參閱 發行私有終端實體憑證和 在 中建立私有 CA AWS 私有 CA。

您無法StandardAttributes搭配 使用 CustomAttributes。不過，您可以將標準 OIDs 做為 的一部分傳遞CustomAttributes。預設主旨名稱 OIDs列於下表中：

主題名稱	物件 ID
Country	2.5.4.6
CommonName	2.5.4.3
DistinguishedNameQualifier	2.5.4.46
GenerationQualifier	2.5.4.44
GivenName	2.5.4.42
Initials	2.5.4.43
Locality	2.5.4.7
組織	2.5.4.10
OrganizationalUnit	2.5.4.11
Pseudonym	2.5.4.65
SerialNumber	2.5.4.5
State	2.5.4.8

主題名稱	物件 ID
Surname	2.5.4.4
Title	2.5.4.12

主題

- 使用 CustomAttribute 建立 CA
- 使用 CustomAttribute 發行憑證

使用 CustomAttribute 建立 CA

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;


import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.ASN1Subject;
import com.amazonaws.services.acmpca.model.CertificateAuthorityConfiguration;
import com.amazonaws.services.acmpca.model.CertificateAuthorityType;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityResult;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityRequest;
import com.amazonaws.services.acmpca.model.CrlConfiguration;
import com.amazonaws.services.acmpca.model.CustomAttribute;
import com.amazonaws.services.acmpca.model.KeyAlgorithm;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Tag;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

import com.amazonaws.AmazonClientException;
```

```
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.InvalidPolicyException;
import com.amazonaws.services.acmpca.model.RevocationConfiguration;


public class CreateCertificateAuthorityWithCustomAttributes {

    public static void main(String[] args) throws Exception {

        // Retrieve your credentials from the C:\Users\name\.aws\credentials file
        // in Windows or the .aws/credentials file in Linux.
        AWSCredentials credentials = null;
        try {
            credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException(
                   "Cannot load the credentials from the credential profiles file. " +
                   "Please make sure that your credentials file is at the correct " +
                   "location (C:\\Users\\joneps\\.aws\\credentials), and is in valid
 format.",
                   e);
        }

        // Define the endpoint for your sample.
        String endpointRegion = "us-west-2";  // Substitute your region here, e.g. "us-
west-2"
        String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
        EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
 endpointRegion);

        // Create a client that you can use to make requests.
        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
            .withEndpointConfiguration(endpoint)
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();

        // Define custom attributes
        List<CustomAttribute> customAttributes = Arrays.asList(
            new CustomAttribute()
                .withObjectIdentifier("2.5.4.6") // Country
                .withValue("US"),
```

```
            new CustomAttribute()
                .withObjectIdentifier("2.5.4.3") // CommonName
                .withValue("CommonName"),
            new CustomAttribute()
                .withObjectIdentifier("1.3.6.1.4.1") // CustomOID
                .withValue("ABCDEFG"),
            new CustomAttribute()
                .withObjectIdentifier("1.3.6.1.4.1") // CustomOID
                .withValue("BCDEFGH")
        );

        // Define a CA subject.
        ASN1Subject subject = new ASN1Subject();
        subject.setCustomAttributes(customAttributes);

        // Define the CA configuration.
        CertificateAuthorityConfiguration configCA = new
CertificateAuthorityConfiguration();
        configCA.withKeyAlgorithm(KeyAlgorithm.RSA_2048);
        configCA.withSigningAlgorithm(SigningAlgorithm.SHA256WITHRSA);
        configCA.withSubject(subject);

        // Define a certificate revocation list configuration.
        CrlConfiguration crlConfigure = new CrlConfiguration();
        crlConfigure.withEnabled(true);
        crlConfigure.withExpirationInDays(365);
        crlConfigure.withCustomCname(null);
        crlConfigure.withS3BucketName("your-bucket-name");

        RevocationConfiguration revokeConfig = new RevocationConfiguration();
        revokeConfig.setCrlConfiguration(crlConfigure);

        // Define a certificate authority type: ROOT or SUBORDINATE
        CertificateAuthorityType caType = CertificateAuthorityType.SUBORDINATE;

        // Create a tag - method 1
        Tag tag1 = new Tag();
        tag1.withKey("PrivateCA");
        tag1.withValue("Sample");

        // Create a tag - method 2
        Tag tag2 = new Tag()
            .withKey("Purpose")
            .withValue("WebServices");
```

```
        // Add the tags to a collection.
        ArrayList<Tag> tags = new ArrayList<Tag>();
        tags.add(tag1);
        tags.add(tag2);

        // Create the request object.
        CreateCertificateAuthorityRequest req = new
 CreateCertificateAuthorityRequest();
        req.withCertificateAuthorityConfiguration(configCA);
        req.withRevocationConfiguration(revokeConfig);
        req.withIdempotencyToken("1234");
        req.withCertificateAuthorityType(caType);
        req.withTags(tags);


        // Create the private CA.
        CreateCertificateAuthorityResult result = null;
        try {
            result = client.createCertificateAuthority(req);
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (InvalidPolicyException ex) {
            throw ex;
        } catch (LimitExceededException ex) {
            throw ex;
        }

        // Retrieve the ARN of the private CA.
        String arn = result.getCertificateAuthorityArn();
        System.out.println(arn);
    }
}
```

使用 CustomAttribute 發行憑證

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
```

```
import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.Arrays;
import java.util.Base64;
import java.util.List;
import java.util.Objects;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;
import com.amazonaws.services.acmpca.model.ASN1Subject;
import com.amazonaws.services.acmpca.model.ApiPassthrough;
import com.amazonaws.services.acmpca.model.CustomAttribute;
import com.amazonaws.services.acmpca.model.Extensions;
import com.amazonaws.services.acmpca.model.IssueCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateResult;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Validity;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.MalformedCSRException;

public class IssueCertificateWithCustomAttributes {
   private static ByteBuffer stringToByteBuffer(final String string) {
      if (Objects.isNull(string)) {
          return null;
      }
      byte[] bytes = string.getBytes(StandardCharsets.UTF_8);
      return ByteBuffer.wrap(bytes);
   }

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
          credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
```

```
            throw new AmazonClientException("Cannot load your credentials from disk", e);
        }

        // Define the endpoint for your sample.
        String endpointRegion = "us-west-2";  // Substitute your region here, e.g. "us-
west-2"
        String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
        EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol, endpointRegion);

        // Create a client that you can use to make requests.
        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
            .withEndpointConfiguration(endpoint)
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();

        // Create a certificate request:
        IssueCertificateRequest req = new IssueCertificateRequest();

        // Set the CA ARN.
        req.withCertificateAuthorityArn("arn:aws:acm-pca:region:account:" +
           "certificate-authority/12345678-1234-1234-1234-123456789012");

        // Specify the certificate signing request (CSR) for the certificate to be signed
 and issued.
        String strCSR =
        "-----BEGIN CERTIFICATE REQUEST-----\n" +
        "base64-encoded CSR\n" +
        "-----END CERTIFICATE REQUEST-----\n";
        ByteBuffer csrByteBuffer = stringToByteBuffer(strCSR);
        req.setCsr(csrByteBuffer);

        // Specify the template for the issued certificate.
        req.withTemplateArn("arn:aws:acm-pca:::template/
EndEntityCertificate_APIPassthrough/V1");

        // Set the signing algorithm.
        req.withSigningAlgorithm(SigningAlgorithm.SHA256WITHRSA);

        // Set the validity period for the certificate to be issued.
        Validity validity = new Validity();
        validity.withValue(100L);
        validity.withType("DAYS");
```

```
        req.withValidity(validity);

        // Set the idempotency token.
        req.setIdempotencyToken("1234");

        // Define custom attributes
        List<CustomAttribute> customAttributes = Arrays.asList(
           new CustomAttribute()
                 .withObjectIdentifier("2.5.4.6") // Country
                 .withValue("US"),
           new CustomAttribute()
                 .withObjectIdentifier("2.5.4.3") // CommonName
                 .withValue("CommonName"),
           new CustomAttribute()
                 .withObjectIdentifier("1.3.6.1.4.1") // CustomOID
                 .withValue("ABCDEFG"),
           new CustomAttribute()
                 .withObjectIdentifier("1.3.6.1.4.1") // CustomOID
                 .withValue("BCDEFGH")
        );

        // Define certificate subject.
        ASN1Subject subject = new ASN1Subject();
        subject.setCustomAttributes(customAttributes);

        // Add subject to api-passthrough
        ApiPassthrough apiPassthrough = new ApiPassthrough();
        apiPassthrough.setSubject(subject);
        req.setApiPassthrough(apiPassthrough);

        // Issue the certificate.
        IssueCertificateResult result = null;
        try {
           result = client.issueCertificate(req);
        } catch (LimitExceededException ex) {
           throw ex;
        } catch (ResourceNotFoundException ex) {
           throw ex;
        } catch (InvalidStateException ex) {
           throw ex;
        } catch (InvalidArnException ex) {
           throw ex;
        } catch (InvalidArgsException ex) {
           throw ex;
```

```
        } catch (MalformedCSRException ex) {
            throw ex;
        }

        // Retrieve and display the certificate ARN.
        String arn = result.getCertificateArn();
        System.out.println(arn);
    }
}
```

建立具有自訂擴充功能的憑證

CustomExtension 物件可讓管理員在私有憑證中設定自訂 X.509 擴充功能。自訂憑證必須使用其中一個 ApiPassthrough 範本建立。如需範本的詳細資訊，請參閱AWS 私有 CA 範本變體。如需使用自訂延伸模組的詳細資訊，請參閱發行私有終端實體憑證。

主題

- 使用 NameConstraints 延伸模組啟用次級 CA
- 使用 QC 陳述式擴充功能發行憑證

使用 NameConstraints 延伸模組啟用次級 CA

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.Arrays;
import java.util.Base64;
import java.util.Objects;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.IssueCertificateRequest;
```

```
import com.amazonaws.services.acmpca.model.IssueCertificateResult;
import com.amazonaws.services.acmpca.model.KeyAlgorithm;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Validity;
import com.amazonaws.waiters.Waiter;
import com.amazonaws.waiters.WaiterParameters;
import com.amazonaws.waiters.WaiterTimedOutException;
import com.amazonaws.waiters.WaiterUnrecoverableException;
import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.RevocationConfiguration;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidPolicyException;
import com.amazonaws.services.acmpca.model.ASN1Subject;
import com.amazonaws.services.acmpca.model.AWSACMPCAException;
import com.amazonaws.services.acmpca.model.ApiPassthrough;
import com.amazonaws.services.acmpca.model.CertificateAuthorityConfiguration;
import com.amazonaws.services.acmpca.model.CertificateAuthorityType;
import com.amazonaws.services.acmpca.model.CertificateMismatchException;
import com.amazonaws.services.acmpca.model.ConcurrentModificationException;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityRequest;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityResult;
import com.amazonaws.services.acmpca.model.CrlConfiguration;
import com.amazonaws.services.acmpca.model.CustomExtension;
import com.amazonaws.services.acmpca.model.Extensions;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCertificateRequest;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCertificateResult;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrRequest;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrResult;
import com.amazonaws.services.acmpca.model.GetCertificateRequest;
import com.amazonaws.services.acmpca.model.GetCertificateResult;
import
 com.amazonaws.services.acmpca.model.ImportCertificateAuthorityCertificateRequest;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.MalformedCSRException;
import com.amazonaws.services.acmpca.model.MalformedCertificateException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.RequestInProgressException;

import org.bouncycastle.asn1.x509.GeneralName;
import org.bouncycastle.asn1.x509.GeneralSubtree;
import org.bouncycastle.asn1.x509.NameConstraints;
```

```
import lombok.SneakyThrows;

public class SubordinateCAActivationWithNameConstraints {
   public static void main(String[] args) throws Exception {
      // Place your own Root CA ARN here.
      String rootCAArn = "arn:aws:acm-pca:region:123456789012:certificate-
authority/12345678-1234-1234-1234-123456789012";

      // Define the endpoint region for your sample.
      String endpointRegion = "us-west-2";  // Substitute your region here, e.g. "us-
west-2"

      // Define a CA subject.
      ASN1Subject subject = new ASN1Subject();
      subject.setOrganization("Example Organization");
      subject.setOrganizationalUnit("Example");
      subject.setCountry("US");
      subject.setState("Virginia");
      subject.setLocality("Arlington");
      subject.setCommonName("SubordinateCA");

      // Define the CA configuration.
      CertificateAuthorityConfiguration configCA = new
 CertificateAuthorityConfiguration();
      configCA.withKeyAlgorithm(KeyAlgorithm.RSA_2048);
      configCA.withSigningAlgorithm(SigningAlgorithm.SHA256WITHRSA);
      configCA.withSubject(subject);

      // Define a certificate revocation list configuration.
      CrlConfiguration crlConfigure = new CrlConfiguration();
      crlConfigure.withEnabled(true);
      crlConfigure.withExpirationInDays(365);
      crlConfigure.withCustomCname(null);
      crlConfigure.withS3BucketName("your-bucket-name");

      // Define a certificate authority type
      CertificateAuthorityType caType = CertificateAuthorityType.SUBORDINATE;

      // ** Execute core code samples for Subordinate CA activation in sequence **
      AWSACMPCA client = ClientBuilder(endpointRegion);
      String rootCertificate = GetCertificateAuthorityCertificate(rootCAArn, client);
      String subordinateCAArn = CreateCertificateAuthority(configCA, crlConfigure,
 caType, client);
```

```
      String csr = GetCertificateAuthorityCsr(subordinateCAArn, client);
      String subordinateCertificateArn = IssueCertificate(rootCAArn, csr, client);
      String subordinateCertificate = GetCertificate(subordinateCertificateArn,
rootCAArn, client);
      ImportCertificateAuthorityCertificate(subordinateCertificate, rootCertificate,
subordinateCAArn, client);
  }

  private static AWSACMPCA ClientBuilder(String endpointRegion) {
      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
          credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
          throw new AmazonClientException(
                  "Cannot load the credentials from the credential profiles file. " +
                  "Please make sure that your credentials file is at the correct " +
                  "location (C:\\Users\\joneps\\.aws\\credentials), and is in valid
format.",
                  e);
      }

      String endpointProtocol = "https://acm-pca." + endpointRegion +
".amazonaws.com/";
      EndpointConfiguration endpoint =
          new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
          .withEndpointConfiguration(endpoint)
          .withCredentials(new AWSStaticCredentialsProvider(credentials))
          .build();

      return client;
  }

  private static String GetCertificateAuthorityCertificate(String rootCAArn, AWSACMPCA
client) {
      // ** GetCertificateAuthorityCertificate **

      // Create a request object and set the certificate authority ARN,
      GetCertificateAuthorityCertificateRequest getCACertificateRequest =
```

```
            new GetCertificateAuthorityCertificateRequest();
      getCACertificateRequest.withCertificateAuthorityArn(rootCAArn);

      // Create a result object.
      GetCertificateAuthorityCertificateResult getCACertificateResult = null;
      try {
          getCACertificateResult =
client.getCertificateAuthorityCertificate(getCACertificateRequest);
      } catch (ResourceNotFoundException ex) {
          throw ex;
      } catch (InvalidStateException ex) {
          throw ex;
      } catch (InvalidArnException ex) {
          throw ex;
      }

      // Retrieve and display the certificate information.
      String rootCertificate = getCACertificateResult.getCertificate();
      System.out.println("Root CA Certificate / Certificate Chain:");
      System.out.println(rootCertificate);

      return rootCertificate;
  }

  private static String CreateCertificateAuthority(CertificateAuthorityConfiguration
configCA, CrlConfiguration crlConfigure, CertificateAuthorityType caType, AWSACMPCA
client) {
      RevocationConfiguration revokeConfig = new RevocationConfiguration();
      revokeConfig.setCrlConfiguration(crlConfigure);

      // Create the request object.
      CreateCertificateAuthorityRequest createCARequest = new
CreateCertificateAuthorityRequest();
      createCARequest.withCertificateAuthorityConfiguration(configCA);
      createCARequest.withRevocationConfiguration(revokeConfig);
      createCARequest.withIdempotencyToken("1234");
      createCARequest.withCertificateAuthorityType(caType);

      // Create the private CA.
      CreateCertificateAuthorityResult createCAResult = null;
      try {
          createCAResult = client.createCertificateAuthority(createCARequest);
      } catch (InvalidArgsException ex) {
          throw ex;
```

```
        } catch (InvalidPolicyException ex) {
            throw ex;
        } catch (LimitExceededException ex) {
            throw ex;
        }

        // Retrieve the ARN of the private CA.
        String subordinateCAArn = createCAResult.getCertificateAuthorityArn();
        System.out.println("Subordinate CA Arn: " + subordinateCAArn);

        return subordinateCAArn;
    }

    private static String GetCertificateAuthorityCsr(String subordinateCAArn, AWSACMPCA
client) {

        // Create the CSR request object and set the CA ARN.
        GetCertificateAuthorityCsrRequest csrRequest = new
GetCertificateAuthorityCsrRequest();
        csrRequest.withCertificateAuthorityArn(subordinateCAArn);

        // Create waiter to wait on successful creation of the CSR file.
        Waiter<GetCertificateAuthorityCsrRequest> getCSRWaiter =
client.waiters().certificateAuthorityCSRCreated();
        try {
            getCSRWaiter.run(new WaiterParameters<>(csrRequest));
        } catch (WaiterUnrecoverableException e) {
            //Explicit short circuit when the recourse transitions into
            //an undesired state.
        } catch (WaiterTimedOutException e) {
            //Failed to transition into desired state even after polling.
        } catch(AWSACMPCAException e) {
            //Unexpected service exception.
        }

        // Retrieve the CSR.
        GetCertificateAuthorityCsrResult csrResult = null;
        try {
            csrResult = client.getCertificateAuthorityCsr(csrRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
```

```
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }

        // Retrieve and display the CSR;
        String csr = csrResult.getCsr();
        System.out.println("Subordinate CSR:");
        System.out.println(csr);

        return csr;
    }

    private static String IssueCertificate(String rootCAArn, String csr, AWSACMPCA
 client) {

        // Create a certificate request:
        IssueCertificateRequest issueRequest = new IssueCertificateRequest();

        // Set the issuing CA ARN.
        issueRequest.withCertificateAuthorityArn(rootCAArn);

        // Set the template ARN.
        issueRequest.withTemplateArn("arn:aws:acm-pca:::template/
SubordinateCACertificate_PathLen0_APIPassthrough/V1");

        ByteBuffer csrByteBuffer = stringToByteBuffer(csr);
        issueRequest.setCsr(csrByteBuffer);

        // Set the signing algorithm.
        issueRequest.withSigningAlgorithm(SigningAlgorithm.SHA256WITHRSA);

        // Set the validity period for the certificate to be issued.
        Validity validity = new Validity();
        validity.withValue(100L);
        validity.withType("DAYS");
        issueRequest.withValidity(validity);

        // Set the idempotency token.
        issueRequest.setIdempotencyToken("1234");

        // Generate Base64 encoded Nameconstraints extension value
        String base64EncodedExtValue = getNameConstraintExtensionValue();
```

```
        // Generate custom extension
        CustomExtension customExtension = new CustomExtension();
        customExtension.setCritical(true);
        customExtension.setObjectIdentifier("2.5.29.30"); // NameConstraints Extension
OID
        customExtension.setValue(base64EncodedExtValue);

        // Add custom extension to api-passthrough
        ApiPassthrough apiPassthrough = new ApiPassthrough();
        Extensions extensions = new Extensions();
        extensions.setCustomExtensions(Arrays.asList(customExtension));
        apiPassthrough.setExtensions(extensions);
        issueRequest.setApiPassthrough(apiPassthrough);

        // Issue the certificate.
        IssueCertificateResult issueResult = null;
        try {
            issueResult = client.issueCertificate(issueRequest);
        } catch (LimitExceededException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (MalformedCSRException ex) {
            throw ex;
        }

        // Retrieve and display the certificate ARN.
        String subordinateCertificateArn = issueResult.getCertificateArn();
        System.out.println("Subordinate Certificate Arn: " + subordinateCertificateArn);

        return subordinateCertificateArn;
    }

    @SneakyThrows
    private static String getNameConstraintExtensionValue() {
       // Generate Base64 encoded Nameconstraints extension value
       GeneralSubtree dnsPrivate = new GeneralSubtree(new
GeneralName(GeneralName.dNSName, ".private"));
```

```
      GeneralSubtree dnsLocal = new GeneralSubtree(new GeneralName(GeneralName.dNSName,
".local"));
      GeneralSubtree dnsCorp = new GeneralSubtree(new GeneralName(GeneralName.dNSName,
".corp"));
      GeneralSubtree dnsSecretCorp = new GeneralSubtree(new
GeneralName(GeneralName.dNSName, ".secret.corp"));
      GeneralSubtree dnsExample = new GeneralSubtree(new
GeneralName(GeneralName.dNSName, ".example.com"));
      GeneralSubtree[] permittedSubTree = new GeneralSubtree[] { dnsPrivate, dnsLocal,
dnsCorp };
      GeneralSubtree[] excludedSubTree = new GeneralSubtree[] { dnsSecretCorp,
dnsExample };
      NameConstraints nameConstraints = new NameConstraints(permittedSubTree,
excludedSubTree);

      return new String(Base64.getEncoder().encode(nameConstraints.getEncoded()));
  }

  private static String GetCertificate(String subordinateCertificateArn, String
rootCAArn, AWSACMPCA client) {

       // Create a request object.
       GetCertificateRequest certificateRequest = new GetCertificateRequest();

       // Set the certificate ARN.
       certificateRequest.withCertificateArn(subordinateCertificateArn);

       // Set the certificate authority ARN.
       certificateRequest.withCertificateAuthorityArn(rootCAArn);

       // Create waiter to wait on successful creation of the certificate file.
       Waiter<GetCertificateRequest> getCertificateWaiter =
client.waiters().certificateIssued();
       try {
           getCertificateWaiter.run(new WaiterParameters<>(certificateRequest));
       } catch (WaiterUnrecoverableException e) {
           //Explicit short circuit when the recourse transitions into
           //an undesired state.
       } catch (WaiterTimedOutException e) {
           //Failed to transition into desired state even after polling.
       } catch (AWSACMPCAException e) {
           //Unexpected service exception.
       }
```

```
        // Retrieve the certificate and certificate chain.
        GetCertificateResult certificateResult = null;
        try {
            certificateResult = client.getCertificate(certificateRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        }

        // Get the certificate and certificate chain and display the result.
        String subordinateCertificate = certificateResult.getCertificate();
        System.out.println("Subordinate CA Certificate:");
        System.out.println(subordinateCertificate);

        return subordinateCertificate;
    }

    private static void ImportCertificateAuthorityCertificate(String
subordinateCertificate, String rootCertificate, String subordinateCAArn, AWSACMPCA
client) {

        // Create the request object and set the signed certificate, chain and CA ARN.
        ImportCertificateAuthorityCertificateRequest importRequest =
            new ImportCertificateAuthorityCertificateRequest();

        ByteBuffer certByteBuffer = stringToByteBuffer(subordinateCertificate);
        importRequest.setCertificate(certByteBuffer);

        ByteBuffer rootCACertByteBuffer = stringToByteBuffer(rootCertificate);
        importRequest.setCertificateChain(rootCACertByteBuffer);

        // Set the certificate authority ARN.
        importRequest.withCertificateAuthorityArn(subordinateCAArn);

        // Import the certificate.
        try {
            client.importCertificateAuthorityCertificate(importRequest);
```

```
        } catch (CertificateMismatchException ex) {
            throw ex;
        } catch (MalformedCertificateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (ConcurrentModificationException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }
        System.out.println("Subordinate CA certificate successfully imported.");
        System.out.println("Subordinate CA activated successfully.");
    }

    private static ByteBuffer stringToByteBuffer(final String string) {
        if (Objects.isNull(string)) {
            return null;
        }
        byte[] bytes = string.getBytes(StandardCharsets.UTF_8);
        return ByteBuffer.wrap(bytes);
    }
}
```

使用 QC 陳述式擴充功能發行憑證

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.Arrays;
import java.util.Base64;
import java.util.Objects;
```

```
import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;
import com.amazonaws.services.acmpca.model.ApiPassthrough;
import com.amazonaws.services.acmpca.model.CustomExtension;
import com.amazonaws.services.acmpca.model.Extensions;
import com.amazonaws.services.acmpca.model.IssueCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateResult;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Validity;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.MalformedCSRException;

import org.bouncycastle.asn1.ASN1EncodableVector;
import org.bouncycastle.asn1.ASN1ObjectIdentifier;
import org.bouncycastle.asn1.DERSequence;
import org.bouncycastle.asn1.DERUTF8String;
import org.bouncycastle.asn1.x509.qualified.ETSIQCObjectIdentifiers;
import org.bouncycastle.asn1.x509.qualified.QCStatement;

import lombok.SneakyThrows;

public class IssueCertificateWithQCStatement {
   private static ByteBuffer stringToByteBuffer(final String string) {
      if (Objects.isNull(string)) {
          return null;
      }
      byte[] bytes = string.getBytes(StandardCharsets.UTF_8);
      return ByteBuffer.wrap(bytes);
   }

   @SneakyThrows
   private static String generateQCStatementBase64ExtValue() {
      DERSequence qcTypeSeq = new DERSequence(ETSIQCObjectIdentifiers.id_etsi_qct_web);
      QCStatement qcType = new QCStatement(ETSIQCObjectIdentifiers.id_etsi_qcs_QcType,
 qcTypeSeq);

      ASN1EncodableVector pspAIVector = new ASN1EncodableVector(2);
```

```
      pspAIVector.add(new ASN1ObjectIdentifier("0.4.0.19495.1.3"));
      pspAIVector.add(new DERUTF8String("PSP_AI"));
      DERSequence pspAISeq = new DERSequence(pspAIVector);

      ASN1EncodableVector pspASVector = new ASN1EncodableVector(2);
      pspASVector.add(new ASN1ObjectIdentifier("0.4.0.19495.1.1"));
      pspASVector.add(new DERUTF8String("PSP_AS"));
      DERSequence pspASSeq = new DERSequence(pspASVector);

      ASN1EncodableVector pspPIVector = new ASN1EncodableVector(2);
      pspPIVector.add(new ASN1ObjectIdentifier("0.4.0.19495.1.2"));
      pspPIVector.add(new DERUTF8String("PSP_PI"));
      DERSequence pspPISeq = new DERSequence(pspPIVector);

      ASN1EncodableVector pspICVector = new ASN1EncodableVector(2);
      pspICVector.add(new ASN1ObjectIdentifier("0.4.0.19495.1.4"));
      pspICVector.add(new DERUTF8String("PSP_IC"));
      DERSequence pspICSeq = new DERSequence(pspICVector);

      ASN1EncodableVector pspSeqVector = new ASN1EncodableVector(4);
      pspSeqVector.add(pspPISeq);
      pspSeqVector.add(pspICSeq);
      pspSeqVector.add(pspASSeq);
      pspSeqVector.add(pspAISeq);
      DERSequence pspSeq = new DERSequence(pspSeqVector);

      ASN1EncodableVector pspVector = new ASN1EncodableVector(3);
      pspVector.add(pspSeq);
      pspVector.add(new DERUTF8String("Your Financial Authority"));
      pspVector.add(new DERUTF8String("AB-CD"));
      DERSequence psp = new DERSequence(pspVector);
      QCStatement qcPSP = new QCStatement(new ASN1ObjectIdentifier("0.4.0.19495.2"),
psp);

      DERSequence qcSeq = new DERSequence(new QCStatement[] { qcType, qcPSP });

      byte[] qcExtValueInBytes = qcSeq.getEncoded();
      return Base64.getEncoder().encodeToString(qcExtValueInBytes);
   }

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
```

```
      AWSCredentials credentials = null;
      try {
          credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
          throw new AmazonClientException("Cannot load your credentials from disk", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "us-west-2";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
          new AwsClientBuilder.EndpointConfiguration(endpointProtocol, endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
          .withEndpointConfiguration(endpoint)
          .withCredentials(new AWSStaticCredentialsProvider(credentials))
          .build();

      // Create a certificate request:
      IssueCertificateRequest req = new IssueCertificateRequest();

      // Set the CA ARN.
      req.withCertificateAuthorityArn("arn:aws:acm-pca:region:account:" +
         "certificate-authority/12345678-1234-1234-1234-123456789012");

      // Specify the certificate signing request (CSR) for the certificate to be signed
 and issued.
      String strCSR =
      "-----BEGIN CERTIFICATE REQUEST-----\n" +
      "base64-encoded CSR\n" +
      "-----END CERTIFICATE REQUEST-----\n";
      ByteBuffer csrByteBuffer = stringToByteBuffer(strCSR);
      req.setCsr(csrByteBuffer);

      // Specify the template for the issued certificate.
      req.withTemplateArn("arn:aws:acm-pca:::template/
EndEntityCertificate_APIPassthrough/V1");

      // Set the signing algorithm.
      req.withSigningAlgorithm(SigningAlgorithm.SHA256WITHRSA);
```

```
      // Set the validity period for the certificate to be issued.
      Validity validity = new Validity();
      validity.withValue(30L);
      validity.withType("DAYS");
      req.withValidity(validity);

      // Set the idempotency token.
      req.setIdempotencyToken("1234");

      // Generate Base64 encoded extension value for QC Statement
      String base64EncodedExtValue = generateQCStatementBase64ExtValue();

      // Generate custom extension
      CustomExtension customExtension = new CustomExtension();
      customExtension.setObjectIdentifier("1.3.6.1.5.5.7.1.3"); // QC Statement
Extension OID
      customExtension.setValue(base64EncodedExtValue);

      // Add custom extension to api-passthrough
      ApiPassthrough apiPassthrough = new ApiPassthrough();
      Extensions extensions = new Extensions();
      extensions.setCustomExtensions(Arrays.asList(customExtension));
      apiPassthrough.setExtensions(extensions);
      req.setApiPassthrough(apiPassthrough);

      // Issue the certificate.
      IssueCertificateResult result = null;
      try {
         result = client.issueCertificate(req);
      } catch (LimitExceededException ex) {
         throw ex;
      } catch (ResourceNotFoundException ex) {
         throw ex;
      } catch (InvalidStateException ex) {
         throw ex;
      } catch (InvalidArnException ex) {
         throw ex;
      } catch (InvalidArgsException ex) {
         throw ex;
      } catch (MalformedCSRException ex) {
         throw ex;
      }

      // Retrieve and display the certificate ARN.
```

```
        String arn = result.getCertificateArn();
        System.out.println(arn);
    }
}
```

使用 AWS 私有 CA 實作事項憑證

您可以使用 AWS 私有憑證授權單位 API 來建立符合事項連線標準的憑證。事項指定憑證組態，可改善跨多個工程平台的物聯網 (IoT) 裝置的安全性和一致性。如需關於事項的詳細資訊，請參閱 https：//buildwithmatter.com。

事項 1.2 於 2023 年 10 月發行，支援使用憑證撤銷清單 (CRLs) 的 DAC 撤銷。為了協助您符合目前的事項標準，當您為發出事項憑證CAs 啟用 CRL 撤銷時，請在 `CrlConfiguration` 物件的 `CrlDistributionPointExtensionConfiguration`結構中，`OmitExtension`將 設定為 `true`。

一般而言，CAs會將 CRL 分佈點 (CDP) 嵌入其發行的憑證中，以便執行憑證鏈驗證的依賴方可以擷取 CRL 並檢查憑證狀態。實際上，CDP URI 不會寫入憑證。反之，使用者會從可信任的事項分散式合規分類帳 (DCL) 資料存放區擷取 CDPs。您必須將 CDP URI 上傳到事項 DCL，以便在驗證 DACs時可以發現它。如需決定 CDP URI 的詳細資訊，請參閱 判斷 CRL 分佈點 (CDP) URI。如需有關事項的詳細資訊，請參閱事項標準首頁。

主題

- 啟用產品鑑證授權機構 (PAA)
- 啟用產品鑑證中級 (PAI)
- 建立裝置證明憑證 (DAC)
- 為節點操作憑證 (NOC) 啟用根 CA。
- 為節點操作憑證 (NOC) 啟用次級 CA
- 建立節點操作憑證 (NOC)

啟用產品鑑證授權機構 (PAA)

此 Java 範例示範如何使用 RootCACertificate_APIPassthrough/V1 定義 範本來建立和安裝適用於產品證明的事項根 CA (PAA) 憑證。對於 PAAs， AuthorityKeyIdentifier (AKI) 擴充功能是選用的。若要設定 AKI，您必須產生 Base64-encoded的 AKI 值，並透過 CustomExtension 傳遞。

此範例會呼叫下列 AWS 私有 CA API 動作：

- CreateCertificateAuthority
- GetCertificateAuthorityCsr
- IssueCertificate
- GetCertificate
- ImportCertificateAuthorityCertificate

如果您遇到問題，請參閱疑難排解一節故障診斷符合 AWS 私有 CA 事項的憑證錯誤中的 。

```
package com.amazonaws.samples.matter;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.samples.GetCertificateAuthorityCertificate;
import com.amazonaws.auth.AWSStaticCredentialsProvider;


import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.ASN1Subject;
import com.amazonaws.services.acmpca.model.ApiPassthrough;
import com.amazonaws.services.acmpca.model.CertificateAuthorityConfiguration;
import com.amazonaws.services.acmpca.model.CertificateAuthorityType;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityResult;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityRequest;
import com.amazonaws.services.acmpca.model.CrlConfiguration;
import com.amazonaws.services.acmpca.model.CustomAttribute;
import com.amazonaws.services.acmpca.model.CustomExtension;
import com.amazonaws.services.acmpca.model.Extensions;
import com.amazonaws.services.acmpca.model.KeyAlgorithm;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Tag;

import java.io.ByteArrayInputStream;
import java.io.InputStreamReader;
import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.Arrays;
```

```
import java.util.Base64;
import java.util.List;
import java.util.Objects;

import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrRequest;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrResult;
import com.amazonaws.services.acmpca.model.GetCertificateRequest;
import com.amazonaws.services.acmpca.model.GetCertificateResult;
import
 com.amazonaws.services.acmpca.model.ImportCertificateAuthorityCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateResult;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Validity;
import com.amazonaws.services.acmpca.model.RevocationConfiguration;
import com.amazonaws.services.acmpca.model.CrlConfiguration;
import com.amazonaws.services.acmpca.model.CrlDistributionPointExtensionConfiguration;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.CertificateMismatchException;
import com.amazonaws.services.acmpca.model.ConcurrentModificationException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidPolicyException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.MalformedCertificateException;
import com.amazonaws.services.acmpca.model.MalformedCSRException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.RequestInProgressException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.AWSACMPCAException;

import com.amazonaws.waiters.Waiter;
import com.amazonaws.waiters.WaiterParameters;
import com.amazonaws.waiters.WaiterTimedOutException;
import com.amazonaws.waiters.WaiterUnrecoverableException;

import org.bouncycastle.asn1.x509.SubjectPublicKeyInfo;
import org.bouncycastle.cert.jcajce.JcaX509ExtensionUtils;
import org.bouncycastle.openssl.PEMParser;
import org.bouncycastle.pkcs.PKCS10CertificationRequest;
import org.bouncycastle.util.io.pem.PemReader;
```

```
import lombok.SneakyThrows;

public class ProductAttestationAuthorityActivation {

    public static void main(String[] args) throws Exception {
        // Define the endpoint region for your sample.
        String endpointRegion = "region";  // Substitute your region here, e.g. "ap-
southeast-2"

        // Define custom attributes
        List<CustomAttribute> customAttributes = Arrays.asList(
            new CustomAttribute()
                .withObjectIdentifier("2.5.4.3") // CommonName
                .withValue("Matter Test PAA"),
            new CustomAttribute()
                .withObjectIdentifier("1.3.6.1.4.1.37244.2.1") // Vendor ID
                .withValue("FFF1")
        );

        // Define a CA subject.
        ASN1Subject subject = new ASN1Subject();
        subject.setCustomAttributes(customAttributes);

        // Define the CA configuration.
        CertificateAuthorityConfiguration configCA = new
 CertificateAuthorityConfiguration();
        configCA.withKeyAlgorithm(KeyAlgorithm.EC_prime256v1);
        configCA.withSigningAlgorithm(SigningAlgorithm.SHA256WITHECDSA);
        configCA.withSubject(subject);

        // Define a CRL distribution point extension configuration
        CrlDistributionPointExtensionConfiguration CDPConfigure = new
 CrlDistributionPointExtensionConfiguration();
        CDPConfigure.withOmitExtension(true);

        // Define a certificate revocation list configuration.
        CrlConfiguration crlConfigure = new CrlConfiguration();
        crlConfigure.withEnabled(true);
        crlConfigure.withExpirationInDays(365);
        crlConfigure.withCustomCname(null);
        crlConfigure.withS3BucketName("your-bucket-name");
        crlConfigure.withS3ObjectAcl("BUCKET_OWNER_FULL_CONTROL");
        crlConfigure.withCrlDistributionPointExtensionConfiguration(CDPConfigure);
```

```
        // Define a certificate authority type
        CertificateAuthorityType CAtype = CertificateAuthorityType.ROOT;

        // ** Execute core code samples for Root CA activation in sequence **
        AWSACMPCA client = ClientBuilder(endpointRegion);
        String rootCAArn = CreateCertificateAuthority(configCA, crlConfigure, CAtype,
client);
        String csr = GetCertificateAuthorityCsr(rootCAArn, client);
        String rootCertificateArn = IssueCertificate(rootCAArn, csr, client);
        String rootCertificate = GetCertificate(rootCertificateArn, rootCAArn, client);
        ImportCertificateAuthorityCertificate(rootCertificate, rootCAArn, client);
    }

    private static AWSACMPCA ClientBuilder(String endpointRegion) {
        // Retrieve your credentials from the C:\Users\name\.aws\credentials file
        // in Windows or the .aws/credentials file in Linux.
        AWSCredentials credentials = null;
        try {
            credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException(
                    "Cannot load the credentials from the credential profiles file. " +
                    "Please make sure that your credentials file is at the correct " +
                    "location (C:\\Users\\joneps\\.aws\\credentials), and is in valid
format.",
                    e);
        }

        String endpointProtocol = "https://acm-pca." + endpointRegion +
".amazonaws.com/";
        EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
endpointRegion);

        // Create a client that you can use to make requests.
        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
            .withEndpointConfiguration(endpoint)
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();

        return client;
    }
```

```
   private static String CreateCertificateAuthority(CertificateAuthorityConfiguration
configCA, CrlConfiguration crlConfigure, CertificateAuthorityType CAtype, AWSACMPCA
client) {
       RevocationConfiguration revokeConfig = new RevocationConfiguration();
       revokeConfig.setCrlConfiguration(crlConfigure);

       // Create the request object.
       CreateCertificateAuthorityRequest createCARequest = new
CreateCertificateAuthorityRequest();
       createCARequest.withCertificateAuthorityConfiguration(configCA);
       createCARequest.withIdempotencyToken("123987");
       createCARequest.withCertificateAuthorityType(CAtype);
       createCARequest.withRevocationConfiguration(revokeConfig);

       // Create the private CA.
       CreateCertificateAuthorityResult createCAResult = null;
       try {
           createCAResult = client.createCertificateAuthority(createCARequest);
       } catch (InvalidArgsException ex) {
           throw ex;
       } catch (InvalidPolicyException ex) {
           throw ex;
       } catch (LimitExceededException ex) {
           throw ex;
       }

       // Retrieve the ARN of the private CA.
       String rootCAArn = createCAResult.getCertificateAuthorityArn();
       System.out.println("Product Attestation Authority (PAA) Arn: " + rootCAArn);

       return rootCAArn;
   }

   private static String GetCertificateAuthorityCsr(String rootCAArn, AWSACMPCA
client) {

       // Create the CSR request object and set the CA ARN.
       GetCertificateAuthorityCsrRequest csrRequest = new
GetCertificateAuthorityCsrRequest();
       csrRequest.withCertificateAuthorityArn(rootCAArn);

       // Create waiter to wait on successful creation of the CSR file.
       Waiter<GetCertificateAuthorityCsrRequest> getCSRWaiter =
client.waiters().certificateAuthorityCSRCreated();
```

```
        try {
            getCSRWaiter.run(new WaiterParameters<>(csrRequest));
        } catch (WaiterUnrecoverableException e) {
            //Explicit short circuit when the recourse transitions into
            //an undesired state.
        } catch (WaiterTimedOutException e) {
            //Failed to transition into desired state even after polling.
        } catch (AWSACMPCAException e) {
            //Unexpected service exception.
        }

        // Retrieve the CSR.
        GetCertificateAuthorityCsrResult csrResult = null;
        try {
            csrResult = client.getCertificateAuthorityCsr(csrRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }

        // Retrieve and display the CSR;
        String csr = csrResult.getCsr();
        System.out.println(csr);

        return csr;
    }

    @SneakyThrows
    private static String generateAuthorityKeyIdentifier(final String csrPEM) {
        PKCS10CertificationRequest csr = getPKCS10CertificationRequest(csrPEM);
        SubjectPublicKeyInfo spki = csr.getSubjectPublicKeyInfo();

        JcaX509ExtensionUtils extensionUtils = new JcaX509ExtensionUtils();
        byte[] akiBytes =
extensionUtils.createAuthorityKeyIdentifier(spki).getEncoded();

        return Base64.getEncoder().encodeToString(akiBytes);
    }
```

```
    @SneakyThrows
    private static PKCS10CertificationRequest getPKCS10CertificationRequest(final
 String csrPEM) {
        ByteArrayInputStream bais = new ByteArrayInputStream(csrPEM.getBytes());
        PemReader pemReader = new PemReader(new InputStreamReader(bais));
        PEMParser parser = new PEMParser(pemReader);
        Object o = parser.readObject();
        if (o instanceof PKCS10CertificationRequest) {
            return (PKCS10CertificationRequest) o;
        }
        return null;
    }

    private static String IssueCertificate(String rootCAArn, String csr, AWSACMPCA
 client) {

        // Create a certificate request:
        IssueCertificateRequest issueRequest = new IssueCertificateRequest();

        // Set the CA ARN.
        issueRequest.withCertificateAuthorityArn(rootCAArn);

        // Set the template ARN.
        issueRequest.withTemplateArn("arn:aws:acm-pca:::template/
RootCACertificate_APIPassthrough/V1");

        ByteBuffer csrByteBuffer = stringToByteBuffer(csr);
        issueRequest.setCsr(csrByteBuffer);

        // Set the signing algorithm.
        issueRequest.withSigningAlgorithm(SigningAlgorithm.SHA256WITHECDSA);

        // Set the validity period for the certificate to be issued.
        Validity validity = new Validity();
        validity.withValue(3650L);
        validity.withType("DAYS");
        issueRequest.withValidity(validity);

        // Set the idempotency token.
        issueRequest.setIdempotencyToken("1234");

        // Generate Base64 encoded extension value for AuthorityKeyIdentifier
        String base64EncodedExtValue = generateAuthorityKeyIdentifier(csr);
```

```
        // Generate custom extension
        CustomExtension customExtension = new CustomExtension();
        customExtension.setObjectIdentifier("2.5.29.35"); // AuthorityKeyIdentifier
Extension OID
        customExtension.setValue(base64EncodedExtValue);

        // Add custom extension to api-passthrough
        ApiPassthrough apiPassthrough = new ApiPassthrough();
        Extensions extensions = new Extensions();
        extensions.setCustomExtensions(Arrays.asList(customExtension));
        apiPassthrough.setExtensions(extensions);
        issueRequest.setApiPassthrough(apiPassthrough);

        // Issue the certificate.
        IssueCertificateResult issueResult = null;
        try {
            issueResult = client.issueCertificate(issueRequest);
        } catch (LimitExceededException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (MalformedCSRException ex) {
            throw ex;
        }

        // Retrieve and display the certificate ARN.
        String rootCertificateArn = issueResult.getCertificateArn();
        System.out.println("Product Attestation Authority (PAA) Certificate Arn: " +
rootCertificateArn);

        return rootCertificateArn;
    }

    private static String GetCertificate(String rootCertificateArn, String rootCAArn,
AWSACMPCA client) {

        // Create a request object.
        GetCertificateRequest certificateRequest = new GetCertificateRequest();
```

```
        // Set the certificate ARN.
        certificateRequest.withCertificateArn(rootCertificateArn);

        // Set the certificate authority ARN.
        certificateRequest.withCertificateAuthorityArn(rootCAArn);

        // Create waiter to wait on successful creation of the certificate file.
        Waiter<GetCertificateRequest> getCertificateWaiter =
client.waiters().certificateIssued();
        try {
            getCertificateWaiter.run(new WaiterParameters<>(certificateRequest));
        } catch (WaiterUnrecoverableException e) {
            //Explicit short circuit when the recourse transitions into
            //an undesired state.
        } catch (WaiterTimedOutException e) {
            //Failed to transition into desired state even after polling.
        } catch (AWSACMPCAException e) {
            //Unexpected service exception.
        }

        // Retrieve the certificate and certificate chain.
        GetCertificateResult certificateResult = null;
        try {
            certificateResult = client.getCertificate(certificateRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        }

        // Get the certificate and certificate chain and display the result.
        String rootCertificate = certificateResult.getCertificate();
        System.out.println(rootCertificate);

        return rootCertificate;
    }
```

```
   private static void ImportCertificateAuthorityCertificate(String rootCertificate,
String rootCAArn, AWSACMPCA client) {

       // Create the request object and set the signed certificate, chain and CA ARN.
       ImportCertificateAuthorityCertificateRequest importRequest =
           new ImportCertificateAuthorityCertificateRequest();

       ByteBuffer certByteBuffer = stringToByteBuffer(rootCertificate);
       importRequest.setCertificate(certByteBuffer);

       importRequest.setCertificateChain(null);

       // Set the certificate authority ARN.
       importRequest.withCertificateAuthorityArn(rootCAArn);

       // Import the certificate.
       try {
           client.importCertificateAuthorityCertificate(importRequest);
       } catch (CertificateMismatchException ex) {
           throw ex;
       } catch (MalformedCertificateException ex) {
           throw ex;
       } catch (InvalidArnException ex) {
           throw ex;
       } catch (ResourceNotFoundException ex) {
           throw ex;
       } catch (RequestInProgressException ex) {
           throw ex;
       } catch (ConcurrentModificationException ex) {
           throw ex;
       } catch (RequestFailedException ex) {
           throw ex;
       }

       System.out.println("Product Attestation Authority (PAA) certificate
successfully imported.");
       System.out.println("Product Attestation Authority (PAA) activated
successfully.");
   }

   private static ByteBuffer stringToByteBuffer(final String string) {
       if (Objects.isNull(string)) {
           return null;
       }
```

```
        byte[] bytes = string.getBytes(StandardCharsets.UTF_8);
        return ByteBuffer.wrap(bytes);
    }
}
```

啟用產品鑑證中級 (PAI)

此 Java 範例示範如何使用 BlankSubordinateCACertificate_PathLen0_APIPassthrough/V1 定義 範本來建立和安裝適用於產品證明的事項次級 CA (PAI) 憑證。您必須產生 Base64-encoded的 KeyUsage 值，並透過 CustomExtension 傳遞。

此範例會呼叫下列 AWS 私有 CA API 動作：

- CreateCertificateAuthority
- GetCertificateAuthorityCsr
- IssueCertificate
- GetCertificate
- ImportCertificateAuthorityCertificate
- GetCertificateAuthorityCertificate

如果您遇到問題，請參閱疑難排解一節故障診斷符合 AWS 私有 CA 事項的憑證錯誤中的 。

```
package com.amazonaws.samples.matter;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;


import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.ASN1Subject;
import com.amazonaws.services.acmpca.model.ApiPassthrough;
import com.amazonaws.services.acmpca.model.CertificateAuthorityConfiguration;
import com.amazonaws.services.acmpca.model.CertificateAuthorityType;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityResult;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityRequest;
```

```
import com.amazonaws.services.acmpca.model.CustomAttribute;
import com.amazonaws.services.acmpca.model.CustomExtension;
import com.amazonaws.services.acmpca.model.Extensions;
import com.amazonaws.services.acmpca.model.KeyAlgorithm;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.RevocationConfiguration;
import com.amazonaws.services.acmpca.model.CrlConfiguration;
import com.amazonaws.services.acmpca.model.CrlDistributionPointExtensionConfiguration;

import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.Arrays;
import java.util.Base64;
import java.util.List;
import java.util.Objects;

import org.bouncycastle.asn1.x509.KeyUsage;
import org.bouncycastle.jce.X509KeyUsage;

import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCertificateRequest;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCertificateResult;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrRequest;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrResult;
import com.amazonaws.services.acmpca.model.GetCertificateRequest;
import com.amazonaws.services.acmpca.model.GetCertificateResult;
import
 com.amazonaws.services.acmpca.model.ImportCertificateAuthorityCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateResult;
import com.amazonaws.services.acmpca.model.Validity;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.CertificateMismatchException;
import com.amazonaws.services.acmpca.model.ConcurrentModificationException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidPolicyException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.MalformedCertificateException;
import com.amazonaws.services.acmpca.model.MalformedCSRException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.RequestInProgressException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
```

```
import com.amazonaws.services.acmpca.model.AWSACMPCAException;

import com.amazonaws.waiters.Waiter;
import com.amazonaws.waiters.WaiterParameters;
import com.amazonaws.waiters.WaiterTimedOutException;
import com.amazonaws.waiters.WaiterUnrecoverableException;

import lombok.SneakyThrows;

public class ProductAttestationIntermediateActivation {

    public static void main(String[] args) throws Exception {
        // Place your own Root CA ARN here.
        String paaArn = "arn:aws:acm-pca:region:123456789012:certificate-
authority/12345678-1234-1234-1234-123456789012";

        // Define the endpoint region for your sample.
        String endpointRegion = "region";  // Substitute your region here, e.g. "ap-
southeast-2"

        // Define custom attributes
        List<CustomAttribute> customAttributes = Arrays.asList(
            new CustomAttribute()
                .withObjectIdentifier("2.5.4.3") // CommonName
                .withValue("Matter Test PAI"),
            new CustomAttribute()
                .withObjectIdentifier("1.3.6.1.4.1.37244.2.1") // Vendor ID
                .withValue("FFF1"),
            new CustomAttribute()
                .withObjectIdentifier("1.3.6.1.4.1.37244.2.2") // Product ID
                .withValue("8000")
        );

        // Define a CA subject.
        ASN1Subject subject = new ASN1Subject();
        subject.setCustomAttributes(customAttributes);

        // Define the CA configuration.
        CertificateAuthorityConfiguration configCA = new
 CertificateAuthorityConfiguration();
        configCA.withKeyAlgorithm(KeyAlgorithm.EC_prime256v1);
        configCA.withSigningAlgorithm(SigningAlgorithm.SHA256WITHECDSA);
        configCA.withSubject(subject);
```

```
        // Define a CRL distribution point extension configuration
        CrlDistributionPointExtensionConfiguration CDPConfigure = new
CrlDistributionPointExtensionConfiguration();
        CDPConfigure.withOmitExtension(true);

        // Define a certificate revocation list configuration.
        CrlConfiguration crlConfigure = new CrlConfiguration();
        crlConfigure.withEnabled(true);
        crlConfigure.withExpirationInDays(365);
        crlConfigure.withCustomCname(null);
        crlConfigure.withS3BucketName("your-bucket-name");
        crlConfigure.withS3ObjectAcl("BUCKET_OWNER_FULL_CONTROL");
        crlConfigure.withCrlDistributionPointConfiguration(CDPConfigure);

        // Define a certificate authority type
        CertificateAuthorityType CAtype = CertificateAuthorityType.SUBORDINATE;

        // ** Execute core code samples for Subordinate CA activation in sequence **
        AWSACMPCA client = ClientBuilder(endpointRegion);
        String rootCertificate = GetCertificateAuthorityCertificate(paaArn, client);
        String subordinateCAArn = CreateCertificateAuthority(configCA, crlConfigure,
CAtype, client);
        String csr = GetCertificateAuthorityCsr(subordinateCAArn, client);
        String subordinateCertificateArn = IssueCertificate(paaArn, csr, client);
        String subordinateCertificate = GetCertificate(subordinateCertificateArn,
paaArn, client);
        ImportCertificateAuthorityCertificate(subordinateCertificate, rootCertificate,
subordinateCAArn, client);

    }

    private static AWSACMPCA ClientBuilder(String endpointRegion) {
        // Retrieve your credentials from the C:\Users\name\.aws\credentials file
        // in Windows or the .aws/credentials file in Linux.
        AWSCredentials credentials = null;
        try {
            credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException(
                    "Cannot load the credentials from the credential profiles file. " +
                    "Please make sure that your credentials file is at the correct " +
                    "location (C:\\Users\\joneps\\.aws\\credentials), and is in valid
format.",
                    e);
```

```
        }

        String endpointProtocol = "https://acm-pca." + endpointRegion +
".amazonaws.com/";
        EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
endpointRegion);

        // Create a client that you can use to make requests.
        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
            .withEndpointConfiguration(endpoint)
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();

        return client;
    }

    private static String GetCertificateAuthorityCertificate(String rootCAArn,
AWSACMPCA client) {
        // ** GetCertificateAuthorityCertificate **

        // Create a request object and set the certificate authority ARN,
        GetCertificateAuthorityCertificateRequest getCACertificateRequest =
        new GetCertificateAuthorityCertificateRequest();
        getCACertificateRequest.withCertificateAuthorityArn(rootCAArn);

        // Create a result object.
        GetCertificateAuthorityCertificateResult getCACertificateResult = null;
        try {
            getCACertificateResult =
client.getCertificateAuthorityCertificate(getCACertificateRequest);
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        }

        // Retrieve and display the certificate information.
        String rootCertificate = getCACertificateResult.getCertificate();
        System.out.println("Product Attestation Authority (PAA) Certificate /
Certificate Chain:");
        System.out.println(rootCertificate);
```

```
        return rootCertificate;
    }

    private static String CreateCertificateAuthority(CertificateAuthorityConfiguration
configCA, CrlConfiguration crlConfigure, CertificateAuthorityType CAtype, AWSACMPCA
client) {
        RevocationConfiguration revokeConfig = new RevocationConfiguration();
        revokeConfig.setCrlConfiguration(crlConfigure);

        // Create the request object.
        CreateCertificateAuthorityRequest createCARequest = new
CreateCertificateAuthorityRequest();
        createCARequest.withCertificateAuthorityConfiguration(configCA);
        createCARequest.withIdempotencyToken("123987");
        createCARequest.withCertificateAuthorityType(CAtype);
        createCARequest.withRevocationConfiguration(revokeConfig);

        // Create the private CA.
        CreateCertificateAuthorityResult createCAResult = null;
        try {
            createCAResult = client.createCertificateAuthority(createCARequest);
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (InvalidPolicyException ex) {
            throw ex;
        } catch (LimitExceededException ex) {
            throw ex;
        }

        // Retrieve the ARN of the private CA.
        String subordinateCAArn = createCAResult.getCertificateAuthorityArn();
        System.out.println("Product Attestation Intermediate (PAI) Arn: " +
subordinateCAArn);

        return subordinateCAArn;
    }

    private static String GetCertificateAuthorityCsr(String subordinateCAArn, AWSACMPCA
client) {

        // Create the CSR request object and set the CA ARN.
        GetCertificateAuthorityCsrRequest csrRequest = new
GetCertificateAuthorityCsrRequest();
```

```
        csrRequest.withCertificateAuthorityArn(subordinateCAArn);

        // Create waiter to wait on successful creation of the CSR file.
        Waiter<GetCertificateAuthorityCsrRequest> getCSRWaiter =
client.waiters().certificateAuthorityCSRCreated();
        try {
            getCSRWaiter.run(new WaiterParameters<>(csrRequest));
        } catch (WaiterUnrecoverableException e) {
            //Explicit short circuit when the recourse transitions into
            //an undesired state.
        } catch (WaiterTimedOutException e) {
            //Failed to transition into desired state even after polling.
        } catch(AWSACMPCAException e) {
            //Unexpected service exception.
        }

        // Retrieve the CSR.
        GetCertificateAuthorityCsrResult csrResult = null;
        try {
            csrResult = client.getCertificateAuthorityCsr(csrRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }

        // Retrieve and display the CSR;
        String csr = csrResult.getCsr();
        System.out.println("Subordinate CSR:");
        System.out.println(csr);

        return csr;
    }

    private static String IssueCertificate(String rootCAArn, String csr, AWSACMPCA
client) {

        // Create a certificate request:
        IssueCertificateRequest issueRequest = new IssueCertificateRequest();
```

```
        // Set the issuing CA ARN.
        issueRequest.withCertificateAuthorityArn(rootCAArn);

        // Set the template ARN.
        issueRequest.withTemplateArn("arn:aws:acm-pca:::template/
BlankSubordinateCACertificate_PathLen0_APIPassthrough/V1");

        ByteBuffer csrByteBuffer = stringToByteBuffer(csr);
        issueRequest.setCsr(csrByteBuffer);

        // Set the signing algorithm.
        issueRequest.withSigningAlgorithm(SigningAlgorithm.SHA256WITHECDSA);

        // Set the validity period for the certificate to be issued.
        Validity validity = new Validity();
        validity.withValue(730L); // Approximately two years
        validity.withType("DAYS");
        issueRequest.withValidity(validity);

        // Set the idempotency token.
        issueRequest.setIdempotencyToken("1234");

        ApiPassthrough apiPassthrough = new ApiPassthrough();

        // Generate Base64 encoded extension value for ExtendedKeyUsage
        String base64EncodedKUValue = generateKeyUsageValue();

        // Generate custom extension
        CustomExtension customKeyUsageExtension = new CustomExtension();
        customKeyUsageExtension.setObjectIdentifier("2.5.29.15");
        customKeyUsageExtension.setValue(base64EncodedKUValue);
        customKeyUsageExtension.setCritical(true);

        // Set KeyUsage extension to api passthrough
        Extensions extensions = new Extensions();
        extensions.setCustomExtensions(Arrays.asList(customKeyUsageExtension));
        apiPassthrough.setExtensions(extensions);
        issueRequest.setApiPassthrough(apiPassthrough);

        // Issue the certificate.
        IssueCertificateResult issueResult = null;
        try {
            issueResult = client.issueCertificate(issueRequest);
        } catch (LimitExceededException ex) {
```

```
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (MalformedCSRException ex) {
            throw ex;
        }

        // Retrieve and display the certificate ARN.
        String subordinateCertificateArn = issueResult.getCertificateArn();
        System.out.println("Subordinate Certificate Arn: " +
subordinateCertificateArn);

        return subordinateCertificateArn;
    }

    @SneakyThrows
    private static String generateKeyUsageValue() {
       KeyUsage keyUsage = new KeyUsage(X509KeyUsage.keyCertSign |
X509KeyUsage.cRLSign);
       byte[] kuBytes = keyUsage.getEncoded();
       return Base64.getEncoder().encodeToString(kuBytes);
    }

    private static String GetCertificate(String subordinateCertificateArn, String
rootCAArn, AWSACMPCA client) {

        // Create a request object.
        GetCertificateRequest certificateRequest = new GetCertificateRequest();

        // Set the certificate ARN.
        certificateRequest.withCertificateArn(subordinateCertificateArn);

        // Set the certificate authority ARN.
        certificateRequest.withCertificateAuthorityArn(rootCAArn);

        // Create waiter to wait on successful creation of the certificate file.
        Waiter<GetCertificateRequest> getCertificateWaiter =
client.waiters().certificateIssued();
```

```
        try {
            getCertificateWaiter.run(new WaiterParameters<>(certificateRequest));
        } catch (WaiterUnrecoverableException e) {
            //Explicit short circuit when the recourse transitions into
            //an undesired state.
        } catch (WaiterTimedOutException e) {
            //Failed to transition into desired state even after polling.
        } catch (AWSACMPCAException e) {
            //Unexpected service exception.
        }

        // Retrieve the certificate and certificate chain.
        GetCertificateResult certificateResult = null;
        try {
            certificateResult = client.getCertificate(certificateRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        }

        // Get the certificate and certificate chain and display the result.
        String subordinateCertificate = certificateResult.getCertificate();
        System.out.println("Subordinate CA Certificate:");
        System.out.println(subordinateCertificate);

        return subordinateCertificate;
    }

    private static void ImportCertificateAuthorityCertificate(String
subordinateCertificate, String rootCertificate, String subordinateCAArn, AWSACMPCA
client) {

        // Create the request object and set the signed certificate, chain and CA ARN.
        ImportCertificateAuthorityCertificateRequest importRequest =
            new ImportCertificateAuthorityCertificateRequest();

        ByteBuffer certByteBuffer = stringToByteBuffer(subordinateCertificate);
```

```
        importRequest.setCertificate(certByteBuffer);

        ByteBuffer rootCACertByteBuffer = stringToByteBuffer(rootCertificate);
        importRequest.setCertificateChain(rootCACertByteBuffer);

        // Set the certificate authority ARN.
        importRequest.withCertificateAuthorityArn(subordinateCAArn);

        // Import the certificate.
        try {
            client.importCertificateAuthorityCertificate(importRequest);
        } catch (CertificateMismatchException ex) {
            throw ex;
        } catch (MalformedCertificateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (ConcurrentModificationException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }
        System.out.println("Product Attestation Intermediate (PAI) certificate
 successfully imported.");
        System.out.println("Product Attestation Intermediate (PAI) activated
 successfully.");
    }

    private static ByteBuffer stringToByteBuffer(final String string) {
        if (Objects.isNull(string)) {
            return null;
        }
        byte[] bytes = string.getBytes(StandardCharsets.UTF_8);
        return ByteBuffer.wrap(bytes);
    }
}
```

建立裝置證明憑證 (DAC)

此 Java 範例說明如何使用 BlankEndEntityCertificate_CriticalBasicConstraints_APIPassthrough/V1 範本來建立事項裝置證明憑證。您必須產生 Base64-encoded的 KeyUsage 值，並透過 CustomExtension 傳遞。

此範例會呼叫下列 AWS 私有 CA API 動作：

- IssueCertificate

如果您遇到問題，請參閱疑難排解一節故障診斷符合 AWS 私有 CA 事項的憑證錯誤中的 。

```
package com.amazonaws.samples.matter;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.Arrays;
import java.util.Base64;
import java.util.List;
import java.util.Objects;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.ASN1Subject;
import com.amazonaws.services.acmpca.model.ApiPassthrough;
import com.amazonaws.services.acmpca.model.CustomAttribute;
import com.amazonaws.services.acmpca.model.CustomExtension;
import com.amazonaws.services.acmpca.model.Extensions;
import com.amazonaws.services.acmpca.model.IssueCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateResult;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Validity;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
```

```
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.MalformedCSRException;

import org.bouncycastle.asn1.x509.KeyUsage;
import org.bouncycastle.jce.X509KeyUsage;

import lombok.SneakyThrows;

public class IssueDeviceAttestationCertificate {
   public static ByteBuffer stringToByteBuffer(final String string) {
      if (Objects.isNull(string)) {
          return null;
      }
      byte[] bytes = string.getBytes(StandardCharsets.UTF_8);
      return ByteBuffer.wrap(bytes);
   }

   @SneakyThrows
   private static String generateKeyUsageValue() {
      KeyUsage keyUsage = new KeyUsage(X509KeyUsage.digitalSignature);
      byte[] kuBytes = keyUsage.getEncoded();
      return Base64.getEncoder().encodeToString(kuBytes);
   }

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
          credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
          throw new AmazonClientException("Cannot load your credentials from disk", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "ap-
southeast-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
```

```
          new AwsClientBuilder.EndpointConfiguration(endpointProtocol, endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
          .withEndpointConfiguration(endpoint)
          .withCredentials(new AWSStaticCredentialsProvider(credentials))
          .build();

      // Create a certificate request:
      IssueCertificateRequest req = new IssueCertificateRequest();

      // Set the CA ARN.
      req.withCertificateAuthorityArn("arn:aws:acm-pca:region:123456789012:certificate-
authority/12345678-1234-1234-1234-123456789012");

      // Specify the certificate signing request (CSR) for the certificate to be signed
 and issued.
      String strCSR =
      "-----BEGIN CERTIFICATE REQUEST-----\n" +
      "base64-encoded certificate\n" +
      "-----END CERTIFICATE REQUEST-----\n";
      ByteBuffer csrByteBuffer = stringToByteBuffer(strCSR);
      req.setCsr(csrByteBuffer);

      // Specify the template for the issued certificate.
      req.withTemplateArn("arn:aws:acm-pca:::template/
BlankEndEntityCertificate_CriticalBasicConstraints_APIPassthrough/V1");

      // Set the signing algorithm.
      req.withSigningAlgorithm(SigningAlgorithm.SHA256WITHECDSA);

      // Set the validity period for the certificate to be issued.
      Validity validity = new Validity();
      validity.withValue(10L);
      validity.withType("DAYS");
      req.withValidity(validity);

      // Set the idempotency token.
      req.setIdempotencyToken("1234");

      // Define custom attributes
      List<CustomAttribute> customAttributes = Arrays.asList(
         new CustomAttribute()
            .withObjectIdentifier("2.5.4.3")
```

```
            .withValue("Matter Test DAC 0001"),
         new CustomAttribute()
            .withObjectIdentifier("1.3.6.1.4.1.37244.2.1")
            .withValue("FFF1"),
         new CustomAttribute()
            .withObjectIdentifier("1.3.6.1.4.1.37244.2.2")
            .withValue("8000")
      );

      // Define a cert subject.
      ASN1Subject subject = new ASN1Subject();
      subject.setCustomAttributes(customAttributes);

      ApiPassthrough apiPassthrough = new ApiPassthrough();
      apiPassthrough.setSubject(subject);

      // Generate Base64 encoded extension value for ExtendedKeyUsage
      String base64EncodedKUValue = generateKeyUsageValue();

      // Generate custom extension
      CustomExtension customKeyUsageExtension = new CustomExtension();
      customKeyUsageExtension.setObjectIdentifier("2.5.29.15"); // KeyUsage Extension
OID
      customKeyUsageExtension.setValue(base64EncodedKUValue);
      customKeyUsageExtension.setCritical(true);

      Extensions extensions = new Extensions();
      extensions.setCustomExtensions(Arrays.asList(customKeyUsageExtension));
      apiPassthrough.setExtensions(extensions);
      req.setApiPassthrough(apiPassthrough);

      // Issue the certificate.
      IssueCertificateResult result = null;
      try {
         result = client.issueCertificate(req);
      } catch (LimitExceededException ex) {
         throw ex;
      } catch (ResourceNotFoundException ex) {
         throw ex;
      } catch (InvalidStateException ex) {
         throw ex;
      } catch (InvalidArnException ex) {
         throw ex;
      } catch (InvalidArgsException ex) {
```

```
            throw ex;
        } catch (MalformedCSRException ex) {
            throw ex;
        }

        // Retrieve and display the certificate ARN.
        String arn = result.getCertificateArn();
        System.out.println(arn);
    }
}
```

為節點操作憑證 (NOC) 啟用根 CA。

此 Java 範例示範如何使用 RootCACertificate_APIPassthrough/V1 定義 範本建立和安裝事項根 CA 憑證來發行 NOCs。對於 NOC 根 CA 憑證， AuthorityKeyIdentifier (AKI) 擴充功能是選用的。若要設定 AKI，您必須產生 Base64-encoded的 AKI 值，並透過 CustomExtension 傳遞。

此範例會呼叫下列 AWS 私有 CA API 動作：

- CreateCertificateAuthority
- GetCertificateAuthorityCsr
- IssueCertificate
- GetCertificate
- ImportCertificateAuthorityCertificate

如果您遇到問題，請參閱疑難排解一節故障診斷符合 AWS 私有 CA 事項的憑證錯誤中的 。

```
package com.amazonaws.samples.matter;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.samples.GetCertificateAuthorityCertificate;
import com.amazonaws.auth.AWSStaticCredentialsProvider;


import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;
```

```
import com.amazonaws.services.acmpca.model.ASN1Subject;
import com.amazonaws.services.acmpca.model.ApiPassthrough;
import com.amazonaws.services.acmpca.model.CertificateAuthorityConfiguration;
import com.amazonaws.services.acmpca.model.CertificateAuthorityType;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityResult;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityRequest;
import com.amazonaws.services.acmpca.model.CrlConfiguration;
import com.amazonaws.services.acmpca.model.CustomAttribute;
import com.amazonaws.services.acmpca.model.CustomExtension;
import com.amazonaws.services.acmpca.model.Extensions;
import com.amazonaws.services.acmpca.model.KeyAlgorithm;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Tag;

import java.io.ByteArrayInputStream;
import java.io.InputStreamReader;
import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Base64;
import java.util.List;
import java.util.Objects;

import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrRequest;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrResult;
import com.amazonaws.services.acmpca.model.GetCertificateRequest;
import com.amazonaws.services.acmpca.model.GetCertificateResult;
import
 com.amazonaws.services.acmpca.model.ImportCertificateAuthorityCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateResult;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Validity;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.CertificateMismatchException;
import com.amazonaws.services.acmpca.model.ConcurrentModificationException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidPolicyException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.MalformedCertificateException;
```

```
import com.amazonaws.services.acmpca.model.MalformedCSRException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.RequestInProgressException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.RevocationConfiguration;
import com.amazonaws.services.acmpca.model.AWSACMPCAException;

import com.amazonaws.waiters.Waiter;
import com.amazonaws.waiters.WaiterParameters;
import com.amazonaws.waiters.WaiterTimedOutException;
import com.amazonaws.waiters.WaiterUnrecoverableException;

import org.bouncycastle.asn1.x509.SubjectPublicKeyInfo;
import org.bouncycastle.cert.jcajce.JcaX509ExtensionUtils;
import org.bouncycastle.openssl.PEMParser;
import org.bouncycastle.pkcs.PKCS10CertificationRequest;
import org.bouncycastle.util.io.pem.PemReader;

import lombok.SneakyThrows;

public class RootCAActivation {
    public static void main(String[] args) throws Exception {
        // Define the endpoint region for your sample.
        String endpointRegion = "region";  // Substitute your region here, e.g. "ap-
southeast-2"

        // Define custom attributes
        List<CustomAttribute> customAttributes = Arrays.asList(
            new CustomAttribute()
                .withObjectIdentifier("1.3.6.1.4.1.37244.1.4")
                .withValue("CACACACA00000001")
        );

        // Define a CA subject.
        ASN1Subject subject = new ASN1Subject();
        subject.setCustomAttributes(customAttributes);

        // Define the CA configuration.
        CertificateAuthorityConfiguration configCA = new
 CertificateAuthorityConfiguration();
        configCA.withKeyAlgorithm(KeyAlgorithm.EC_prime256v1);
        configCA.withSigningAlgorithm(SigningAlgorithm.SHA256WITHECDSA);
        configCA.withSubject(subject);
```

```
        // Define a certificate authority type
        CertificateAuthorityType CAtype = CertificateAuthorityType.ROOT;

        // ** Execute core code samples for Root CA activation in sequence **
        AWSACMPCA client = ClientBuilder(endpointRegion);
        String rootCAArn = CreateCertificateAuthority(configCA, CAtype, client);
        String csr = GetCertificateAuthorityCsr(rootCAArn, client);
        String rootCertificateArn = IssueCertificate(rootCAArn, csr, client);
        String rootCertificate = GetCertificate(rootCertificateArn, rootCAArn, client);
        ImportCertificateAuthorityCertificate(rootCertificate, rootCAArn, client);
    }

    private static AWSACMPCA ClientBuilder(String endpointRegion) {
        // Retrieve your credentials from the C:\Users\name\.aws\credentials file
        // in Windows or the .aws/credentials file in Linux.
        AWSCredentials credentials = null;
        try {
            credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException(
                    "Cannot load the credentials from the credential profiles file. " +
                    "Please make sure that your credentials file is at the correct " +
                    "location (C:\\Users\\joneps\\.aws\\credentials), and is in valid
format.",
                    e);
        }

        String endpointProtocol = "https://acm-pca." + endpointRegion +
".amazonaws.com/";
        EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
endpointRegion);

        // Create a client that you can use to make requests.
        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
            .withEndpointConfiguration(endpoint)
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();

        return client;
    }

    private static String CreateCertificateAuthority(CertificateAuthorityConfiguration
configCA, CertificateAuthorityType CAtype, AWSACMPCA client) {
```

```
        // Create the request object.
        CreateCertificateAuthorityRequest createCARequest = new
CreateCertificateAuthorityRequest();
        createCARequest.withCertificateAuthorityConfiguration(configCA);
        createCARequest.withIdempotencyToken("123987");
        createCARequest.withCertificateAuthorityType(CAtype);

        // Create the private CA.
        CreateCertificateAuthorityResult createCAResult = null;
        try {
            createCAResult = client.createCertificateAuthority(createCARequest);
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (InvalidPolicyException ex) {
            throw ex;
        } catch (LimitExceededException ex) {
            throw ex;
        }

        // Retrieve the ARN of the private CA.
        String rootCAArn = createCAResult.getCertificateAuthorityArn();
        System.out.println("Root CA Arn: " + rootCAArn);

        return rootCAArn;
    }

    private static String GetCertificateAuthorityCsr(String rootCAArn, AWSACMPCA
client) {

        // Create the CSR request object and set the CA ARN.
        GetCertificateAuthorityCsrRequest csrRequest = new
GetCertificateAuthorityCsrRequest();
        csrRequest.withCertificateAuthorityArn(rootCAArn);

        // Create waiter to wait on successful creation of the CSR file.
        Waiter<GetCertificateAuthorityCsrRequest> getCSRWaiter =
client.waiters().certificateAuthorityCSRCreated();
        try {
            getCSRWaiter.run(new WaiterParameters<>(csrRequest));
        } catch (WaiterUnrecoverableException e) {
            //Explicit short circuit when the recourse transitions into
            //an undesired state.
        } catch (WaiterTimedOutException e) {
            //Failed to transition into desired state even after polling.
```

```
        } catch (AWSACMPCAException e) {
            //Unexpected service exception.
        }

        // Retrieve the CSR.
        GetCertificateAuthorityCsrResult csrResult = null;
        try {
            csrResult = client.getCertificateAuthorityCsr(csrRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }

        // Retrieve and display the CSR;
        String csr = csrResult.getCsr();
        System.out.println(csr);

        return csr;
    }

    @SneakyThrows
    private static String generateAuthorityKeyIdentifier(final String csrPEM) {
        PKCS10CertificationRequest csr = getPKCS10CertificationRequest(csrPEM);
        SubjectPublicKeyInfo spki = csr.getSubjectPublicKeyInfo();

        JcaX509ExtensionUtils extensionUtils = new JcaX509ExtensionUtils();
        byte[] akiBytes =
extensionUtils.createAuthorityKeyIdentifier(spki).getEncoded();

        return Base64.getEncoder().encodeToString(akiBytes);
    }

    @SneakyThrows
    private static PKCS10CertificationRequest getPKCS10CertificationRequest(final
String csrPEM) {
        ByteArrayInputStream bais = new ByteArrayInputStream(csrPEM.getBytes());
        PemReader pemReader = new PemReader(new InputStreamReader(bais));
        PEMParser parser = new PEMParser(pemReader);
        Object o = parser.readObject();
```

```
        if (o instanceof PKCS10CertificationRequest) {
            return (PKCS10CertificationRequest) o;
        }
        return null;
    }

    private static String IssueCertificate(String rootCAArn, String csr, AWSACMPCA
 client) {

        // Create a certificate request:
        IssueCertificateRequest issueRequest = new IssueCertificateRequest();

        // Set the CA ARN.
        issueRequest.withCertificateAuthorityArn(rootCAArn);

        // Set the template ARN.
        issueRequest.withTemplateArn("arn:aws:acm-pca:::template/
RootCACertificate_APIPassthrough/V1");

        ByteBuffer csrByteBuffer = stringToByteBuffer(csr);
        issueRequest.setCsr(csrByteBuffer);

        // Set the signing algorithm.
        issueRequest.withSigningAlgorithm(SigningAlgorithm.SHA256WITHECDSA);

        // Set the validity period for the certificate to be issued.
        Validity validity = new Validity();
        validity.withValue(3650L);
        validity.withType("DAYS");
        issueRequest.withValidity(validity);

        // Set the idempotency token.
        issueRequest.setIdempotencyToken("1234");

        // Generate Base64 encoded extension value for AuthorityKeyIdentifier
        String base64EncodedExtValue = generateAuthorityKeyIdentifier(csr);

        // Generate custom extension
        CustomExtension customExtension = new CustomExtension();
        customExtension.setObjectIdentifier("2.5.29.35"); // AuthorityKeyIdentifier
 Extension OID
        customExtension.setValue(base64EncodedExtValue);

        // Add custom extension to api-passthrough
```

```
        ApiPassthrough apiPassthrough = new ApiPassthrough();
        Extensions extensions = new Extensions();
        extensions.setCustomExtensions(Arrays.asList(customExtension));
        apiPassthrough.setExtensions(extensions);
        issueRequest.setApiPassthrough(apiPassthrough);

        // Issue the certificate.
        IssueCertificateResult issueResult = null;
        try {
            issueResult = client.issueCertificate(issueRequest);
        } catch (LimitExceededException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (MalformedCSRException ex) {
            throw ex;
        }

        // Retrieve and display the certificate ARN.
        String rootCertificateArn = issueResult.getCertificateArn();
        System.out.println("Root Certificate Arn: " + rootCertificateArn);

        return rootCertificateArn;
    }

    private static String GetCertificate(String rootCertificateArn, String rootCAArn,
AWSACMPCA client) {

        // Create a request object.
        GetCertificateRequest certificateRequest = new GetCertificateRequest();

        // Set the certificate ARN.
        certificateRequest.withCertificateArn(rootCertificateArn);

        // Set the certificate authority ARN.
        certificateRequest.withCertificateAuthorityArn(rootCAArn);

        // Create waiter to wait on successful creation of the certificate file.
```

```
        Waiter<GetCertificateRequest> getCertificateWaiter =
client.waiters().certificateIssued();
        try {
            getCertificateWaiter.run(new WaiterParameters<>(certificateRequest));
        } catch (WaiterUnrecoverableException e) {
            //Explicit short circuit when the recourse transitions into
            //an undesired state.
        } catch (WaiterTimedOutException e) {
            //Failed to transition into desired state even after polling.
        } catch (AWSACMPCAException e) {
            //Unexpected service exception.
        }

        // Retrieve the certificate and certificate chain.
        GetCertificateResult certificateResult = null;
        try {
            certificateResult = client.getCertificate(certificateRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        }

        // Get the certificate and certificate chain and display the result.
        String rootCertificate = certificateResult.getCertificate();
        System.out.println(rootCertificate);

        return rootCertificate;
    }

    private static void ImportCertificateAuthorityCertificate(String rootCertificate,
String rootCAArn, AWSACMPCA client) {

        // Create the request object and set the signed certificate, chain and CA ARN.
        ImportCertificateAuthorityCertificateRequest importRequest =
            new ImportCertificateAuthorityCertificateRequest();

        ByteBuffer certByteBuffer = stringToByteBuffer(rootCertificate);
```

```
        importRequest.setCertificate(certByteBuffer);

        importRequest.setCertificateChain(null);

        // Set the certificate authority ARN.
        importRequest.withCertificateAuthorityArn(rootCAArn);

        // Import the certificate.
        try {
            client.importCertificateAuthorityCertificate(importRequest);
        } catch (CertificateMismatchException ex) {
            throw ex;
        } catch (MalformedCertificateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (ConcurrentModificationException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }

        System.out.println("Root CA certificate successfully imported.");
        System.out.println("Root CA activated successfully.");
    }

    private static ByteBuffer stringToByteBuffer(final String string) {
        if (Objects.isNull(string)) {
            return null;
        }
        byte[] bytes = string.getBytes(StandardCharsets.UTF_8);
        return ByteBuffer.wrap(bytes);
    }
}
```

為節點操作憑證 (NOC) 啟用次級 CA

此 Java 範例示範如何使用 BlankSubordinateCACertificate_PathLen0_APIPassthrough/V1 定義 範本發行和安裝事項次級 CA 憑證來發行 NOCs。您必須產生 Base64-encoded的 KeyUsage 值，並透過 CustomExtension 傳遞。

此範例會呼叫下列 AWS 私有 CA API 動作：

- CreateCertificateAuthority
- GetCertificateAuthorityCsr
- IssueCertificate
- GetCertificate
- ImportCertificateAuthorityCertificate
- GetCertificateAuthorityCertificate

如果發生問題，請參閱疑難排解一節故障診斷符合 AWS 私有 CA 事項的憑證錯誤中的 。

```
package com.amazonaws.samples.matter;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;


import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.ASN1Subject;
import com.amazonaws.services.acmpca.model.ApiPassthrough;
import com.amazonaws.services.acmpca.model.CertificateAuthorityConfiguration;
import com.amazonaws.services.acmpca.model.CertificateAuthorityType;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityResult;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityRequest;
import com.amazonaws.services.acmpca.model.CustomAttribute;
import com.amazonaws.services.acmpca.model.CustomExtension;
import com.amazonaws.services.acmpca.model.Extensions;
import com.amazonaws.services.acmpca.model.KeyAlgorithm;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
```

```
import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.Arrays;
import java.util.Base64;
import java.util.List;
import java.util.Objects;

import org.bouncycastle.asn1.x509.KeyUsage;
import org.bouncycastle.jce.X509KeyUsage;

import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCertificateRequest;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCertificateResult;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrRequest;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrResult;
import com.amazonaws.services.acmpca.model.GetCertificateRequest;
import com.amazonaws.services.acmpca.model.GetCertificateResult;
import
 com.amazonaws.services.acmpca.model.ImportCertificateAuthorityCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateResult;
import com.amazonaws.services.acmpca.model.Validity;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.CertificateMismatchException;
import com.amazonaws.services.acmpca.model.ConcurrentModificationException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidPolicyException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.MalformedCertificateException;
import com.amazonaws.services.acmpca.model.MalformedCSRException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.RequestInProgressException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.AWSACMPCAException;

import com.amazonaws.waiters.Waiter;
import com.amazonaws.waiters.WaiterParameters;
import com.amazonaws.waiters.WaiterTimedOutException;
import com.amazonaws.waiters.WaiterUnrecoverableException;

import lombok.SneakyThrows;
```

```
public class IntermediateCAActivation {

    public static void main(String[] args) throws Exception {
        // Place your own Root CA ARN here.
        String rootCAArn = "arn:aws:acm-pca:region:123456789012:certificate-
authority/12345678-1234-1234-1234-123456789012";

        // Define the endpoint region for your sample.
        String endpointRegion = "region";  // Substitute your region here, e.g. "ap-
southeast-2"

        // Define custom attributes
        List<CustomAttribute> customAttributes = Arrays.asList(
            new CustomAttribute()
                .withObjectIdentifier("1.3.6.1.4.1.37244.1.3")
                .withValue("CACACACA00000003")
        );

        // Define a CA subject.
        ASN1Subject subject = new ASN1Subject();
        subject.setCustomAttributes(customAttributes);

        // Define the CA configuration.
        CertificateAuthorityConfiguration configCA = new
 CertificateAuthorityConfiguration();
        configCA.withKeyAlgorithm(KeyAlgorithm.EC_prime256v1);
        configCA.withSigningAlgorithm(SigningAlgorithm.SHA256WITHECDSA);
        configCA.withSubject(subject);

        // Define a certificate authority type
        CertificateAuthorityType CAtype = CertificateAuthorityType.SUBORDINATE;

        // ** Execute core code samples for Subordinate CA activation in sequence **
        AWSACMPCA client = ClientBuilder(endpointRegion);
        String rootCertificate = GetCertificateAuthorityCertificate(rootCAArn, client);
        String subordinateCAArn = CreateCertificateAuthority(configCA, CAtype, client);
        String csr = GetCertificateAuthorityCsr(subordinateCAArn, client);
        String subordinateCertificateArn = IssueCertificate(rootCAArn, csr, client);
        String subordinateCertificate = GetCertificate(subordinateCertificateArn,
 rootCAArn, client);
        ImportCertificateAuthorityCertificate(subordinateCertificate, rootCertificate,
 subordinateCAArn, client);
```

```
    }

    private static AWSACMPCA ClientBuilder(String endpointRegion) {
        // Get your credentials from the C:\Users\name\.aws\credentials file
        // in Windows or the .aws/credentials file in Linux.
        AWSCredentials credentials = null;
        try {
            credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException(
                    "Cannot load the credentials from the credential profiles file. " +
                    "Please make sure that your credentials file is at the correct " +
                    "location (C:\\Users\\joneps\\.aws\\credentials), and is in valid
format.",
                    e);
        }

        String endpointProtocol = "https://acm-pca." + endpointRegion +
".amazonaws.com/";
        EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
endpointRegion);

        // Create a client that you can use to make requests.
        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
            .withEndpointConfiguration(endpoint)
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();

        return client;
    }

    private static String GetCertificateAuthorityCertificate(String rootCAArn,
AWSACMPCA client) {
        // ** GetCertificateAuthorityCertificate **

        // Create a request object and set the certificate authority ARN,
        GetCertificateAuthorityCertificateRequest getCACertificateRequest =
        new GetCertificateAuthorityCertificateRequest();
        getCACertificateRequest.withCertificateAuthorityArn(rootCAArn);

        // Create a result object.
        GetCertificateAuthorityCertificateResult getCACertificateResult = null;
        try {
```

```
            getCACertificateResult =
client.getCertificateAuthorityCertificate(getCACertificateRequest);
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        }

        // Get and display the certificate information.
        String rootCertificate = getCACertificateResult.getCertificate();
        System.out.println("Root CA Certificate / Certificate Chain:");
        System.out.println(rootCertificate);

        return rootCertificate;
    }

    private static String CreateCertificateAuthority(CertificateAuthorityConfiguration
configCA, CertificateAuthorityType CAtype, AWSACMPCA client) {
        // Create the request object.
        CreateCertificateAuthorityRequest createCARequest = new
CreateCertificateAuthorityRequest();
        createCARequest.withCertificateAuthorityConfiguration(configCA);
        createCARequest.withIdempotencyToken("123987");
        createCARequest.withCertificateAuthorityType(CAtype);

        // Create the private CA.
        CreateCertificateAuthorityResult createCAResult = null;
        try {
            createCAResult = client.createCertificateAuthority(createCARequest);
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (InvalidPolicyException ex) {
            throw ex;
        } catch (LimitExceededException ex) {
            throw ex;
        }

        // Retrieve the ARN of the private CA.
        String subordinateCAArn = createCAResult.getCertificateAuthorityArn();
        System.out.println("Subordinate CA Arn: " + subordinateCAArn);

        return subordinateCAArn;
```

```
    }

    private static String GetCertificateAuthorityCsr(String subordinateCAArn, AWSACMPCA
client) {

        // Create the CSR request object and set the CA ARN.
        GetCertificateAuthorityCsrRequest csrRequest = new
GetCertificateAuthorityCsrRequest();
        csrRequest.withCertificateAuthorityArn(subordinateCAArn);

        // Create waiter to wait on successful creation of the CSR file.
        Waiter<GetCertificateAuthorityCsrRequest> getCSRWaiter =
client.waiters().certificateAuthorityCSRCreated();
        try {
            getCSRWaiter.run(new WaiterParameters<>(csrRequest));
        } catch (WaiterUnrecoverableException e) {
            //Explicit short circuit when the recourse transitions into
            //an undesired state.
        } catch (WaiterTimedOutException e) {
            //Failed to transition into desired state even after polling.
        } catch(AWSACMPCAException e) {
            //Unexpected service exception.
        }

        // Get the CSR.
        GetCertificateAuthorityCsrResult csrResult = null;
        try {
            csrResult = client.getCertificateAuthorityCsr(csrRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }

        // Get and display the CSR;
        String csr = csrResult.getCsr();
        System.out.println("Subordinate CSR:");
        System.out.println(csr);

        return csr;
```

```
    }

    private static String IssueCertificate(String rootCAArn, String csr, AWSACMPCA
 client) {

        // Create a certificate request:
        IssueCertificateRequest issueRequest = new IssueCertificateRequest();

        // Set the issuing CA ARN.
        issueRequest.withCertificateAuthorityArn(rootCAArn);

        // Set the template ARN.
        issueRequest.withTemplateArn("arn:aws:acm-pca:::template/
BlankSubordinateCACertificate_PathLen0_APIPassthrough/V1");

        ByteBuffer csrByteBuffer = stringToByteBuffer(csr);
        issueRequest.setCsr(csrByteBuffer);

        // Set the signing algorithm.
        issueRequest.withSigningAlgorithm(SigningAlgorithm.SHA256WITHECDSA);

        // Set the validity period for the certificate to be issued.
        Validity validity = new Validity();
        validity.withValue(730L); // Approximately two years
        validity.withType("DAYS");
        issueRequest.withValidity(validity);

        // Set the idempotency token.
        issueRequest.setIdempotencyToken("1234");

        ApiPassthrough apiPassthrough = new ApiPassthrough();

        // Generate base64 encoded extension value for ExtendedKeyUsage
        String base64EncodedKUValue = generateKeyUsageValue();

        // Generate custom extension
        CustomExtension customKeyUsageExtension = new CustomExtension();
        customKeyUsageExtension.setObjectIdentifier("2.5.29.15");
        customKeyUsageExtension.setValue(base64EncodedKUValue);
        customKeyUsageExtension.setCritical(true);

        // Set KeyUsage extension to api passthrough
        Extensions extensions = new Extensions();
        extensions.setCustomExtensions(Arrays.asList(customKeyUsageExtension));
```

```
        apiPassthrough.setExtensions(extensions);
        issueRequest.setApiPassthrough(apiPassthrough);

        // Issue the certificate.
        IssueCertificateResult issueResult = null;
        try {
            issueResult = client.issueCertificate(issueRequest);
        } catch (LimitExceededException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (MalformedCSRException ex) {
            throw ex;
        }

        // Get and display the certificate ARN.
        String subordinateCertificateArn = issueResult.getCertificateArn();
        System.out.println("Subordinate Certificate Arn: " +
subordinateCertificateArn);

        return subordinateCertificateArn;
    }

    @SneakyThrows
    private static String generateKeyUsageValue() {
       KeyUsage keyUsage = new KeyUsage(X509KeyUsage.keyCertSign |
X509KeyUsage.cRLSign);
       byte[] kuBytes = keyUsage.getEncoded();
       return Base64.getEncoder().encodeToString(kuBytes);
    }

    private static String GetCertificate(String subordinateCertificateArn, String
rootCAArn, AWSACMPCA client) {

        // Create a request object.
        GetCertificateRequest certificateRequest = new GetCertificateRequest();

        // Set the certificate ARN.
```

```
        certificateRequest.withCertificateArn(subordinateCertificateArn);

        // Set the certificate authority ARN.
        certificateRequest.withCertificateAuthorityArn(rootCAArn);

        // Create waiter to wait on successful creation of the certificate file.
        Waiter<GetCertificateRequest> getCertificateWaiter =
client.waiters().certificateIssued();
        try {
            getCertificateWaiter.run(new WaiterParameters<>(certificateRequest));
        } catch (WaiterUnrecoverableException e) {
            //Explicit short circuit when the recourse transitions into
            //an undesired state.
        } catch (WaiterTimedOutException e) {
            //Failed to transition into desired state even after polling.
        } catch (AWSACMPCAException e) {
            //Unexpected service exception.
        }

        // Get the certificate and certificate chain.
        GetCertificateResult certificateResult = null;
        try {
            certificateResult = client.getCertificate(certificateRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        }

        // Get the certificate and certificate chain and display the result.
        String subordinateCertificate = certificateResult.getCertificate();
        System.out.println("Subordinate CA Certificate:");
        System.out.println(subordinateCertificate);

        return subordinateCertificate;
    }
```

```
    private static void ImportCertificateAuthorityCertificate(String
subordinateCertificate, String rootCertificate, String subordinateCAArn, AWSACMPCA
client) {

        // Create the request object and set the signed certificate, chain and CA ARN.
        ImportCertificateAuthorityCertificateRequest importRequest =
            new ImportCertificateAuthorityCertificateRequest();

        ByteBuffer certByteBuffer = stringToByteBuffer(subordinateCertificate);
        importRequest.setCertificate(certByteBuffer);

        ByteBuffer rootCACertByteBuffer = stringToByteBuffer(rootCertificate);
        importRequest.setCertificateChain(rootCACertByteBuffer);

        // Set the certificate authority ARN.
        importRequest.withCertificateAuthorityArn(subordinateCAArn);

        // Import the certificate.
        try {
            client.importCertificateAuthorityCertificate(importRequest);
        } catch (CertificateMismatchException ex) {
            throw ex;
        } catch (MalformedCertificateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (ConcurrentModificationException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }
        System.out.println("Subordinate CA certificate successfully imported.");
        System.out.println("Subordinate CA activated successfully.");
    }

    private static ByteBuffer stringToByteBuffer(final String string) {
        if (Objects.isNull(string)) {
            return null;
        }
        byte[] bytes = string.getBytes(StandardCharsets.UTF_8);
```

```
        return ByteBuffer.wrap(bytes);
    }
}
```

建立節點操作憑證 (NOC)

此 Java 範例說明如何使用 BlankEndEntityCertificate_CriticalBasicConstraints_APIPassthrough/V1 範本來建立事項節點操作憑證。您必須產生 Base64-encoded的 KeyUsage 值，並透過 CustomExtension 傳遞。

此範例會呼叫下列 AWS 私有 CA API 動作：

- IssueCertificate

如果您遇到問題，請參閱疑難排解一節故障診斷符合 AWS 私有 CA 事項的憑證錯誤中的 。

```
package com.amazonaws.samples.matter;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.Arrays;
import java.util.Base64;
import java.util.List;
import java.util.Objects;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.ASN1Subject;
import com.amazonaws.services.acmpca.model.ApiPassthrough;
import com.amazonaws.services.acmpca.model.CustomAttribute;
import com.amazonaws.services.acmpca.model.CustomExtension;
import com.amazonaws.services.acmpca.model.Extensions;
import com.amazonaws.services.acmpca.model.IssueCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateResult;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
```

```
import com.amazonaws.services.acmpca.model.Validity;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.MalformedCSRException;

import org.bouncycastle.asn1.x509.ExtendedKeyUsage;
import org.bouncycastle.asn1.x509.KeyPurposeId;
import org.bouncycastle.asn1.x509.KeyUsage;
import org.bouncycastle.jce.X509KeyUsage;

import lombok.SneakyThrows;

public class IssueNodeOperatingCertificate {
   public static ByteBuffer stringToByteBuffer(final String string) {
      if (Objects.isNull(string)) {
          return null;
      }
      byte[] bytes = string.getBytes(StandardCharsets.UTF_8);
      return ByteBuffer.wrap(bytes);
   }

   @SneakyThrows
   private static String generateExtendedKeyUsageValue() {
      KeyPurposeId[] keyPurposeIds = new KeyPurposeId[]
 { KeyPurposeId.id_kp_clientAuth, KeyPurposeId.id_kp_serverAuth };
      ExtendedKeyUsage eku = new ExtendedKeyUsage(keyPurposeIds);
      byte[] ekuBytes = eku.getEncoded();
      return Base64.getEncoder().encodeToString(ekuBytes);
   }

   @SneakyThrows
   private static String generateKeyUsageValue() {
      KeyUsage keyUsage = new KeyUsage(X509KeyUsage.digitalSignature);
      byte[] kuBytes = keyUsage.getEncoded();
      return Base64.getEncoder().encodeToString(kuBytes);
   }

   public static void main(String[] args) throws Exception {
```

```
      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
          credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
          throw new AmazonClientException("Cannot load your credentials from disk", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "ap-
southeast-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
          new AwsClientBuilder.EndpointConfiguration(endpointProtocol, endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
          .withEndpointConfiguration(endpoint)
          .withCredentials(new AWSStaticCredentialsProvider(credentials))
          .build();

      // Create a certificate request:
      IssueCertificateRequest req = new IssueCertificateRequest();

      // Set the CA ARN.
      req.withCertificateAuthorityArn("arn:aws:acm-pca:region:123456789012:certificate-
authority/12345678-1234-1234-1234-123456789012");

      // Specify the certificate signing request (CSR) for the certificate to be signed
 and issued.
      String strCSR =
      "-----BEGIN CERTIFICATE REQUEST-----\n" +
      "base64-encoded certificate\n" +
      "-----END CERTIFICATE REQUEST-----\n";
      ByteBuffer csrByteBuffer = stringToByteBuffer(strCSR);
      req.setCsr(csrByteBuffer);

      // Specify the template for the issued certificate.
      req.withTemplateArn("arn:aws:acm-pca:::template/
BlankEndEntityCertificate_CriticalBasicConstraints_APIPassthrough/V1");

      // Set the signing algorithm.
```

```
      req.withSigningAlgorithm(SigningAlgorithm.SHA256WITHECDSA);

      // Set the validity period for the certificate to be issued.
      Validity validity = new Validity();
      validity.withValue(10L);
      validity.withType("DAYS");
      req.withValidity(validity);

      // Set the idempotency token.
      req.setIdempotencyToken("1234");

      // Define custom attributes
      List<CustomAttribute> customAttributes = Arrays.asList(
         new CustomAttribute()
            .withObjectIdentifier("1.3.6.1.4.1.37244.1.1")
            .withValue("DEDEDEDE00010001"),
         new CustomAttribute()
            .withObjectIdentifier("1.3.6.1.4.1.37244.1.5")
            .withValue("FAB000000000001D")
      );

      // Define a cert subject.
      ASN1Subject subject = new ASN1Subject();
      subject.setCustomAttributes(customAttributes);

      ApiPassthrough apiPassthrough = new ApiPassthrough();
      apiPassthrough.setSubject(subject);

      // Generate Base64 encoded extension value for ExtendedKeyUsage
      String base64EncodedKUValue = generateKeyUsageValue();

      // Generate custom extension
      CustomExtension customKeyUsageExtension = new CustomExtension();
      customKeyUsageExtension.setObjectIdentifier("2.5.29.15");
      customKeyUsageExtension.setValue(base64EncodedKUValue);
      customKeyUsageExtension.setCritical(true);

      // Generate Base64 encoded extension value for ExtendedKeyUsage
      String base64EncodedEKUValue = generateExtendedKeyUsageValue();

      CustomExtension customExtendedKeyUsageExtension = new CustomExtension();
      customExtendedKeyUsageExtension.setObjectIdentifier("2.5.29.37"); //
 ExtendedKeyUsage Extension OID
      customExtendedKeyUsageExtension.setValue(base64EncodedEKUValue);
```

```
        customExtendedKeyUsageExtension.setCritical(true);

        // Set KeyUsage and ExtendedKeyUsage extension to api-passthrough
        Extensions extensions = new Extensions();
        extensions.setCustomExtensions(Arrays.asList(customKeyUsageExtension,
 customExtendedKeyUsageExtension));
        apiPassthrough.setExtensions(extensions);
        req.setApiPassthrough(apiPassthrough);

        // Issue the certificate.
        IssueCertificateResult result = null;
        try {
           result = client.issueCertificate(req);
        } catch (LimitExceededException ex) {
           throw ex;
        } catch (ResourceNotFoundException ex) {
           throw ex;
        } catch (InvalidStateException ex) {
           throw ex;
        } catch (InvalidArnException ex) {
           throw ex;
        } catch (InvalidArgsException ex) {
           throw ex;
        } catch (MalformedCSRException ex) {
           throw ex;
        }

        // Retrieve and display the certificate ARN.
        String arn = result.getCertificateArn();
        System.out.println(arn);
    }
}
```

使用 AWS 私有 CA 實作 mDL 憑證

您可以使用 AWS 私有憑證授權單位 API 來建立符合 ISO/IEC 行動駕照標準 (mDL) 的憑證。此標準會建立介面規格，以實作與行動裝置相關的駕照，包括憑證組態。

主題

- 啟用發行授權機構憑證授權機構 (IACA) 憑證
- 建立文件簽署者憑證

啟用發行授權機構憑證授權機構 (IACA) 憑證

此 Java 範例示範如何使用 BlankRootCACertificate_PathLen0_APIPassthrough/V1 定義 範本來建立和安裝 ISO/IEC mDL 標準合規發行授權機構憑證授權機構 (IACA) 憑證。您必須為 KeyUsage、IssuerAlternativeName和 產生 base64 編碼值CRLDistributionPoint，並透過 傳遞這些值CustomExtensions。

> **Note**
>
> IACA 連結憑證會建立從舊 IACA 根憑證到新 IACA 根憑證的信任路徑。發行授權機構可以在 IACA 重新金鑰程序期間產生和分發 IACA 連結憑證。您無法透過使用 IACA 根憑證搭配pathLen=0集合來發行 IACA 連結憑證。

此範例會呼叫下列 AWS 私有 CA API 動作：

- CreateCertificateAuthority
- GetCertificateAuthorityCsr
- IssueCertificate
- GetCertificate
- ImportCertificateAuthorityCertificate

```
package com.amazonaws.samples.mdl;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.ASN1Subject;
import com.amazonaws.services.acmpca.model.ApiPassthrough;
import com.amazonaws.services.acmpca.model.CertificateAuthorityConfiguration;
import com.amazonaws.services.acmpca.model.CertificateAuthorityType;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityResult;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityRequest;
import com.amazonaws.services.acmpca.model.CustomExtension;
import com.amazonaws.services.acmpca.model.Extensions;
```

```
import com.amazonaws.services.acmpca.model.KeyAlgorithm;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;

import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.Arrays;
import java.util.Base64;
import java.util.Objects;

import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrRequest;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrResult;
import com.amazonaws.services.acmpca.model.GetCertificateRequest;
import com.amazonaws.services.acmpca.model.GetCertificateResult;
import
 com.amazonaws.services.acmpca.model.ImportCertificateAuthorityCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateResult;
import com.amazonaws.services.acmpca.model.Validity;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.CertificateMismatchException;
import com.amazonaws.services.acmpca.model.ConcurrentModificationException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidPolicyException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.MalformedCertificateException;
import com.amazonaws.services.acmpca.model.MalformedCSRException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.RequestInProgressException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.AWSACMPCAException;

import com.amazonaws.waiters.Waiter;
import com.amazonaws.waiters.WaiterParameters;
import com.amazonaws.waiters.WaiterTimedOutException;
import com.amazonaws.waiters.WaiterUnrecoverableException;

import org.bouncycastle.asn1.x509.GeneralNames;
import org.bouncycastle.asn1.x509.GeneralName;
import org.bouncycastle.asn1.x509.CRLDistPoint;
import org.bouncycastle.asn1.x509.DistributionPoint;
import org.bouncycastle.asn1.x509.DistributionPointName;
```

```
import org.bouncycastle.asn1.x509.KeyUsage;
import org.bouncycastle.jce.X509KeyUsage;

import lombok.SneakyThrows;

public class IssuingAuthorityCertificateAuthorityActivation {
    public static void main(String[] args) throws Exception {
        // Define the endpoint region for your sample.
        String endpointRegion = null;  // Substitute your region here, e.g. "ap-
southeast-2"
        if (endpointRegion == null) throw new Exception("Region cannot be null");

        // Define a CA subject.
        ASN1Subject subject = new ASN1Subject()
                .withCountry("US") // mDL spec requires ISO 3166-1-alpha-2 country code
 e.g. "US"
                .withCommonName("mDL Test IACA");

        // Define the CA configuration.
        CertificateAuthorityConfiguration configCA = new
 CertificateAuthorityConfiguration()
                .withKeyAlgorithm(KeyAlgorithm.EC_prime256v1)
                .withSigningAlgorithm(SigningAlgorithm.SHA256WITHECDSA)
                .withSubject(subject);

        // Define a certificate authority type
        CertificateAuthorityType CAType = CertificateAuthorityType.ROOT;

        // Execute core code samples for Root CA activation in sequence
        AWSACMPCA client = buildClient(endpointRegion);
        String rootCAArn = createCertificateAuthority(configCA, CAType, client);
        String csr = getCertificateAuthorityCsr(rootCAArn, client);
        String rootCertificateArn = issueCertificate(rootCAArn, csr, client);
        String rootCertificate = getCertificate(rootCertificateArn, rootCAArn, client);
        importCertificateAuthorityCertificate(rootCertificate, rootCAArn, client);
    }

    private static AWSACMPCA buildClient(String endpointRegion) {
        // Get your credentials from the C:\Users\name\.aws\credentials file
        // in Windows or the .aws/credentials file in Linux.
        AWSCredentials credentials = null;
        try {
            credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
```

```
            throw new AmazonClientException("Cannot load your credentials from disk",
e);
        }

        // Create a client that you can use to make requests.
        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
                .withRegion(endpointRegion)
                .withCredentials(new AWSStaticCredentialsProvider(credentials))
                .build();

        return client;
    }

    private static String createCertificateAuthority(CertificateAuthorityConfiguration
configCA, CertificateAuthorityType CAtype, AWSACMPCA client) {
        // Create the request object.
        CreateCertificateAuthorityRequest createCARequest = new
CreateCertificateAuthorityRequest()
                .withCertificateAuthorityConfiguration(configCA)
                .withIdempotencyToken("123987")
                .withCertificateAuthorityType(CAtype);

        // Create the private CA.
        CreateCertificateAuthorityResult createCAResult = null;
        try {
            createCAResult = client.createCertificateAuthority(createCARequest);
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (InvalidPolicyException ex) {
            throw ex;
        } catch (LimitExceededException ex) {
            throw ex;
        }

        // Get the ARN of the private CA.
        String rootCAArn = createCAResult.getCertificateAuthorityArn();
        System.out.println("Issuing Authority Certificate Authority (IACA) Arn: " +
rootCAArn);

        return rootCAArn;
    }

    private static String getCertificateAuthorityCsr(String rootCAArn, AWSACMPCA
client) {
```

```
        // Create the CSR request object and set the CA ARN.
        GetCertificateAuthorityCsrRequest csrRequest = new
GetCertificateAuthorityCsrRequest()
                .withCertificateAuthorityArn(rootCAArn);

        // Create waiter to wait on successful creation of the CSR file.
        Waiter<GetCertificateAuthorityCsrRequest> getCSRWaiter =
client.waiters().certificateAuthorityCSRCreated();
        try {
            getCSRWaiter.run(new WaiterParameters<>(csrRequest));
        } catch (WaiterUnrecoverableException e) {
            // Explicit short circuit when the recourse transitions into
            // an undesired state.
        } catch (WaiterTimedOutException e) {
            // Failed to transition into desired state even after polling.
        } catch (AWSACMPCAException e) {
            // Unexpected service exception.
        }

        // Get the CSR.
        GetCertificateAuthorityCsrResult csrResult = null;
        try {
            csrResult = client.getCertificateAuthorityCsr(csrRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }

        // Get and display the CSR;
        String csr = csrResult.getCsr();
        System.out.println("CSR:");
        System.out.println(csr);

        return csr;
    }

    @SneakyThrows
```

```
    private static String issueCertificate(String rootCAArn, String csr, AWSACMPCA
 client) {
        IssueCertificateRequest issueRequest = new IssueCertificateRequest()
                .withCertificateAuthorityArn(rootCAArn)
                .withTemplateArn("arn:aws:acm-pca:::template/
BlankRootCACertificate_PathLen0_APIPassthrough/V1")
                .withSigningAlgorithm(SigningAlgorithm.SHA256WITHECDSA)
                .withIdempotencyToken("1234");

        // Set the CSR.
        ByteBuffer csrByteBuffer = stringToByteBuffer(csr);
        issueRequest.setCsr(csrByteBuffer);

        // Set the validity period for the certificate to be issued.
        Validity validity = new Validity()
                .withValue(3650L)
                .withType("DAYS");
        issueRequest.setValidity(validity);

        // Generate base64 encoded extension value for KeyUsage
        KeyUsage keyUsage = new KeyUsage(X509KeyUsage.keyCertSign +
 X509KeyUsage.cRLSign);
        byte[] kuBytes = keyUsage.getEncoded();
        String base64EncodedKUValue = Base64.getEncoder().encodeToString(kuBytes);

        CustomExtension keyUsageCustomExtension = new CustomExtension()
                .withObjectIdentifier("2.5.29.15") // KeyUsage Extension OID
                .withValue(base64EncodedKUValue)
                .withCritical(true);

        // Generate base64 encoded extension value for IssuerAlternativeName
        GeneralNames issuerAlternativeName = new GeneralNames(new
 GeneralName(GeneralName.uniformResourceIdentifier, "https://issuer-alternative-
name.com"));
        String base64EncodedIANValue =
 Base64.getEncoder().encodeToString(issuerAlternativeName.getEncoded());

        CustomExtension ianCustomExtension = new CustomExtension()
                .withValue(base64EncodedIANValue)
                .withObjectIdentifier("2.5.29.18"); // IssuerAlternativeName Extension
 OID

        // Generate base64 encoded extension value for CRLDistributionPoint
```

```
        CRLDistPoint crlDistPoint = new CRLDistPoint(new DistributionPoint[]{new
DistributionPoint(new DistributionPointName(
                new GeneralNames(new GeneralName(GeneralName.uniformResourceIdentifier,
"dummycrl.crl"))), null, null)});
        String base64EncodedCDPValue =
Base64.getEncoder().encodeToString(crlDistPoint.getEncoded());

        CustomExtension cdpCustomExtension = new CustomExtension()
                .withValue(base64EncodedCDPValue)
                .withObjectIdentifier("2.5.29.31"); // CRLDistributionPoint Extension
OID

        // Add custom extension to api-passthrough
        Extensions extensions = new Extensions()
                .withCustomExtensions(Arrays.asList(keyUsageCustomExtension,
ianCustomExtension, cdpCustomExtension));
        ApiPassthrough apiPassthrough = new ApiPassthrough()
                .withExtensions(extensions);
        issueRequest.setApiPassthrough(apiPassthrough);

        // Issue the certificate.
        IssueCertificateResult issueResult = null;
        try {
            issueResult = client.issueCertificate(issueRequest);
        } catch (LimitExceededException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (MalformedCSRException ex) {
            throw ex;
        }

        // Get and display the certificate ARN.
        String rootCertificateArn = issueResult.getCertificateArn();
        System.out.println("mDL IACA Certificate Arn: " + rootCertificateArn);

        return rootCertificateArn;
    }
```

```
   private static String getCertificate(String rootCertificateArn, String rootCAArn,
AWSACMPCA client) {

       // Create a request object.
       GetCertificateRequest certificateRequest = new GetCertificateRequest()
               .withCertificateArn(rootCertificateArn)
               .withCertificateAuthorityArn(rootCAArn);

       // Create waiter to wait on successful creation of the certificate file.
       Waiter<GetCertificateRequest> getCertificateWaiter =
client.waiters().certificateIssued();
       try {
           getCertificateWaiter.run(new WaiterParameters<>(certificateRequest));
       } catch (WaiterUnrecoverableException e) {
           // Explicit short circuit when the recourse transitions into
           // an undesired state.
       } catch (WaiterTimedOutException e) {
           // Failed to transition into desired state even after polling.
       } catch (AWSACMPCAException e) {
           // Unexpected service exception.
       }

       // Get the certificate and certificate chain.
       GetCertificateResult certificateResult = null;
       try {
           certificateResult = client.getCertificate(certificateRequest);
       } catch (RequestInProgressException ex) {
           throw ex;
       } catch (RequestFailedException ex) {
           throw ex;
       } catch (ResourceNotFoundException ex) {
           throw ex;
       } catch (InvalidArnException ex) {
           throw ex;
       } catch (InvalidStateException ex) {
           throw ex;
       }

       // Get the certificate and certificate chain and display the result.
       String rootCertificate = certificateResult.getCertificate();
       System.out.println(rootCertificate);

       return rootCertificate;
```

```
    }

    private static void importCertificateAuthorityCertificate(String rootCertificate,
String rootCAArn, AWSACMPCA client) {

        // Create the request object and set the signed certificate, chain and CA ARN.
        ImportCertificateAuthorityCertificateRequest importRequest =
                new ImportCertificateAuthorityCertificateRequest()
                        .withCertificateChain(null)
                        .withCertificateAuthorityArn(rootCAArn);

        ByteBuffer certByteBuffer = stringToByteBuffer(rootCertificate);
        importRequest.setCertificate(certByteBuffer);

        // Import the certificate.
        try {
            client.importCertificateAuthorityCertificate(importRequest);
        } catch (CertificateMismatchException ex) {
            throw ex;
        } catch (MalformedCertificateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (ConcurrentModificationException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }

        System.out.println("Root CA certificate successfully imported.");
        System.out.println("Root CA activated successfully.");
    }

    private static ByteBuffer stringToByteBuffer(final String string) {
        if (Objects.isNull(string)) {
            return null;
        }
        byte[] bytes = string.getBytes(StandardCharsets.UTF_8);
        return ByteBuffer.wrap(bytes);
    }
```

```
}
```

建立文件簽署者憑證

此 Java 範例示範如何使用 BlankEndEntityCertificate_APIPassthrough/V1 範本來建立 ISO/IEC mDL 標準合規文件簽署者憑證。您必須為 KeyUsage、 和 產生 base64 編碼值IssuerAlternativeName，CRLDistributionPoint並透過 傳遞這些值CustomExtensions。

此範例會呼叫下列 AWS 私有 CA API 動作：

- IssueCertificate

```
package com.amazonaws.samples.mdl;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.Arrays;
import java.util.Base64;
import java.util.Objects;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.ASN1Subject;
import com.amazonaws.services.acmpca.model.ApiPassthrough;
import com.amazonaws.services.acmpca.model.ExtendedKeyUsage;
import com.amazonaws.services.acmpca.model.CustomExtension;
import com.amazonaws.services.acmpca.model.Extensions;
import com.amazonaws.services.acmpca.model.IssueCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateResult;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Validity;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
```

```
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.MalformedCSRException;

import org.bouncycastle.asn1.x509.GeneralNames;
import org.bouncycastle.asn1.x509.GeneralName;
import org.bouncycastle.asn1.x509.CRLDistPoint;
import org.bouncycastle.asn1.x509.DistributionPoint;
import org.bouncycastle.asn1.x509.DistributionPointName;
import org.bouncycastle.asn1.x509.KeyUsage;
import org.bouncycastle.jce.X509KeyUsage;

public class IssueDocumentSignerCertificate {
    public static ByteBuffer stringToByteBuffer(final String string) {
        if (Objects.isNull(string)) {
            return null;
        }
        byte[] bytes = string.getBytes(StandardCharsets.UTF_8);
        return ByteBuffer.wrap(bytes);
    }

    public static void main(String[] args) throws Exception {

        // Get your credentials from the C:\Users\name\.aws\credentials file
        // in Windows or the .aws/credentials file in Linux.
        AWSCredentials credentials = null;
        try {
            credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException("Cannot load your credentials from disk",
 e);
        }

        // Create a client that you can use to make requests.
        String endpointRegion = null;  // Substitute your region here, e.g. "ap-
southeast-2"
        if (endpointRegion == null) throw new Exception("Region cannot be null");

        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
                .withRegion(endpointRegion)
                .withCredentials(new AWSStaticCredentialsProvider(credentials))
                .build();
```

```
        // Create a certificate request:
        String caArn = null;
        if (caArn == null) throw new Exception("Certificate authority ARN cannot be
 null");

        IssueCertificateRequest req = new IssueCertificateRequest()
                .withCertificateAuthorityArn(caArn)
                .withTemplateArn("arn:aws:acm-pca:::template/
BlankEndEntityCertificate_APIPassthrough/V1")
                .withSigningAlgorithm(SigningAlgorithm.SHA256WITHECDSA)
                .withIdempotencyToken("1234");

        // Specify the certificate signing request (CSR) for the certificate to be
 signed and issued.
        // Format: "-----BEGIN CERTIFICATE REQUEST-----\n" +
        //         "base64-encoded certificate\n" +
        //         "-----END CERTIFICATE REQUEST-----\n";
        String strCSR = null;
        if (strCSR == null) throw new Exception("CSR string cannot be null");

        ByteBuffer csrByteBuffer = stringToByteBuffer(strCSR);
        req.setCsr(csrByteBuffer);

        // Set the validity period for the certificate to be issued.
        Validity validity = new Validity()
                .withValue(365L)
                .withType("DAYS");
        req.setValidity(validity);

        // Define a cert subject.
        ASN1Subject subject = new ASN1Subject()
                .withCountry("US") // mDL spec requires ISO 3166-1-alpha-2 country code
 e.g. "US"
                .withCommonName("mDL Test DS");

        ApiPassthrough apiPassthrough = new ApiPassthrough()
                .withSubject(subject);

        // Generate base64 encoded extension value for KeyUsage
        KeyUsage keyUsage = new KeyUsage(X509KeyUsage.digitalSignature);
        byte[] kuBytes = keyUsage.getEncoded();
        String base64EncodedKUValue = Base64.getEncoder().encodeToString(kuBytes);

        CustomExtension customKeyUsageExtension = new CustomExtension()
```

```
                .withObjectIdentifier("2.5.29.15") // KeyUsage Extension OID
                .withValue(base64EncodedKUValue)
                .withCritical(true);

        // Generate base64 encoded extension value for IssuerAlternativeName
        GeneralNames issuerAlternativeName = new GeneralNames(new
 GeneralName(GeneralName.uniformResourceIdentifier, "https://issuer-alternative-
name.com"));
        String base64EncodedIANValue =
 Base64.getEncoder().encodeToString(issuerAlternativeName.getEncoded());

        CustomExtension ianCustomExtension = new CustomExtension()
                .withValue(base64EncodedIANValue)
                .withObjectIdentifier("2.5.29.18"); // IssuerAlternativeName Extension
 OID

        // Generate base64 encoded extension value for CRLDistributionPoint
        CRLDistPoint crlDistPoint = new CRLDistPoint(new DistributionPoint[]{new
 DistributionPoint(new DistributionPointName(
                new GeneralNames(new GeneralName(GeneralName.uniformResourceIdentifier,
 "dummycrl.crl"))), null, null)});
        String base64EncodedCDPValue =
 Base64.getEncoder().encodeToString(crlDistPoint.getEncoded());

        CustomExtension cdpCustomExtension = new CustomExtension()
                .withValue(base64EncodedCDPValue)
                .withObjectIdentifier("2.5.29.31"); // CRLDistributionPoint Extension
 OID

        // Generate EKU
        ExtendedKeyUsage eku = new ExtendedKeyUsage()
                .withExtendedKeyUsageObjectIdentifier("1.0.18013.5.1.2"); // EKU value
 reserved for mDL DS

        // Set KeyUsage, ExtendedKeyUsage, IssuerAlternativeName, CRL Distribution
 Point extensions to api-passthrough
        Extensions extensions = new Extensions()
                .withCustomExtensions(Arrays.asList(customKeyUsageExtension,
 ianCustomExtension, cdpCustomExtension))
                .withExtendedKeyUsage(Arrays.asList(eku));
        apiPassthrough.setExtensions(extensions);
        req.setApiPassthrough(apiPassthrough);

        // Issue the certificate.
```

```
        IssueCertificateResult result = null;
        try {
            result = client.issueCertificate(req);
        } catch (LimitExceededException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (MalformedCSRException ex) {
            throw ex;
        }

        // Get and display the certificate ARN.
        String arn = result.getCertificateArn();
        System.out.println("mDL DS Certificate Arn: " + arn);
    }
}
```

建構您的 解決方案 AWS 私有 CA

AWS 私有 CA 可讓您完全控制組織的私有 PKI （公有金鑰基礎設施），從根憑證授權機構 (CA) 延伸到次級 CAs 到終端實體憑證。對於安全、可維護、可擴展且符合組織需求的 PKI 而言，詳盡的規劃非常重要。本節提供設計 CA 階層、管理私有 CA 和私有終端實體憑證生命週期，及套用安全最佳實務的指導。

本節說明如何在建立私有憑證授權機構 (CA) 之前 AWS 私有 CA 準備使用 。它還說明了透過線上憑證狀態通訊協定 (OCSP) 或憑證撤銷清單 (CRL) 新增撤銷支援的選項。

此外，您應該判斷您的組織是否偏好在內部部署託管其私有根 CA 憑證，而不是使用 AWS。在這種情況下，您需要在使用前設定並保護自我管理的私有 PKI AWS 私有 CA。在此案例中，您會在 中建立由外部父 CA AWS 私有 CA 支援的次級 CA AWS 私有 CA。如需詳細資訊，請參閱安裝由外部父 CA 簽署的次級 CA 憑證。

主題

- 設計 CA 階層
- 管理私有 CA 生命週期
- 規劃您的 AWS 私有 CA 憑證撤銷方法
- 了解 AWS 私有 CA CA 模式
- 在 中規劃彈性 AWS 私有 CA

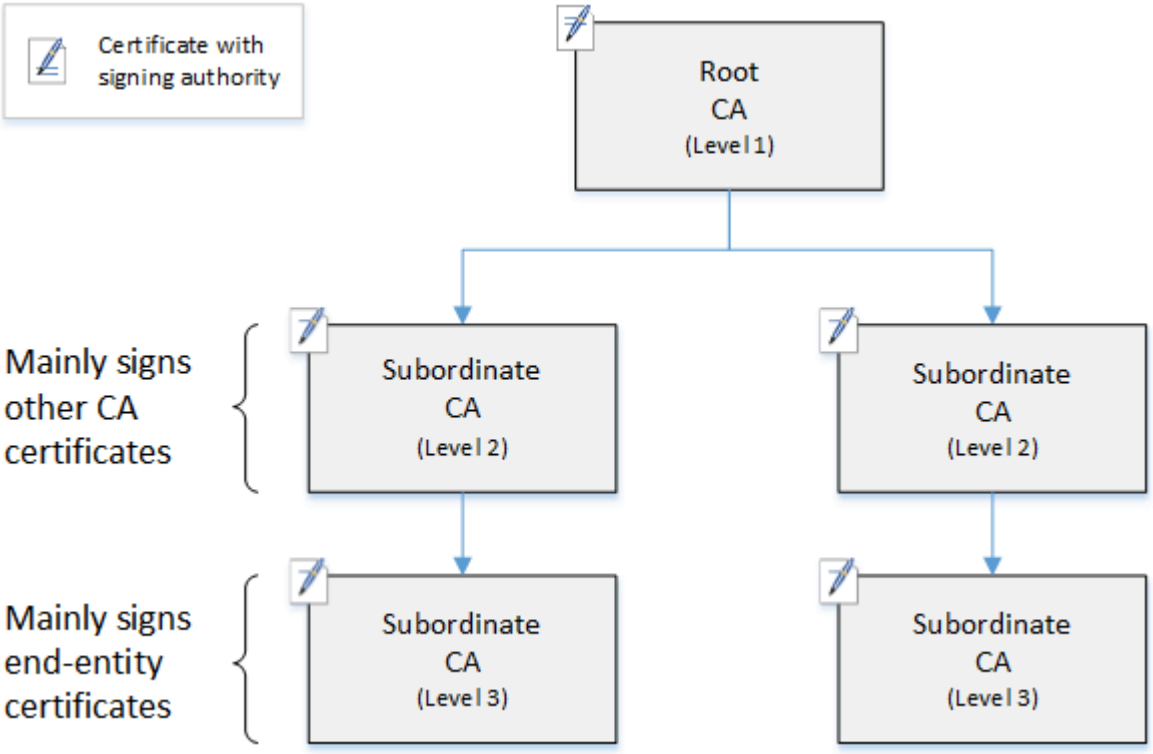
設計 CA 階層

使用 AWS 私有 CA，您可以建立最多五個層級的憑證授權單位階層。位於階層樹狀結構頂端的根 CA，可以有任何數量的分支。根 CA 在每個分支上最多可以有四個層級的次級 CA。您也可以建立多個階層，每個階層都具有自己的根。

設計完善的 CA 階層具有下列優點：

- 適用於每個 CA 的精細安全控制
- 具有更佳負載平衡與安全的管理任務分工
- 使用有限且可撤銷信任的 CA 進行日常操作
- 有效期間和憑證路徑限制

下圖說明簡單的三層 CA 階層。



樹狀結構中的每個 CA，都受到一個具有簽署授權機構的 X.509 v3 憑證支援 (以筆與紙張圖示表示)。這表示其做為 CA 可以簽署其他從屬於其下方的憑證。當 CA 簽署較低層級的 CA 憑證時，會對已簽署的憑證授予有限且可撤銷的授權。層級 1 中的根 CA，會簽署層級 2 中的高階次級 CA 憑證。這些 CA 會依次簽署層級 3 中 CA 的憑證，而這些 CA 則是由管理終端實體憑證的 PKI (公有金鑰基礎設施) 管理員所使用。

CA 階層中的安全，應在樹狀目錄的頂端設為最強。此設定可保護根 CA 憑證及其私有金鑰。根 CA 會錨定所有次級 CA 及其下方終端實體憑證的信任。雖然對終端實體憑證造成的危害會導致局部損害，但對根造成的危害會破壞整個 PKI 中的信任。根和高階次級 CAs只會不常使用 （通常用於簽署其他 CA 憑證）。因此，這些 CA 會受到嚴格控制與稽核，以確保降低危害風險。階層中的較低層級，對安全方面的限制則較少。這種方法可讓您執行例行管理任務，例如發行及撤銷使用者、電腦主機和軟體服務的終端實體憑證。

> **Note**
>
> 使用根 CA 簽署次級憑證是罕見事件，只發生在少數情況下：
>
> - 建立 PKI 時
> - 需要更換高階憑證授權機構時
> - 需要設定憑證撤銷清單 (CRL) 或線上憑證狀態通訊協定 (OCSP) 回應程式時

根和其他高階 CA，需要高度安全的操作程序和存取控制通訊協定。

主題

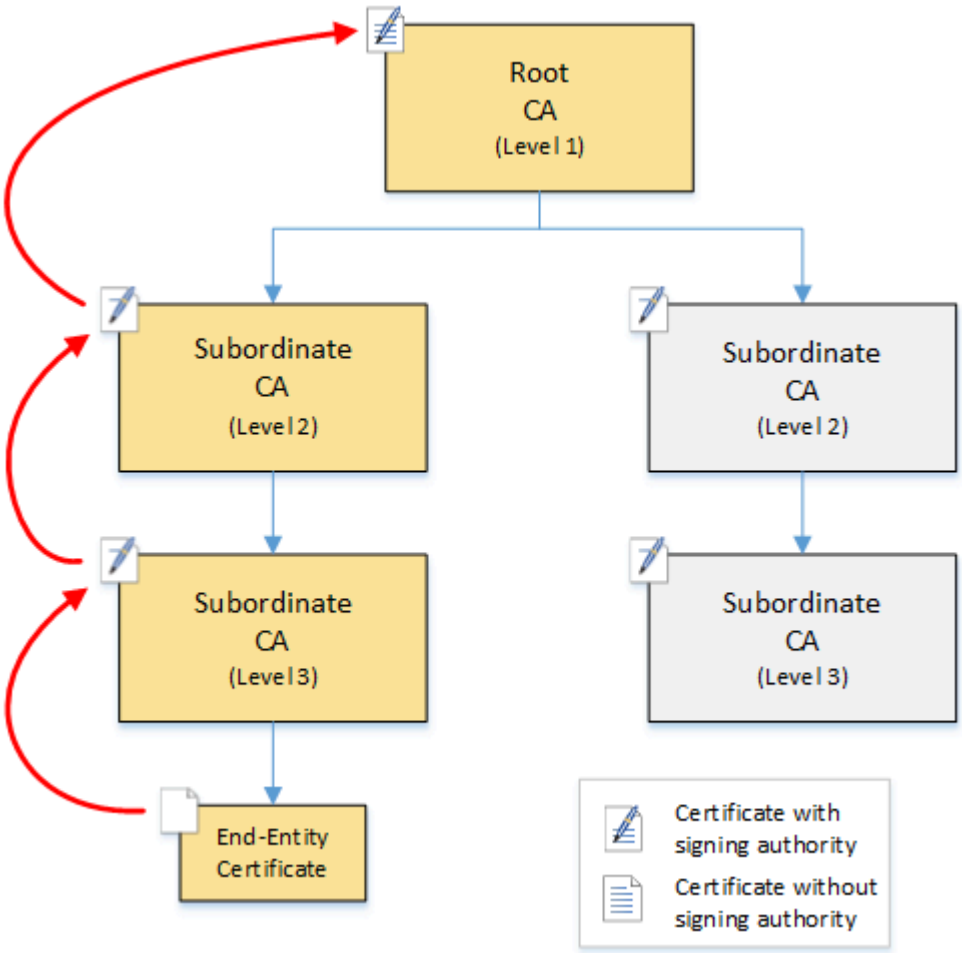
- 驗證終端實體憑證
- 規劃 CA 階層的結構
- 設定認證路徑的長度限制

驗證終端實體憑證

終端實體憑證會從透過次級 CA 回到根 CA 的認證路徑取得信任。當 Web 瀏覽器或其他用戶端提供終端實體憑證時，其會嘗試建構信任鏈。例如，其可能會檢查憑證的「發行者辨別名稱」和「主體辨別名稱」是否與發行 CA 憑證的對應欄位相符。接著會繼續在階層的每個連續層級往上繼續比對，直到用戶端到達其信任存放區中所包含的信任根目錄為止。

信任存放區是瀏覽器或作業系統所包含的信任 CA 資源庫。針對私有 PKI，您組織的 IT 必須確保每個瀏覽器或系統先前已將私有根 CA 新增至其信任存放區。否則會無法驗證認證路徑，而導致用戶端錯誤。

下圖顯示在提供終端實體 X.509 憑證時，瀏覽器會遵循的驗證路徑。請注意，終端實體憑證缺少簽署授權機構，並且僅可用於驗證擁有該憑證的實體。



瀏覽器檢查終端實體憑證。瀏覽器發現憑證提供次級 CA (層級 3) 的簽章做為其信任登入資料。次級 CA 的憑證必須包含在相同 PEM 檔案中。或者，也可以位於包含組成信任鏈結之憑證的個別檔案中。找到這些資訊後，瀏覽器檢查次級 CA (層級 3) 的憑證，並發現其提供次級 CA (層級 2) 的簽章。反過來，次級 CA (層級 2) 會提供來自根 CA (層級 1) 的簽章做為其信任登入資料。如果瀏覽器發現預先安裝在其信任存放區中的私有根 CA 憑證複本，則會將終端實體憑證驗證為受信任。

一般來說，瀏覽器也會根據憑證撤銷清單 (CRL) 來檢查每個憑證。已過期、已撤銷或設定錯誤的憑證會遭拒絕，且驗證會失敗。

規劃 CA 階層的結構

一般而言，您的 CA 階層應該要反映出組織的結構。路徑長度 （即 CA 層級數量） 的目標是不超過委派管理和安全角色所需的值。將 CA 新增至階層，表示增加認證路徑中的憑證數量，而這會增加驗證時間。在驗證終端實體憑證時，將路徑長度保持在最短，也會減少從伺服器傳送至用戶端的憑證數量。

理論上，沒有 pathLenConstraint 參數的根 CA 可以授權不限層級的次級 CAs。次級 CA 可以有與其內部 configuration 允許的次級 CAs 數量一樣多。 AWS 私有 CA 受管階層支援最多五個層級深的 CA 認證路徑。

設計完善的 CA 結構有幾項優點：

- 不同組織單位的個別管理控制
- 將存取權委派給次級 CA 的能力
- 可透過額外安全控制來保護較高層級 CA 的階層式結構

有兩種常見的 CA 結構可以達成上述所有項目：

- 兩個 CA 層級：根 CA 和次級 CA

  這是最簡單的 CA 結構，可允許根 CA 和次級 CA 的個別系統管理、控制和安全政策。您可以在為根 CA 維護嚴格控制和政策的同時，給予次級 CA 較寬鬆的存取權。後者用於大量發行終端實體憑證。

- 三個 CA 層級：根 CA 和兩個次級 CA 層級

  與上述情況類似，此結構會增加一層額外的 CA，以進一步將根 CA 與低層 CA 操作分開。中間層 CA 僅用於簽署執行終端實體憑證發行的次級 CA。

較不常見的 CA 結構包括下列項目：

- 四個或更多 CA 層級

  雖然比三層階層較少見，但具有四層或更多層級的 CA 階層仍可能存在，且可能需要允許管理委派。

- 一個 CA 層級：僅限根 CA

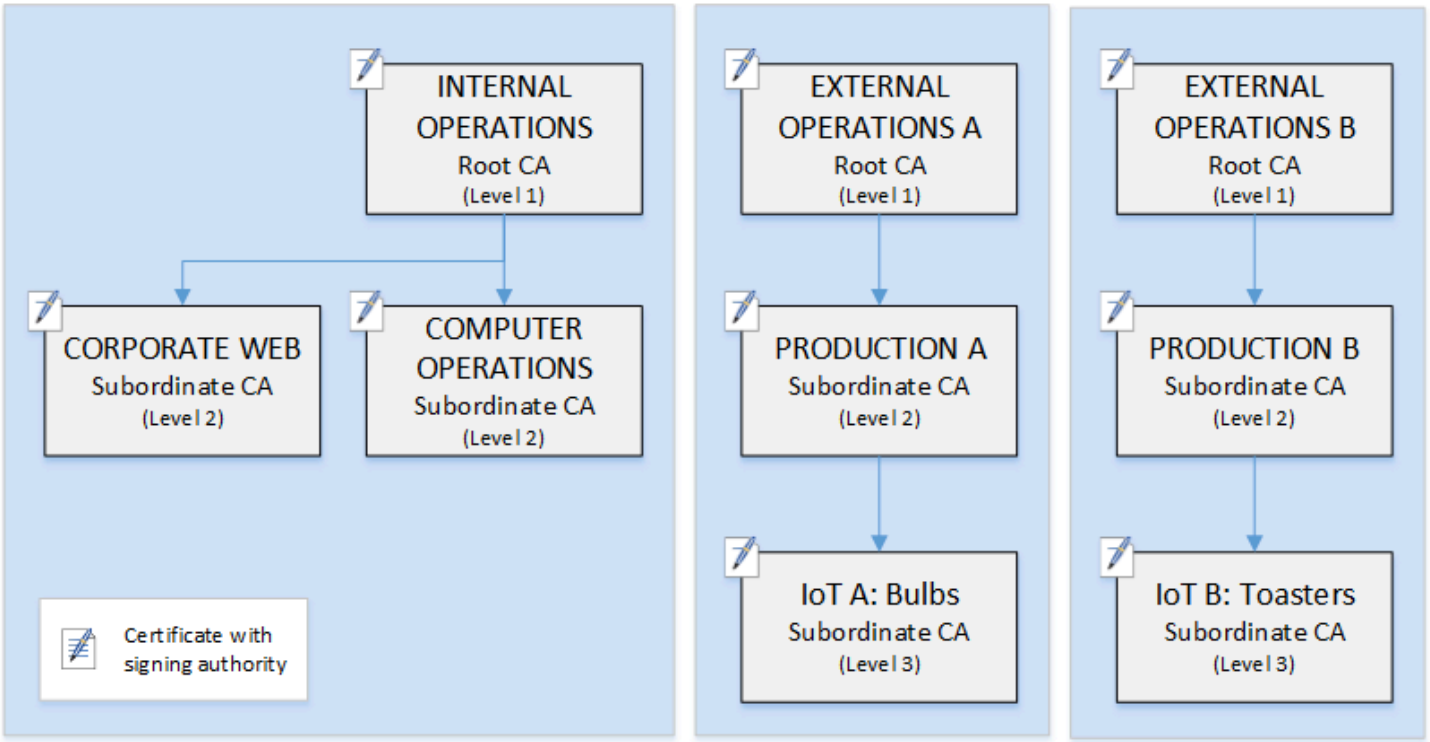
  此結構通常用於進行開發及測試，而不需要完整的信任鏈時。在生產環境中使用並不常見。此外，其違反了針對根 CA 及發行終端實體憑證的 CA 維護個別安全政策的最佳實務。

  不過，如果您已經直接從根 CA 發行憑證，則可以遷移至 AWS 私有 CA。這樣做可提供與使用透過 OpenSSL 或其他軟體管理的根 CA 相比的安全性和控制優勢。

製造商的私有 PKI 範例

在此範例中，一家虛構的科技公司生產兩種物聯網產品：智慧型燈泡和智慧型烤麵包機。在生產過程中，會將終端實體憑證發行給每部裝置，以便裝置能夠透過網際網路與製造商安全通訊。公司 PKI 也會確保其電腦基礎設施的安全，包括內部網站和各種自我裝載電腦服務，這些服務負責執行財務和業務營運。

因此，CA 階層會密切為業務的這些管理和操作層面建立模型。



此階層包含三個根目錄，一個用於內部操作，另外兩個用於外部操作 (每個產品系列都有一個根 CA)。它還說明了多個認證路徑長度，其中兩層 CA 用於內部操作，三層用於外部操作。

在外部操作端使用分離的根 CAs和其他次級 CA 層，是滿足商業和安全性需求的設計決策。具備多個 CA 樹狀結構時，PKI 就可以適應未來的企業重新組織、撤資或併購。在發生變更時，整個根 CA 階層都可以隨著其保護的部門而整潔地移動。由於具備了兩個層級的次級 CA，根 CA 擁有高度隔離的效果，可與負責大量簽署數千或數百萬個製造項目之憑證的層級 3 CA 區隔。

在內部方面，企業 Web 和內部電腦操作可完成兩個層級的階層。這些層級可讓 Web 管理員和操作工程師獨立管理其工作領域的憑證發行。將 PKI 區分為不同的功能領域是安全最佳實務，可保護每個領域免受可能影響另一個網域的危害。Web 管理員會發行終端實體憑證供整個公司的 Web 瀏覽器使用，以驗證及加密內部網站上的通訊。操作工程師會發行終端實體憑證，以互相驗證資料中心主機和電腦服務。此系統會藉由在 LAN 上加密資料以協助保護敏感資料的安全。

設定認證路徑的長度限制

CA 階層架構的結構，由每個憑證所包含的「基本限制條件延伸」來定義及強制執行。延伸會定義兩個限制條件：

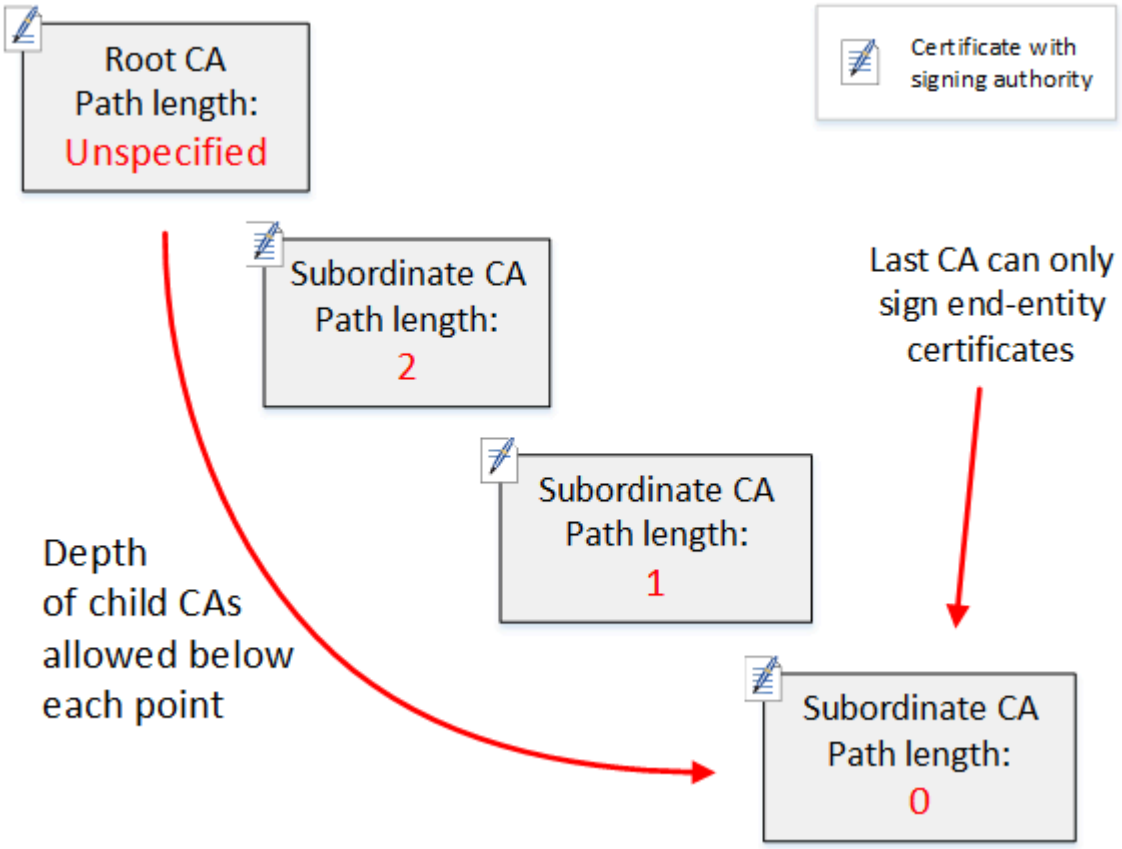
- cA – 憑證是否定義 CA。如果此值為 False (預設值)，則憑證為終端實體憑證。
- pathLenConstraint – 可存在於有效信任鏈中的較低層級次級 CAs 數目上限。不會計算終端實體憑證，因為它不是 CA 憑證。

根 CA 憑證需要具備最大的彈性，且不包含路徑長度限制條件。這可讓根定義任何長度的認證路徑。

> **Note**
>
> AWS 私有 CA 會將認證路徑限制為五個層級。

次級 CA 的 pathLenConstraint 值會等於或大於零，取決於階層放置中的位置與所需功能。例如，在具有三個 CA 的階層中，不會對根 CA 指定任何路徑限制條件。第一個次級 CA 的路徑長度為 1，因此可以簽署子 CA。每個子 CA 都必須具有值為零的 pathLenConstraint。這表示其可以簽署終端實體憑證，但無法發行其他 CA 憑證。限制建立新 CA 的能力，是一項重要的安全控制。

下圖說明了階層中受限授權能力的這種傳播方式。



在四層階層中，根目錄是不受限制的 (如同往常)。但第一個次級 CA 的 `pathLenConstraint` 值為 2，這會限制其子 CA 深入超過兩個層級。因此，針對有效的認證路徑，限制條件值必須在接下來的兩個層級遞減為零。如果 Web 瀏覽器遇到來自此分支的終端實體憑證，且路徑長度大於 4，則驗證就會失敗。這類憑證可能是意外建立的 CA、設定錯誤的 CA 或未經授權的發行所造成。

使用 範本管理路徑長度

AWS 私有 CA 提供用於發行根憑證、次級憑證和終端實體憑證的範本。這些範本納入了基本限制條件值 (包括路徑長度) 的最佳實務。範本包括下列項目：

- RootCACertificate/V1
- SubordinateCACertificate_PathLen0/V1
- SubordinateCACertificate_PathLen1/V1
- SubordinateCACertificate_PathLen2/V1
- SubordinateCACertificate_PathLen3/V1
- EndEntityCertificate/V1

如果您嘗試建立路徑長度大於或等於其發行 CA 憑證路徑長度的 CA，則 `IssueCertificate` API 將傳回錯誤。

如需憑證範本的詳細資訊，請參閱使用 AWS 私有 CA 憑證範本。

使用 自動化 CA 階層設定 AWS CloudFormation

當您完成 CA 階層的設計時，您可以進行測試，並使用 AWS CloudFormation 範本將其投入生產環境。如需這類範本的範例，請參閱CloudFormation 《 使用者指南》中的宣告私有 CA 階層。

管理私有 CA 生命週期

CA 憑證有固定的生命週期或有效期間。當 CA 憑證過期時，在 CA 階層中由次級 CA 直接或間接發行的所有憑證都會失效。您可以預先進行規劃，以避免 CA 憑證過期。

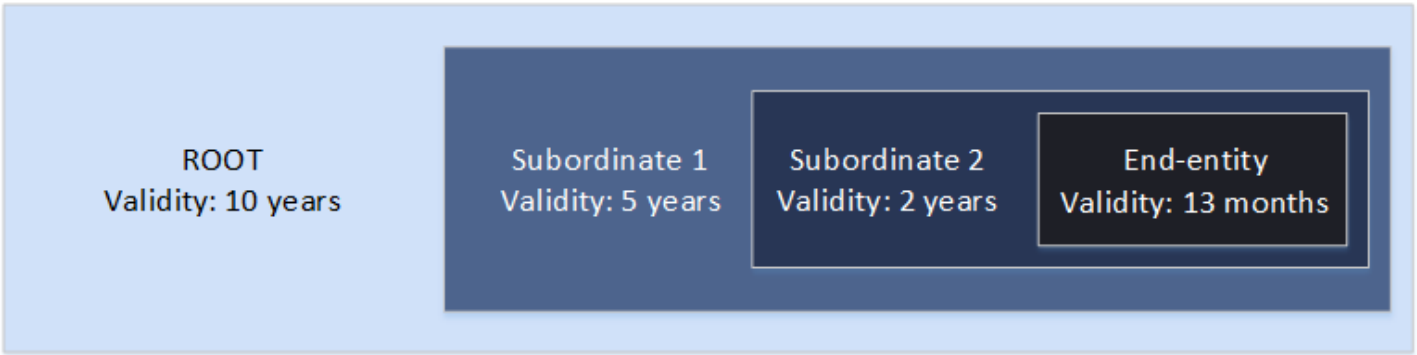
選擇有效期間

X.509 憑證的有效期間，是必要的基本憑證欄位。其會決定發行 CA 認證憑證可以受信任的時間範圍 (除了撤銷之外)。(根憑證是自我簽署的，會認證其本身的有效期間。)

AWS 私有 CA 和 AWS Certificate Manager 協助設定憑證有效期間，受下列限制約束：

- 由 管理的憑證，其有效期間 AWS 私有 CA 必須小於或等於發行憑證的 CA 有效期間。換句話說，子 CA 和終端實體憑證的有效期間不能超過其父憑證。嘗試使用 `IssueCertificate` API 來發行有效期間長於或等於父 CA 的 CA 憑證會失敗。
- 由 發行和管理的憑證 AWS Certificate Manager (ACM 產生私有金鑰的憑證） 的有效期為 13 個月 (395 天）。ACM 會管理這些憑證的續約程序。如果您使用 AWS 私有 CA 直接發行憑證，您可以選擇任何有效期間。

下圖顯示巢狀有效期間的一般組態。根憑證的有效期間最長，終端實體憑證相對較短，而次級 CA 的有效期間範圍則位於這兩者之間。



在您規劃 CA 階層時，請判斷 CA 憑證的最佳生命週期。從想要發行之終端實體憑證的所需生命週期向後推算。

終端實體憑證

終端實體憑證應具有適用於使用案例的有效期。短暫的生命週期，可在憑證私有金鑰遺失或遭竊時，盡量減少洩漏憑證的機會。然而，短暫的生命周期也代表必須頻繁續約。若無法續約即將到期的憑證，就可能會導致停機。

如果存在安全性漏洞，以分發方式使用終端實體憑證也可能會造成後勤問題。您的規劃應該考量更新和分配憑證、撤銷遭危害的憑證，以及撤銷傳播至依賴憑證之用戶端的速度。

透過 ACM 發行之終端實體憑證的預設有效期間為 13 個月 (395 天）。在 中 AWS 私有 CA，您可以使用 `IssueCertificate` API 來套用任何有效期間，只要其小於發行 CA 的有效期間即可。

次級 CA 憑證

次級 CA 憑證的有效期間，應該比其發行的憑證要長得多。CA 憑證有效性的良好範圍，是其發行之任何子 CA 憑證或終端實體憑證的二到五倍。例如，假設您有兩個層級的 CA 階層 (根 CA 和一個次級 CA)。如果您想要發行生命週期為一年的終端實體憑證，您可以將次級發行 CA 生命週期設為三年。這是 中次級 CA 憑證的預設有效期間 AWS 私有 CA。次級 CA 憑證可以在不取代根 CA 憑證的情況下進行變更。

根憑證

根 CA 憑證的變更會影響整個 PKI (公有金鑰基礎設施)，並要求您更新所有相依的用戶端作業系統和瀏覽器信任存放區。若要將操作影響降至最低，您應該為根憑證選擇較長的有效期間。根憑證的 AWS 私有 CA 預設值為 10 年。

管理 CA 接續

您有兩種方式可以管理 CA 繼承：取代舊的 CA，或使用新的有效期間重新發行 CA。

取代舊 CA

若要取代舊 CA，您可以建立新的 CA，並將其鏈結至相同的父 CA。之後，您就可以從新的 CA 發出憑證。

從新 CA 發行的憑證，具有新 CA 鏈結。建立新的 CA 後，您就可以停用舊 CA，以防止其發行新的憑證。停用時，舊 CA 支援撤銷從 CA 發行的舊憑證，如果設定為這樣做，它會繼續透過 OCSP 和/或憑證撤銷清單 (CRLs) 驗證憑證。從舊 CA 發出的最後一個憑證到期時，您就可以刪除舊的 CA。您可以為 CA 發行的所有憑證產生稽核報告，以確認所有已發行的憑證都已過期。如果舊 CA 具有次級 CA，

您也必須取代這些次級 CA，因為次級 CA 會與其父 CA 同時或在之前過期。首先，請取代階層中需要取代的最高層級 CA。然後在每個後續的較低層級中，建立替換用的新次級 CA。

AWS 建議您視需要在 CA 的名稱中包含 CAs產生識別符。例如，假設您將第一代 CA 命名為「Corporate Root CA」。當您建立第二代 CA 時，請將其命名為「Corporate Root CA G2」。這種簡單的命名慣例，有助於避免在兩個 CA 都未過期時產生混淆。

這是較佳的 CA 繼承方法，因為會輪替 CA 的私有金鑰。輪替私有金鑰是 CA 金鑰的最佳實務。輪替的頻率應該與金鑰使用頻率成正比：發行較多憑證的 CA，應該更頻繁地進行輪替。

> **Note**
>
> 如果您取代 CA，則無法續約透過 ACM 發行的私有憑證。如果您使用 ACM 進行發行和續約，則必須重新發行 CA 憑證，以延長 CA 的生命週期。

重新發行舊 CA

當 CA 快要過期時，另一種延長其生命週期的方法是使用新的過期日期重新發行 CA 憑證。重新發行會保留所有 CA 中繼資料，並保留現有的私有和公有金鑰。在此案例中，CA 發行的現有憑證鏈和未過期的終端實體憑證會保持有效，直到過期為止。新的憑證發行也可以繼續而不會中斷。若要使用重新發行的憑證更新 CA，請遵循中所述的一般安裝程序安裝 CA 憑證。

> **Note**
>
> 我們建議您取代即將到期的 CA，而不是重新發行憑證，因為輪換至新的金鑰對可獲得安全優勢。

撤銷 CA

您可以透過撤銷 CA 的基礎憑證來撤銷 CA。這也會有效地撤銷 CA 發行的所有憑證。撤銷資訊會透過 OCSP 或 CRL 分發給用戶端。只有在您想要撤銷其所有發行的最終實體和次級 CA 憑證時，才應該撤銷 CA 憑證。

規劃您的 AWS 私有 CA 憑證撤銷方法

當您使用 規劃私有 PKI 時 AWS 私有 CA，應考慮如何處理您不再希望端點信任已發行憑證的情況，例如當端點的私有金鑰公開時。此問題的常見方法是使用短期憑證或設定憑證撤銷。短期憑證會在短時間

內過期，以小時或天為單位，撤銷並不合理，因為憑證在 中會變成無效，大約與通知撤銷端點相同。本節說明 AWS 私有 CA 客戶的撤銷選項，包括組態和最佳實務。

尋找撤銷方法的客戶可以選擇線上憑證狀態通訊協定 (OCSP)、憑證撤銷清單 CRLs) 或兩者。

> **Note**
>
> 如果您在未設定撤銷的情況下建立 CA，您可以隨時稍後進行設定。如需詳細資訊，請參閱在中更新私有 CA AWS 私有憑證授權單位。

- 線上憑證狀態通訊協定 (OCSP)

  AWS 私有 CA 提供全受管 OCSP 解決方案，通知端點憑證已撤銷，而客戶不需要自行操作基礎設施。客戶可以使用 AWS 私有 CA 主控台、API、CLI 或透過 ，透過單一操作在新的或現有的 CAs 上啟用 OCSP CloudFormation。雖然 CRLs存放在端點上並進行處理，而且可能會過時，但 OCSP 儲存和處理需求會在回應程式後端同步處理。

  當您為 CA 啟用 OCSP 時， 會將 OCSP 回應程式的 URL AWS 私有 CA 包含在發行的每個新憑證的 Authority Information Access (AIA) 延伸中。延伸可讓 Web 瀏覽器等用戶端查詢回應者，並判斷是否可以信任終端實體或次級 CA 憑證。回應者傳回以密碼編譯方式簽署的狀態訊息，以確保其真實性。

  OCSP AWS 私有 CA 回應程式符合 RFC 5019。

  OCSP 考量事項

  - OCSP 狀態訊息是使用與發行 CA 設定為使用的相同簽署演算法來簽署。在主控台中建立的 AWS 私有 CA CAs 預設會使用 SHA256WITHRSA 簽章演算法。您可以在 CertificateAuthorityConfiguration API 文件中找到其他支援的演算法。
  - 如果啟用 OCSP 回應程式，APIPassthrough 和 CSRPassthrough 憑證範本將無法與 AIA 延伸模組搭配使用。
  - 受管 OCSP 服務的端點可在公有網際網路上存取。想要 OCSP 但不想擁有公有端點的客戶，將需要操作自己的 OCSP 基礎設施。
- 憑證撤銷清單 (CRLs)

  憑證撤銷清單 (CRL) 是一個檔案，其中包含在其排定的過期日期之前撤銷的憑證清單。CRL 包含不應再受信任的憑證清單、撤銷原因和其他相關資訊。

當您設定憑證授權機構 (CA) 時，您可以選擇 AWS 私有 CA 是否建立完整或分割的 CRL。您的選擇決定憑證授權單位可以發行和撤銷的憑證數量上限。如需詳細資訊，請參閱 AWS 私有 CA 配額。

CRL 考量事項

- 記憶體和頻寬考量：由於本機下載和處理需求，CRLs 需要比 OCSP 更多的記憶體。不過，相較於 OCSP，CRLs 可能會透過快取撤銷清單來減少網路頻寬，而不是檢查每個連線的狀態。對於記憶體受限的裝置，例如特定 IoT 裝置，請考慮使用分割CRLs。
- 變更 CRL 類型：從完整變更為分割的 CRL 時， AWS 私有 CA 會視需要建立新的分割區，並將 IDP 延伸新增至所有 CRLs，包括原始分割區。從分割變更為僅完成單一 CRL 更新，並防止日後撤銷與先前分割區相關聯的憑證。

> **Note**
>
> OCSP 和 CRLs撤銷和狀態變更的可用性之間都顯示一些延遲。
>
> - 當您撤銷憑證時，OCSP 回應最多可能需要 60 分鐘才能反映新狀態。一般而言，OCSP 傾向於支援更快速的撤銷資訊分佈，因為與用戶端可以快取數天CRLs 不同，用戶端通常不會快取 OCSP 回應。
> - CRL 通常在憑證撤銷後約 30 分鐘更新。如果 CRL 更新因任何原因失敗， AWS 私有 CA 會每 15 分鐘進一步嘗試一次。

撤銷組態的一般要求

下列要求適用於所有撤銷組態。

- 停用 CRL 或 OCSP 的組態必須只包含 `Enabled=False` 參數，如果包含其他參數 (例如 `CustomCname` 或 `ExpirationInDays`)，則會失敗。
- 在 CRL 組態中， `S3BucketName` 參數必須符合 Amazon Simple Storage Service 儲存貯體命名規則。
- 包含 CRLs或 OCSP 自訂正式名稱 (CNAME) 參數的組態必須符合在 CNAME 中使用特殊字元的 RFC7230 限制。
- 在 CRL 或 OCSP 組態中，CNAME 參數的值不得包含通訊協定字首，例如 "http://" 或 "https://"。

主題

- 設定 的 CRL AWS 私有 CA
- 自訂 的 OCSP URL AWS 私有 CA

設定 的 CRL AWS 私有 CA

在您將憑證撤銷清單 (CRL) 設定為 CA 建立程序的一部分之前，可能需要一些先前的設定。本節說明在建立已連接 CRL 的 CA 之前，您應該了解的先決條件和選項。

如需使用線上憑證狀態通訊協定 (OCSP) 做為 CRL 的替代方案或補充，請參閱 Certificate revocation options和 自訂 的 OCSP URL AWS 私有 CA。

主題

- CRL 類型
- CRL 結構
- Amazon S3 中 CRLs存取政策
- 使用 CloudFront 啟用 S3 封鎖公開存取 (BPA)
- 判斷 CRL 分佈點 (CDP) URI
-

CRL 類型

- 完成 - 預設設定。 會為 CA 發行且已撤銷的所有未過期憑證 AWS 私有 CA 維護單一、未分割的 CRL 檔案。問題的每個憑證 AWS 私有 CA 都會透過其 CRL 分佈點 (CDP) 延伸繫結至特定 CRL，如 RFC 5280 所定義。對於每個啟用完整 CRL 的 CA，您最多可以擁有 100 萬個私有憑證。如需詳細資訊，請參閱AWS 私有 CA 配額。
- 分割 - 與完成 CRLs相比，分割的 CRLs會大幅增加私有 CA 可以發行的憑證數量，並讓您免於頻繁輪換 CAs。

> **Important**
>
> 使用分割CRLs 時，您必須驗證 CRL 相關聯的發行分佈點 (IDP) URI 符合憑證的 CDP URI，以確保已擷取正確的 CRL。 會將 IDP 延伸 AWS 私有 CA 標記為關鍵，您的用戶端必須能夠處理。

CRL 結構

每個 CRL 皆為 DER 編碼檔案。若要下載檔案並使用 OpenSSL 進行檢視，請使用類似以下的命令：

```
openssl crl -inform DER -in path-to-crl-file -text -noout
```

CRL 具有下列格式：

```
Certificate Revocation List (CRL):
          Version 2 (0x1)
      Signature Algorithm: sha256WithRSAEncryption
          Issuer: /C=US/ST=WA/L=Seattle/O=Example Company CA/OU=Corporate/
CN=www.example.com
          Last Update: Feb 26 19:28:25 2018 GMT
          Next Update: Feb 26 20:28:25 2019 GMT
          CRL extensions:
              X509v3 Authority Key Identifier:
                  keyid:AA:6E:C1:8A:EC:2F:8F:21:BC:BE:80:3D:C5:65:93:79:99:E7:71:65

              X509v3 CRL Number:
                  1519676905984
  Revoked Certificates:
      Serial Number: E8CBD2BEDB122329F97706BCFEC990F8
          Revocation Date: Feb 26 20:00:36 2018 GMT
          CRL entry extensions:
              X509v3 CRL Reason Code:
                  Key Compromise
      Serial Number: F7D7A3FD88B82C6776483467BBF0B38C
          Revocation Date: Jan 30 21:21:31 2018 GMT
          CRL entry extensions:
              X509v3 CRL Reason Code:
                  Key Compromise
      Signature Algorithm: sha256WithRSAEncryption
           82:9a:40:76:86:a5:f5:4e:1e:43:e2:ea:83:ac:89:07:49:bf:
           c2:fd:45:7d:15:d0:76:fe:64:ce:7b:3d:bb:4c:a0:6c:4b:4f:
           9e:1d:27:f8:69:5e:d1:93:5b:95:da:78:50:6d:a8:59:bb:6f:
           49:9b:04:fa:38:f2:fc:4c:0d:97:ac:02:51:26:7d:3e:fe:a6:
           c6:83:34:b4:84:0b:5d:b1:c4:25:2f:66:0a:2e:30:f6:52:88:
           e8:d2:05:78:84:09:01:e8:9d:c2:9e:b5:83:bd:8a:3a:e4:94:
           62:ed:92:e0:be:ea:d2:59:5b:c7:c3:61:35:dc:a9:98:9d:80:
           1c:2a:f7:23:9b:fe:ad:6f:16:7e:22:09:9a:79:8f:44:69:89:
           2a:78:ae:92:a4:32:46:8d:76:ee:68:25:63:5c:bd:41:a5:5a:
           57:18:d7:71:35:85:5c:cd:20:28:c6:d5:59:88:47:c9:36:44:
```

```
        53:55:28:4d:6b:f8:6a:00:eb:b4:62:de:15:56:c8:9c:45:d7:
        83:83:07:21:84:b4:eb:0b:23:f2:61:dd:95:03:02:df:0d:0f:
        97:32:e0:9d:38:de:7c:15:e4:36:66:7a:18:da:ce:a3:34:94:
        58:a6:5d:5c:04:90:35:f1:8b:55:a9:3c:dd:72:a2:d7:5f:73:
        5a:2c:88:85
```

> **Note**
>
> 只有在發出參考 CRL 的憑證之後，CRL 才會存放在 Amazon S3 中。在此之前，Amazon S3 儲存貯體中只會顯示 `acm-pca-permission-test-key` 檔案。

Amazon S3 中 CRLs存取政策

如果您打算建立 CRL，則需要準備 Amazon S3 儲存貯體以將其存放在 中。 AWS 私有 CA 會自動將 CRL 存放在您指定的 Amazon S3 儲存貯體中，並定期更新。如需詳細資訊，請參閱建立儲存貯體。

您的 S3 儲存貯體必須受到連接的 IAM 許可政策保護。授權使用者和服務主體需要`Put`允許 AWS 私有 CA 將物件放入儲存貯體的許可，以及擷取物件的`Get`許可。

> **Note**
>
> IAM 政策組態取決於所 AWS 區域 涉及的 。區域分為兩個類別：
>
> - 預設啟用區域 – 預設為所有啟用的區域 AWS 帳戶。
> - 預設停用區域 – 預設停用但可由客戶手動啟用的區域。
>
> 如需詳細資訊和預設停用區域的清單，請參閱管理 AWS 區域。如需 IAM 內容中服務主體的討論，請參閱AWS 選擇加入區域中的服務主體。
> 當您將 CRLs設定為憑證撤銷方法時， AWS 私有 CA 會建立 CRL 並將其發佈至 S3 儲存貯體。S3 儲存貯體需要允許 AWS 私有 CA 服務主體寫入儲存貯體的 IAM 政策。服務主體的名稱會根據所使用的區域而有所不同，但並非所有可能性都受到支援。

PCA	S3	服務主體
位於相同 區域中的兩者		`acm-pca.amazonaws.com`
已啟用	已啟用	`acm-pca.amazonaws.com`

PCA	S3	服務主體
Disabled	已啟用	acm-pca.*Region*.amazonaws.com
已啟用	Disabled	不支援

預設政策不會對 CA 套用SourceArn限制。我們建議您套用較不寬鬆的政策，例如下列政策，這會限制對特定 AWS 帳戶和特定私有 CA 的存取。或者，您可以使用 aws：SourceOrgID 條件金鑰來限制對 中特定組織的存取 AWS Organizations。如需儲存貯體政策的詳細資訊，請參閱 Amazon Simple Storage Service 的儲存貯體政策。

如果您選擇允許預設政策，您稍後可以隨時修改它。

使用 CloudFront 啟用 S3 封鎖公開存取 (BPA)

新的 Amazon S3 儲存貯體預設為啟用封鎖公開存取 (BPA) 功能。BPA 包含在 Amazon S3 安全最佳實務中，是一組存取控制，客戶可用來微調對 S3 儲存貯體中物件和整個儲存貯體的存取。當 BPA 處於作用中且設定正確時，只有經過授權和驗證 AWS 的使用者才能存取儲存貯體及其內容。

AWS 建議在所有 S3 儲存貯體上使用 BPA，以避免敏感資訊暴露給潛在的對手。不過，如果您的 PKI 用戶端透過公有網際網路 （即未登入 AWS 帳戶時） 擷取 CRLs，則需要額外的規劃。本節說明如何使用內容交付網路 (CDN) Amazon CloudFront 設定私有 PKI 解決方案，以提供 CRLs而不需要對 S3 儲存貯體進行身分驗證的用戶端存取。

> **Note**
>
> 使用 CloudFront 會在 AWS 您的帳戶產生額外費用。如需詳細資訊，請參閱 Amazon CloudFront 定價。
> 如果您選擇將 CRL 存放在已啟用 BPA 的 S3 儲存貯體中，而且不使用 CloudFront，則必須建置另一個 CDN 解決方案，以確保 PKI 用戶端可以存取 CRL。

設定 BPA 的 CloudFront

建立可存取私有 S3 儲存貯體的 CloudFront 分佈，並可為未驗證的用戶端提供 CRLs。

設定 CRL 的 CloudFront 分佈

1. 使用《Amazon CloudFront 開發人員指南》中 Amazon CloudFront建立分佈中的程序建立新的 CloudFront 分佈。

   完成程序時，請套用下列設定：

   - 在原始伺服器網域名稱中，選擇您的 S3 儲存貯體。
   - 針對限制儲存貯體存取選擇是。
   - 選擇為原始存取身分建立新身分。
   - 選擇是，在授予儲存貯體的讀取許可下更新儲存貯體政策。

   > **Note**
   >
   > 在此程序中，CloudFront 會修改儲存貯體政策，以允許其存取儲存貯體物件。請考慮編輯此政策，僅允許存取 `crl` 資料夾下的物件。

2. 分佈初始化後，請在 CloudFront 主控台中尋找其網域名稱，並將其儲存以進行下一個程序。

   > **Note**
   >
   > 如果您的 S3 儲存貯體是在 us-east-1 以外的區域中新建立的，當您透過 CloudFront 存取已發佈的應用程式時，可能會收到 HTTP 307 暫時重新導向錯誤。儲存貯體的地址可能需要幾個小時才能傳播。

設定 BPA 的 CA

設定新的 CA 時，請將別名納入 CloudFront 分佈。

使用 CloudFront 的 CNAME 設定 CA

- 使用 建立您的 CA在 中建立私有 CA AWS 私有 CA。

  當您執行程序時，撤銷檔案`revoke_config.txt`應包含下列幾行，以指定非公有 CRL 物件，並為 CloudFront 中的分發端點提供 URL：

  ```
  "S3ObjectAcl":"BUCKET_OWNER_FULL_CONTROL",
   "CustomCname":"abcdef012345.cloudfront.net"
  ```

之後，當您使用此 CA 發行憑證時，它們將包含如下所示的區塊：

```
X509v3 CRL Distribution Points:
 Full Name:
 URI:http://abcdef012345.cloudfront.net/crl/01234567-89ab-
cdef-0123-456789abcdef.crl
```

> **Note**
>
> 如果您有此 CA 發行的舊憑證，他們將無法存取 CRL。

判斷 CRL 分佈點 (CDP) URI

如果您需要在工作流程中使用 CRL 分佈點 (CDP) URI，您可以使用該憑證上的 CRL URI 發行憑證，或使用下列方法。這僅適用於完整的 CRLs。分割CRLs 會附加隨機的 GUID。

如果您使用 S3 儲存貯體做為 CA 的 CRL 分佈點 (CDP)，CDP URI 可以是下列其中一種格式。

- `http://amzn-s3-demo-bucket.s3.region-code.amazonaws.com/crl/CA-ID.crl`
- `http://s3.region-code.amazonaws.com/amzn-s3-demo-bucket/crl/CA-ID.crl`

如果您已使用自訂 CNAME 設定 CA，CDP URI 將包含 CNAME，例如，`http://alternative.example.com/crl/CA-ID.crl`

根據預設， 會使用IPv4-only`amazonaws.com`端點 AWS 私有 CA 寫入 CDP 擴充功能。若要透過 IPv6 使用 CRLs，請執行下列其中一個步驟，以便使用指向 S3 雙堆疊端點的 URLs 寫入 CDPs：

- 將您的 CRL 自訂名稱設定為 S3 dualstack 端點網域。例如 `bucketname.s3.dualstack.region-code.amazonaws.com`
- 在相關的 S3 雙堆疊端點設定您自己的 CNAME DNS 記錄，然後將其用作 CRL 自訂名稱

自訂 的 OCSP URL AWS 私有 CA

> **Note**
>
> 本主題適用於想要自訂線上憑證狀態協定 (OCSP) 回應者端點的公有 URL 以建立品牌或其他用途的客戶。如果您計劃使用 AWS 私有 CA 受管 OCSP 的預設組態，您可以略過本主題，並遵循設定撤銷中的組態指示。

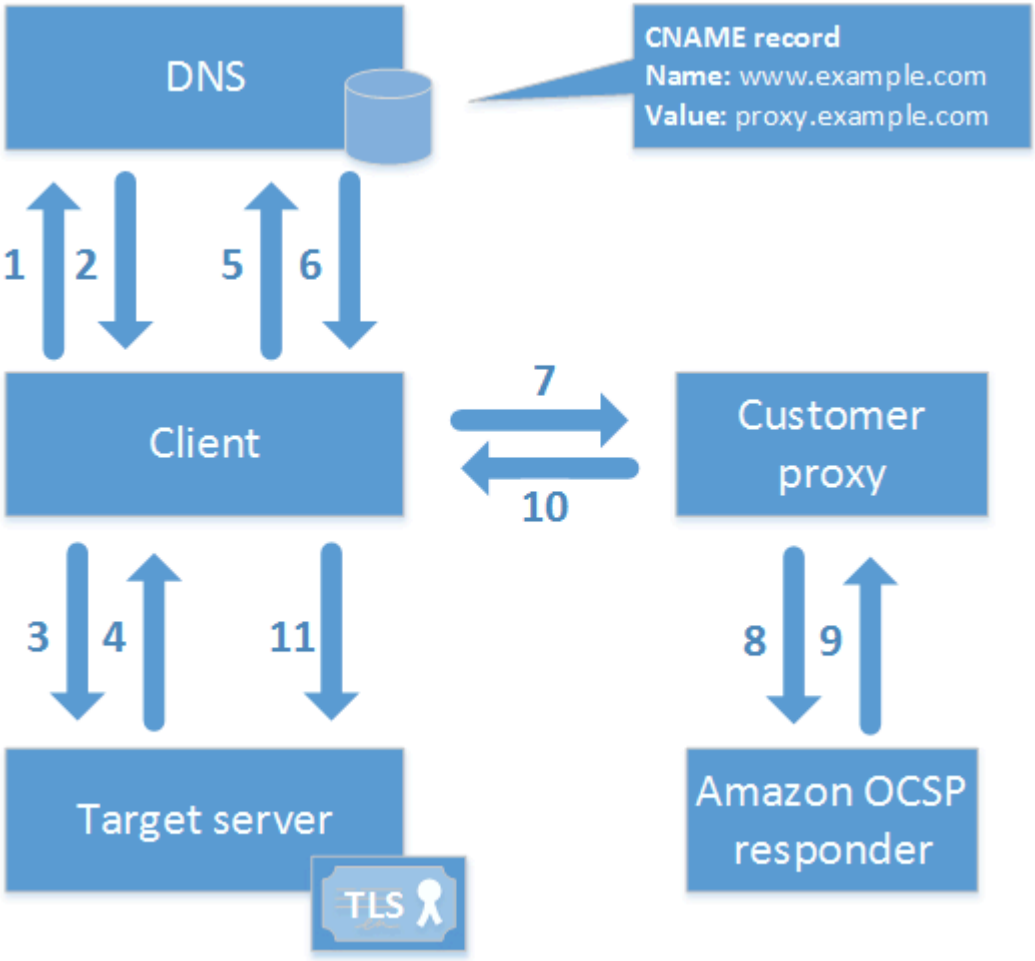
根據預設，當您為 啟用 OCSP 時 AWS 私有 CA，您發行的每個憑證都會包含 OCSP AWS 回應程式的 URL。這可讓請求密碼編譯安全連線的用戶端將 OCSP 驗證查詢直接傳送到 AWS。不過，在某些情況下，建議您在憑證中陳述不同的 URL，同時最終仍向 提交 OCSP 查詢 AWS。

> **Note**
>
> 如需使用憑證撤銷清單 (CRL) 作為 OCSP 替代方案或補充的資訊，請參閱設定撤銷和規劃憑證撤銷清單 (CRL)。

設定 OCSP 的自訂 URL 涉及三個元素。

- CA 組態 – 在 中`RevocationConfiguration`為您的 CA 指定自訂 OCSP URL，如 範例 2：建立已啟用 OCSP 和自訂 CNAME 的 CA中所述在 中建立私有 CA AWS 私有 CA。
- DNS – 將 CNAME 記錄新增至您的網域組態，將憑證中出現的 URL 映射至代理伺服器 URL。如需詳細資訊，請參閱 在 中建立私有 CA AWS 私有 CA 中的 範例 2：建立已啟用 OCSP 和自訂 CNAME 的 CA。
- 轉送代理伺服器 – 設定代理伺服器，以透明的方式將接收的 OCSP 流量轉送至 AWS OCSP 回應程式。

下圖說明這些元素如何一起運作。



如圖所示，自訂的 OCSP 驗證程序包含下列步驟：

1. 用戶端會查詢目標網域的 DNS。
2. 用戶端會收到目標 IP。
3. 用戶端會開啟與目標的 TCP 連線。
4. 用戶端會收到目標 TLS 憑證。
5. 用戶端會查詢 DNS 以取得憑證中列出的 OCSP 網域。
6. 用戶端接收代理 IP。
7. 用戶端將 OCSP 查詢傳送至代理。
8. Proxy 會將查詢轉送至 OCSP 回應者。
9. 回應者將憑證狀態傳回代理。
10. Proxy 會將憑證狀態轉送至用戶端。
11. 如果憑證有效，用戶端會開始 TLS 交握。

> **Tip**
>
> 在您設定 CA 後，可以使用 Amazon CloudFront 和 Amazon Route 53 實作此範例，如上所述。
>
> 1. 在 CloudFront 中，建立分佈並將其設定如下：
>   - 建立符合您自訂 CNAME 的替代名稱。
>   - 繫結您的憑證。
>   - 將 `ocsp.acm-pca.<region>.amazonaws.com`設定為原始伺服器。
>   - 若要使用 IPv6 連線，請使用雙堆疊端點 `acm-pca-ocsp.<region>.api.aws`
>   - 套用`Managed-CachingDisabled`政策。
>   - 將檢視器通訊協定政策設定為 HTTP 和 HTTPS。
>   - 將允許的 HTTP 方法設定為 GET、HEAD、OPONS、PUT、POST、PATCH、DELETE。
> 2. 在 Route 53 中，建立將自訂 CNAME 映射至 CloudFront 分佈 URL 的 DNS 記錄。

透過 IPv6 使用 OCSP

預設 OCSP AWS 私有 CA 回應程式 URL IPv4-only。若要透過 IPv6 使用 OCSP，請為您的 CA 設定自訂 OCSP URL。URL 可以是：

- 雙堆疊 PCA OCSP 回應程式的 FQDN，採用 形式 `acm-pca-ocsp.region-name.api.aws`
- 您設定為指向雙堆疊 OCSP 回應程式的 CNAME 記錄，如上所述。

了解 AWS 私有 CA CA 模式

AWS 私有 CA 支援以兩種模式之一建立憑證授權單位 (CA)。模式、一般用途和短期憑證會影響 CA 所發行憑證的允許有效期間。

> **Note**
>
> AWS 私有 CA 不會對根 CA 憑證執行有效性檢查。

一般用途 （預設）

此模式允許 CA 發行任何有效期間的憑證。大多數應用程式都會使用此類型的憑證。一般而言，CA 也會指定撤銷機制。

短期憑證

此模式定義一個 CA，專門發行有效期間最長為 7 天的憑證。這些短期憑證過期的速度太快，因此可以在沒有撤銷機制的情況下進行部署。對於某些應用程式，經常部署短期憑證比產生網路和處理撤銷的額外負荷更合理。

短期憑證必須是憑證階層中的最後一個 CA。由於私有 CA 必須每七天續約一次，因此有很大的額外負荷。

短期憑證模式CAs 成本低於一般用途 CAs。如需詳細資訊，請參閱AWS 私有憑證授權單位 定價。

若要建立發行短期憑證的 CA，請使用建立 CA 程序來建立 CA，將 UsageMode 參數設定為短期憑證。

> **Note**
>
> AWS Certificate Manager 無法發行由具有短期模式的私有 CA 簽署的憑證。

下列 AWS 服務支援使用短期憑證：

- Amazon AppStream
- Amazon WorkSpaces

在 中規劃彈性 AWS 私有 CA

AWS 全球基礎設施是以 AWS 區域和可用區域為基礎建置。 AWS 區域提供多個實體分隔和隔離的可用區域，這些可用區域以低延遲、高輸送量和高備援聯網連接。透過可用區域，您可以設計與操作的應用程式和資料庫，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、容錯能力和擴展能力，均較單一或多個資料中心的傳統基礎設施還高。

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。

備援和災難復原

在規劃 CA 階層時，請考慮備援和 DR。 AWS 私有 CA 可在多個區域中使用，這可讓您在多個區域中建立備援 CAs。 AWS 私有 CA 服務以 99.9% 可用性的服務水準協議 (SLA) 運作。至少有兩種方法可供您考量備援和災難復原。您可以在根 CA 或最高次級 CA 設定備援。每種方法都有優缺點。

1. 您可以在兩個不同的 AWS 區域中建立兩個根 CAs，以進行備援和災難復原。使用此組態，每個根 CA 會在 區域中獨立運作 AWS ，在發生單一區域災難時保護您。不過，建立備援根 CA 會增加操作複雜性：您必須將這兩個根 CA 憑證分配至您環境中瀏覽器和作業系統的信任存放區。
2. 您也可以建立備援次級 CAs 以在每個 AWS 區域中部署，並將其連結到單一 AWS 區域中的相同唯一根 CA。採用這種方法的好處是，您只需要將單一根 CA 憑證分配至環境中的信任存放區即可。限制是發生會影響根 CA 所在 AWS 區域的災難時，您沒有備援根 CA。

中的憑證授權單位 AWS 私有 CA

使用 AWS 私有憑證授權單位，您可以建立根和次級憑證授權單位 (CAs的完全 AWS 託管階層，以供組織內部使用。若要管理憑證撤銷，您可以啟用線上憑證狀態通訊協定 (OCSP)、憑證撤銷清單 (CRLs) 或兩者。 會 AWS 私有 CA 存放和管理 CA 憑證、CRLs 和 OCSP 回應，且根授權單位的私有金鑰會由 安全存放 AWS。

> **Note**
>
> 中的 OCSP 實作 AWS 私有 CA 不支援 OCSP 請求延伸。如果您提交包含多個憑證的 OCSP AWS 批次查詢，OCSP 回應程式只會處理佇列中的第一個憑證，並捨棄其他憑證。撤銷最多可能需要一小時才會出現在 OCSP 回應中。

您可以使用 AWS 私有 CA AWS 管理主控台 AWS CLI、 和 AWS 私有 CA API 存取 。下列主題說明如何使用主控台和 CLI。若要進一步了解 API，請參閱 AWS 私有憑證授權單位 API 參考。如需如何使用 API 的 Java 範例，請參閱 AWS 私有 CA 搭配 使用 適用於 Java 的 AWS SDK。

建立作用中私有 CA 並設定其存取權後，您可以發行和擷取憑證，如 中所述在 中發行和管理憑證 AWS 私有 CA。

主題

- 設定 以使用 AWS 私有 CA
- 在 中建立私有 CA AWS 私有 CA
- 安裝 CA 憑證
- 控制私有 CA 的存取
- 列出私有 CAs
- 檢視私有 CA
- 為您的私有 CA 新增標籤
- 了解 AWS 私有 CA CA 狀態
- 在 中更新私有 CA AWS 私有憑證授權單位
- 刪除您的私有 CA
- 還原私有 CA
- 使用外部簽署的私有 CA 憑證

設定 以使用 AWS 私有 CA

如果您尚未是 Amazon Web Services (AWS) 客戶，您必須註冊才能使用 AWS 私有 CA。您的帳戶會自動具備存取所有可用服務的權限，但是您只需要針對所使用的服務付費。

主題

- 註冊 AWS 帳戶
- 建立具有管理存取權的使用者
- 安裝 AWS Command Line Interface

註冊 AWS 帳戶

如果您沒有 AWS 帳戶，請完成下列步驟來建立一個。

註冊 AWS 帳戶

1. 開啟 https://portal.aws.amazon.com/billing/signup。
2. 請遵循線上指示進行。

   部分註冊程序需接收來電或簡訊，並在電話鍵盤輸入驗證碼。

   當您註冊 時 AWS 帳戶，AWS 帳戶根使用者會建立 。根使用者有權存取該帳戶中的所有 AWS 服務 和資源。作為安全最佳實務，請將管理存取權指派給使用者，並且僅使用根使用者來執行需要根使用者存取權的任務。

AWS 會在註冊程序完成後傳送確認電子郵件給您。您可以隨時登錄 https://aws.amazon.com/ 並選擇我的帳戶，以檢視您目前的帳戶活動並管理帳戶。

建立具有管理存取權的使用者

註冊 後 AWS 帳戶，請保護 AWS 帳戶根使用者、啟用 AWS IAM Identity Center和建立管理使用者，以免將根使用者用於日常任務。

保護您的 AWS 帳戶根使用者

1. 選擇根使用者並輸入 AWS 帳戶 您的電子郵件地址，以帳戶擁有者AWS 管理主控台身分登入 。在下一頁中，輸入您的密碼。

如需使用根使用者登入的說明，請參閱 AWS 登入 使用者指南中的以根使用者身分登入。

2. 若要在您的根使用者帳戶上啟用多重要素驗證 (MFA)。

   如需說明，請參閱《IAM 使用者指南》中的為您的 AWS 帳戶 根使用者 （主控台） 啟用虛擬 MFA 裝置。

建立具有管理存取權的使用者

1. 啟用 IAM Identity Center。

   如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的啟用 AWS IAM Identity Center。

2. 在 IAM Identity Center 中，將管理存取權授予使用者。

   如需使用 IAM Identity Center 目錄 做為身分來源的教學課程，請參閱AWS IAM Identity Center 《 使用者指南》中的使用預設值設定使用者存取 IAM Identity Center 目錄。

以具有管理存取權的使用者身分登入

- 若要使用您的 IAM Identity Center 使用者簽署，請使用建立 IAM Identity Center 使用者時傳送至您電子郵件地址的簽署 URL。

  如需使用 IAM Identity Center 使用者登入的說明，請參閱AWS 登入 《 使用者指南》中的登入 AWS 存取入口網站。

指派存取權給其他使用者

1. 在 IAM Identity Center 中，建立一個許可集來遵循套用最低權限的最佳實務。

   如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的建立許可集。

2. 將使用者指派至群組，然後對該群組指派單一登入存取權。

   如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的新增群組。

安裝 AWS Command Line Interface

如果您尚未安裝 ， AWS CLI 但想要使用它，請遵循 中的指示AWS Command Line Interface。在本指南中，我們假設您已設定端點、區域和身分驗證詳細資訊，並從範例命令中省略這些參數。

在 中建立私有 CA AWS 私有 CA

您可以使用本節中的程序來建立根 CAs或次級 CAs，進而產生符合您組織需求的可稽核信任關係階層。您可以使用 AWS 管理主控台、 或 的 PCA 部分來建立 CA AWS CLI AWS CloudFormation。

如需更新您已建立之 CA 組態的相關資訊，請參閱 在 中更新私有 CA AWS 私有憑證授權單位。

如需有關使用 CA 為您的使用者、裝置和應用程式簽署終端實體憑證的資訊，請參閱 發行私有終端實體憑證。

> **Note**
>
> 從您帳戶的建立時間開始，會針對每個私有 CA 每月向您收取費用。
> 如需最新的 AWS 私有 CA 定價資訊，請參閱 AWS 私有憑證授權單位 定價。您也可以使用AWS 定價計算器來估計成本。

主題

- 建立私有 CA 的 CLI 範例

Console

使用 主控台建立私有 CA

1. 完成下列步驟，以使用 建立私有 CA AWS 管理主控台。

   開始使用 主控台

   登入 AWS 您的帳戶，並在 開啟 AWS 私有 CA 主控台**https://console.aws.amazon.com/acm-pca/home**。

   - 如果您在沒有私有 CAs的區域中開啟主控台，則會顯示簡介頁面。選擇建立私有 CA。
   - 如果您要在已建立 CA 的區域中開啟主控台，則私有憑證授權機構頁面會開啟，其中包含您的 CAs清單。選擇建立 CA。

2. 在模式選項下，選擇 CA 發行之憑證的過期模式。

   - 一般用途 – 發行可設定任何過期日期的憑證。這是預設值。

- 短期憑證 – 發行有效期間上限為 7 天的憑證。在某些情況下，短期有效期間可以取代撤銷機制。

3. 在主控台的類型選項區段中，選擇您要建立的私有憑證授權單位類型。

   - 選擇根會建立新的 CA 階層。這個 CA 是以自我簽署的憑證為基礎的。它可做為階層中其他 CAs和終端實體憑證的最終簽署授權。
   - 選擇次級會建立 CA，該 CA 必須由階層中其上方的父 CA 簽署。次級 CAs通常用於建立其他次級 CAs或向使用者、電腦和應用程式發行終端實體憑證。

   > **Note**
   >
   > AWS 私有 CA 當您的次級 CA 的父 CA 也由 託管時， 會提供自動簽署程序 AWS 私有 CA。您只需選擇要使用的父 CA。
   > 您的次級 CA 可能需要由外部信任服務提供者簽署。若是如此， AWS 私有 CA 會為您提供憑證簽署請求 (CSR)，您必須下載並使用該請求來取得已簽署的 CA 憑證。如需詳細資訊，請參閱安裝由外部父 CA 簽署的次級 CA 憑證。

4. 在主體辨別名稱選項下，設定私有 CA 的主體名稱。您必須為下列至少一個選項輸入值：

   - Organization (O) – 例如，公司名稱
   - 組織單位 (OU) – 例如，公司內的一個部門
   - 國家名稱 (C) – 兩個字母的國家代碼
   - 狀態或省名稱 – 狀態或省的全名
   - 地區名稱 – 城市的名稱
   - Common Name (CN) – 識別 CA 的人類可讀取字串。

   > **Note**
   >
   > 您可以在發出時套用 APIPassthrough 範本，進一步自訂憑證的主旨名稱。如需詳細資訊和詳細範例，請參閱 使用 APIPassthrough 範本發行具有自訂主旨名稱的憑證。

   由於備份憑證是自我簽署的，因此您為私有 CA 提供的主題資訊可能比公有 CA 包含的內容更為稀疏。如需構成主體辨別名稱的每個值的詳細資訊，請參閱 RFC 5280。

5. 在金鑰演算法選項下，選擇金鑰演算法和演算法強度。預設值為 RSA 2048。您可以從下列演算法中選擇：

   - ML-DSA-44
   - ML-DSA-65
   - ML-DSA-87
   - RSA 2048
   - RSA 3072
   - RSA 4096
   - ECDSA P256
   - ECDSA P384
   - ECDSA P521

6. 在憑證撤銷選項下，您可以從兩種方法中選擇與使用憑證的用戶端共用撤銷狀態：

   - 啟用 CRL 分佈
   - 開啟 OCSP

   您可以為您的 CA 設定這些撤銷選項之一、兩者都無法設定。雖然是選用的，但建議使用 受管撤銷做為最佳實務。在完成此步驟之前，請參閱 規劃您的 AWS 私有 CA 憑證撤銷方法 以取得每個方法的優點、可能需要的初步設定，以及其他撤銷功能的相關資訊。

   > **Note**
   >
   > 如果您在未設定撤銷的情況下建立 CA，您可以隨時稍後進行設定。如需詳細資訊，請參閱在 中更新私有 CA AWS 私有憑證授權單位。

   若要設定憑證撤銷選項，請執行下列步驟。

   a. 在憑證撤銷選項下，選擇啟用 CRL 分佈。

   b. 在 S3 儲存貯體 URI 下，從清單中選擇現有的儲存貯體。

      當您指定現有的儲存貯體時，您必須確保為帳戶和儲存貯體停用 BPA。否則，建立 CA 的操作會失敗。如果成功建立 CA，您仍然必須手動連接政策，才能開始產生 CRLs。使用

中所述的其中一個政策模式Amazon S3 中 CRLs存取政策。如需詳細資訊，請參閱使用 Amazon S3 主控台新增儲存貯體政策。

c. 展開其他組態選項的 CRL 設定。

- 選擇啟用分割以啟用 CRLs的分割。如果您未啟用分割，您的 CA 會受到撤銷憑證數量上限的限制。如需詳細資訊，請參閱 AWS 私有憑證授權單位 配額。如需分割 CRLs的詳細資訊，請參閱 CRL 類型。
- 新增自訂 CRL 名稱，為您的 Amazon S3 儲存貯體建立別名。此名稱包含在由 RFC 5280 定義的「CRL 分佈點」延伸中 CA 發行的憑證中。若要透過 IPv6 使用 CRLs，請將其設定為儲存貯體的雙堆疊 S3 端點，如透過 IPv6 使用 CRLs中所述。
- 新增自訂路徑，為 Amazon S3 儲存貯體中的檔案路徑建立 DNS 別名。
- 輸入 CRL 將保持有效的有效天數。預設值為 7 天。對於線上 CRLs，有效期為 2-7 天很常見。 AWS 私有 CA 會嘗試在指定期間的中點重新產生 CRL。

7. 針對憑證撤銷選項，選擇開啟 OCSP。

- 在自訂 OCSP 端點 - 選用欄位中，您可以為非 Amazon OCSP 端點提供完整網域名稱 (FQDN)。若要透過 IPv6 使用 OCSP，請將此欄位設定為雙堆疊端點，如透過 IPv6 使用 OCSP 中所述。

  當您在此欄位中提供 FQDN 時， 會將 FQDN AWS 私有 CA 插入每個發行憑證的 Authority Information Access 延伸，以取代 OCSP AWS 回應程式的預設 URL。當端點收到包含自訂 FQDN 的憑證時，它會查詢 OCSP 回應的定址。若要讓此機制正常運作，您需要採取兩個額外的動作：

  - 使用代理伺服器將到達自訂 FQDN 的流量轉送至 OCSP AWS 回應程式。
  - 將對應的 CNAME 記錄新增至您的 DNS 資料庫。

  > **Tip**
  >
  > 如需使用自訂 CNAME 實作完整 OCSP 解決方案的詳細資訊，請參閱 自訂 的 OCSP URL AWS 私有 CA。

  例如，以下是自訂 OCSP 的 CNAME 記錄，如 Amazon Route 53 所示。

記錄名稱	Type	路由政策	差異化器	值/將流量路由到
alternative.example.com	CNAME	簡便	-	proxy.example.com

> **Note**
>
> CNAME 的值不得包含通訊協定字首，例如 "http：//" 或 "https：//"。

8. 在新增標籤下，您可以選擇為 CA 加上標籤。標籤是做為中繼資料的鍵/值對，可用來識別和整理 AWS 資源。如需 AWS 私有 CA 標籤參數的清單，以及如何在建立後將標籤新增至 CAs的說明，請參閱 為您的私有 CA 新增標籤。

> **Note**
>
> 若要在建立程序期間將標籤連接至私有 CA，CA 管理員必須先將內嵌 IAM 政策與 `CreateCertificateAuthority`動作建立關聯，並明確允許標記。如需詳細資訊，請參閱Tag-on-create：在建立時將標籤連接至 CA。

9. 在 CA 許可選項下，您可以選擇將自動續約許可委派給 AWS Certificate Manager 服務主體。只有在授予此許可時，ACM 才能自動續約此 CA 產生的私有終端實體憑證。您可以隨時使用 AWS 私有 CA CreatePermission API 或 create-permission CLI 命令指派續約許可。

   預設是啟用這些許可。

> **Note**
>
> AWS Certificate Manager 不支援自動續約短期憑證。

10. 在定價下，確認您了解私有 CA 的定價。

> **Note**
>
> 如需最新的 AWS 私有 CA 定價資訊，請參閱 AWS 私有憑證授權單位 定價。您也可以使用AWS 定價計算器來預估成本。

11. 在您檢查所有輸入資訊的準確性之後，請選擇建立 CA。CA 的詳細資訊頁面會開啟，並將其狀態顯示為待定憑證。

> **Note**
>
> 在詳細資訊頁面上，您可以選擇動作、安裝 CA 憑證，或稍後返回私有憑證授權單位清單並完成適用於您案例的安裝程序，以完成設定 CA：
>
> - 安裝根 CA 憑證
> - 安裝由 託管的次級 CA 憑證 AWS 私有 CA
> - 安裝由外部父 CA 簽署的次級 CA 憑證

CLI

使用 create-certificate-authority 命令來建立私有 CA。您必須指定 CA 組態（包含演算法和主體名稱資訊）、撤銷組態（如果您計劃使用 OCSP 和/或 CRL），以及 CA 類型（根或次級）。組態和撤銷組態詳細資訊包含在您提供做為命令引數的兩個檔案中。或者，您也可以設定 CA 使用模式（用於發行標準或短期憑證）、連接標籤，並提供冪等字符。

如果您要設定 CRL，則必須先有安全的 Amazon S3 儲存貯體，才能發出create-certificate-authority命令。如需詳細資訊，請參閱Amazon S3 中 CRLs存取政策。

CA 組態檔案會指定下列資訊：

- 演算法的名稱
- 用於建立 CA 私密金鑰的金鑰大小
- CA 用來簽署自己的憑證簽署請求、CRLs 和 OCSP 回應的簽署演算法類型
- X.500 主旨資訊

OCSP 的撤銷組態會定義具有下列資訊的`OcspConfiguration`物件：

- `Enabled` 旗標設定為「true」。
- （選用）宣告為 值的自訂 CNAME`OcspCustomCname`。

CRL 的撤銷組態會定義具有下列資訊的`CrlConfiguration`物件：

- `Enabled` 旗標設定為「true」。
- CRL 過期期間，以天為單位 (CRL 的有效期間）。
- 將包含 CRL 的 Amazon S3 儲存貯體。
- （選用）決定 CRL 是否可公開存取的 S3ObjectAcl 值。在此範例中，公開存取會遭到封鎖。如需詳細資訊，請參閱使用 CloudFront 啟用 S3 封鎖公開存取 (BPA)。
- （選用）S3 儲存貯體的 CNAME 別名，包含在 CA 發行的憑證中。如果 CRL 無法公開存取，這將指向分佈機制，例如 Amazon CloudFront。
- （選用）具有下列資訊的`CrlDistributionPointExtensionConfiguration`物件：
  - `OmitExtension` 旗標設定為「true」或「false」。這會控制 CDP 延伸項目的預設值是否寫入 CA 發行的憑證。如需 CDP 延伸模組的詳細資訊，請參閱 判斷 CRL 分佈點 (CDP) URI。如果 OmitExtension 為「true」，則無法設定 CustomCname。
- （選用）S3 儲存貯體中 CRL 的自訂路徑。
- （選用）CrlType 值，決定 CRL 是完成還是分割。如果未提供，CRL 將預設為完成。

> **Note**
>
> 您可以透過同時定義`OcspConfiguration`物件和`CrlConfiguration`物件，在相同的 CA 上啟用這兩個撤銷機制。如果您未提供--revocation-configuration參數，則預設會停用這兩個機制。如果您稍後需要撤銷驗證支援，請參閱 更新 CA (CLI)。

如需 CLI 範例，請參閱下一節。

建立私有 CA 的 CLI 範例

下列範例假設您已使用有效的預設區域、端點和登入資料來設定`.aws`組態目錄。如需有關設定 AWS CLI 環境的資訊，請參閱組態和登入資料檔案設定。為了便於閱讀，我們在範例命令中提供 CA 組態和撤銷輸入做為 JSON 檔案。視需要修改範例檔案以供您使用。

除非另有說明，否則所有範例都使用下列`ca_config.txt`組態檔案。

檔案：ca_config.txt

```
{
   "KeyAlgorithm":"RSA_2048",
   "SigningAlgorithm":"SHA256WITHRSA",
   "Subject":{
      "Country":"US",
      "Organization":"Example Corp",
      "OrganizationalUnit":"Sales",
      "State":"WA",
      "Locality":"Seattle",
      "CommonName":"www.example.com"
   }
}
```

範例 1：建立已啟用 OCSP 的 CA

在此範例中，撤銷檔案會啟用預設 OCSP 支援，這會使用 AWS 私有 CA 回應程式來檢查憑證狀態。

檔案： revoke_config.txt for OCSP

```
{
   "OcspConfiguration":{
      "Enabled":true
   }
}
```

命令

```
$ aws acm-pca create-certificate-authority \
     --certificate-authority-configuration file://ca_config.txt \
     --revocation-configuration file://revoke_config.txt \
     --certificate-authority-type "ROOT" \
     --idempotency-token 01234567 \
     --tags Key=Name,Value=MyPCA
```

如果成功，此命令會輸出新 CA 的 Amazon Resource Name (ARN)。

```
{
 "CertificateAuthorityArn":"arn:aws:acm-pca:region:account:
       certificate-authority/CA_ID"
}
```

命令

```
$ aws acm-pca create-certificate-authority \
 --certificate-authority-configuration file://ca_config.txt \
 --revocation-configuration file://revoke_config.txt \
 --certificate-authority-type "ROOT" \
 --idempotency-token 01234567 \
 --tags Key=Name,Value=MyPCA-2
```

如果成功，此命令會輸出 CA 的 Amazon Resource Name (ARN)。

```
{
    "CertificateAuthorityArn":"arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
}
```

使用下列命令來檢查 CA 的組態。

```
$ aws acm-pca describe-certificate-authority \
      --certificate-authority-arn "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566" \
      --output json
```

此描述應包含下列區段。

```
"RevocationConfiguration": {
   ...
   "OcspConfiguration": {
      "Enabled": true
   }
   ...
}
```

範例 2：建立已啟用 OCSP 和自訂 CNAME 的 CA

在此範例中，撤銷檔案會啟用自訂的 OCSP 支援。`OcspCustomCname` 參數會採用完整網域名稱 (FQDN) 做為其值。

當您在此欄位中提供 FQDN 時， 會將 FQDN AWS 私有 CA 插入每個發行憑證的 Authority Information Access 延伸，以取代 OCSP AWS 回應程式的預設 URL。當端點收到包含自訂 FQDN 的憑證時，它會查詢 OCSP 回應的定址。若要讓此機制正常運作，您需要採取兩個額外的動作：

- 使用代理伺服器將到達自訂 FQDN 的流量轉送至 OCSP AWS 回應程式。
- 將對應的 CNAME 記錄新增至您的 DNS 資料庫。

> **Tip**
>
> 如需使用自訂 CNAME 實作完整 OCSP 解決方案的詳細資訊，請參閱 自訂 的 OCSP URL AWS 私有 CA。

例如，以下是自訂 OCSP 的 CNAME 記錄，如 Amazon Route 53 所示。

記錄名稱	Type	路由政策	差異化器	值/將流量路由到
alternati ve.example.com	CNAME	簡便	-	proxy.exa mple.com

> **Note**
>
> CNAME 的值不得包含通訊協定字首，例如 "http：//" 或 "https：//"。

檔案： revoke_config.txt for OCSP

```
{
   "OcspConfiguration":{
      "Enabled":true,
      "OcspCustomCname":"alternative.example.com"
   }
}
```

命令

```
$ aws acm-pca create-certificate-authority \
 --certificate-authority-configuration file://ca_config.txt \
 --revocation-configuration file://revoke_config.txt \
 --certificate-authority-type "ROOT" \
 --idempotency-token 01234567 \
```

```
 --tags Key=Name,Value=MyPCA-3
```

如果成功，此命令會輸出 CA 的 Amazon Resource Name (ARN)。

```
{
    "CertificateAuthorityArn":"arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
}
```

使用下列命令來檢查 CA 的組態。

```
$ aws acm-pca describe-certificate-authority \
      --certificate-authority-arn "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566" \
      --output json
```

此描述應包含下列區段。

```
"RevocationConfiguration": {
   ...
   "OcspConfiguration": {
      "Enabled": true,
      "OcspCustomCname": "alternative.example.com"
   }
   ...
}
```

範例 3：使用連接的 CRL 建立 CA

在此範例中，撤銷組態會定義 CRL 參數。

檔案： revoke_config.txt

```
{
   "CrlConfiguration":{
      "Enabled":true,
      "ExpirationInDays":7,
      "S3BucketName":"amzn-s3-demo-bucket"
   }
}
```

命令

```
$ aws acm-pca create-certificate-authority \
      --certificate-authority-configuration file://ca_config.txt \
      --revocation-configuration file://revoke_config.txt \
      --certificate-authority-type "ROOT" \
      --idempotency-token 01234567 \
      --tags Key=Name,Value=MyPCA-1
```

如果成功，此命令會輸出 CA 的 Amazon Resource Name (ARN)。

```
{
    "CertificateAuthorityArn":"arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
}
```

使用下列命令來檢查 CA 的組態。

```
$ aws acm-pca describe-certificate-authority \
      --certificate-authority-arn "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566" \
      --output json
```

此描述應包含下列區段。

```
"RevocationConfiguration": {
   ...
   "CrlConfiguration": {
      "Enabled": true,
      "ExpirationInDays": 7,
      "S3BucketName": "amzn-s3-demo-bucket"
   },
   ...
}
```

範例 4：建立已連接 CRL 且已啟用自訂 CNAME 的 CA

在此範例中，撤銷組態會定義包含自訂 CNAME 的 CRL 參數。

檔案： revoke_config.txt

```
{
   "CrlConfiguration":{
      "Enabled":true,
```

```
      "ExpirationInDays":7,
      "CustomCname": "alternative.example.com",
      "S3BucketName":"amzn-s3-demo-bucket"
   }
}
```

命令

```
$ aws acm-pca create-certificate-authority \
      --certificate-authority-configuration file://ca_config.txt \
      --revocation-configuration file://revoke_config.txt \
      --certificate-authority-type "ROOT" \
      --idempotency-token 01234567 \
      --tags Key=Name,Value=MyPCA-1
```

如果成功，此命令會輸出 CA 的 Amazon Resource Name (ARN)。

```
{
    "CertificateAuthorityArn":"arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
}
```

使用下列命令來檢查 CA 的組態。

```
$ aws acm-pca describe-certificate-authority \
      --certificate-authority-arn "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566" \
      --output json
```

此描述應包含下列區段。

```
"RevocationConfiguration": {
   ...
   "CrlConfiguration": {
      "Enabled": true,
      "ExpirationInDays": 7,
      "CustomCname": "alternative.example.com",
      "S3BucketName": "amzn-s3-demo-bucket",
   ...
   }
}
```

範例 5：建立 CA 並指定使用模式

在此範例中，建立 CA 時會指定 CA 用量模式。如果未指定，則用量模式參數預設為 GENERAL_PURPOSE。在此範例中， 參數設定為 SHORT_LIVED_CERTIFICATE，這表示 CA 將發行有效期間最長為 7 天的憑證。在設定撤銷不方便的情況下，快速遭到入侵的短期憑證會在正常操作中過期。因此，此範例 CA 缺少撤銷機制。

> **Note**
>
> AWS 私有 CA 不會對根 CA 憑證執行有效性檢查。

```
$ aws acm-pca create-certificate-authority \
      --certificate-authority-configuration file://ca_config.txt \
      --certificate-authority-type "ROOT" \
      --usage-mode SHORT_LIVED_CERTIFICATE \
      --tags Key=usageMode,Value=SHORT_LIVED_CERTIFICATE
```

使用 中的 describe-certificate-authority命令 AWS CLI 來顯示所產生 CA 的詳細資訊，如下列命令所示：

```
$ aws acm-pca describe-certificate-authority \
      --certificate-authority-arn arn:aws:acm:region:account:certificate-
authority/CA_ID
```

```
{
    "CertificateAuthority":{
       "Arn":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID",
       "CreatedAt":"2022-09-30T09:53:42.769000-07:00",
       "LastStateChangeAt":"2022-09-30T09:53:43.784000-07:00",
       "Type":"ROOT",
       "UsageMode":"SHORT_LIVED_CERTIFICATE",
       "Serial":"serial_number",
       "Status":"PENDING_CERTIFICATE",
       "CertificateAuthorityConfiguration":{
          "KeyAlgorithm":"RSA_2048",
          "SigningAlgorithm":"SHA256WITHRSA",
          "Subject":{
             "Country":"US",
             "Organization":"Example Corp",
             "OrganizationalUnit":"Sales",
```

```
            "State":"WA",
            "Locality":"Seattle",
            "CommonName":"www.example.com"
         }
      },
      "RevocationConfiguration":{
         "CrlConfiguration":{
            "Enabled":false
         },
         "OcspConfiguration":{
            "Enabled":false
         }
      },
...
```

範例 6：為 Active Directory 登入建立 CA

您可以建立適合在 Microsoft Active Directory (AD) 的 Enterprise NTAuth 存放區中使用的私有 CA，在該處可以發行卡片登入或網域控制程式憑證。如需將 CA 憑證匯入 AD 的資訊，請參閱如何將第三方憑證授權機構 (CA) 憑證匯入 Enterprise NTAuth 存放區。

Microsoft certutil 工具可透過叫用 -dspublish選項，在 AD 中發佈 CA 憑證。在整個樹系中信任使用 certutil 發佈至 AD 的憑證。使用群組政策，您也可以限制對整個樹系子集的信任，例如網域中的單一網域或一組電腦。若要讓登入正常運作，發行 CA 也必須在 NTAuth 存放區中發佈。如需詳細資訊，請參閱使用群組政策將憑證分發至用戶端電腦。

此範例使用下列ca_config_AD.txt組態檔案。

檔案：ca_config_AD.txt

```
{
   "KeyAlgorithm":"RSA_2048",
   "SigningAlgorithm":"SHA256WITHRSA",
   "Subject":{
      "CustomAttributes":[
         {
            "ObjectIdentifier":"2.5.4.3",
            "Value":"root CA"
         },
         {
            "ObjectIdentifier":"0.9.2342.19200300.100.1.25",
            "Value":"example"
         },
```

```
         {
            "ObjectIdentifier":"0.9.2342.19200300.100.1.25",
            "Value":"com"
         }
      ]
   }
}
```

命令

```
$ aws acm-pca create-certificate-authority \
      --certificate-authority-configuration file://ca_config_AD.txt \
      --certificate-authority-type "ROOT" \
      --tags Key=application,Value=ActiveDirectory
```

如果成功，此命令會輸出 CA 的 Amazon Resource Name (ARN)。

```
{
 "CertificateAuthorityArn":"arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
 }
```

使用下列命令來檢查 CA 的組態。

```
$ aws acm-pca describe-certificate-authority \
      --certificate-authority-arn "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566" \
      --output json
```

此描述應包含下列區段。

```
...

"Subject":{
   "CustomAttributes":[
      {
         "ObjectIdentifier":"2.5.4.3",
         "Value":"root CA"
      },
      {
         "ObjectIdentifier":"0.9.2342.19200300.100.1.25",
         "Value":"example"
```

```
      },
      {
         "ObjectIdentifier":"0.9.2342.19200300.100.1.25",
         "Value":"com"
      }
   ]
}
...
```

範例 7：使用連接的 CRL 和從發行的憑證中省略的 CDP 延伸來建立事項 CA

您可以建立私有 CA，以適配發行 事物智慧家庭標準的憑證。在此範例中， 中的 CA 組態會`ca_config_PAA.txt`定義事項產品認證授權機構 (PAA)，並將廠商 ID (VID) 設定為 FFF1。

檔案：ca_config_PAA.txt

```
{
   "KeyAlgorithm":"EC_prime256v1",
   "SigningAlgorithm":"SHA256WITHECDSA",
   "Subject":{
      "Country":"US",
      "Organization":"Example Corp",
      "OrganizationalUnit":"SmartHome",
      "State":"WA",
      "Locality":"Seattle",
      "CommonName":"Example Corp Matter PAA",
   "CustomAttributes":[
      {
        "ObjectIdentifier":"1.3.6.1.4.1.37244.2.1",
        "Value":"FFF1"
      }
    ]
  }
}
```

撤銷組態會啟用 CRLs，並設定 CA 從任何發行的憑證中省略預設 CDP URL。

檔案： revoke_config.txt

```
{
   "CrlConfiguration":{
      "Enabled":true,
      "ExpirationInDays":7,
```

```
      "S3BucketName":"amzn-s3-demo-bucket",
   "CrlDistributionPointExtensionConfiguration":{
  "OmitExtension":true
   }
   }
}
```

命令

```
$ aws acm-pca create-certificate-authority \
      --certificate-authority-configuration file://ca_config_PAA.txt \
      --revocation-configuration file://revoke_config.txt \
      --certificate-authority-type "ROOT" \
      --idempotency-token 01234567 \
      --tags Key=Name,Value=MyPCA-1
```

如果成功，此命令會輸出 CA 的 Amazon Resource Name (ARN)。

```
{
    "CertificateAuthorityArn":"arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
}
```

使用下列命令來檢查 CA 的組態。

```
$ aws acm-pca describe-certificate-authority \
      --certificate-authority-arn "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566" \
      --output json
```

此描述應包含下列區段。

```
"RevocationConfiguration": {
   ...
   "CrlConfiguration": {
      "Enabled": true,
      "ExpirationInDays": 7,
      "S3BucketName": "amzn-s3-demo-bucket",
   "CrlDistributionPointExtensionConfiguration":{
  "OmitExtension":true
   }
   },
```

```
    ...
}
...
```

安裝 CA 憑證

完成下列程序來建立及安裝您的私有 CA 憑證。您的 CA 接著便會準備好可供使用。

AWS 私有 CA 支援三種安裝 CA 憑證的案例：

- 安裝由 託管之根 CA 的憑證 AWS 私有 CA
- 安裝父系授權單位是由 AWS 私有 CA託管的次級 CA 憑證
- 安裝父系授權單位是於外部進行託管的次級 CA 憑證

下列各節說明每個案例的程序。主控台程序會從主控台頁面的 Private CAs (私有 CA) 上開始。

相容簽署演算法

CA 憑證的簽署演算法支援取決於父 CA 的簽署演算法和 AWS 區域。下列限制適用於主控台和 AWS CLI 操作。

- 具有 RSA 金鑰演算法的父 CA 可以使用下列簽署演算法發行憑證：
  - SHA256 RSA
  - SHA384 RSA
  - SHA512 RSA
- 在舊版中 AWS 區域，具有 EDCSA 金鑰演算法的父 CA 可以使用下列簽署演算法發行憑證：
  - SHA256 ECDSA
  - SHA384 ECDSA
  - SHA512 ECDSA

  舊版 AWS 區域 包括：

區域名稱	地理位置
eu-north-1	歐洲 (斯德哥爾摩)

區域名稱	地理位置
me-south-1	Middle East (Bahrain)
ap-south-1	亞太地區 (孟買)
eu-west-3	Europe (Paris)
us-east-2	美國東部 (俄亥俄)
af-south-1	非洲 (開普敦)
eu-west-1	歐洲 (愛爾蘭)
eu-central-1	歐洲 (法蘭克福)
sa-east-1	南美洲 (聖保羅)
ap-east-1	亞太地區 (香港)
us-east-1	美國東部 (維吉尼亞北部)
ap-northeast-2	亞太地區 (首爾)
eu-west-2	歐洲 (倫敦)
ap-northeast-1	亞太地區 (東京)
us-gov-east-1	AWS GovCloud （美國東部）

區域名稱	地理位置
us-gov-west-1	AWS GovCloud （美國西部）
us-west-2	美國西部 (奧勒岡)
us-west-1	美國西部 (加利佛尼亞北部)
ap-southeast-1	亞太地區 (新加坡)
ap-southeast-2	亞太地區 (悉尼)

- 在非舊版中 AWS 區域，下列規則適用於 EDCSA：
  - 具有 EC_prime256v1 簽署演算法的父 CA 可以使用 ECDSA P256 發行憑證。
  - 具有 EC_secp384r1 簽署演算法的父 CA 可以使用 ECDSA P384 發行憑證。
- 在每個 中 AWS 區域，下列規則適用於 EDCSA：
  - 具有 EC_secp521r1 簽署演算法的父 CA 可以使用 ECDSA P521 發行憑證。

安裝根 CA 憑證

您可以從 AWS 管理主控台 或 安裝根 CA 憑證 AWS CLI。

為您的私有根 CA 建立和安裝憑證 （主控台）

1. （選用） 如果您尚未在 CA 的詳細資訊頁面上，請開啟位於 https：//https://console.aws.amazon.com/acm-pca/home 的 AWS 私有 CA 主控台。在私有憑證授權單位頁面上，選擇狀態為待定憑證或作用中的根 CA。
2. 選擇動作、安裝 CA 憑證以開啟安裝根 CA 憑證頁面。
3. 在指定根 CA 憑證參數下，指定下列憑證參數：
   - 有效性 — 指定 CA 憑證的過期日期和時間。根 CA 憑證 AWS 私有 CA 的預設有效期間為 10 年。

- 簽章演算法 — 指定根 CA 發行新憑證時要使用的簽署演算法。可用的選項會根據您建立 CA AWS 區域 的 而有所不同。如需詳細資訊，請參閱 CertificateAuthorityConfiguration 相容簽署演算法中的 中的支援密碼編譯演算法 AWS 私有憑證授權單位、 和 SigningAlgorithm。
  - SHA256 RSA
  - SHA384 RSA
  - SHA512 RSA

檢閱您的設定是否正確，然後選擇確認並安裝。 AWS 私有 CA 匯出 CA 的 CSR，使用根 CA 憑證範本產生憑證，然後自我簽署憑證。 AWS 私有 CA 然後匯入自我簽署的根 CA 憑證。

4. CA 的詳細資訊頁面會在頂端顯示安裝狀態（成功或失敗）。如果安裝成功，新完成的根 CA 會在一般窗格中顯示作用中狀態。

為您的私有根 CA 建立和安裝憑證 (AWS CLI)

1. 產生憑證簽署請求 (CSR)。

```
$ aws acm-pca get-certificate-authority-csr \
     --certificate-authority-arn arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566 \
     --output text \
     --region region > ca.csr
```

產生的檔案 `ca.csr`為以 base64 格式編碼的 PEM 檔案，其外觀如下。

```
-----BEGIN CERTIFICATE REQUEST-----
MIIC1DCCAbwCAQAwbTELMAkGA1UEBhMCVVMxFTATBgNVBAoMDEV4YW1wbGUgQ29y
cDEOMAwGA1UECwwFU2FsZXMxCzAJBgNVBAgMAldBMRgwFgYDVQQDDA93d3cuZXhh
bXBsZS5jb20xEDAOBgNVBAcMB1NlYXR0bGUwggEiMA0GCSqGSIb3DQEBAQUAA4IB
DwAwggEKAoIBAQDD+7eQChWUO2m6pHslI7AVSFkWvbQofKIHvbvy7wm8VO9/BuI7
LE/jrnd1jGoyI7jaMHKXPtEP3uNlCzv+oEza07OjgjqPZVehtA6a3/3vdQ1qCoD2
rXpv6VIzcq2onx2X7m+Zixwn2oY1l1ELXP7I5g0GmUStymq+pY5VARPy3vTRMjgC
JEiz8w7VvC15uIsHFAWa2/NvKyndQMPaCNft238wesV5s2cXOUS173jghIShg99o
ymf0TRUgvAGQMCXvsW07MrP5VDmBU7k/AZ9ExsUfMe2OB++fhfQWr2N7/lpC4+DP
qJTfXTEexLfRTLeLuGEaJL+c6fMyG+Yk53tZAgMBAAGgIjAgBgkqhkiG9w0BCQ4x
EzARMA8GA1UdEwEB/wQFMAMBAf8wDQYJKoZIhvcNAQELBQADggEBAA7xxLVI5s1B
qmXMMT44y1DZtQx3RDPanMNGLGO1TmLtyqqnUH49Tla+2p7nrl0tojUf/3PaZ52F
QN09SrFk8qtYSKnMGd5PZL0A+NFsNW+w4BAQNKlg9m617YEsnkztbfKRloaJNYoA
HZaRvbA0lMQ/tU2PKZR2vnao444Ugm0O/t3jx5rj817b31hQcHHQ0lQuXV2kyTrM
```

```
ohWeLf2fL+K0xJ9ZgXD4KYnY0zarpreA5RBeO5xs3Ms+oGWc13qQfMBx33vrrz2m
dw5iKjg71uuUUmtDV6ewwGa/VO5hNinYAfogdu5aGuVbnTFT3n45B8WHz2+9r0dn
bA7xUel1SuQ=
-----END CERTIFICATE REQUEST-----
```

您可以使用 OpenSSL 來檢視和驗證 CSR 的內容。

```
openssl req -text -noout -verify -in ca.csr
```

這會產生類似以下的輸出。

```
verify OK
Certificate Request:
    Data:
        Version: 0 (0x0)
        Subject: C=US, O=Example Corp, OU=Sales, ST=WA, CN=www.example.com,
 L=Seattle
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:c3:fb:b7:90:0a:15:94:3b:69:ba:a4:7b:25:23:
                    b0:15:48:59:16:bd:b4:28:7c:a2:07:bd:bb:f2:ef:
                    09:bc:54:ef:7f:06:e2:3b:2c:4f:e3:ae:77:75:8c:
                    6a:32:23:b8:da:30:72:97:3e:d1:0f:de:e3:65:0b:
                    3b:fe:a0:4c:da:d3:b3:a3:82:3a:8f:65:57:a1:b4:
                    0e:9a:df:fd:ef:75:0d:6a:0a:80:f6:ad:7a:6f:e9:
                    52:33:72:ad:a8:9f:1d:97:ee:6f:99:8b:1c:27:da:
                    86:35:97:51:0b:5c:fe:c8:e6:0d:06:99:44:ad:ca:
                    6a:be:a5:8e:55:01:13:f2:de:f4:d1:32:38:02:24:
                    48:b3:f3:0e:d5:bc:2d:79:b8:8b:07:14:05:9a:db:
                    f3:6f:2b:29:dd:40:c3:da:08:d7:ed:db:7f:30:7a:
                    c5:79:b3:67:17:39:44:b5:ef:78:e0:84:84:a1:83:
                    df:68:ca:67:f4:4d:15:20:bc:01:90:30:25:ef:b1:
                    6d:3b:32:b3:f9:54:39:81:53:b9:3f:01:9f:44:c6:
                    c5:1f:31:ed:8e:07:ef:9f:85:f4:16:af:63:7b:fe:
                    5a:42:e3:e0:cf:a8:94:df:5d:31:1e:c4:b7:d1:4c:
                    b7:8b:b8:61:1a:24:bf:9c:e9:f3:32:1b:e6:24:e7:
                    7b:59
                Exponent: 65537 (0x10001)
        Attributes:
        Requested Extensions:
```

```
            X509v3 Basic Constraints: critical
                CA:TRUE
    Signature Algorithm: sha256WithRSAEncryption
         0e:f1:c4:b5:48:e6:cd:41:aa:65:cc:31:3e:38:cb:50:d9:b5:
         0c:77:44:33:da:9c:c3:46:2c:63:b5:4e:62:ed:ca:aa:a7:50:
         7e:3d:4e:56:be:da:9e:e7:ae:5d:2d:a2:35:1f:ff:73:da:67:
         9d:85:40:dd:3d:4a:b1:64:f2:ab:58:48:a9:cc:19:de:4f:64:
         bd:00:f8:d1:6c:35:6f:b0:e0:10:10:34:a9:60:f6:6e:b5:ed:
         81:2c:9e:4c:ed:6d:f2:91:96:86:89:35:8a:00:1d:96:91:bd:
         b0:34:94:c4:3f:b5:4d:8f:29:94:76:be:76:a8:e3:8e:14:82:
         6d:0e:fe:dd:e3:c7:9a:e3:f3:5e:db:df:58:50:70:71:d0:d2:
         54:2e:5d:5d:a4:c9:3a:cc:a2:15:9e:2d:fd:9f:2f:e2:b4:c4:
         9f:59:81:70:f8:29:89:d8:d3:36:ab:a6:b7:80:e5:10:5e:3b:
         9c:6c:dc:cb:3e:a0:65:9c:d7:7a:90:7c:c0:71:df:7b:eb:af:
         3d:a6:77:0e:62:2a:38:3b:d6:eb:94:52:6b:43:57:a7:b0:c0:
         66:bf:54:ee:61:36:29:d8:01:fa:20:76:ee:5a:1a:e5:5b:9d:
         31:53:de:7e:39:07:c5:87:cf:6f:bd:af:47:67:6c:0e:f1:51:
         e9:75:4a:e4
```

2. 使用上一個步驟的 CSR 做為 `--csr` 參數的引數，發行根憑證。

> **Note**
>
> 如果您使用的是 1.6.3 AWS CLI 版或更新版本，請在指定所需的輸入檔案`fileb://`時使用 字首。這可確保 AWS 私有 CA 正確剖析 Base64-encoded的資料。

```
$ aws acm-pca issue-certificate \
     --certificate-authority-arn arn:aws:acm-pca:region:account:certificate-
authority/CA_ID \
     --csr file://ca.csr \
     --signing-algorithm SHA256WITHRSA \
     --template-arn arn:aws:acm-pca:::template/RootCACertificate/V1 \
     --validity Value=365,Type=DAYS
```

3. 擷取根憑證。

```
$ aws acm-pca get-certificate \
 --certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566 \
 --certificate-arn arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID \
```

```
 --output text > cert.pem
```

產生的檔案 cert.pem為以 base64 格式編碼的 PEM 檔案，其外觀如下。

```
-----BEGIN CERTIFICATE-----
MIIDpzCCAo+gAwIBAgIRAIIuOarlQETlUQEOZJGZYdIwDQYJKoZIhvcNAQELBQAw
bTELMAkGA1UEBhMCVVMxFTATBgNVBAoMDEV4YW1wbGUgQ29ycDEOMAwGA1UECwwF
U2FsZXMxCzAJBgNVBAgMAldBMRgwFgYDVQQDDA93d3cuZXhhbXBsZS5jb20xEDAO
BgNVBAcMB1NlYXR0bGUwHhcNMjEwMzA4MTU0NjI3WhcNMjIwMzA4MTY0NjI3WjBt
MQswCQYDVQQGEwJVUzEVMBMGA1UECgwMRXhhbXBsZSBDb3JwMQ4wDAYDVQQLDAVT
YWxlczELMAkGA1UECAwCV0ExGDAWBgNVBAMMD3d3dy5leGFtcGxlLmNvbTEQMA4G
A1UEBwwHU2VhdHRsZTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAMP7
t5AKFZQ7abqkeyUjsBVIWRa9tCh8oge9u/LvCbxU738G4jssT+Oud3WMajIjuNow
cpc+0Q/e42ULO/6gTNrTs6OCOo9lV6G0Dprf/e91DWoKgPatem/pUjNyraifHZfu
b5mLHCfahjWXUQtc/sjmDQaZRK3Kar6ljlUBE/Le9NEyOAIkSLPzDtW8LXm4iwcU
BZrb828rKd1Aw9oI1+3bfzB6xXmzZxc5RLXveOCEhKGD32jKZ/RNFSC8AZAwJe+x
bTsys/lUOYFTuT8Bn0TGxR8x7Y4H75+F9BavY3v+WkLj4M+olN9dMR7Et9FMt4u4
YRokv5zp8zIb5iTne1kCAwEAAaNCMEAwDwYDVR0TAQH/BAUwAwEB/zAdBgNVHQ4E
FgQUaW3+r328uTLokog2TklmoBK+yt4wDgYDVR0PAQH/BAQDAgGGMA0GCSqGSIb3
DQEBCwUAA4IBAQAXjd/7UZ8RDE+PLWSDNGQdLemOBTcawF+tK+PzA4Evlmn9VuNc
g+x3oZvVZSDQBANUz0b9oPeo54aE38dW1zQm2qfTab8822aqeWMLyJ1dMsAgqYX2
t9+u6w3NzRCw8Pvz18V69+dFE5AeXmNP0Z5/gdz8H/NSpctjlzopbScRZKCSlPid
Rf3ZOPm9QP92YpWyYDkfAU04xdDo1vR0MYjKPkl4LjRqSU/tcCJnPMbJiwq+bWpX
2WJoEBXB/p15Kn6JxjI0ze2SnSI48JZ8it4fvxrhOo0VoLNIuCuNXJOwU17Rdl1W
YJidaq7je6k18AdgPA0Kh8y1XtfUH3fTaVw4
-----END CERTIFICATE-----
```

您可以使用 OpenSSL 來檢視和驗證憑證的內容。

```
openssl x509 -in cert.pem -text -noout
```

這會產生類似以下的輸出。

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            82:2e:39:aa:e5:40:44:e5:51:01:0e:64:91:99:61:d2
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=US, O=Example Corp, OU=Sales, ST=WA, CN=www.example.com,
 L=Seattle
        Validity
```

```
            Not Before: Mar  8 15:46:27 2021 GMT
            Not After : Mar  8 16:46:27 2022 GMT
        Subject: C=US, O=Example Corp, OU=Sales, ST=WA, CN=www.example.com,
L=Seattle
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:c3:fb:b7:90:0a:15:94:3b:69:ba:a4:7b:25:23:
                    b0:15:48:59:16:bd:b4:28:7c:a2:07:bd:bb:f2:ef:
                    09:bc:54:ef:7f:06:e2:3b:2c:4f:e3:ae:77:75:8c:
                    6a:32:23:b8:da:30:72:97:3e:d1:0f:de:e3:65:0b:
                    3b:fe:a0:4c:da:d3:b3:a3:82:3a:8f:65:57:a1:b4:
                    0e:9a:df:fd:ef:75:0d:6a:0a:80:f6:ad:7a:6f:e9:
                    52:33:72:ad:a8:9f:1d:97:ee:6f:99:8b:1c:27:da:
                    86:35:97:51:0b:5c:fe:c8:e6:0d:06:99:44:ad:ca:
                    6a:be:a5:8e:55:01:13:f2:de:f4:d1:32:38:02:24:
                    48:b3:f3:0e:d5:bc:2d:79:b8:8b:07:14:05:9a:db:
                    f3:6f:2b:29:dd:40:c3:da:08:d7:ed:db:7f:30:7a:
                    c5:79:b3:67:17:39:44:b5:ef:78:e0:84:84:a1:83:
                    df:68:ca:67:f4:4d:15:20:bc:01:90:30:25:ef:b1:
                    6d:3b:32:b3:f9:54:39:81:53:b9:3f:01:9f:44:c6:
                    c5:1f:31:ed:8e:07:ef:9f:85:f4:16:af:63:7b:fe:
                    5a:42:e3:e0:cf:a8:94:df:5d:31:1e:c4:b7:d1:4c:
                    b7:8b:b8:61:1a:24:bf:9c:e9:f3:32:1b:e6:24:e7:
                    7b:59
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints: critical
                CA:TRUE
            X509v3 Subject Key Identifier:
                69:6D:FE:AF:7D:BC:B9:32:E8:92:88:36:4E:49:66:A0:12:BE:CA:DE
            X509v3 Key Usage: critical
                Digital Signature, Certificate Sign, CRL Sign
    Signature Algorithm: sha256WithRSAEncryption
         17:8d:df:fb:51:9f:11:0c:4f:8f:2d:64:83:34:64:1d:2d:e9:
         8e:05:37:1a:c0:5f:ad:2b:e3:f3:03:81:2f:96:69:fd:56:e3:
         5c:83:ec:77:a1:9b:d5:65:20:d0:04:03:54:cf:46:fd:a0:f7:
         a8:e7:86:84:df:c7:56:d7:34:26:da:a7:d3:69:bf:3c:db:66:
         aa:79:63:0b:c8:9d:5d:32:c0:20:a9:85:f6:b7:df:ae:eb:0d:
         cd:cd:10:b0:f0:fb:f3:d7:c5:7a:f7:e7:45:13:90:1e:5e:63:
         4f:d1:9e:7f:81:dc:fc:1f:f3:52:a5:cb:63:97:3a:29:6d:27:
         11:64:a0:92:94:f8:9d:45:fd:d9:38:f9:bd:40:ff:76:62:95:
         b2:60:39:1f:01:4d:38:c5:d0:e8:d6:f4:74:31:88:ca:3e:49:
```

```
        78:2e:34:6a:49:4f:ed:70:22:67:3c:c6:c9:8b:0a:be:6d:6a:
        57:d9:62:68:10:15:c1:fe:9d:79:2a:7e:89:c6:32:34:cd:ed:
        92:9d:22:38:f0:96:7c:8a:de:1f:bf:1a:e1:3a:8d:15:a0:b3:
        48:b8:2b:8d:5c:93:b0:53:5e:d1:76:5d:56:60:98:9d:6a:ae:
        e3:7b:a9:35:f0:07:60:3c:0d:0a:87:cc:b5:5e:d7:d4:1f:77:
        d3:69:5c:38
```

4. 匯入根 CA 憑證以將其安裝在 CA 上。

> **Note**
>
> 如果您使用的是 1.6.3 AWS CLI 版或更新版本，請在指定所需的輸入檔案`fileb://`時使用 字首。這可確保 AWS 私有 CA 正確剖析 Base64-encoded的資料。

```
$ aws acm-pca import-certificate-authority-certificate \
     --certificate-authority-arn arn:aws:acm-pca:region:account:certificate-
authority/CA_ID \
     --certificate file://cert.pem
```

檢查 CA 的新狀態。

```
$ aws acm-pca describe-certificate-authority \
 --certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566 \
 --output json
```

狀態現在會顯示為 ACTIVE。

```
{
    "CertificateAuthority": {
        "Arn": "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
        "CreatedAt": "2021-03-05T14:24:12.867000-08:00",
        "LastStateChangeAt": "2021-03-08T12:37:14.235000-08:00",
        "Type": "ROOT",
        "Serial": "serial_number",
        "Status": "ACTIVE",
        "NotBefore": "2021-03-08T07:46:27-08:00",
        "NotAfter": "2022-03-08T08:46:27-08:00",
```

```
        "CertificateAuthorityConfiguration": {
            "KeyAlgorithm": "RSA_2048",
            "SigningAlgorithm": "SHA256WITHRSA",
            "Subject": {
                "Country": "US",
                "Organization": "Example Corp",
                "OrganizationalUnit": "Sales",
                "State": "WA",
                "CommonName": "www.example.com",
                "Locality": "Seattle"
            }
        },
        "RevocationConfiguration": {
            "CrlConfiguration": {
                "Enabled": true,
                "ExpirationInDays": 7,
                "CustomCname": "alternative.example.com",
                "S3BucketName": "amzn-s3-demo-bucket"
            },
            "OcspConfiguration": {
                "Enabled": false
            }
        }
    }
}
```

安裝由 託管的次級 CA 憑證 AWS 私有 CA

您可以使用 AWS 管理主控台 來建立和安裝託管 AWS 私有 CA 次級 CA 的憑證。

為您的 AWS 私有 CA 託管次級 CA 建立並安裝憑證

1. （選用）如果您尚未在 CA 的詳細資訊頁面上，請開啟位於 https：//https://console.aws.amazon.com/acm-pca/home 的 AWS 私有 CA 主控台。在私有憑證授權單位頁面上，選擇狀態為待定憑證或作用中的次級 CA。
2. 選擇動作、安裝 CA 憑證以開啟安裝次級 CA 憑證頁面。
3. 在安裝次級 CA 憑證頁面的選取 CA 類型下，選擇AWS 私有 CA安裝由 管理的憑證 AWS 私有 CA。
4. 在選取父 CA 下，從父私有 CA 清單中選擇 CA。系統會篩選清單，以顯示符合下列條件CAs：
   - 您有使用 CA 的許可。

- CA 不會自行簽署。
- CA 處於 狀態ACTIVE。
- CA 模式為 GENERAL_PURPOSE。

5. 在指定次級 CA 憑證參數下，指定下列憑證參數：

   - 有效性 — 指定 CA 憑證的過期日期和時間。
   - 簽章演算法 — 指定根 CA 發行新憑證時要使用的簽署演算法。選項為：
     - SHA256 RSA
     - SHA384 RSA
     - SHA512 RSA
   - 路徑長度 — 次級 CA 在簽署新憑證時可新增的信任層數目。路徑長度為零 （預設） 表示只能建立終端實體憑證，而不是 CA 憑證。長度為一以上的路徑表示次級 CA 可發行憑證來建立其他 CA 次級。
   - 範本 ARN — 顯示此 CA 憑證組態範本的 ARN。如果您變更了指定的 Path length (路徑長度)，範本也會變更。如果您使用 CLI issue-certificate 命令或 API IssueCertificate 動作建立憑證，您必須手動指定 ARN。如需可用 CA 憑證範本的資訊，請參閱 使用 AWS 私有 CA 憑證範本。

6. 檢閱您的設定是否正確，然後選擇確認並安裝。 AWS 私有 CA 匯出 CSR，使用次級 CA 憑證範本產生憑證，然後使用選取的父 CA 簽署憑證。 AWS 私有 CA 然後匯入已簽章的次級 CA 憑證。
7. CA 的詳細資訊頁面會在頂端顯示安裝狀態 （成功或失敗）。如果安裝成功，新完成的次級 CA 會在一般窗格中顯示作用中狀態。

安裝由外部父 CA 簽署的次級 CA 憑證

如 中所述建立次級私有 CA 之後在 中建立私有 CA AWS 私有 CA，您可以選擇安裝外部簽署授權單位簽署的 CA 憑證來啟用它。使用外部 CA 簽署次級 CA 憑證需要您先將外部信任服務提供者設定為簽署授權單位，或安排使用第三方供應商。

> **Note**
>
> 建立或取得外部信任服務提供者的程序不在本指南的範圍內。

在您建立次級 CA 並有權存取外部簽署授權之後，請完成下列任務：

1. 從 取得憑證簽署請求 (CSR) AWS 私有 CA。
2. 將 CSR 提交給您的外部簽署授權機構，並取得已簽署的 CA 憑證以及任何鏈結憑證。
3. 將 CA 憑證和鏈結匯入 ， AWS 私有 CA 以啟用您的次級 CA。

如需詳細程序，請參閱使用外部簽署的私有 CA 憑證。

控制私有 CA 的存取

對私有 CA 具有必要許可的任何使用者可以 AWS 私有 CA 使用該 CA 簽署其他憑證。CA 擁有者可以發行憑證，或將發行憑證所需的許可委派給位於相同 的 AWS Identity and Access Management (IAM) 使用者 AWS 帳戶。如果 CA 擁有者透過以資源為基礎的政策授權，則位於不同 AWS 帳戶中的使用者也可以發行憑證。

無論單一帳戶或跨帳戶，授權使用者都可以在發行憑證時使用 AWS 私有 CA 或 AWS Certificate Manager 資源。從 AWS 私有 CA IssueCertificate API 或 issue-certificate CLI 命令發出的憑證不受管理。這類憑證需要在目標裝置上手動安裝，並在過期時手動續約。從 ACM 主控台、ACM RequestCertificate API 或 request-certificate CLI 命令發出的憑證會受到管理。這類憑證可以輕鬆安裝在與 ACM 整合的 服務中。如果 CA 管理員允許，且發行者的帳戶具有適用於 ACM 的服務連結角色，則受管憑證會在過期時自動續約。

主題

- 為 IAM 使用者建立單一帳戶許可
- 連接跨帳戶存取的政策

為 IAM 使用者建立單一帳戶許可

當 CA 管理員 （即 CA 擁有者） 和憑證發行者位於單一 AWS 帳戶中時，最佳實務是建立具有有限許可的 AWS Identity and Access Management (IAM) 使用者來區隔發行者和管理員角色。如需搭配 使用 IAM AWS 私有 CA以及範例許可的詳細資訊，請參閱 的 Identity and Access Management (IAM) AWS 私有憑證授權單位。

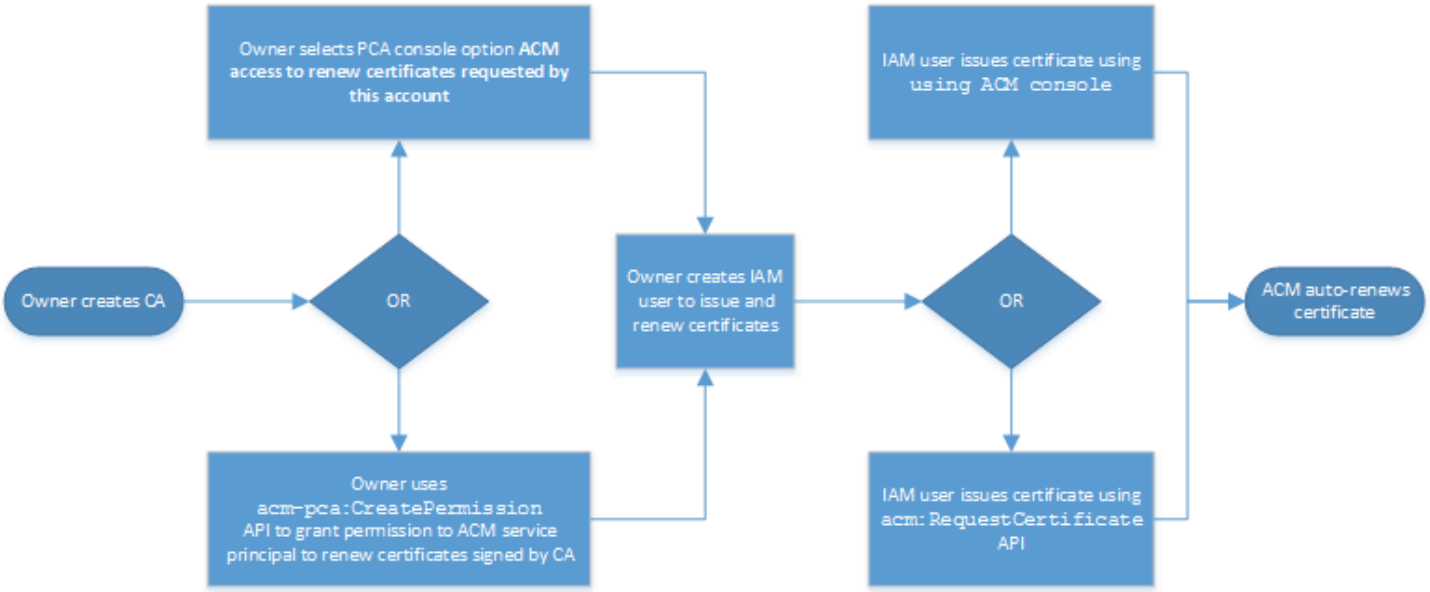
單一帳戶案例 1：發行未受管憑證

在此情況下，帳戶擁有者會建立私有 CA，然後建立具有許可的 IAM 使用者，以發行私有 CA 簽署的憑證。IAM 使用者透過呼叫 `IssueCertificate` API AWS 私有 CA 發出憑證。

以這種方式發行的憑證不受管理，這表示管理員必須匯出憑證，並將其安裝在要使用它們的裝置上。它們也必須在過期時手動續約。使用此 API 發行憑證需要 OpenSSL 或類似程式 AWS 私有 CA 在 外部產生的憑證簽署請求 (CSR) 和金鑰對。如需詳細資訊，請參閱 `IssueCertificate` 文件。

單一帳戶案例 2：透過 ACM 發行受管憑證

第二個案例涉及來自 ACM 和 PCA 的 API 操作。帳戶擁有者會像以前一樣建立私有 CA 和 IAM 使用者。帳戶擁有者接著會將許可授予 ACM 服務主體，以自動續約此 CA 簽署的任何憑證。IAM 使用者再次發出憑證，但這次呼叫 ACM `RequestCertificate` API 來處理 CSR 和金鑰產生。當憑證過期時，ACM 會自動執行續約工作流程。



帳戶擁有者可以選擇在建立 CA 或使用 PCA `CreatePermission` API 期間或之後，透過管理主控台授予續約許可。從此工作流程建立的受管憑證可在 上使用與 ACM 整合的 AWS 服務。

下一節包含授予續約許可的程序。

將憑證續約許可指派給 ACM

在 AWS Certificate Manager (ACM) 中使用受管續約，您可以自動化公有和私有憑證的憑證續約程序。為了讓 ACM 自動續約私有 CA 產生的憑證，ACM 服務主體必須由 CA 本身授予所有可能的許可。如果 ACM 沒有這些續約許可，則 CA 擁有者 （或授權代表） 必須在到期時手動重新發行每個私有憑證。

> **Important**
>
> 指派續約許可的這些程序僅適用於 CA 擁有者和憑證發行者位於相同 AWS 帳戶中的情況。如需跨帳戶案例，請參閱 連接跨帳戶存取的政策。

您可以在建立私有 CA 的期間，或是在其之後 CA 處於 ACTIVE 狀態的任何時間委派續約許可。

您可以透過 AWS 私有 CA 主控台、AWS Command Line Interface (AWS CLI)，或 AWS 私有 CA API 來管理私有 CA 許可：

將私有 CA 許可指派給 ACM （主控台）

1. 登入 AWS 您的帳戶，並在 https：//https://console.aws.amazon.com/acm-pca/home 開啟 AWS 私有 CA 主控台。
2. 在私有憑證授權單位頁面上，從清單中選擇私有 CA。
3. 選擇動作、設定 CA 許可。
4. 選取授權 ACM 存取以續約此帳戶請求的憑證。
5. 選擇儲存。

在 AWS 私有 CA (AWS CLI) 中管理 ACM 許可

使用 create-permission 命令將許可指派給 ACM。您必須指派必要的許可 (IssueCertificate、GetCertificate和 ListPermissions)，ACM 才能自動續約您的憑證。

```
$ aws acm-pca create-permission \
     --certificate-authority-arn arn:aws:acm-pca:region:account:certificate-
authority/CA_ID \
     --actions IssueCertificate GetCertificate ListPermissions \
     --principal acm.amazonaws.com
```

使用 list-permissions 命令來列出 CA 委派的許可。

```
$ aws acm-pca list-permissions \
     --certificate-authority-arn arn:aws:acm-pca:region:account:certificate-
authority/CA_ID
```

使用 delete-permission 命令撤銷 CA 指派給 AWS 服務主體的許可。

```
$ aws acm-pca delete-permission \
     --certificate-authority-arn arn:aws:acm-pca:region:account:certificate-
authority/CA_ID \
     --principal acm.amazonaws.com
```

連接跨帳戶存取的政策

當 CA 管理員和憑證發行者位於不同的 AWS 帳戶中時，CA 管理員必須共用 CA 存取權。這可透過將資源型政策連接至 CA 來完成。政策會將發行許可授予特定委託人，該委託人可以是 AWS 帳戶擁有者、IAM 使用者、 AWS Organizations ID 或組織單位 ID。

CA 管理員可以透過下列方式連接和管理政策：

- 在 管理主控台中，使用 AWS Resource Access Manager (RAM)，這是跨帳戶共用 AWS 資源的標準方法。當您在 中 AWS RAM 與其他帳戶中的委託人共用 CA 資源時，所需的資源型政策會自動連接到 CA。如需 RAM 的詳細資訊，請參閱AWS RAM 《 使用者指南》。

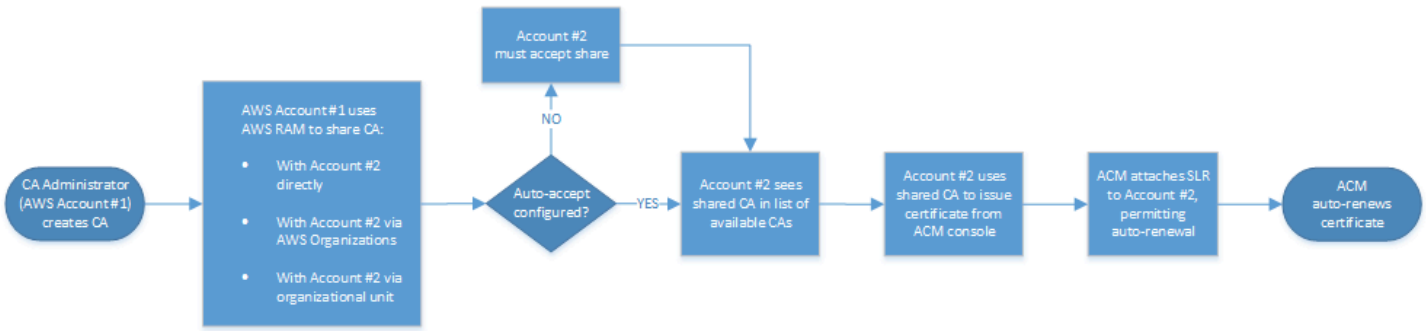
  > **Note**
  >
  > 您可以選擇 CA，然後選擇動作、管理資源共享，即可輕鬆開啟 RAM 主控台。

- 以程式設計方式使用 PCA APIsPutPolicy、GetPolicy 和 DeletePolicy。
- 手動使用 中的 PCA 命令 put-policy、get-policy 和 delete-policy AWS CLI。

只有主控台方法需要 RAM 存取。

跨帳戶案例 1：從主控台發行受管憑證

在此情況下，CA 管理員會使用 AWS Resource Access Manager (AWS RAM) 與其他 AWS 帳戶共用 CA 存取權，以允許該帳戶發行受管 ACM 憑證。圖表顯示 AWS RAM 可以直接與帳戶共用 CA，或透過帳戶為成員的 AWS Organizations ID 間接共用 CA。

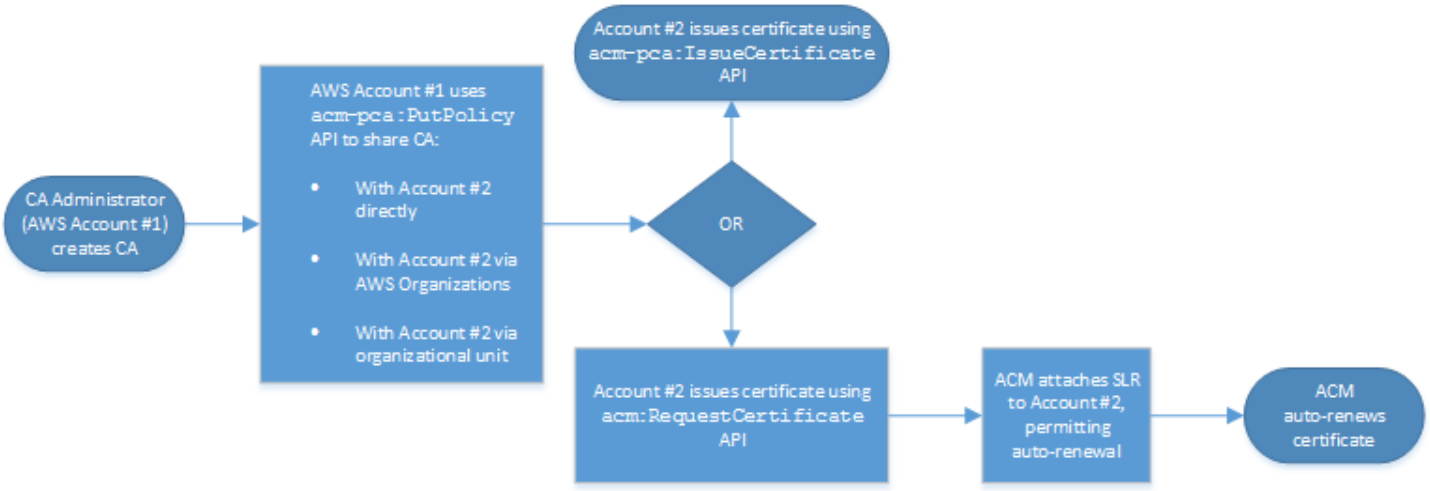


RAM 透過 共用資源後 AWS Organizations，收件人委託人必須接受資源才能生效。收件人可以設定 AWS Organizations 自動接受提供的共用。

> **Note**
>
> 收件人帳戶負責在 ACM 中設定自動續約。一般而言，在第一次使用共用 CA 時，ACM 會安裝服務連結角色，允許它在 上進行自動憑證呼叫 AWS 私有 CA。如果失敗（通常是由於缺少許可），則不會自動續約來自 CA 的憑證。只有 ACM 使用者可以解決問題，而不是 CA 管理員。如需詳細資訊，請參閱搭配 ACM 使用服務連結角色 (SLR)。

跨帳戶案例 2：使用 API 或 CLI 發行受管和未受管憑證

第二個案例示範可使用 和 AWS 私有 CA API 的共用 AWS Certificate Manager 和發行選項。所有這些操作也可以使用對應的 AWS CLI 命令來執行。



由於在此範例中直接使用 API 操作，憑證發行者可以選擇兩種 API 操作來發行憑證。PCA API 動作`IssueCertificate`會產生未受管憑證，不會自動續約，且必須匯出並手動安裝。ACM API 動作 RequestCertificate 會產生可輕易安裝在 ACM 整合服務上的受管憑證，並自動續約。

> **Note**
>
> 收件人帳戶負責在 ACM 中設定自動續約。一般而言，在第一次使用共用 CA 時，ACM 會安裝服務連結角色，允許它在 上進行無人看管憑證呼叫 AWS 私有 CA。如果失敗（通常是由於缺少許可），則 CA 的憑證不會自動續約，只有 ACM 使用者可以解決問題，而不是 CA 管理員。如需詳細資訊，請參閱搭配 ACM 使用服務連結角色 (SLR)。

列出私有 CAs

您可以使用 AWS 私有 CA 主控台或 AWS CLI 列出您擁有或可存取CAs。

使用主控台列出可用的 CAs

1. 登入 AWS 您的帳戶，並在 https：//https://console.aws.amazon.com/acm-pca/home 開啟 AWS 私有 CA 主控台。
2. 檢閱私有憑證授權單位清單中的資訊。您可以使用右上角的頁碼瀏覽多個 CAs 頁面。每個 CA 都會佔用一個資料列，其中顯示以下部分或全部資料欄：

- 主旨 – CA 辨別名稱資訊的摘要。
- Id – CA 的 32 位元組十六進位唯一識別符。
- 狀態 – CA 狀態。可能的值包括建立、待定憑證、作用中、已刪除、已停用、已過期和失敗。
- 類型 – CA 的類型。可能的值為根和次級。
- 模式 – CA 的模式。可能的值為一般用途（發行可設定任何過期日期的憑證）和短期憑證（發行有效期間最長為七天的憑證）。在某些情況下，短期有效期間可以取代撤銷機制。預設為一般用途。
- 擁有者 – 擁有 CA AWS 的帳戶。這可能是您的帳戶或已將 CA 管理許可委派給您的帳戶。
- 金鑰演算法 – CA 支援的公有金鑰演算法。可能的值為 ML_DSA_44、ML_DSA_65、ML_DSA_87、RSA_2048、RSA_3072、RSA_4096、EC_prime256v1、EC_ 和 EC_secp521r1。
- 簽署演算法 – CA 用來簽署憑證請求的演算法。（不要與發行憑證時用來簽署憑證的 `SigningAlgorithm` 參數混淆。）可能的值為 ML_DSA_44、ML_DSA_65、ML_DSA_87、SHA256WITHRSA、SHA384WITHRSA、SHA512WITHRSA、和 SHA512WITHECDSA。

> **Note**
>
> 您可以選擇主控台右上角的設定圖示，以自訂要顯示的資料欄和其他設定。

使用 列出可用的 CAs AWS CLI

使用 list-certificate-authorities 命令列出可用的 CAs，如下列範例所示：

```
$ aws acm-pca list-certificate-authorities --max-items 10
```

此命令會傳回與以下內容相似的資訊：

```
{
   "CertificateAuthorities":[
      {
         "Arn":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID",
         "CreatedAt":"2022-05-02T11:59:02.022000-07:00",
         "LastStateChangeAt":"2022-05-02T11:59:18.498000-07:00",
         "Type":"ROOT",
         "Serial":"serial_number",
         "Status":"ACTIVE",
         "NotBefore":"2022-05-02T10:59:17-07:00",
         "NotAfter":"2032-05-02T11:59:17-07:00",
         "CertificateAuthorityConfiguration":{
            "KeyAlgorithm":"RSA_2048",
            "SigningAlgorithm":"SHA256WITHRSA",
            "Subject":{
               "Organization":"testing_com"
            }
         },
         "RevocationConfiguration":{
            "CrlConfiguration":{
               "Enabled":false
            }
         }
      }
 ...
   ]
}
```

檢視私有 CA

您可以使用 ACM 主控台或 AWS CLI 檢視私有 CA 的詳細中繼資料，並視需要變更數個值。如需更新 CAs的詳細資訊，請參閱 在 中更新私有 CA AWS 私有憑證授權單位。

在主控台中檢視 CA 詳細資訊

1. 登入 AWS 您的帳戶，並在 https：//https://console.aws.amazon.com/acm-pca/home 開啟 AWS 私有 CA 主控台。

2. 檢閱私有憑證授權單位清單。您可以使用右上角的頁碼瀏覽多個 CAs 頁面。
3. 若要顯示列出的 CA 的詳細中繼資料，請依您要檢查的 CA 選擇選項按鈕。這會開啟具有下列標籤式檢視的詳細資訊窗格：

   - 主旨標籤 – CA 辨別名稱的相關資訊。如需詳細資訊，請參閱Subject distinguished name。顯示的欄位包括：
     - 主旨 – 提供的名稱資訊欄位摘要
     - Organization (O) – 例如，公司名稱
     - 組織單位 (OU) – 例如，公司內的一個部門
     - 國家名稱 (C) – 兩個字母的國家代碼
     - 狀態或省名稱 – 狀態或省的全名
     - 地區名稱 – 城市的名稱
     - Common Name (CN) – 識別 CA 的人類可讀取字串。
   - CA 憑證索引標籤 – CA 憑證有效性的相關資訊
     - 有效期限 - CA 憑證有效之前的日期和時間
     - 過期時間 – 過期前的天數
   - 撤銷組態索引標籤 – 您目前的憑證撤銷選項選擇。選擇編輯以更新。
     - 憑證撤銷清單 (CRL) 分佈 – 啟用或停用狀態
     - 線上憑證狀態通訊協定 (OCSP) – 啟用或停用的狀態
   - 許可索引標籤 – 您目前為此 CA 透過 AWS Certificate Manager (ACM) 選擇憑證續約權限。選擇編輯以更新。
   - 續約的 ACM 授權 – 授權或未經授權的狀態
   - 標籤索引標籤 – 您目前為此 CA 指派的可自訂標籤。選擇要更新的管理標籤。
   - 資源共享索引標籤 – 您目前透過 AWS Resource Access Manager (RAM) 為此 CA 指派的資源共享。選擇管理要更新的資源共用。
     - 名稱 – 資源共用的名稱
     - 狀態 – 資源共用的狀態
4. 選擇您要檢查的 CA ID 欄位，以開啟一般窗格。CA 的 32 位元組十六進位唯一識別符會顯示在頂端。窗格提供下列額外資訊：

   - 狀態 – CA 狀態。可能的值包括建立、待定憑證、作用中、已刪除、已停用、已過期和失敗。
   - ARN – CA 的 Amazon Resource Name。

- 擁有者 – 擁有 CA AWS 的帳戶。這可能是您的帳戶 (自我) 或已將 CA 管理許可委派給您的帳戶。
- CA 類型 – CA 的類型。可能的值為根和次級。
- 建立時間：建立 CA 的日期和時間。
- 過期日期 – CA 憑證過期的日期和時間。
- 模式 – CA 的模式。可能的值為一般用途 （ 可使用任何過期日期設定的憑證 ） 和短期憑證 （ 有效期間上限為 7 天的憑證 ）。在某些情況下，短期有效期間可以取代撤銷機制。預設為一般用途。
- 金鑰演算法 – CA 支援的公有金鑰演算法。可能的值為 ML-DSA-44、ML-DSA-65、ML-DSA-87、RSA 2048、RSA 3072、RSA 4096、ECDSA P256、ECDSA P384 和 ECDSA P521。
- 簽署演算法 – CA 用來簽署自己的憑證簽署請求、CRLs 和 OCSP 回應的演算法 （ 不要與 IssueCertificate API 中使用的`SigningAlgorithm`參數混淆 ）。可能的值為 ML-DSA-44、ML-DSA-65、ML-DSA-87、SHA256 RSA、SHA384 RSA 和 SHA512 RSA、SHA256 ECDSA、SHA384 ECDSA、SHA512 ECDSA
- 金鑰儲存安全標準 – 聯邦資訊處理標準 (FIPS) 一致性等級。您可以從這些值中選擇：FIPS 140-2 第 2 級或更高、FIPS 140-2 第 3 級或更高，以及 CCPC 第 1 級或更高。此參數因 AWS 區域而異。

> **Note**
>
> 自 2023 年 1 月 26 日起， AWS Private CA 會使用符合 FIPS PUB 140-2 第 3 級的硬體安全模組 (HSMs) 保護非中國區域中的所有 CA 私有金鑰。

使用 檢視和修改 CA 詳細資訊 AWS CLI

使用 AWS CLI 中的 describe-certificate-authority命令來顯示 CA 的詳細資訊，如下列命令所示：

```
$ aws acm-pca describe-certificate-authority --certificate-authority-arn
 arn:aws:acm:region:account:certificate-authority/CA_ID
```

此命令會傳回與以下內容相似的資訊：

```
{
   "CertificateAuthority":{
      "Arn":"arn:aws:acm:region:account:certificate-authority/CA_ID",
```

```
      "CreatedAt":"2022-05-02T11:59:02.022000-07:00",
      "LastStateChangeAt":"2022-05-02T11:59:18.498000-07:00",
      "Type":"ROOT",
      "Serial":"serial_number",
      "Status":"ACTIVE",
      "NotBefore":"2022-05-02T10:59:17-07:00",
      "NotAfter":"2031-05-02T11:59:17-07:00",
      "CertificateAuthorityConfiguration":{
         "KeyAlgorithm":"RSA_2048",
         "SigningAlgorithm":"SHA256WITHRSA",
         "Subject":{
            "Organization":"testing_com"
         }
      },
      "RevocationConfiguration":{
         "CrlConfiguration":{
            "Enabled":false
         }
      }
   }
}
```

如需從命令列更新私有 CA 的資訊，請參閱 更新 CA (CLI)。

為您的私有 CA 新增標籤

標籤是做為中繼資料的單字或片語，用於識別和組織您的 AWS 資源。每個標籤皆包含鍵與值。您可以使用 AWS 私有 CA 主控台、 AWS Command Line Interface (AWS CLI) 或 PCA API 來新增、檢視或移除私有 CAs的標籤。

您可以隨時新增或移除私有 CA 的自訂標籤。例如，您可以使用類似 `Environment=Prod`或 的鍵值對來標記私有 CAs，`Environment=Beta`以識別 CA 要用於哪個環境。如需詳細資訊，請參閱建立私有 CA。

> **Note**
>
> 若要在建立程序期間將標籤連接至私有 CA，CA 管理員必須先將內嵌 IAM 政策與 `CreateCertificateAuthority`動作建立關聯，並明確允許標記。如需詳細資訊，請參閱Tag-on-create：在建立時將標籤連接至 CA。

其他 AWS 資源也支援標記。您可以將相同的標籤指派給不同的資源，以指出這些資源是相關的。例如，您可以將 等標籤指派給 Website=example.com CA、Elastic Load Balancing 負載平衡器和其他相關資源。如需標記 AWS 資源的詳細資訊，請參閱《Amazon EC2 使用者指南》中的標記 Amazon EC2 資源。

下列基本限制適用於 AWS 私有 CA 標籤：

- 每個私有 CA 的標籤數上限為 50。
- 標籤鍵的長度上限為 128 個字元。
- 標籤值的長度上限為 256 個字元。
- 標籤鍵和值可以包含下列字元：A-Z、a-z 和 .:+= @_%- (連字號)。
- 標籤鍵與值皆區分大小寫。
- aws: 和 rds: 字首是保留給 AWS 使用的字詞；您無法新增、編輯或刪除鍵是以 aws: 或 rds: 開頭的標籤。開頭為 aws: 和 rds: 的預設標籤不會計入您每個資源的標籤配額中。
- 如果您計劃跨多個 服務和資源使用標記結構描述，請記住，其他 服務對允許的字元可能有不同的限制。請參閱文件以了解該服務。
- AWS 私有 CA 標籤不適用於 中的資源群組和標籤編輯器 AWS 管理主控台。

您可以從 AWS 私有 CA 主控台、AWS Command Line Interface (AWS CLI) 或 AWS 私有 CA API 來標記私有 CA。

標記私有 CA (主控台)

1. 登入 AWS 您的帳戶，並在 https：//https://console.aws.amazon.com/acm-pca/home 開啟 AWS 私有 CA 主控台。
2. 在私有憑證授權單位頁面上，從清單中選擇您的私有 CA。
3. 在清單下方的詳細資訊區域中，選擇標籤索引標籤。隨即顯示現有標籤的清單。
4. 選擇管理標籤。
5. 選擇 Add new tag (新增標籤)。
6. 輸入鍵值組。
7. 選擇儲存。

標記私有 CA (AWS CLI)

使用 tag-certificate-authority 命令來為私有 CA 加上標籤。

```
$ aws acm-pca tag-certificate-authority \
     --certificate-authority-arn arn:aws:acm-pca:region:account:certificate-
authority/CA_ID \
     --tags Key=Admin,Value=Alice
```

使用 list-tags 命令來列出私有 CA 的標籤。

```
$ aws acm-pca list-tags \
     --certificate-authority-arn arn:aws:acm-pca:region:account:certificate-
authority/CA_ID \
     --max-results 10
```

使用 untag-certificate-authority 命令來移除私有 CA 的標籤。

```
$ aws acm-pca untag-certificate-authority \
     --certificate-authority-arn arn:aws:acm-pca:aregion:account:certificate-
authority/CA_ID \
     --tags Key=Purpose,Value=Website
```

了解 AWS 私有 CA CA 狀態

由使用者動作 AWS 私有 CA 的結果所管理的 CA 狀態，或在某些情況下來自服務動作。例如，CA 狀態會在過期時變更。CA 管理員可使用的狀態選項會因 CA 目前的狀態而有所不同。

AWS 私有 CA 可以報告下列狀態值。資料表顯示每個狀態中可用的 CA 功能。

> **Note**
>
> 對於 DELETED和 以外的所有狀態值FAILED，您需要支付 CA 的費用。

狀態	發行憑證	使用 OCSP 驗證憑證	產生 CRLs	產生稽核	您可以更新 CA 憑證	憑證可以撤銷	您需支付 CA 的費用
CREATING – 正在建立 CA。	否	否	否	否	否	否	是

狀態	發行憑證	使用 OCSP 驗證憑證	產生 CRLs	產生稽核	您可以更新 CA 憑證	憑證可以撤銷	您需支付 CA 的費用
PENDING_CERTIFICATE – 已建立 CA，且需要憑證才能運作。*	否	否	否	否	否	否	是
ACTIVE	是	是	是	是	是	是	是
DISABLED – 您已手動停用 CA。	否	是	是	是	否	是	是
EXPIRED – CA 憑證已過期。**	否	否	否	否	是	否	是
FAILED	CreateCertificateAuthority 動作失敗。這可能是因為網路中斷、後端 AWS 故障或其他錯誤而發生。失敗的 CA 無法復原。請刪除 CA 並建立新的 CA。						否
DELETED	您的 CA 處於還原期間，長度可能為 7-30 天。在此期間之後，CA 將會永久刪除。 • 如果您對狀態為 DELETED 且憑證已過期的 CA 呼叫 RestoreCertificateAuthority API，則 CA 將會設為 EXPIRED。 • 如需刪除 CA 的詳細資訊，請參閱 刪除您的私有 CA。						否

若要完成啟用，您需要產生 CSR、從 CA 取得已簽署的 CA 憑證，以及將憑證匯入 AWS 私有 CA。CSR 可以提交至新的 CA（用於自我簽署），或提交至內部部署根 CA 或次級 CA。如需詳細資訊，請參閱安裝 CA 憑證。

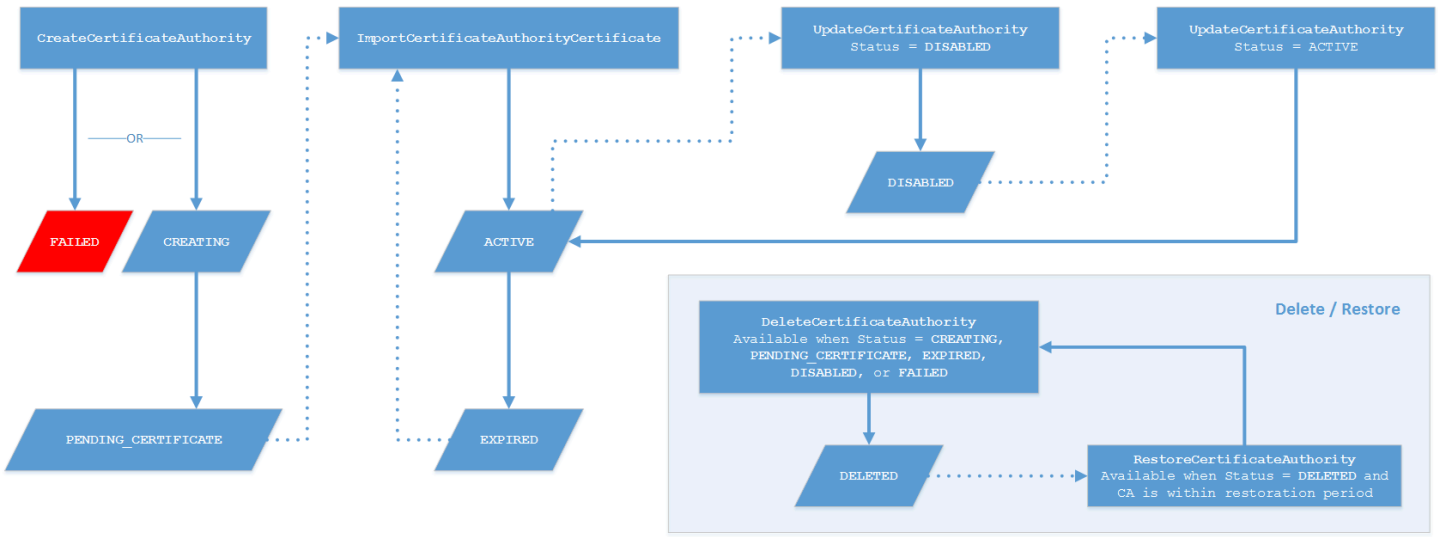
您無法直接變更已過期 CA 的狀態。如果您匯入 CA 的新憑證， 會將狀態 AWS 私有 CA 重設為 ，ACTIVE除非在憑證過期DISABLED之前將其設定為 。

過期 CA 憑證的其他考量事項：

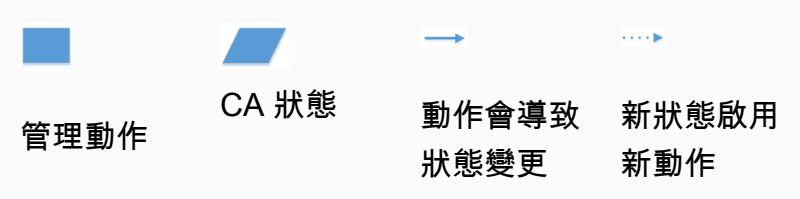
- CA 憑證不會自動續約。如需透過 自動續約的詳細資訊 AWS Certificate Manager，請參閱 將憑證續約許可指派給 ACM。
- 如果您嘗試使用過期的 CA 發行新憑證，則 `IssueCertificate` API 會傳回 `InvalidStateException`。過期的根 CA 必須先自我簽署新的根 CA 憑證，才能發行新的次級憑證。
- 如果 CA 憑證已過期，The `ListCertificateAuthorities` 和 `DescribeCertificateAuthority` API 便會傳回 `EXPIRED` 狀態，無論 CA 狀態是設為 `ACTIVE` 或是 `DISABLED`。但是，如果過期的 CA 已設為 `DELETED`，則狀態會傳回 `DELETED`。
- `UpdateCertificateAuthority` API 無法更新已過期 CA 的狀態。
- `RevokeCertificate` API 無法用來撤銷任何過期的憑證，包括 CA 憑證。

CA 狀態與 CA 生命週期之間的關係

下圖會將 CA 的生命週期做為 CA 狀態與管理動作的互動顯示。



圖表索引鍵



在圖表頂端，管理動作會透過 AWS 私有 CA 主控台、CLI 或 API 套用。這些動作會帶領 CA 經歷建立、啟用、過期和續約。CA 狀態會在回應中變更為手動動作或自動更新 (以實線顯示)。在大多數的情況下，新的狀態會產生可讓 CA 管理員套用的新可能動作 (以虛線顯示)。右下方的內凹顯示可能的狀態值，允許刪除和還原動作。

在 中更新私有 CA AWS 私有憑證授權單位

您可以更新私有 CA 的狀態，或在建立後變更其撤銷組態。本主題提供 CA 狀態和 CA 生命週期的詳細資訊，以及 CAs的主控台和 CLI 更新範例。

更新 CA （主控台）

下列程序說明如何使用 更新現有的 CA 組態 AWS 管理主控台。

更新 CA 狀態 （主控台）

在此範例中，已啟用 CA 的狀態變更為已停用。

更新 CA 的狀態

1. 登入 AWS 您的帳戶，並在 https：//https://console.aws.amazon.com/acm-pca/home 開啟 AWS 私有 CA 主控台
2. 在私有憑證授權單位頁面上，從清單中選擇目前作用中的私有 CA。
3. 在動作功能表中，選擇停用以停用私有 CA。

更新 CA 的撤銷組態 （主控台）

您可以更新私有 CA 的撤銷組態，例如新增或移除 OCSP 或 CRL 支援，或修改其設定。

> **Note**
>
> CA 撤銷組態的變更不會影響已發行的憑證。若要讓受管撤銷正常運作，必須重新發行較舊的憑證。

對於 OCSP，您可以變更下列設定：

- 啟用或停用 OCSP。
- 啟用或停用自訂 OCSP 完整網域名稱 (FQDN)。

- 變更 FQDN。

對於 CRL，您可以變更下列任何設定：

- CRL 類型（完整或分割）
- 私有 CA 是否會產生憑證撤銷清單 (CRL)
- CRL 到期的天數。請注意， 會在您指定的天數的一半時 AWS 私有 CA 開始嘗試重新產生 CRL。
- 儲存 CRL 的 Amazon S3 儲存貯體名稱。
- 從公有檢視中隱藏 Amazon S3 儲存貯體名稱的別名。

> **Important**
>
> 變更任何上述參數可能會產生負面影響。範例包括停用 CRL 產生、變更有效期間，或在您將私有 CA 放入生產環境後變更 S3 儲存貯體。這類變更可能會破壞取決於 CRL 和目前 CRL 組態的現有憑證。您可以安全地變更別名，只要舊別名仍連結到正確的儲存貯體。

更新撤銷設定

1. 登入 AWS 您的帳戶，並在 https：//https://console.aws.amazon.com/acm-pca/home 開啟 AWS 私有 CA 主控台。
2. 在私有憑證授權單位頁面上，從清單中選擇私有 CA。這會開啟 CA 的詳細資訊面板。
3. 選擇撤銷組態索引標籤，然後選擇編輯。
4. 在憑證撤銷選項下，會顯示兩個選項：

   - 啟用 CRL 分佈
   - 開啟 OCSP

   您可以為 CA 設定這些撤銷機制，或兩者都無法設定。雖然是選用的，但建議使用 受管撤銷做為最佳實務。在完成此步驟之前，請參閱 規劃您的 AWS 私有 CA 憑證撤銷方法 以取得每個方法的優點、可能需要的初步設定，以及其他撤銷功能的相關資訊。

設定 CRL

1. 選取啟用 CRL 分佈。

2. 若要為您的 CRL 項目建立 Amazon S3 儲存貯體，請選取建立新的 S3 儲存貯體。提供唯一的儲存貯體名稱。(您不需要包含指向儲存貯體的路徑。) 否則，請保持未選取此選項，然後從 S3 儲存貯體名稱清單中選擇現有的儲存貯體。

   如果您建立新的儲存貯體， 會 AWS 私有 CA 建立所需的存取政策並將其連接至該儲存貯體。如果您決定使用現有的儲存貯體，您必須先連接存取政策，才能開始產生 CRLs。使用中所述的其中一個政策模式Amazon S3 中 CRLs存取政策。如需連接政策的資訊，請參閱使用 Amazon S3 主控台新增儲存貯體政策。

   > **Note**
   >
   > 當您使用 AWS 私有 CA 主控台時，如果下列兩個條件都適用，則嘗試建立 CA 會失敗：
   >
   > - 您正在對 Amazon S3 儲存貯體或帳戶強制執行封鎖公開存取設定。
   > - 您要求 自動建立 Amazon S3 儲存貯 AWS 私有 CA 體。
   >
   > 在這種情況下，主控台預設會嘗試建立可公開存取的儲存貯體，Amazon S3 會拒絕此動作。如果發生這種情況，請檢查您的 Amazon S3 設定。如需詳細資訊，請參閱封鎖對 Amazon S3 儲存體的公開存取。

3. 展開 Advanced (進階) 以取得其他組態選項。

   - 選擇啟用分割以啟用 CRLs的分割。如果您未啟用分割，您的 CA 會受限於AWS 私有憑證授權單位 配額上顯示的撤銷憑證數量上限。如需分割 CRLs的詳細資訊，請參閱 CRL 類型。
   - 新增自訂 CRL 名稱，為您的 Amazon S3 儲存貯體建立別名。此名稱包含在由 RFC 5280 定義的「CRL 分佈點」延伸中 CA 發行的憑證中。若要透過 IPv6 使用 CRLs，請將其設定為儲存貯體的雙堆疊 S3 端點，如透過 IPv6 使用 CRLs中所述。
   - 新增自訂路徑，為 Amazon S3 儲存貯體中的檔案路徑建立 DNS 別名。
   - 輸入 CRL 將保持有效的有效天數。預設值為 7 天。對於線上 CRLs，有效期為 2-7 天很常見。AWS 私有 CA 會嘗試在指定期間的中點重新產生 CRL。

4. 完成後選擇儲存變更。

設定 OCSP

1. 在憑證撤銷頁面上，選擇開啟 OCSP。

2. （選用）在自訂 OCSP 端點欄位中，為您的 OCSP 端點提供完整網域名稱 (FQDN)。若要透過 IPv6 使用 OCSP，請將此欄位設定為雙堆疊端點，如透過 IPv6 使用 OCSP 中所述。

   當您在此欄位中提供 FQDN 時， 會將 FQDN AWS 私有 CA 插入每個發行憑證的 Authority Information Access 延伸，以取代 OCSP AWS 回應程式的預設 URL。當端點收到包含自訂 FQDN 的憑證時，它會查詢 OCSP 回應的定址。若要讓此機制正常運作，您需要採取兩個額外的動作：

   - 使用代理伺服器將到達自訂 FQDN 的流量轉送至 OCSP AWS 回應程式。
   - 將對應的 CNAME 記錄新增至您的 DNS 資料庫。

   > **Tip**
   >
   > 如需使用自訂 CNAME 實作完整 OCSP 解決方案的詳細資訊，請參閱 自訂 的 OCSP URL AWS 私有 CA。

   例如，以下是自訂 OCSP 的 CNAME 記錄，如 Amazon Route 53 所示。

記錄名稱	Type	路由政策	差異化器	值/將流量路由到
alternative.example.com	CNAME	簡便	-	proxy.example.com

   > **Note**
   >
   > CNAME 的值不得包含通訊協定字首，例如 "http：//" 或 "https：//"。

3. 完成後，選擇儲存變更。

更新 CA (CLI)

下列程序說明如何使用 更新現有 CA 的狀態和撤銷組態 AWS CLI。

> **Note**
>
> CA 撤銷組態的變更不會影響已發行的憑證。若要讓受管撤銷正常運作，必須重新發行較舊的憑證。

更新私有 CA 的狀態 (AWS CLI)

請使用 update-certificate-authority 命令。

當您現有的 CA 具有您要設定為 的狀態時DISABLED，此功能非常有用ACTIVE。若要開始，請使用下列命令確認 CA 的初始狀態。

```
$ aws acm-pca describe-certificate-authority \
      --certificate-authority-arn "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566" \
      --output json
```

這會產生類似以下的輸出。

```
{
    "CertificateAuthority": {
        "Arn": "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
        "CreatedAt": "2021-03-05T14:24:12.867000-08:00",
        "LastStateChangeAt": "2021-03-08T13:17:40.221000-08:00",
        "Type": "ROOT",
        "Serial": "serial_number",
        "Status": "DISABLED",
        "NotBefore": "2021-03-08T07:46:27-08:00",
        "NotAfter": "2022-03-08T08:46:27-08:00",
        "CertificateAuthorityConfiguration": {
            "KeyAlgorithm": "RSA_2048",
            "SigningAlgorithm": "SHA256WITHRSA",
            "Subject": {
                "Country": "US",
                "Organization": "Example Corp",
                "OrganizationalUnit": "Sales",
                "State": "WA",
                "CommonName": "www.example.com",
                "Locality": "Seattle"
            }
```

```
        },
        "RevocationConfiguration": {
            "CrlConfiguration": {
                "Enabled": true,
                "ExpirationInDays": 7,
                "CustomCname": "alternative.example.com",
                "S3BucketName": "amzn-s3-demo-bucket"
   },
            "OcspConfiguration": {
                "Enabled": false
            }
        }
    }
}
```

下列命令會將私有 CA 的狀態設定為 ACTIVE。只有在 CA 上安裝有效的憑證時，才能這麼做。

```
$ aws acm-pca update-certificate-authority \
      --certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566 \
      --status "ACTIVE"
```

檢查 CA 的新狀態。

```
$ aws acm-pca describe-certificate-authority \
      --certificate-authority-arn "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566" \
      --output json
```

狀態現在會顯示為 ACTIVE。

```
{
    "CertificateAuthority": {
        "Arn": "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
        "CreatedAt": "2021-03-05T14:24:12.867000-08:00",
        "LastStateChangeAt": "2021-03-08T13:23:09.352000-08:00",
        "Type": "ROOT",
        "Serial": "serial_number",
        "Status": "ACTIVE",
        "NotBefore": "2021-03-08T07:46:27-08:00",
        "NotAfter": "2022-03-08T08:46:27-08:00",
```

```
        "CertificateAuthorityConfiguration": {
            "KeyAlgorithm": "RSA_2048",
            "SigningAlgorithm": "SHA256WITHRSA",
            "Subject": {
                "Country": "US",
                "Organization": "Example Corp",
                "OrganizationalUnit": "Sales",
                "State": "WA",
                "CommonName": "www.example.com",
                "Locality": "Seattle"
            }
        },
        "RevocationConfiguration": {
            "CrlConfiguration": {
                "Enabled": true,
                "ExpirationInDays": 7,
                "CustomCname": "alternative.example.com",
                "S3BucketName": "amzn-s3-demo-bucket"
            },
            "OcspConfiguration": {
                "Enabled": false
            }
        }
    }
}
```

在某些情況下，您的作用中 CA 可能未設定撤銷機制。如果您想要開始使用憑證撤銷清單 (CRL)，請使用下列程序。

將 CRL 新增至現有的 CA (AWS CLI)

1. 使用下列命令來檢查 CA 的目前狀態。

```
$ aws acm-pca describe-certificate-authority
 --certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566
 --output json
```

輸出會確認 CA 具有狀態，ACTIVE但未設定為使用 CRL。

```
{
    "CertificateAuthority": {
```

```
        "Arn": "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
        "CreatedAt": "2021-03-08T14:36:26.449000-08:00",
        "LastStateChangeAt": "2021-03-08T14:50:52.224000-08:00",
        "Type": "ROOT",
        "Serial": "serial_number",
        "Status": "ACTIVE",
        "NotBefore": "2021-03-08T13:46:50-08:00",
        "NotAfter": "2022-03-08T14:46:50-08:00",
        "CertificateAuthorityConfiguration": {
            "KeyAlgorithm": "RSA_2048",
            "SigningAlgorithm": "SHA256WITHRSA",
            "Subject": {
                "Country": "US",
                "Organization": "Example Corp",
                "OrganizationalUnit": "Sales",
                "State": "WA",
                "CommonName": "www.example.com",
                "Locality": "Seattle"
            }
        },
        "RevocationConfiguration": {
            "CrlConfiguration": {
                "Enabled": false
            },
            "OcspConfiguration": {
                "Enabled": false
            }
        }
    }
}
```

2. 建立並儲存名稱為 的檔案，例如 revoke_config.txt以定義 CRL 組態參數。

```
{
   "CrlConfiguration":{
      "Enabled": true,
      "ExpirationInDays": 7,
      "S3BucketName": "amzn-s3-demo-bucket"
   }
}
```

> **Note**
>
> 更新事項裝置認證 CA 以啟用 CRLs時，您必須將其設定為省略發行憑證中的 CDP 延伸模組，以協助符合目前的事項標準。若要這樣做，請定義 CRL 組態參數，如下所示：
>
> ```
> {
>    "CrlConfiguration":{
>       "Enabled": true,
>       "ExpirationInDays": 7,
>       "S3BucketName": "amzn-s3-demo-bucket"
>       "CrlDistributionPointExtensionConfiguration":{
>          "OmitExtension": true
>       }
>    }
> }
> ```

3. 使用 update-certificate-authority 命令和撤銷組態檔案來更新 CA。

```
$ aws acm-pca update-certificate-authority \
      --certificate-authority-arn arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566 \
      --revocation-configuration file://revoke_config.txt
```

4. 再次檢查 CA 的狀態。

```
$ aws acm-pca describe-certificate-authority
 --certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566
 --output json
```

輸出會確認 CA 現在已設定為使用 CRL。

```
{
    "CertificateAuthority": {
        "Arn": "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
        "CreatedAt": "2021-03-08T14:36:26.449000-08:00",
        "LastStateChangeAt": "2021-03-08T14:50:52.224000-08:00",
        "Type": "ROOT",
        "Serial": "serial_number",
```

```
        "Status": "ACTIVE",
        "NotBefore": "2021-03-08T13:46:50-08:00",
        "NotAfter": "2022-03-08T14:46:50-08:00",
        "CertificateAuthorityConfiguration": {
            "KeyAlgorithm": "RSA_2048",
            "SigningAlgorithm": "SHA256WITHRSA",
            "Subject": {
                "Country": "US",
                "Organization": "Example Corp",
                "OrganizationalUnit": "Sales",
                "State": "WA",
                "CommonName": "www.example.com",
                "Locality": "Seattle"
            }
        },
        "RevocationConfiguration": {
            "CrlConfiguration": {
                "Enabled": true,
                "ExpirationInDays": 7,
                "S3BucketName": "amzn-s3-demo-bucket",
            },
            "OcspConfiguration": {
                "Enabled": false
            }
        }
    }
}
```

在某些情況下，您可能想要新增 OCSP 撤銷支援，而不是如先前程序中啟用 CRL。在這種情況下，請使用下列步驟。

將 OCSP 支援新增至現有的 CA (AWS CLI)

1. 使用 等名稱建立並儲存檔案，`revoke_config.txt`以定義您的 OCSP 參數。

```
{
   "OcspConfiguration":{
      "Enabled":true
   }
}
```

2. 使用 update-certificate-authority 命令和撤銷組態檔案來更新 CA。

```
$ aws acm-pca update-certificate-authority \
      --certificate-authority-arn arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566 \
      --revocation-configuration file://revoke_config.txt
```

3. 再次檢查 CA 的狀態。

```
$ aws acm-pca describe-certificate-authority
 --certificate-authority-arnarn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566
 --output json
```

輸出會確認 CA 現在已設定為使用 OCSP。

```
{
    "CertificateAuthority": {
        "Arn": "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
        "CreatedAt": "2021-03-08T14:36:26.449000-08:00",
        "LastStateChangeAt": "2021-03-08T14:50:52.224000-08:00",
        "Type": "ROOT",
        "Serial": "serial_number",
        "Status": "ACTIVE",
        "NotBefore": "2021-03-08T13:46:50-08:00",
        "NotAfter": "2022-03-08T14:46:50-08:00",
        "CertificateAuthorityConfiguration": {
            "KeyAlgorithm": "RSA_2048",
            "SigningAlgorithm": "SHA256WITHRSA",
            "Subject": {
                "Country": "US",
                "Organization": "Example Corp",
                "OrganizationalUnit": "Sales",
                "State": "WA",
                "CommonName": "www.example.com",
                "Locality": "Seattle"
            }
        },
        "RevocationConfiguration": {
            "CrlConfiguration": {
                "Enabled": false
            },
            "OcspConfiguration": {
```

```
                "Enabled": true
            }
        }
    }
}
```

> **Note**
>
> 您也可以在 CA 上同時設定 CRL 和 OCSP 支援。

刪除您的私有 CA

您可以從 AWS 管理主控台 或 AWS CLI 永久刪除私有 CA。例如，您可能想要刪除某 CA，將其取代成具有新私有金鑰的新 CA。若要安全地刪除 CA，請遵循下列步驟：

1. 建立替換 CA。
2. 一旦新的私有 CA 在生產環境中，請停用舊的私有 CA，但不要立即刪除。
3. 請將舊 CA 保持停用，直到其發行的所有憑證皆過期為止。
4. 刪除舊 CA。

AWS 私有 CA 在處理刪除請求之前， 不會檢查所有發行的憑證是否已過期。您可以產生稽核報告來判斷哪些憑證已過期。雖然 CA 已停用，您仍可撤銷憑證，但無法發行新的憑證。

如果您必須在所有發行的憑證過期前刪除私有 CA，我們建議您撤銷 CA 憑證。CA 憑證將會列於上層 CA 的 CRL 中，私有 CA 將不受用戶端信任。

> **Important**
>
> 私有 CA 處於 PENDING_CERTIFICATE、CREATING、EXPIRED、DISABLED 或 FAILED 狀態時，可以刪除。若要刪除處於 ACTIVE 狀態的 CA，您必須先進行停用，否則刪除請求會導致例外狀況。如果您要刪除處於 PENDING_CERTIFICATE或 DISABLED 狀態的私有 CA，您可以將其還原期間的長度設定為 7-30 天，預設值為 30。在此期間，狀態會設為 DELETED 且 CA 可進行還原。在 CREATING或 FAILED 狀態時刪除的私有 CA 沒有指派的還原期間，而且無法還原。如需詳細資訊，請參閱還原私有 CA。

私有 CA 刪除後，您不需付費。不過，如果還原已刪除的 CA，您需支付刪除到還原這段期間的費用。如需詳細資訊，請參閱的定價 AWS 私有憑證授權單位。

刪除私有 CA (主控台)

1. 登入 AWS 您的帳戶，並在 https：//https://console.aws.amazon.com/acm-pca/home 開啟 AWS 私有 CA 主控台。
2. 在私有憑證授權單位頁面上，從清單中選擇您的私有 CA。
3. 如果您的 CA 處於 ACTIVE 狀態，您必須先停用它。在 Actions (動作) 選單上，選擇 Disable (停用)。出現提示時，選擇我了解風險，繼續。
4. 對於未處於 ACTIVE 狀態的 CA，請選擇動作、刪除。
5. 如果您的 CA 處於 DISABLED、 EXPIRED或 PENDING_CERTIFICATE 狀態，刪除 CA 頁面可讓您指定 7-30 天的還原期間。如果您的私有 CA 未處於這些狀態之一，則無法稍後還原，且刪除是永久的。
6. 選擇 刪除。
7. 如果您確定要刪除私有 CA，請在系統提示時選擇 Permanently delete (永久刪除)。私有 CA 的狀態會變更為 DELETED。不過，您可以在還原期間結束之前還原私有 CA。若要檢查DELETED處於狀態之私有 CA 的還原期間，請呼叫 DescribeCerticateAuthority 或 ListCertificateAuthorities API 操作。

刪除私有 CA (AWS CLI)

使用 delete-certificate-authority 命令來刪除私有 CA。

```
$ aws acm-pca delete-certificate-authority \
     --certificate-authority-arn arn:aws:acm-pca:region:account:certificate-
authority/CA_ID \
     --permanent-deletion-time-in-days 16
```

還原私有 CA

您可以還原已刪除的私有 CA，只要 CA 在刪除後仍處於您指定的還原期間內即可。還原期間為 7-30 天。此期間結束後，私有 CA 將永久刪除。如需詳細資訊，請參閱刪除您的私有 CA。您不能還原已永久刪除的私有 CA。

> **Note**
>
> 私有 CA 刪除後，您不需付費。不過，如果還原已刪除的 CA，您需支付刪除到還原這段期間的費用。如需詳細資訊，請參閱的定價 AWS 私有憑證授權單位。

還原私有 CA （主控台）

您可以使用 AWS 管理主控台 來還原私有 CA。

若要還原私有 CA (主控台)

1. 登入 AWS 您的帳戶，並在 https：//https://console.aws.amazon.com/acm-pca/home 開啟 AWS 私有 CA 主控台。
2. 在私有憑證授權單位頁面上，從清單中選擇已刪除的私有 CA。
3. 在操作功能表上，選擇 Restore (還原)。
4. 在還原 CA 頁面上，再次選擇還原。
5. 如果成功，私有 CA 的狀態會設為其刪除前狀態。選擇動作、啟用和再次啟用，將其狀態變更為 ACTIVE。如果私有 CA 在刪除時處於 PENDING_CERTIFICATE 狀態，您必須先匯入 CA 憑證到私有 CA，才能進行啟用。

還原私有 CA (AWS CLI)

使用 restore-certificate-authority 命令來還原處於 DELETED 狀態的已刪除私有 CA。以下步驟討論刪除、還原，然後重新啟用私有 CA 所需的整個程序。

若要刪除、還原和重新啟用私有 CA (AWS CLI)

1. 刪除私有 CA。

   執行 delete-certificate-authority 命令來刪除私有 CA。如果私有 CA 的狀態為 DISABLED或 PENDING_CERTIFICATE，您可以設定 --permanent-deletion-time-in-days 參數來指定私有 CA 的還原期間，從 7 到 30 天。如果您未指定還原期間，預設為 30 天。如果成功，這個命令會將私有 CA 的狀態設為 DELETED。

> **Note**
>
> 若要可供還原，在刪除當時的私有 CA 狀態必須為 DISABLED 或 PENDING_CERTIFICATE。

```
$ aws acm-pca delete-certificate-authority \
     --certificate-authority-arn arn:aws:acm-pca:region:account:certificate-
authority/CA_ID \
     --permanent-deletion-time-in-days 16
```

2. 還原私有 CA。

   執行 restore-certificate-authority 命令來還原私有 CA。您必須在您以 delete-certificate-authority 命令設定的還原期間過期前執行命令。如果成功，此命令會將私有 CA 的狀態設為其刪除前狀態。

```
$ aws acm-pca restore-certificate-authority \
     --certificate-authority-arn arn:aws:acm-pca:region:account:certificate-
authority/CA_ID
```

3. 將私有 CA 設為 ACTIVE。

   執行 update-certificate-authority 命令來將私有 CA 的狀態變更為 ACTIVE。

```
$ aws acm-pca update-certificate-authority \
     --certificate-authority-arn arn:aws:acm-pca:region:account:certificate-
authority/CA_ID \
     --status ACTIVE
```

使用外部簽署的私有 CA 憑證

如果您私有 CA 階層的信任根必須是外部的 CA AWS 私有 CA，您可以建立並自我簽署自己的根 CA。或者，您也可以取得由您組織營運之外部私有 CA 所簽署的私有 CA 憑證。無論來源為何，您都可以使用此外部取得的 CA 來簽署 AWS 私有 CA 管理的私有次級 CA 憑證。

> **Note**
>
> 建立或取得外部信任服務提供者的程序不在本指南的範圍內。

使用外部父 CA 搭配 AWS 私有 CA 可讓您強制執行 RFC 5280 名稱限制區段中定義的 CA 名稱限制。名稱限制可讓 CA 管理員限制憑證中的主體名稱。

如果您計劃使用外部 CA 簽署私有次級 CA 憑證，在您擁有工作 CA 之前，需要完成三個任務 AWS 私有 CA：

1. 產生憑證簽署請求 (CSR)。
2. 將 CSR 提交至外部簽署授權機構，並傳回已簽署的憑證和憑證鏈。
3. 在 中安裝已簽署的憑證 AWS 私有 CA。

下列程序說明如何使用 AWS 管理主控台 或 來完成這些任務 AWS CLI。

取得並安裝外部簽署的 CA 憑證 （主控台）

1. （選用）如果您尚未在 CA 的詳細資訊頁面上，請開啟位於 https：//https://console.aws.amazon.com/acm-pca/home 的 AWS 私有 CA 主控台。在私有憑證授權單位頁面上，選擇狀態為待定憑證、作用中、已停用或已過期的次級 CA。
2. 選擇動作、安裝 CA 憑證以開啟安裝次級 CA 憑證頁面。
3. 在安裝次級 CA 憑證頁面的選取 CA 類型下，選擇外部私有 CA。
4. 在此 CA 的 CSR 下，主控台會顯示 CSR 的 Base64-encoded ASCII 文字。您可以使用複製按鈕複製文字，也可以選擇將 CSR 匯出至檔案，並將其儲存在本機。

   > **Note**
   >
   > 複製和貼上時，必須保留 CSR 文字的確切格式。

5. 如果您無法立即執行離線步驟，從外部簽署授權單位取得已簽署的憑證，您可以關閉頁面，並在您擁有已簽署的憑證和憑證鏈後返回頁面。

   否則，如果您已準備好，請執行下列其中一項操作：

   - 將憑證內文和憑證鏈的 Base64-encoded ASCII 文字貼入各自的文字方塊。

- 選擇上傳，將憑證內文和憑證鏈從本機檔案載入各自的文字方塊。

6. 選擇確認並安裝。

取得並安裝外部簽署的 CA 憑證 (CLI)

1. 使用 get-certificate-authority-csr 命令，為您的私有 CA 擷取憑證簽署要求 (CSR)。如果您想要傳送 CSR 至顯示內容，請使用 `--output text` 選項，消除每一行結尾處的 CR/LF 字元。若要傳送 CSR 至檔案，請使用重新導向選項 (>)，後面加上檔案名稱。

```
$ aws acm-pca get-certificate-authority-csr \
--certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566 \
--output text
```

將 CSR 儲存為本機檔案後，您可以使用下列 OpenSSL 命令來檢查它：

```
openssl req -in path_to_CSR_file -text -noout
```

此命令會產生類似如下的輸出。請注意，CA 延伸為 TRUE，表示該 CSR 適用於 CA 憑證。

```
Certificate Request:
Data:
Version: 0 (0x0)
Subject: O=ExampleCompany, OU=Corporate Office, CN=Example CA 1
Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
        Public-Key: (2048 bit)
        Modulus:
            00:d4:23:51:b3:dd:01:09:01:0b:4c:59:e4:ea:81:
            1d:7f:48:36:ef:2a:e9:45:82:ec:95:1d:c6:d7:c9:
            7f:19:06:73:c5:cd:63:43:14:eb:c8:03:82:f8:7b:
            c7:89:e6:8d:03:eb:b6:76:58:70:f2:cb:c3:4c:67:
            ea:50:fd:b9:17:84:b8:60:2c:64:9d:2e:d5:7d:da:
            46:56:38:34:a9:0d:57:77:85:f1:6f:b8:ce:73:eb:
            f7:62:a7:8e:e6:35:f5:df:0c:f7:3b:f5:7f:bd:f4:
            38:0b:95:50:2c:be:7d:bf:d9:ad:91:c3:81:29:23:
            b2:5e:a6:83:79:53:f3:06:12:20:7e:a8:fa:18:d6:
            a8:f3:a3:89:a5:a3:6a:76:da:d0:97:e5:13:bc:84:
            a6:5c:d6:54:1a:f0:80:16:dd:4e:79:7b:ff:6d:39:
```

```
            b5:67:56:cb:02:6b:14:c3:17:06:0e:7d:fb:d2:7e:
            1c:b8:7d:1d:83:13:59:b2:76:75:5e:d1:e3:23:6d:
            8a:5e:f5:85:ca:d7:e9:a3:f1:9b:42:9f:ed:8a:3c:
            14:4d:1f:fc:95:2b:51:6c:de:8f:ee:02:8c:0c:b6:
            3e:2d:68:e5:f8:86:3f:4f:52:ec:a6:f0:01:c4:7d:
            68:f3:09:ae:b9:97:d6:fc:e4:de:58:58:37:09:9a:
            f6:27
        Exponent: 65537 (0x10001)
Attributes:
Requested Extensions:
    X509v3 Basic Constraints:
        CA:TRUE
Signature Algorithm: sha256WithRSAEncryption
 c5:64:0e:6c:cf:11:03:0b:b7:b8:9e:48:e1:04:45:a0:7f:cc:
 a7:fd:e9:4d:c9:00:26:c5:6e:d0:7e:69:7a:fb:17:1f:f3:5d:
 ac:f3:65:0a:96:5a:47:3c:c1:ee:45:84:46:e3:e6:05:73:0c:
 ce:c9:a0:5e:af:55:bb:89:46:21:92:7b:10:96:92:1b:e6:75:
 de:02:13:2d:98:72:47:bd:b1:13:1a:3d:bb:71:ae:62:86:1a:
 ee:ae:4e:f4:29:2e:d6:fc:70:06:ac:ca:cf:bb:ee:63:68:14:
 8e:b2:8f:e3:8d:e8:8f:e0:33:74:d6:cf:e2:e9:41:ad:b6:47:
 f8:2e:7d:0a:82:af:c6:d8:53:c2:88:a0:32:05:09:e0:04:8f:
 79:1c:ac:0d:d4:77:8e:a6:b2:5f:07:f8:1b:e3:98:d4:12:3d:
 28:32:82:b5:50:92:a4:b2:4c:28:fc:d2:73:75:75:ff:10:33:
 2c:c0:67:4b:de:fd:e6:69:1c:a8:bb:e8:31:93:07:35:69:b7:
 d6:53:37:53:d5:07:dd:54:35:74:50:50:f9:99:7d:38:b7:b6:
 7f:bd:6c:b8:e4:2a:38:e5:04:00:a8:a3:d9:e5:06:38:e0:38:
 4c:ca:a9:3c:37:6d:ba:58:38:11:9c:30:08:93:a5:62:00:18:
 d1:83:66:40
```

2. 將 CSR 提交至外部簽署授權機構，並取得包含 Base64 PEM 編碼簽章憑證和憑證鏈的檔案。
3. 使用 import-certificate-authority-certificate 命令，將私有 CA 憑證檔案和鏈結檔案匯入 AWS 私有 CA。

```
$ aws acm-pca import-certificate-authority-certificate \
--certificate-authority-arn arn:aws:acm-pca:region:account:\
certificate-authority/12345678-1234-1234-1234-123456789012 \
--certificate file://C:\example_ca_cert.pem \
--certificate-chain file://C:\example_ca_cert_chain.pem
```

在 中發行和管理憑證 AWS 私有 CA

在您建立並啟用私有憑證授權機構 (CA) 並設定存取後，您或您的授權使用者可以發行和管理憑證。如果您尚未為 CA 設定 AWS Identity and Access Management (IAM) 政策，您可以在本指南的 Identity and Access Management 區段中進一步了解如何設定這些政策。如需在單一帳戶和跨帳戶案例中設定 CA 存取的資訊，請參閱 控制私有 CA 的存取。

主題

- 發行私有終端實體憑證
- 擷取私有憑證
- 列出私有憑證
- 匯出私有憑證及其私密金鑰
- 撤銷私有憑證
- 自動化匯出續約的憑證
- 使用 AWS 私有 CA 憑證範本

發行私有終端實體憑證

使用私有 CA 時，您可以從 AWS Certificate Manager (ACM) 或 請求私有終端實體憑證 AWS 私有 CA。下表會比較這兩種服務的功能。

功能	ACM	AWS 私有 CA
發行終端實體憑證	✓（使用 RequestCertificate 或主控台）	✓（使用 IssueCertificate)
與負載平衡器和面向網際網路 AWS 的服務建立關聯	✓	不支援
受管憑證續約	✓	透過 ACM 間接支援
主控台支援	✓	不支援
API 支援	✓	✓
CLI 支援	✓	✓

當 AWS 私有 CA 建立憑證時，它會遵循指定憑證類型和路徑長度的範本。如果未將範本 ARN 提供給建立憑證的 API 或 CLI 陳述式，則預設會套用 EndEntityCertificate/V1 範本。如需可用憑證範本的詳細資訊，請參閱使用 AWS 私有 CA 憑證範本。

雖然 ACM 憑證是以公有信任為基礎設計，但 AWS 私有 CA 可滿足您的私有 PKI 需求。因此，您可以使用 AWS 私有 CA API 和 CLI，以 ACM 不允許的方式設定憑證。這些索引標籤包括以下項目：

- 使用任何主體名稱建立憑證。
- 使用任何支援的私有金鑰演算法和金鑰長度。
- 使用任何支援的簽署演算法。
- 指定私有 CA 和私有憑證的任何有效期間。

使用 建立私有 TLS 憑證之後 AWS 私有 CA，您可以將其匯入 ACM，並搭配支援 AWS 的服務使用。

> **Note**
>
> 使用下列程序、 issue-certificate命令或 IssueCertificate API 動作建立的憑證無法直接匯出以供外部使用 AWS。不過，您可以使用私有 CA 來簽署透過 ACM 發行的憑證，而且這些憑證可以與其私密金鑰一起匯出。如需詳細資訊，請參閱《ACM 使用者指南》中的請求私有憑證和匯出私有憑證。

發行標準憑證 (AWS CLI)

您可以使用 AWS 私有 CA CLI 命令 issue-certificate 或 API 動作 IssueCertificate 來請求終端實體憑證。此命令需要您要使用的私有 CA 的 Amazon Resource Name (ARN) 來發行憑證。您還必須使用 OpenSSL 等程式產生憑證簽署請求 (CSR)。

如果您使用 AWS 私有 CA API 或 AWS CLI 發行私有憑證，憑證將不受管理，這表示您無法使用 ACM 主控台、ACM CLI 或 ACM API 來檢視或匯出憑證，而且憑證不會自動續約。不過，您可以使用 PCA get-certificate 命令來擷取憑證詳細資訊，如果您擁有 CA，則可以建立稽核報告。

建立憑證時的考量事項

- 為了符合 RFC 5280，您提供的網域名稱（技術上是通用名稱）的長度不能超過 64 個八位元組（字元），包括句點。若要新增較長的網域名稱，請在主體別名欄位中指定，該欄位支援長度上限為 253 個八位元組的名稱。

- 如果您使用的是 1.6.3 AWS CLI 版或更新版本，請在指定 base64 編碼的輸入檔案`fileb://`時，使用 字首，例如 CSRs。這可確保 AWS 私有 CA 正確剖析資料。

下列 OpenSSL 命令會產生憑證的 CSR 和私有金鑰：

```
$ openssl req -out csr.pem -new -newkey rsa:2048 -nodes -keyout private-key.pem
```

您可以檢查 CSR 的內容，如下所示：

```
$ openssl req -in csr.pem -text -noout
```

產生的輸出應類似下列縮寫範例：

```
Certificate Request:
    Data:
        Version: 0 (0x0)
        Subject: C=US, O=Big Org, CN=example.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:ca:85:f4:3a:b7:5f:e2:66:be:fc:d8:97:65:3d:
                    a4:3d:30:c6:02:0a:9e:1c:ca:bb:15:63:ca:22:81:
                    00:e1:a9:c0:69:64:75:57:56:53:a1:99:ee:e1:cd:
                    ...
                    aa:38:73:ff:3d:b7:00:74:82:8e:4a:5d:da:5f:79:
                    5a:89:52:e7:de:68:95:e0:16:9b:47:2d:57:49:2d:
                    9b:41:53:e2:7f:e1:bd:95:bf:eb:b3:a3:72:d6:a4:
                    d3:63
                Exponent: 65537 (0x10001)
        Attributes:
            a0:00
    Signature Algorithm: sha256WithRSAEncryption
         74:18:26:72:33:be:ef:ae:1d:1e:ff:15:e5:28:db:c1:e0:80:
         42:2c:82:5a:34:aa:1a:70:df:fa:4f:19:e2:5a:0e:33:38:af:
         21:aa:14:b4:85:35:9c:dd:73:98:1c:b7:ce:f3:ff:43:aa:11:
         ....
         3c:b2:62:94:ad:94:11:55:c2:43:e0:5f:3b:39:d3:a6:4b:47:
         09:6b:9d:6b:9b:95:15:10:25:be:8b:5c:cc:f1:ff:7b:26:6b:
         fa:81:df:e4:92:e5:3c:e5:7f:0e:d8:d9:6f:c5:a6:67:fb:2b:
         0b:53:e5:22
```

下列命令會建立憑證。由於未指定範本，預設會發出基本終端實體憑證。

```
$ aws acm-pca issue-certificate \
      --certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566 \
      --csr fileb://csr.pem \
      --signing-algorithm "SHA256WITHRSA" \
      --validity Value=365,Type="DAYS"
```

已發行憑證的 ARN 會傳回：

```
{
   "CertificateArn":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID"
}
```

> **Note**
>
> AWS 私有 CA 收到issue-certificate命令時， 會立即傳回具有序號的 ARN。不過，憑證處理會以非同步方式進行，但仍可能會失敗。如果發生這種情況，使用新 ARN 的get-certificate命令也會失敗。

使用 APIPassthrough 範本發行具有自訂主旨名稱的憑證

在此範例中，會發出包含自訂主旨名稱元素的憑證。除了提供 中類似 的 CSR 之外發行標準憑證 (AWS CLI)，您還會將兩個額外的引數傳遞至 issue-certificate命令：APIPassthrough 範本的 ARN，以及指定自訂屬性及其物件識別符 (OIDs JSON 組態檔案。您無法`StandardAttributes`搭配 使用`CustomAttributes`。不過，您可以將標準 OIDs 做為 的一部分傳遞`CustomAttributes`。預設主旨名稱 OIDs 會列在下表中（來自 RFC 4519 和全域 OID 參考資料庫的資訊）：

主旨名稱	縮寫	物件 ID
countryName	c	2.5.4.6
commonName	cn	2.5.4.3
dnQualifier 【辨別名稱限定詞】		2.5.4.46

主旨名稱	縮寫	物件 ID
generationQualifier		2.5.4.44
givenName		2.5.4.42
初始		2.5.4.43
地區性	l	2.5.4.7
organizationName	o	2.5.4.10
organizationalUnitName	ou	2.5.4.11
假名		2.5.4.65
serialNumber		2.5.4.5
st 【狀態】		2.5.4.8
姓氏	sn	2.5.4.4
標題		2.5.4.12
domainComponent	dc	0.9.2342.19200300.100.1.25
userid		0.9.2342.19200300.100.1.1

範例組態檔案`api_passthrough_config.txt`包含下列程式碼：

```
{
  "Subject": {
    "CustomAttributes": [
      {
        "ObjectIdentifier": "2.5.4.6",
        "Value": "US"
      },
      {
        "ObjectIdentifier": "1.3.6.1.4.1.37244.1.1",
        "Value": "BCDABCDA12341234"
      },
      {
```

```
        "ObjectIdentifier": "1.3.6.1.4.1.37244.1.5",
        "Value": "CDABCDAB12341234"
      }
    ]
  }
}
```

使用下列命令來發行憑證：

```
$ aws acm-pca issue-certificate \
      --validity Type=DAYS,Value=10
      --signing-algorithm "SHA256WITHRSA" \
      --csr fileb://csr.pem \
      --api-passthrough file://api_passthrough_config.txt \
      --template-arn arn:aws:acm-pca:::template/
BlankEndEntityCertificate_APIPassthrough/V1 \
      --certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566
```

已發行憑證的 ARN 會傳回：

```
{
   "CertificateArn":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID"
}
```

在本機擷取憑證，如下所示：

```
$ aws acm-pca get-certificate \
      --certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566 \
      --certificate-arn arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID | \
      jq -r .'Certificate' > cert.pem
```

您可以使用 OpenSSL 檢查憑證的內容：

```
$ openssl x509 -in cert.pem -text -noout
```

> **Note**
>
> 您也可以建立私有 CA，將自訂屬性傳遞給其發行的每個憑證。

使用 APIPassthrough 範本發行具有自訂擴充功能的憑證

在此範例中，會發出包含自訂延伸項目的憑證。為此，您需要將三個引數傳遞至 issue-certificate命令：APIPassthrough 範本的 ARN，以及指定自訂副檔名的 JSON 組態檔案，以及類似 中所示的 CSR發行標準憑證 (AWS CLI)。

範例組態檔案`api_passthrough_config.txt`包含下列程式碼：

```
{
  "Extensions": {
    "CustomExtensions": [
      {
        "ObjectIdentifier": "2.5.29.30",
        "Value": "MBWgEzARgg8ucGVybWl0dGVkLnRlc3Q=",
        "Critical": true
      }
    ]
  }
}
```

自訂憑證的發行方式如下：

```
$ aws acm-pca issue-certificate \
      --validity Type=DAYS,Value=10
      --signing-algorithm "SHA256WITHRSA" \
      --csr fileb://csr.pem \
      --api-passthrough file://api_passthrough_config.txt \
      --template-arn arn:aws:acm-pca:::template/EndEntityCertificate_APIPassthrough/V1
 \
      --certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566
```

已發行憑證的 ARN 會傳回：

```
{
```

```
   "CertificateArn":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID"
}
```

在本機擷取憑證，如下所示：

```
$ aws acm-pca get-certificate \
      --certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566 \
      --certificate-arn arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID | \
      jq -r .'Certificate' > cert.pem
```

您可以使用 OpenSSL 檢查憑證的內容：

```
$ openssl x509 -in cert.pem -text -noout
```

擷取私有憑證

您可以使用 AWS 私有 CA API 和來 AWS CLI 發行私有憑證。如果您這麼做，您可以使用 AWS CLI 或 AWS 私有 CA API 來擷取該憑證。如果您使用 ACM 建立私有 CA 並請求憑證，則必須使用 ACM 匯出憑證和加密的私有金鑰。如需詳細資訊，請參閱匯出私有憑證。

擷取終端實體憑證

使用 get-certificate AWS CLI 命令來擷取私有終端實體憑證。您也可以使用 GetCertificate API 操作。我們建議使用類似 Sed 的剖析器 jq 格式化輸出。

> **Note**
>
> 如果您想要撤銷憑證，可以使用 get-certificate 命令以十六進制格式擷取序號。您也可以建立稽核報告以擷取十六進位序號。如需詳細資訊，請參閱將稽核報告與私有 CA 搭配使用。

```
$ aws acm-pca get-certificate \
      --certificate-arn arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID \
      --certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566 | \
      jq -r '.Certificate, .CertificateChain'
```

此命令會以下列標準格式輸出憑證和憑證鏈。

```
-----BEGIN CERTIFICATE-----
...base64-encoded certificate...
-----END CERTIFICATE----
-----BEGIN CERTIFICATE-----
...base64-encoded certificate...
-----END CERTIFICATE----
-----BEGIN CERTIFICATE-----
...base64-encoded certificate...
-----END CERTIFICATE----
```

擷取 CA 憑證

您可以使用 AWS 私有 CA API 和 AWS CLI 來擷取私有 CA 的憑證授權單位 (CA) 憑證。執行 get-certificate-authority-certificate 命令。您也可以呼叫 GetCertificateAuthorityCertificate 操作。建議您使用類似 Sed 的剖析器 jq 格式化輸出。

```
$ aws acm-pca get-certificate-authority-certificate \
     --certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566 \
     | jq -r '.Certificate'
```

此命令會以下列標準格式輸出 CA 憑證。

```
-----BEGIN CERTIFICATE-----
...base64-encoded certificate...
-----END CERTIFICATE----
```

列出私有憑證

若要列出您的私有憑證，請產生稽核報告、從其 S3 儲存貯體擷取報告，並視需要剖析報告內容。如需建立 AWS 私有 CA 稽核報告的資訊，請參閱 將稽核報告與私有 CA 搭配使用。如需有關從 S3 儲存貯體擷取物件的資訊，請參閱《Amazon Simple Storage Service 使用者指南》中的下載物件。

下列範例說明建立稽核報告的方法，並剖析這些報告以取得有用的資料。結果會以 JSON 格式，並使用類似 Sed 的剖析器 jq 來篩選資料。

1. 建立稽核報告。

下列命令會為指定的 CA 產生稽核報告。

```
$ aws acm-pca create-certificate-authority-audit-report \
     --region region \
     --certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566 \
     --s3-bucket-name bucket_name \
     --audit-report-response-format JSON
```

成功時，命令會傳回新稽核報告的 ID 和位置。

```
{
   "AuditReportId":"audit_report_ID",
   "S3Key":"audit-report/CA_ID/audit_report_ID.json"
}
```

2. 擷取並格式化稽核報告。

此命令會擷取稽核報告、在標準輸出中顯示其內容，並篩選結果，以僅顯示 2020-12-01 當天或之後發行的憑證。

```
$ aws s3api get-object \
     --region region \
     --bucket bucket_name \
     --key audit-report/CA_ID/audit_report_ID.json \
     /dev/stdout | jq '.[] | select(.issuedAt >= "2020-12-01")'
```

傳回的項目如下所示：

```
{
   "awsAccountId":"account",
   "certificateArn":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID",
   "serial":"serial_number",
   "subject":"CN=pca.alpha.root2.leaf5",
   "notBefore":"2020-12-21T21:28:09+0000",
   "notAfter":"9999-12-31T23:59:59+0000",
   "issuedAt":"2020-12-21T22:28:09+0000",
   "templateArn":"arn:aws:acm-pca:::template/EndEntityCertificate/V1"
}
```

3. 在本機儲存稽核報告。

如果您想要執行多個查詢，將稽核報告儲存至本機檔案相當方便。

```
$ aws s3api get-object \
     --region region \
     --bucket bucket_name \
     --key audit-report/CA_ID/audit_report_ID.json > my_local_audit_report.json
```

與 相同的篩選條件會產生相同的輸出：

```
$ cat my_local_audit_report.json | jq '.[] | select(.issuedAt >= "2020-12-01")'
{
   "awsAccountId":"account",
   "certificateArn":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID",
   "serial":"serial_number",
   "subject":"CN=pca.alpha.root2.leaf5",
   "notBefore":"2020-12-21T21:28:09+0000",
   "notAfter":"9999-12-31T23:59:59+0000",
   "issuedAt":"2020-12-21T22:28:09+0000",
   "templateArn":"arn:aws:acm-pca:::template/EndEntityCertificate/V1"
}
```

4. 日期範圍內的查詢

您可以查詢日期範圍內發行的憑證，如下所示：

```
$ cat my_local_audit_report.json | jq '.[] | select(.issuedAt >= "2020-11-01"
 and .issuedAt <= "2020-11-10")'
```

篩選的內容會顯示在標準輸出中：

```
{
   "awsAccountId": "account",
   "certificateArn": "arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID",
   "serial": "serial_number",
   "subject": "CN=pca.alpha.root2.leaf1",
   "notBefore": "2020-11-06T19:18:21+0000",
   "notAfter": "9999-12-31T23:59:59+0000",
   "issuedAt": "2020-11-06T20:18:22+0000",
   "templateArn": "arn:aws:acm-pca:::template/EndEntityCertificate/V1"
}
{
   "awsAccountId": "account",
```

```
   "certificateArn": "arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID",
   "serial": "serial_number",
   "subject": "CN=pca.alpha.root2.rsa2048sha256",
   "notBefore": "2020-11-06T19:15:46+0000",
   "notAfter": "9999-12-31T23:59:59+0000",
   "issuedAt": "2020-11-06T20:15:46+0000",
   "templateArn": "arn:aws:acm-pca:::template/RootCACertificate/V1"
}
{
   "awsAccountId": "account",
   "certificateArn": "arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID",
   "serial": "serial_number",
   "subject": "CN=pca.alpha.root2.leaf2",
   "notBefore": "2020-11-06T20:04:39+0000",
   "notAfter": "9999-12-31T23:59:59+0000",
   "issuedAt": "2020-11-06T21:04:39+0000",
   "templateArn": "arn:aws:acm-pca:::template/EndEntityCertificate/V1"
}
```

5. 搜尋指定範本之後的憑證。

下列命令會使用範本 ARN 篩選報告內容：

```
$ cat my_local_audit_report.json | jq '.[] | select(.templateArn == "arn:aws:acm-
pca:::template/RootCACertificate/V1")'
```

輸出會顯示相符的憑證記錄：

```
{
   "awsAccountId": "account",
   "certificateArn": "arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID",
   "serial": "serial_number",
   "subject": "CN=pca.alpha.root2.rsa2048sha256",
   "notBefore": "2020-11-06T19:15:46+0000",
   "notAfter": "9999-12-31T23:59:59+0000",
   "issuedAt": "2020-11-06T20:15:46+0000",
   "templateArn": "arn:aws:acm-pca:::template/RootCACertificate/V1"
}
```

6. 篩選已撤銷的憑證

若要尋找所有已撤銷的憑證，請使用下列命令：

```
$ cat my_local_audit_report.json | jq '.[] | select(.revokedAt != null)'
```

撤銷的憑證會顯示如下：

```
{
   "awsAccountId": "account",
   "certificateArn": "arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID",
   "serial": "serial_number",
   "subject": "CN=pca.alpha.root2.leaf2",
   "notBefore": "2020-11-06T20:04:39+0000",
   "notAfter": "9999-12-31T23:59:59+0000",
   "issuedAt": "2020-11-06T21:04:39+0000",
   "revokedAt": "2021-05-27T18:57:32+0000",
   "revocationReason": "UNSPECIFIED",
   "templateArn": "arn:aws:acm-pca:::template/EndEntityCertificate/V1"
}
```

7. 使用規則表達式進行篩選。

下列命令會搜尋包含字串 "leaf" 的主題名稱：

```
$ cat my_local_audit_report.json | jq '.[] | select(.subject|test("leaf"))'
```

將傳回相符的憑證記錄，如下所示：

```
{
   "awsAccountId": "account",
   "certificateArn": "arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID",
   "serial": "serial_number",
   "subject": "CN=pca.alpha.roo2.leaf4",
   "notBefore": "2020-11-16T18:17:10+0000",
   "notAfter": "9999-12-31T23:59:59+0000",
   "issuedAt": "2020-11-16T19:17:12+0000",
   "templateArn": "arn:aws:acm-pca:::template/EndEntityCertificate/V1"
}
{
   "awsAccountId": "account",
```

```
   "certificateArn": "arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID",
   "serial": "serial_number",
   "subject": "CN=pca.alpha.root2.leaf5",
   "notBefore": "2020-12-21T21:28:09+0000",
   "notAfter": "9999-12-31T23:59:59+0000",
   "issuedAt": "2020-12-21T22:28:09+0000",
   "templateArn": "arn:aws:acm-pca:::template/EndEntityCertificate/V1"
}
{
   "awsAccountId": "account",
   "certificateArn": "arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID",
   "serial": "serial_number",
   "subject": "CN=pca.alpha.root2.leaf1",
   "notBefore": "2020-11-06T19:18:21+0000",
   "notAfter": "9999-12-31T23:59:59+0000",
   "issuedAt": "2020-11-06T20:18:22+0000",
   "templateArn": "arn:aws:acm-pca:::template/EndEntityCertificate/V1"
}
```

匯出私有憑證及其私密金鑰

AWS 私有 CA 無法直接匯出已簽署並發行的私有憑證。不過，您可以使用 AWS Certificate Manager 匯出這類憑證及其加密的私密金鑰。然後，憑證可以完全可攜式，以便在私有 PKI 的任何位置進行部署。如需詳細資訊，請參閱 AWS Certificate Manager 《使用者指南》中的匯出私有憑證。

為增加效益， 為使用 ACM 主控台發行的私有憑證、ACM API `RequestCertificate`的動作或 的 ACM 區段中的 request-certificate命令 AWS Certificate Manager 提供受管續約 AWS CLI。如需續約的詳細資訊，請參閱在私有 PKI 中續約憑證。

撤銷私有憑證

您可以使用 revoke-certificate AWS CLI 命令或 RevokeCertificate API 動作來撤銷 AWS 私有 CA 憑證。例如，如果憑證的私密金鑰遭到入侵或其相關聯的網域變成無效，則憑證可能需要在排程過期之前撤銷。為了讓撤銷生效，使用憑證的用戶端需要一種方法來檢查每次嘗試建置安全網路連線時的撤銷狀態。

AWS 私有 CA 提供兩種完全受管的機制來支援撤銷狀態檢查：線上憑證狀態通訊協定 (OCSP) 和憑證撤銷清單 (CRLs)。使用 OCSP，用戶端會查詢授權撤銷資料庫，以即時傳回狀態。使用 CRL 時，用戶端會根據其定期下載和存放的已撤銷憑證清單來檢查憑證。用戶端拒絕接受已撤銷的憑證。

OCSP 和 CRLs都取決於內嵌在憑證中的驗證資訊。因此，發行 CA 必須設定為在發行之前支援其中一個或兩個機制。如需透過 選取和實作受管撤銷的資訊 AWS 私有 CA，請參閱 規劃您的 AWS 私有 CA 憑證撤銷方法。

撤銷的憑證一律會記錄在 AWS 私有 CA 稽核報告中。

> **Note**
>
> 對於跨帳戶發起人，需要具有 AWSRAMRevokeCertificateCertificateAuthority許可的共享。撤銷許可不包含在 中AWSRAMDefaultPermissionCertificateAuthority。若要啟用跨帳戶發行者撤銷，CA 管理員必須建立兩個 RAM 共用，兩個共用都指向相同的 CA：
>
> 1. 具有 AWSRAMRevokeCertificateCertificateAuthority許可的共用。
> 2. 具有 AWSRAMDefaultPermissionCertificateAuthority許可的共用。

撤銷憑證

使用 RevokeCertificate API 動作或 revoke-certificate 命令來撤銷私有 PKI 憑證。序號必須為十六進位格式。您可以透過呼叫 get-certificate 命令來擷取序號。revoke-certificate 命令不會傳回回應。

```
$ aws acm-pca revoke-certificate \
     --certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566 \
     --certificate-serial serial_number \
     --revocation-reason "KEY_COMPROMISE"
```

撤銷的憑證和 OCSP

當您撤銷憑證時，OCSP 回應最多可能需要 60 分鐘才能反映新狀態。一般而言，OCSP 傾向於支援更快的撤銷資訊分佈，因為與用戶端可以快取幾天的 CRLs 不同，用戶端通常不會快取 OCSP 回應。

CRL 中的已撤銷憑證

CRL 通常在憑證撤銷後約 30 分鐘更新。如果 CRL 更新因任何原因失敗， AWS 私有 CA 會每 15 分鐘進一步嘗試一次。

使用 Amazon CloudWatch，您可以為 CRLGenerated 和 MisconfiguredCRLBucket 指標建立警示。如需詳細資訊，請參閱支援的 CloudWatch 指標。如需建立及設定 CRL 的詳細資訊，請參閱 設定的 CRL AWS 私有 CA。

以下範例會示範憑證撤銷清單 (CRL) 中的已撤銷憑證。

```
Certificate Revocation List (CRL):
        Version 2 (0x1)
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: /C=US/ST=WA/L=Seattle/O=Examples LLC/OU=Corporate Office/
CN=www.example.com
        Last Update: Jan 10 19:28:47 2018 GMT
        Next Update: Jan  8 20:28:47 2028 GMT
        CRL extensions:
            X509v3 Authority key identifier:
                keyid:3B:F0:04:6B:51:54:1F:C9:AE:4A:C0:2F:11:E6:13:85:D8:84:74:67

            X509v3 CRL Number:
                1515616127629
Revoked Certificates:
    Serial Number: B17B6F9AE9309C51D5573BCA78764C23
        Revocation Date: Jan  9 17:19:17 2018 GMT
        CRL entry extensions:
            X509v3 CRL Reason Code:
                Key Compromise
    Signature Algorithm: sha256WithRSAEncryption
         21:2f:86:46:6e:0a:9c:0d:85:f6:b6:b6:db:50:ce:32:d4:76:
         99:3e:df:ec:6f:c7:3b:7e:a3:6b:66:a7:b2:83:e8:3b:53:42:
         f0:7a:bc:ba:0f:81:4d:9b:71:ee:14:c3:db:ad:a0:91:c4:9f:
         98:f1:4a:69:9a:3f:e3:61:36:cf:93:0a:1b:7d:f7:8d:53:1f:
         2e:f8:bd:3c:7d:72:91:4c:36:38:06:bf:f9:c7:d1:47:6e:8e:
         54:eb:87:02:33:14:10:7f:b2:81:65:a1:62:f5:fb:e1:79:d5:
         1d:4c:0e:95:0d:84:31:f8:5d:59:5d:f9:2b:6f:e4:e6:60:8b:
         58:7d:b2:a9:70:fd:72:4f:e7:5b:e4:06:fc:e7:23:e7:08:28:
         f7:06:09:2a:a1:73:31:ec:1c:32:f8:dc:03:ea:33:a8:8e:d9:
         d4:78:c1:90:4c:08:ca:ba:ec:55:c3:00:f4:2e:03:b2:dd:8a:
         43:13:fd:c8:31:c9:cd:8d:b3:5e:06:c6:cc:15:41:12:5d:51:
         a2:84:61:16:a0:cf:f5:38:10:da:a5:3b:69:7f:9c:b0:aa:29:
         5f:fc:42:68:b8:fb:88:19:af:d9:ef:76:19:db:24:1f:eb:87:
         65:b2:05:44:86:21:e0:b4:11:5c:db:f6:a2:f9:7c:a6:16:85:
         0e:81:b2:76
```

稽核報告中的已撤銷憑證

所有憑證 (包括撤銷的憑證) 皆包含在私有 CA 的稽核報告中。以下範例會示範具有一個已發行憑證和一個已撤銷憑證的稽核報告。如需詳細資訊，請參閱將稽核報告與私有 CA 搭配使用。

```
[
   {
      "awsAccountId":"account",
      "certificateArn":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID",
      "serial":"serial_number",

 "Subject":"1.2.840.113549.1.9.1=#161173616c6573406578616d706c652e636f6d,CN=www.example1.com,OU
 Company,L=Seattle,ST=Washington,C=US",
      "notBefore":"2018-02-26T18:39:57+0000",
      "notAfter":"2019-02-26T19:39:57+0000",
      "issuedAt":"2018-02-26T19:39:58+0000",
      "revokedAt":"2018-02-26T20:00:36+0000",
      "revocationReason":"KEY_COMPROMISE"
   },
   {
      "awsAccountId":"account",
      "certificateArn":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID",
      "serial":"serial_number",

 "Subject":"1.2.840.113549.1.9.1=#161970726f64407777772e70616c6f75736573616c65732e636f6d,CN=www
 Company,L=Seattle,ST=Washington,C=US",
      "notBefore":"2018-01-22T20:10:49+0000",
      "notAfter":"2019-01-17T21:10:49+0000",
      "issuedAt":"2018-01-22T21:10:49+0000"
   }
]
```


自動化匯出續約的憑證

當您使用 AWS 私有 CA 建立 CA 時，您可以將該 CA 匯入 AWS Certificate Manager ，並讓 ACM 管理憑證發行和續約。如果要續約的憑證與整合的服務相關聯，則該服務會無縫套用新的憑證。不過，如果憑證最初匯出https://docs.aws.amazon.com/acm/latest/userguide/export-private.html以用於 PKI 環境的其他位置（例如，在內部部署伺服器或設備中），則需要在續約後再次匯出。

如需使用 Amazon EventBridge 和 AWS Lambda 自動化 ACM 匯出程序的範例解決方案，請參閱自動化匯出更新的憑證。

使用 AWS 私有 CA 憑證範本

AWS 私有 CA 使用組態範本來同時發行 CA 憑證和終端實體憑證。當您從 PCA 主控台發出 CA 憑證時，會自動套用適當的根憑證或次級 CA 憑證範本。

如果您使用 CLI 或 API 發行憑證，您可以提供範本 ARN 做為 `IssueCertificate`動作的參數。如果您未提供 ARN，則預設會套用`EndEntityCertificate/V1`範本。如需詳細資訊，請參閱 IssueCertificate API 和 issue-certificate 命令文件。

> **Note**
>
> AWS Certificate Manager 具有私有 CA 跨帳戶共用存取權的 (ACM) 使用者可以發行由 CA 簽署的受管憑證。跨帳戶發行者受到資源型政策的限制，只能存取下列終端實體憑證範本：
>
> - EndEntityCertificate/V1
> - EndEntityClientAuthCertificate/V1
> - EndEntityServerAuthCertificate/V1
> - BlankEndEntityCertificate_APIPassthrough/V1
> - BlankEndEntityCertificate_APICSRPassthrough/V1
> - SubordinateCACertificate_PathLen0/V1
>
> 如需詳細資訊，請參閱資源型政策。

主題

- AWS 私有 CA 範本變體
- AWS 私有 CA 範本操作順序
- AWS 私有 CA 範本定義

AWS 私有 CA 範本變體

AWS 私有 CA 支援四種類型的範本。

- 基本範本

  不允許傳遞參數的預先定義範本。

- CSRPassthrough 範本

  透過允許 CSR 傳遞來擴展其對應基礎範本版本的範本。用於發行憑證的 CSR 中的延伸項目會複製到發行的憑證。如果 CSR 包含與範本定義衝突的延伸值，則範本定義一律具有較高的優先順序。如需優先順序的詳細資訊，請參閱 AWS 私有 CA 範本操作順序。

- APIPassthrough 範本

  允許 API 傳遞來擴展其對應基礎範本版本的範本。請求憑證的實體可能無法得知管理員或其他中繼系統已知的動態值，可能無法在範本中定義，也可能無法在 CSR 中使用。不過，CA 管理員可以從其他資料來源擷取其他資訊，例如 Active Directory，以完成請求。例如，如果機器不知道屬於哪個組織單位，管理員可以在 Active Directory 中查詢資訊，並在 JSON 結構中包含資訊，將其新增至憑證請求。

  `IssueCertificate` 動作 `ApiPassthrough` 參數中的值會複製到發行的憑證。如果 `ApiPassthrough` 參數包含與範本定義衝突的資訊，則範本定義一律具有較高的優先順序。如需優先順序的詳細資訊，請參閱 AWS 私有 CA 範本操作順序。

- APICSRPassthrough 範本

  透過允許 API 和 CSR 傳遞來擴展其對應基礎範本版本的範本。用於發行憑證的 CSR 中的延伸項目會複製到發行的憑證，而 `IssueCertificate`動作 `ApiPassthrough` 參數中的值也會透過 複製。如果範本定義、API 傳遞值和 CSR 傳遞延伸出現衝突，則範本定義具有最高優先順序，後面接著 API 傳遞值，後面接著 CSR 傳遞延伸。如需優先順序的詳細資訊，請參閱 AWS 私有 CA 範本操作順序。

下表列出 支援的所有範本類型 AWS 私有 CA ，其中包含其定義的連結。

> **Note**
>
> 如需 GovCloud 區域中範本 ARNs 的相關資訊，請參閱AWS GovCloud (US) 《 使用者指南AWS 私有憑證授權單位》中的 。

基本範本

範本名稱	範本 ARN	憑證類型
CodeSigningCertificate/V1	arn:aws:acm-pca:::template/CodeSigningCertificate/V1	程式碼簽署
EndEntityCertificate/V1	arn:aws:acm-pca:::template/EndEntityCertificate/V1	終端實體
EndEntityClientAuthCertificate/V1	arn:aws:acm-pca:::template/EndEntityClientAuthCertificate/V1	終端實體
EndEntityServerAuthCertificate/V1	arn:aws:acm-pca:::template/EndEntityServerAuthCertificate/V1	終端實體
OCSPSigningCertificate/V1	arn:aws:acm-pca:::template/OCSPSigningCertificate/V1	OCSP 簽署
RootCACertificate/V1	arn:aws:acm-pca:::template/RootCACertificate/V1	CA
SubordinateCACertificate_PathLen0/V1	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen0/V1	CA
SubordinateCACertificate_PathLen1/V1	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen1/V1	CA

範本名稱	範本 ARN	憑證類型
SubordinateCACertificate_PathLen2/V1	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen2/V1	CA
SubordinateCACertificate_PathLen3/V1	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen3/V1	CA

CSRPassthrough 範本

範本名稱	範本 ARN	憑證類型
BlankEndEntityCertificate_CSRPassthrough/V1	arn:aws:acm-pca:::template/BlankEndEntityCertificate_CSRPassthrough/V1	終端實體
BlankEndEntityCertificate_CriticalBasicConstraints_CSRPassthrough/V1	arn:aws:acm-pca:::template/BlankEndEntityCertificate_CriticalBasicConstraints_CSRPassthrough/V1	終端實體
BlankSubordinateCACertificate_PathLen0_CSRPassthrough/V1	arn:aws:acm-pca:::template/BlankSubordinateCACertificate_PathLen0_CSRPassthrough/V1	CA
BlankSubordinateCACertificate_PathLen1_CSRPassthrough/V1	arn:aws:acm-pca:::template/BlankSubordinateCACertifica	CA

範本名稱	範本 ARN	憑證類型
	te_PathLen1_CSRPas sthrough/V1	
BlankSubordinateCACertifica te_PathLen2_CSRPassthrough/V1	arn:aws:acm-pca::: template/BlankSubo rdinateCACertifica te_PathLen2_CSRPas sthrough/V1	CA
BlankSubordinateCACertifica te_PathLen3_CSRPassthrough/V1	arn:aws:acm-pca::: template/BlankSubo rdinateCACertifica te_PathLen3_CSRPas sthrough/V1	CA
CodeSigningCertificate_CSRP assthrough/V1	arn:aws:acm-pca::: template/CodeSigni ngCertificate_CSRP assthrough/V1	程式碼簽署
EndEntityCertificate_CSRPas sthrough/V1	arn:aws:acm-pca::: template/EndEntity Certificate_CSRPas sthrough/V1	終端實體
EndEntityClientAuthCertific ate_CSRPassthrough/V1	arn:aws:acm-pca::: template/EndEntity ClientAuthCertific ate_CSRPassthrough/V1	終端實體
EndEntityServerAuthCertific ate_CSRPassthrough/V1	arn:aws:acm-pca::: template/EndEntity ServerAuthCertific ate_CSRPassthrough/V1	終端實體

範本名稱	範本 ARN	憑證類型
OCSPSigningCertificate_CSRPassthrough/V1	arn:aws:acm-pca:::template/OCSPSigningCertificate_CSRPassthrough/V1	OCSP 簽署
SubordinateCACertificate_PathLen0_CSRPassthrough/V1	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen0_CSRPassthrough/V1	CA
SubordinateCACertificate_PathLen1_CSRPassthrough/V1	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen1_CSRPassthrough/V1	CA
SubordinateCACertificate_PathLen2_CSRPassthrough/V1	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen2_CSRPassthrough/V1	CA
SubordinateCACertificate_PathLen3_CSRPassthrough/V1	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen3_CSRPassthrough/V1	CA

APIPassthrough 範本

範本名稱	範本 ARN	憑證類型
BlankEndEntityCertificate_APIPassthrough/V1	arn:aws:acm-pca:::template/BlankEndE	終端實體

範本名稱	範本 ARN	憑證類型
	ntityCertificate_APIPassthrough/V1	
BlankEndEntityCertificate_CriticalBasicConstraints_APIPassthrough/V1	arn:aws:acm-pca:::template/BlankEndEntityCertificate_CriticalBasicConstraints_APIPassthrough/V1	終端實體
CodeSigningCertificate_APIPassthrough/V1	arn:aws:acm-pca:::template/CodeSigningCertificate_APIPassthrough/V1	程式碼簽署
EndEntityCertificate_APIPassthrough/V1	arn:aws:acm-pca:::template/EndEntityCertificate_APIPassthrough/V1	終端實體
EndEntityClientAuthCertificate_APIPassthrough/V1	arn:aws:acm-pca:::template/EndEntityClientAuthCertificate_APIPassthrough/V1	終端實體
EndEntityServerAuthCertificate_APIPassthrough/V1	arn:aws:acm-pca:::template/EndEntityServerAuthCertificate_APIPassthrough/V1	終端實體
OCSPSigningCertificate_APIPassthrough/V1	arn:aws:acm-pca:::template/OCSPSigningCertificate_APIPassthrough/V1	OCSP 簽署

範本名稱	範本 ARN	憑證類型
RootCACertificate_APIPassthrough/V1	arn:aws:acm-pca:::template/RootCACertificate_APIPassthrough/V1	CA
BlankRootCACertificate_APIPassthrough/V1	arn:aws:acm-pca:::template/BlankRootCACertificate_APIPassthrough/V1	CA
BlankRootCACertificate_PathLen0_APIPassthrough/V1	arn:aws:acm-pca:::template/BlankRootCACertificate_PathLen0_APIPassthrough/V1	CA
BlankRootCACertificate_PathLen1_APIPassthrough/V1	arn:aws:acm-pca:::template/BlankRootCACertificate_PathLen1_APIPassthrough/V1	CA
BlankRootCACertificate_PathLen2_APIPassthrough/V1	arn:aws:acm-pca:::template/BlankRootCACertificate_PathLen2_APIPassthrough/V1	CA
BlankRootCACertificate_PathLen3_APIPassthrough/V1	arn:aws:acm-pca:::template/BlankRootCACertificate_PathLen3_APIPassthrough/V1	CA
SubordinateCACertificate_PathLen0_APIPassthrough/V1	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen0_APIPassthrough/V1	CA

範本名稱	範本 ARN	憑證類型
BlankSubordinateCACertificate_PathLen0_APIPassthrough/V1	arn:aws:acm-pca:::template/BlankSubordinateCACertificate_PathLen0_APIPassthrough/V1	CA
SubordinateCACertificate_PathLen1_APIPassthrough/V1	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen1_APIPassthrough/V1	CA
BlankSubordinateCACertificate_PathLen1_APIPassthrough/V1	arn:aws:acm-pca:::template/BlankSubordinateCACertificate_PathLen1_APIPassthrough/V1	CA
SubordinateCACertificate_PathLen2_APIPassthrough/V1	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen2_APIPassthrough/V1	CA
BlankSubordinateCACertificate_PathLen2_APIPassthrough/V1	arn:aws:acm-pca:::template/BlankSubordinateCACertificate_PathLen2_APIPassthrough/V1	CA
SubordinateCACertificate_PathLen3_APIPassthrough/V1	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen3_APIPassthrough/V1	CA

範本名稱	範本 ARN	憑證類型
BlankSubordinateCACertificate_PathLen3_APIPassthrough/V1	arn:aws:acm-pca:::template/BlankSubordinateCACertificate_PathLen3_APIPassthrough/V1	CA

APICSRPassthrough 範本

範本名稱	範本 ARN	憑證類型
BlankEndEntityCertificate_APICSRPassthrough/V1	arn:aws:acm-pca:::template/BlankEndEntityCertificate_APICSRPassthrough/V1	終端實體
BlankEndEntityCertificate_CriticalBasicConstraints_APICSRPassthrough/V1	arn:aws:acm-pca:::template/BlankEndEntityCertificate_CriticalBasicConstraints_APICSRPassthrough/V1	終端實體
CodeSigningCertificate_APICSRPassthrough/V1	arn:aws:acm-pca:::template/CodeSigningCertificate_APICSRPassthrough/V1	程式碼簽署
EndEntityCertificate_APICSRPassthrough/V1	arn:aws:acm-pca:::template/EndEntityCertificate_APICSRPassthrough/V1	終端實體
EndEntityClientAuthCertificate_APICSRPassthrough/V1	arn:aws:acm-pca:::template/EndEntity	終端實體

範本名稱	範本 ARN	憑證類型
	ClientAuthCertificate_APICSRPassthrough/V1	
EndEntityServerAuthCertificate_APICSRPassthrough/V1	arn:aws:acm-pca:::template/EndEntityServerAuthCertificate_APICSRPassthrough/V1	終端實體
OCSPSigningCertificate_APICSRPassthrough/V1	arn:aws:acm-pca:::template/OCSPSigningCertificate_APICSRPassthrough/V1	OCSP 簽署
SubordinateCACertificate_PathLen0_APICSRPassthrough/V1	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen0_APICSRPassthrough/V1	CA
BlankSubordinateCACertificate_PathLen0_APICSRPassthrough/V1	arn:aws:acm-pca:::template/BlankSubordinateCACertificate_PathLen0_APICSRPassthrough/V1	CA
SubordinateCACertificate_PathLen1_APICSRPassthrough/V1	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen1_APICSRPassthrough/V1	CA

範本名稱	範本 ARN	憑證類型
BlankSubordinateCACertificate_PathLen1_APICSRPassthrough/V1	arn:aws:acm-pca:::template/BlankSubordinateCACertificate_PathLen1_APICSRPassthrough/V1	CA
SubordinateCACertificate_PathLen2_APICSRPassthrough/PathLen3_APIPassthroughV1	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen2_APICSRPassthrough/V1	CA
BlankSubordinateCACertificate_PathLen2_APICSRPassthrough/V1	arn:aws:acm-pca:::template/BlankSubordinateCACertificate_PathLen2_APICSRPassthrough/V1	CA
SubordinateCACertificate_PathLen3_APICSRPassthrough/V1	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen3_APICSRPassthrough/V1	CA
BlankSubordinateCACertificate_PathLen3_APICSRPassthrough/V1	arn:aws:acm-pca:::template/BlankSubordinateCACertificate_PathLen3_APICSRPassthrough/V1	CA

AWS 私有 CA 範本操作順序

已發行憑證中包含的資訊可以來自四個來源：範本定義、API 傳遞、CSR 傳遞和 CA 組態。

只有在您使用 API 傳遞或 APICSR 傳遞範本時，才會遵守 API 傳遞值。只有在您使用 CSRPassthrough 或 APICSR 傳遞範本時，才遵守 CSR 傳遞。當這些資訊來源發生衝突時，通常適用一般規則：對於每個延伸值，範本定義具有最高的優先順序，後面接著 API 傳遞值，後面接著 CSR 傳遞延伸。

範例

1. EndEntityClientAuthCertificate_APIPassthrough 的範本定義定義了 ExtendedKeyUsage 延伸，其值為「TLS Web 伺服器身分驗證、TLS Web 用戶端身分驗證」。如果 ExtendedKeyUsage 在 CSR 或 `IssueCertificate` `ApiPassthrough` 參數中定義，則由於範本定義優先，所以會忽略 ExtendedKeyUsage `ApiPassthrough`的值，並且會忽略 ExtendedKeyUsage 值的 CSR 值，因為範本不是 CSR 傳遞多樣性。

   > **Note**
   >
   > 不過，範本定義會複製至 CSR 中的其他值，例如主旨和主旨別名。即使範本不是 CSR 傳遞變體，這些值仍會從 CSR 取得，因為範本定義始終具有最高優先順序。

2. EndEntityClientAuthCertificate_APICSRPassthrough 的範本定義會將主體別名 (SAN) 延伸模組定義為從 API 或 CSR 複製。如果 SAN 延伸在 CSR 中定義並在 `IssueCertificate` `ApiPassthrough` 參數中提供，則 API 傳遞值會優先，因為 API 傳遞值會優先於 CSR 傳遞值。

AWS 私有 CA 範本定義

下列各節提供受支援 AWS 私有 CA 憑證範本的組態詳細資訊。

BlankEndEntityCertificate_APIPassthrough/V1 定義

使用空白的終端實體憑證範本，您可以發行僅存在 X.509 基本限制條件的終端實體憑證。這是 AWS 私有 CA 可以發行的最簡單終端實體憑證，但可以使用 API 結構進行自訂。基本限制延伸定義憑證是否為 CA 憑證。空白的終端實體憑證範本會對基本限制強制執行 FALSE 值，以確保發行終端實體憑證，而不是 CA 憑證。

您可以使用空白傳遞範本來發行智慧卡憑證，這些憑證需要金鑰用量 (KU) 和擴充金鑰用量 (EKU) 的特定值。例如，擴充金鑰用量可能需要用戶端身分驗證和智慧卡登入，而金鑰用量可能需要數位簽章、非複寫和金鑰加密。與其他傳遞範本不同，空白終端實體憑證範本允許 KU 和 EKU 延伸模組的組態，其中 KU 可以是九個支援值中的任何一個 (digitalSignature、nonRepudiation、keyEncipherment、dataEncipherment、keyAgreement、keyCertSign、

和 decipherOnly)，而 EKU 可以是任何支援的值 (serverAuth、clientAuth、codesigning、emailProtection、timestamping 和 OCSPSigning) 以及自訂延伸。

BlankEndEntityCertificate_APIPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 API 或 CSR 傳遞】
主旨	【從 API 或 CSR 傳遞】
基本限制條件	CA：FALSE
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】
CRL 分佈點*	【從 CA 組態傳遞】

* 只有在 CA 設定為啟用 CRL 產生時，CRL 分佈點才會包含在範本中。

BlankEndEntityCertificate_APICSRPassthrough/V1 定義

如需空白範本的一般資訊，請參閱 BlankEndEntityCertificate_APIPassthrough/V1 定義。

BlankEndEntityCertificate_APICSRPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 API 或 CSR 傳遞】
主旨	【從 API 或 CSR 傳遞】
基本限制條件	CA：FALSE
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】
CRL 分佈點*	【從 CA 組態或 CSR 傳遞】

* 只有在 CA 設定為啟用 CRL 產生時，才會在範本中包含 CRL 分佈點。

BlankEndEntityCertificate_CriticalBasicConstraints_APICSRPassthrough/V1 定義

如需空白範本的一般資訊，請參閱 BlankEndEntityCertificate_APIPassthrough/V1 定義。

BlankEndEntityCertificate_CriticalBasicConstraints_APICSRPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 API 或 CSR 傳遞】
主旨	【從 API 或 CSR 傳遞】
基本限制條件	Critical， CA：FALSE
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】
CRL 分佈點*	【從 CA 組態、API 或 CSR 傳遞】

* 只有在 CA 設定為啟用 CRL 產生時，才會在範本中包含 CRL 分佈點。

BlankEndEntityCertificate_CriticalBasicConstraints_APIPassthrough/V1 定義

如需空白範本的一般資訊，請參閱 BlankEndEntityCertificate_APIPassthrough/V1 定義。

BlankEndEntityCertificate_CriticalBasicConstraints_APIPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 API 或 CSR 傳遞】
主旨	【從 API 或 CSR 傳遞】
基本限制條件	Critical， CA：FALSE
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】
CRL 分佈點*	【從 CA 組態或 API 傳遞】

* 只有在 CA 設定為啟用 CRL 產生時，才會在範本中包含 CRL 分佈點。

BlankEndEntityCertificate_CriticalBasicConstraints_CSRPassthrough/V1 定義

如需空白範本的一般資訊，請參閱 BlankEndEntityCertificate_APIPassthrough/V1 定義。

BlankEndEntityCertificate_CriticalBasicConstraints_CSRPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 API 或 CSR 傳遞】
主旨	【從 API 或 CSR 傳遞】
基本限制條件	Critical， CA：FALSE
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】
CRL 分佈點*	【從 CA 組態或 CSR 傳遞】

* 只有在 CA 設定為啟用 CRL 產生時，才會在範本中包含 CRL 分佈點。

BlankEndEntityCertificate_CSRPassthrough/V1 定義

如需空白範本的一般資訊，請參閱 BlankEndEntityCertificate_APIPassthrough/V1 定義。

BlankEndEntityCertificate_CSRPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 CSR 傳遞】
主旨	【從 CSR 傳遞】
基本限制條件	CA：FALSE
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】
CRL 分佈點*	【從 CA 組態或 CSR 傳遞】

* 只有在 CA 設定為啟用 CRL 產生時，才會在範本中包含 CRL 分佈點。

BlankSubordinateCACertificate_PathLen0_CSRPassthrough/V1 定義

如需空白範本的一般資訊，請參閱 BlankEndEntityCertificate_APIPassthrough/V1 定義。

BlankSubordinateCACertificate_PathLen0_CSRPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 CSR 傳遞】
主旨	【從 CSR 傳遞】
基本限制條件	關鍵、CA:TRUE、pathlen: 0
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】
CRL 分佈點*	【從 CA 組態或 CSR 傳遞】

* 只有在 CA 設定為啟用 CRL 產生時，才會在範本中包含 CRL 分佈點。

BlankSubordinateCACertificate_PathLen0_APICSRPassthrough/V1 定義

如需空白範本的一般資訊，請參閱 BlankEndEntityCertificate_APIPassthrough/V1 定義。

BlankSubordinateCACertificate_PathLen0_APICSRPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 API 或 CSR 傳遞】
主旨	【從 API 或 CSR 傳遞】
基本限制條件	關鍵、CA:TRUE、pathlen: 0
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】

X509v3 參數	Value
CRL 分佈點*	【從 CA 組態或 CSR 傳遞】

* 只有在 CA 設定為啟用 CRL 產生時，才會在範本中包含 CRL 分佈點。

BlankSubordinateCACertificate_PathLen0_APIPassthrough/V1 定義

如需空白範本的一般資訊，請參閱 BlankEndEntityCertificate_APIPassthrough/V1 定義。

BlankSubordinateCACertificate_PathLen0_APIPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 API 或 CSR 傳遞】
主旨	【從 API 或 CSR 傳遞】
基本限制條件	關鍵、CA:TRUE、pathlen: 0
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】
CRL 分佈點*	【從 CA 組態傳遞】

BlankSubordinateCACertificate_PathLen1_APIPassthrough/V1 定義

如需空白範本的一般資訊，請參閱 BlankEndEntityCertificate_APIPassthrough/V1 定義。

BlankSubordinateCACertificate_PathLen1_APIPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 API 或 CSR 傳遞】
主旨	【從 API 或 CSR 傳遞】
基本限制條件	關鍵、CA:TRUE、pathlen: 1
授權金鑰識別符	【來自 CA 憑證的 SKI】

X509v3 參數	Value
主旨金鑰識別符	【衍生自 CSR】
CRL 分佈點*	【從 CA 組態傳遞】

* 只有在 CA 設定為啟用 CRL 產生時，才會在範本中包含 CRL 分佈點。

BlankSubordinateCACertificate_PathLen1_CSRPassthrough/V1 定義

如需空白範本的一般資訊，請參閱 BlankEndEntityCertificate_APIPassthrough/V1 定義。

BlankSubordinateCACertificate_PathLen1_CSRPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 CSR 傳遞】
主旨	【從 CSR 傳遞】
基本限制條件	關鍵、CA:TRUE、pathlen: 1
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】
CRL 分佈點*	【從 CA 組態或 CSR 傳遞】

* 只有在 CA 設定為啟用 CRL 產生時，才會在範本中包含 CRL 分佈點。

BlankSubordinateCACertificate_PathLen1_APICSRPassthrough/V1 定義

如需空白範本的一般資訊，請參閱 BlankEndEntityCertificate_APIPassthrough/V1 定義。

BlankSubordinateCACertificate_PathLen1_APICSRPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 API 或 CSR 傳遞】
主旨	【從 API 或 CSR 傳遞】

X509v3 參數	Value
基本限制條件	關鍵、CA:TRUE、pathlen: 1
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】
CRL 分佈點*	【從 CA 組態或 CSR 傳遞】

* 只有在 CA 設定為啟用 CRL 產生時，才會在範本中包含 CRL 分佈點。

BlankSubordinateCACertificate_PathLen2_APIPassthrough/V1 定義

如需空白範本的一般資訊，請參閱 BlankEndEntityCertificate_APIPassthrough/V1 定義。

BlankSubordinateCACertificate_PathLen2_APIPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 API 或 CSR 傳遞】
主旨	【從 API 或 CSR 傳遞】
基本限制條件	關鍵、CA:TRUE、pathlen: 2
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】
CRL 分佈點*	【從 CA 組態傳遞】

* 只有在 CA 設定為啟用 CRL 產生時，才會在範本中包含 CRL 分佈點。

BlankSubordinateCACertificate_PathLen2_CSRPassthrough/V1 定義

如需空白範本的一般資訊，請參閱 BlankEndEntityCertificate_APIPassthrough/V1 定義。

BlankSubordinateCACertificate_PathLen2_CSRPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 CSR 傳遞】
主旨	【從 CSR 傳遞】
基本限制條件	關鍵、CA:TRUE、pathlen: 2
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】
CRL 分佈點*	【從 CA 組態或 CSR 傳遞】

* 只有在 CA 設定為啟用 CRL 產生時，才會在範本中包含 CRL 分佈點。

BlankSubordinateCACertificate_PathLen2_APICSRPassthrough/V1 定義

如需空白範本的一般資訊，請參閱 BlankEndEntityCertificate_APIPassthrough/V1 定義。

BlankSubordinateCACertificate_PathLen2_APICSRPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 API 或 CSR 傳遞】
主旨	【從 API 或 CSR 傳遞】
基本限制條件	關鍵、CA:TRUE、pathlen: 2
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】
CRL 分佈點*	【從 CA 組態或 CSR 傳遞】

* 只有在 CA 設定為啟用 CRL 產生時，才會在範本中包含 CRL 分佈點。

BlankSubordinateCACertificate_PathLen3_APIPassthrough/V1 定義

如需空白範本的一般資訊，請參閱 BlankEndEntityCertificate_APIPassthrough/V1 定義。

BlankSubordinateCACertificate_PathLen3_APIPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 API 或 CSR 傳遞】
主旨	【從 API 或 CSR 傳遞】
基本限制條件	關鍵、CA:TRUE、pathlen: 3
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】
CRL 分佈點*	【從 CA 組態傳遞】

* 只有在 CA 設定為啟用 CRL 產生時，才會在範本中包含 CRL 分佈點。

BlankSubordinateCACertificate_PathLen3_CSRPassthrough/V1 定義

如需空白範本的一般資訊，請參閱 BlankEndEntityCertificate_APIPassthrough/V1 定義。

BlankSubordinateCACertificate_PathLen3_CSRPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 CSR 傳遞】
主旨	【從 CSR 傳遞】
基本限制條件	關鍵、CA:TRUE、pathlen: 3
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】
CRL 分佈點*	【從 CA 組態或 CSR 傳遞】

* 只有在 CA 設定為啟用 CRL 產生時，才會在範本中包含 CRL 分佈點。

BlankSubordinateCACertificate_PathLen3_APICSRPassthrough/V1 定義

如需空白範本的一般資訊，請參閱 BlankEndEntityCertificate_APIPassthrough/V1 定義。

BlankSubordinateCACertificate_PathLen3_APICSRPassthrough

X509v3 參數	Value
主體替代名稱	【從 API 或 CSR 傳遞】
主旨	【從 API 或 CSR 傳遞】
基本限制條件	關鍵、CA:TRUE、pathlen: 3
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】
CRL 分佈點*	【從 CA 組態或 CSR 傳遞】

* 只有在 CA 設定為啟用 CRL 產生時，才會在範本中包含 CRL 分佈點。

CodeSigningCertificate/V1 定義

此範本用來建立用於進行程式碼簽署的憑證。您可以將來自 的程式碼簽署憑證 AWS 私有 CA 與以私有 CA 基礎設施為基礎的任何程式碼簽署解決方案搭配使用。例如，使用 Code Signing for 的客戶 AWS IoT 可以使用 產生程式碼簽署憑證， AWS 私有 CA 並將其匯入 AWS Certificate Manager。如需詳細資訊，請參閱什麼是程式碼簽署 AWS IoT？以及取得和匯入程式碼簽署憑證。

CodeSigningCertificate/V1

X509v3 參數	Value
主體替代名稱	【從 CSR 傳遞】
主旨	【從 CSR 傳遞】
基本限制條件	CA:FALSE
授權金鑰識別符	【來自 CA 憑證的 SKI】

X509v3 參數	Value
主旨金鑰識別符	【衍生自 CSR】
金鑰用途	關鍵數位簽章
擴充金鑰用量	關鍵、程式碼簽署
CRL 分佈點*	【從 CA 組態傳遞】

* 只有在 CA 已設為啟用產生 CRL 時，CRL 分佈點才會包含在範本中。

CodeSigningCertificate_APICSRPassthrough/V1 定義

此範本擴展 CodeSigningCertificate/V1，以支援 API 和 CSR 傳遞值。

CodeSigningCertificate_APICSRPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 API 或 CSR 傳遞】
主旨	【從 API 或 CSR 傳遞】
基本限制條件	CA:FALSE
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】
金鑰用途	關鍵數位簽章
擴充金鑰用量	關鍵、程式碼簽署
CRL 分佈點*	【從 CA 組態或 CSR 傳遞】

* 只有在 CA 設定為啟用 CRL 產生時，才會在範本中包含 CRL 分佈點。

CodeSigningCertificate_APIPassthrough/V1 定義

此範本與具有一個差異的CodeSigningCertificate範本相同：在此範本中，如果未在範本中指定擴充功能， 會透過 API 將其他擴充功能 AWS 私有 CA 傳遞至憑證。範本中指定的延伸項目一律會覆寫 API 中的延伸項目。

CodeSigningCertificate_APIPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 API 或 CSR 傳遞】
主旨	【從 API 或 CSR 傳遞】
基本限制條件	CA:FALSE
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】
金鑰用途	關鍵數位簽章
擴充金鑰用量	關鍵、程式碼簽署
CRL 分佈點*	【從 CA 組態傳遞】

* 只有在 CA 設定為啟用 CRL 產生時，才會在範本中包含 CRL 分佈點。

CodeSigningCertificate_CSRPassthrough/V1 定義

此範本與具有一個差異的CodeSigningCertificate範本相同：在此範本中，如果未在範本中指定延伸， 會將憑證簽署請求 (CSR) 中的其他延伸 AWS 私有 CA 傳遞至憑證。範本中指定的延伸一律會覆寫 CSR 中的延伸。

CodeSigningCertificate_CSRPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 CSR 傳遞】
主旨	【從 CSR 傳遞】

X509v3 參數	Value
基本限制條件	CA:FALSE
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】
金鑰用途	關鍵數位簽章
擴充金鑰用量	關鍵、程式碼簽署
CRL 分佈點*	【從 CA 組態或 CSR 傳遞】

* 只有在 CA 已設為啟用產生 CRL 時，CRL 分佈點才會包含在範本中。

EndEntityCertificate/V1 定義

此範本用來建立終端實體 (例如作業系統或 Web 伺服器) 的憑證。

EndEntityCertificate/V1

X509v3 參數	Value
主體替代名稱	【從 CSR 傳遞】
主旨	【從 CSR 傳遞】
基本限制條件	CA：FALSE
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】
金鑰用途	關鍵、數位簽章、金鑰加密
擴充金鑰用量	TLS Web 伺服器身分驗證、TLS Web 用戶端身分驗證
CRL 分佈點*	【從 CA 組態傳遞】

* 只有在 CA 已設為啟用產生 CRL 時，CRL 分佈點才會包含在範本中。

EndEntityCertificate_APICSRPassthrough/V1 定義

此範本延伸 EndEntityCertificate/V1，以支援 API 和 CSR 傳遞值。

EndEntityCertificate_APICSRPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 API 或 CSR 傳遞】
主旨	【從 API 或 CSR 傳遞】
基本限制條件	CA：FALSE
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】
金鑰用途	關鍵、數位簽章、金鑰加密
擴充金鑰用量	TLS Web 伺服器身分驗證、TLS Web 用戶端身分驗證
CRL 分佈點*	【從 CA 組態或 CSR 傳遞】

* 只有在 CA 設定為啟用 CRL 產生時，才會在範本中包含 CRL 分佈點。

EndEntityCertificate_APIPassthrough/V1 定義

此範本與具有一個差異的`EndEntityCertificate`範本相同：在此範本中，如果未在範本中指定延伸項目，則透過 API 將其他延伸項目 AWS 私有 CA 傳遞至憑證。範本中指定的延伸項目一律會覆寫 API 中的延伸項目。

EndEntityCertificate_APIPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 API 或 CSR 傳遞】
主旨	【從 API 或 CSR 傳遞】

X509v3 參數	Value
基本限制條件	CA：FALSE
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】
金鑰用途	關鍵、數位簽章、金鑰加密
擴充金鑰用量	TLS Web 伺服器身分驗證、TLS Web 用戶端身分驗證
CRL 分佈點*	【從 CA 組態傳遞】

* 只有在 CA 設定為啟用 CRL 產生時，才會在範本中包含 CRL 分佈點。

EndEntityCertificate_CSRPassthrough/V1 定義

此範本與具有一個差異的`EndEntityCertificate`範本相同：在此範本中，如果未在範本中指定延伸，則 會將憑證簽署請求 (CSR) 中的其他延伸 AWS 私有 CA 傳遞至憑證。範本中指定的延伸一律會覆寫 CSR 中的延伸。

EndEntityCertificate_CSRPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 CSR 傳遞】
主旨	【從 CSR 傳遞】
基本限制條件	CA：FALSE
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】
金鑰用途	關鍵、數位簽章、金鑰加密
擴充金鑰用量	TLS Web 伺服器身分驗證、TLS Web 用戶端身分驗證

X509v3 參數	Value
CRL 分佈點*	【從 CA 組態或 CSR 傳遞】

* 只有在 CA 已設為啟用產生 CRL 時，CRL 分佈點才會包含在範本中。

EndEntityClientAuthCertificate/V1 定義

此範本與擴展金鑰用量值中`EndEntityCertificate`唯一的 不同，這會將其限制為 TLS Web 用戶端身分驗證。

EndEntityClientAuthCertificate/V1

X509v3 參數	Value
主體替代名稱	【從 CSR 傳遞】
主旨	【從 CSR 傳遞】
基本限制條件	CA：FALSE
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】
金鑰用途	關鍵、數位簽章、金鑰加密
擴充金鑰用量	TLS Web 用戶端身分驗證
CRL 分佈點*	【從 CA 組態或 CSR 傳遞】

* 只有在 CA 已設為啟用產生 CRL 時，CRL 分佈點才會包含在範本中。

EndEntityClientAuthCertificate_APICSRPassthrough/V1 定義

此範本延伸 EndEntityClientAuthCertificate/V1，以支援 API 和 CSR 傳遞值。

EndEntityClientAuthCertificate_APICSRPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 API 或 CSR 傳遞】
主旨	【從 API 或 CSR 傳遞】
基本限制條件	CA：FALSE
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】
金鑰用途	關鍵、數位簽章、金鑰加密
擴充金鑰用量	TLS Web 用戶端身分驗證
CRL 分佈點*	【從 CA 組態或 CSR 傳遞】

* 只有在 CA 設定為啟用 CRL 產生時，CRL 分佈點才會包含在範本中。

EndEntityClientAuthCertificate_APIPassthrough/V1 定義

此範本與 `EndEntityClientAuthCertificate` 範本之間唯一的不同點在於，在此範本中，如果未在範本中指定擴充功能， 會透過 API 將其他擴充功能 AWS 私有 CA 傳遞至憑證。範本中指定的延伸項目一律會覆寫 API 中的延伸項目。

EndEntityClientAuthCertificate_APIPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 API 或 CSR 傳遞】
主旨	【從 API 或 CSR 傳遞】
基本限制條件	CA：FALSE
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】

X509v3 參數	Value
金鑰用途	關鍵、數位簽章、金鑰加密
擴充金鑰用量	TLS Web 用戶端身分驗證
CRL 分佈點*	【從 CA 組態傳遞】

* 只有在 CA 設定為啟用 CRL 產生時，CRL 分佈點才會包含在範本中。

EndEntityClientAuthCertificate_CSRPassthrough/V1 定義

此範本與 `EndEntityClientAuthCertificate` 範本之間唯一的不同點在於，在此範本中，如果未在範本中指定擴充功能， 會從憑證簽署請求 (CSR) AWS 私有 CA 傳遞其他擴充功能到憑證。範本中指定的延伸一律會覆寫 CSR 中的延伸。

EndEntityClientAuthCertificate_CSRPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 CSR 傳遞】
主旨	【從 CSR 傳遞】
基本限制條件	CA：FALSE
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】
金鑰用途	關鍵、數位簽章、金鑰加密
擴充金鑰用量	TLS Web 用戶端身分驗證
CRL 分佈點*	【從 CA 組態或 CSR 傳遞】

* 只有在 CA 已設為啟用產生 CRL 時，CRL 分佈點才會包含在範本中。

EndEntityServerAuthCertificate/V1 定義

此範本與擴展金鑰用量值中EndEntityCertificate唯一的 不同，這會將其限制為 TLS Web 伺服器身分驗證。

EndEntityServerAuthCertificate/V1

X509v3 參數	Value
主體替代名稱	【從 CSR 傳遞】
主旨	【從 CSR 傳遞】
基本限制條件	CA：FALSE
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】
金鑰用途	關鍵、數位簽章、金鑰加密
擴充金鑰用量	TLS Web 伺服器身分驗證
CRL 分佈點*	【從 CA 組態傳遞】

* 只有在 CA 已設為啟用產生 CRL 時，CRL 分佈點才會包含在範本中。

EndEntityServerAuthCertificate_APICSRPassthrough/V1 定義

此範本延伸 EndEntityServerAuthCertificate/V1，以支援 API 和 CSR 傳遞值。

EndEntityServerAuthCertificate_APICSRPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 API 或 CSR 傳遞】
主旨	【從 API 或 CSR 傳遞】
基本限制條件	CA：FALSE
授權金鑰識別符	【來自 CA 憑證的 SKI】

X509v3 參數	Value
主旨金鑰識別符	【衍生自 CSR】
金鑰用途	關鍵、數位簽章、金鑰加密
擴充金鑰用量	TLS Web 伺服器身分驗證
CRL 分佈點*	【從 CA 組態或 CSR 傳遞】

* 只有在 CA 設定為啟用 CRL 產生時，CRL 分佈點才會包含在範本中。

EndEntityServerAuthCertificate_APIPassthrough/V1 定義

此範本與 `EndEntityServerAuthCertificate` 範本之間唯一的不同點在於，在此範本中，如果未在範本中指定擴充功能， 會透過 API 將其他擴充功能 AWS 私有 CA 傳遞至憑證。範本中指定的延伸項目一律會覆寫 API 中的延伸項目。

EndEntityServerAuthCertificate_APIPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 API 或 CSR 傳遞】
主旨	【從 API 或 CSR 傳遞】
基本限制條件	CA：FALSE
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】
金鑰用途	關鍵、數位簽章、金鑰加密
擴充金鑰用量	TLS Web 伺服器身分驗證
CRL 分佈點*	【從 CA 組態傳遞】

* 只有在 CA 設定為啟用 CRL 產生時，CRL 分佈點才會包含在範本中。

EndEntityServerAuthCertificate_CSRPassthrough/V1 定義

此範本與 EndEntityServerAuthCertificate 範本之間唯一的不同點在於，在此範本中，如果未在範本中指定擴充功能， 會從憑證簽署請求 (CSR) AWS 私有 CA 傳遞其他擴充功能到憑證。範本中指定的延伸一律會覆寫 CSR 中的延伸。

EndEntityServerAuthCertificate_CSRPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 CSR 傳遞】
主旨	【從 CSR 傳遞】
基本限制條件	CA：FALSE
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】
金鑰用途	關鍵、數位簽章、金鑰加密
擴充金鑰用量	TLS Web 伺服器身分驗證
CRL 分佈點*	【從 CA 組態或 CSR 傳遞】

* 只有在 CA 已設為啟用產生 CRL 時，CRL 分佈點才會包含在範本中。

OCSPSigningCertificate/V1 定義

此範本用來建立用於簽署 OCSP 回應的憑證。範本與CodeSigningCertificate範本相同，但擴充金鑰用量值指定 OCSP 簽署而非程式碼簽署。

OCSPSigningCertificate/V1

X509v3 參數	Value
主體替代名稱	【從 CSR 傳遞】
主旨	【從 CSR 傳遞】
基本限制條件	CA:FALSE

X509v3 參數	Value
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】
金鑰用途	關鍵數位簽章
擴充金鑰用量	關鍵、OCSP 簽署
CRL 分佈點*	【從 CA 組態傳遞】

* 只有在 CA 已設為啟用產生 CRL 時，CRL 分佈點才會包含在範本中。

OCSPSigningCertificate_APICSRPassthrough/V1 定義

此範本擴展 OCSPSigningCertificate/V1，以支援 API 和 CSR 傳遞值。

OCSPSigningCertificate_APICSRPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 API 或 CSR 傳遞】
主旨	【從 API 或 CSR 傳遞】
基本限制條件	CA:FALSE
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】
金鑰用途	關鍵數位簽章
擴充金鑰用量	關鍵、OCSP 簽署
CRL 分佈點*	【從 CA 組態或 CSR 傳遞】

* 只有在 CA 設定為啟用 CRL 產生時，CRL 分佈點才會包含在範本中。

OCSPSigningCertificate_APIPassthrough/V1 定義

此範本與 `OCSPSigningCertificate` 範本之間唯一的不同點在於，在此範本中，如果未在範本中指定擴充功能， 會透過 API 將其他擴充功能 AWS 私有 CA 傳遞至憑證。範本中指定的延伸項目一律會覆寫 API 中的延伸項目。

OCSPSigningCertificate_APIPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 API 或 CSR 傳遞】
主旨	【從 API 或 CSR 傳遞】
基本限制條件	`CA:FALSE`
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】
金鑰用途	關鍵數位簽章
擴充金鑰用量	關鍵、OCSP 簽署
CRL 分佈點*	【從 CA 組態傳遞】

* 只有在 CA 設定為啟用 CRL 產生時，CRL 分佈點才會包含在範本中。

OCSPSigningCertificate_CSRPassthrough/V1 定義

此範本與 `OCSPSigningCertificate` 範本之間唯一的不同點在於，在此範本中，如果未在範本中指定擴充功能， 會從憑證簽署請求 (CSR) AWS 私有 CA 傳遞其他擴充功能到憑證。範本中指定的延伸一律會覆寫 CSR 中的延伸。

OCSPSigningCertificate_CSRPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 CSR 傳遞】
主旨	【從 CSR 傳遞】

X509v3 參數	Value
基本限制條件	CA:FALSE
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】
金鑰用途	關鍵數位簽章
擴充金鑰用量	關鍵、OCSP 簽署
CRL 分佈點*	【從 CA 組態或 CSR 傳遞】

* 只有在 CA 已設為啟用產生 CRL 時，CRL 分佈點才會包含在範本中。

RootCACertificate/V1 定義

此範本用來發行自我簽署的根 CA 憑證。CA 憑證包含關鍵的基本限制條件延伸，其中會將 CA 欄位設為 TRUE，指定憑證可以用來發行 CA 憑證。範本不會指定路徑長度 (pathLenConstraint)，因為這可能會抑制階層的未來擴展。其中已排除延伸金鑰使用方式，以防止使用 CA 憑證做為 TLS 用戶端或伺服器憑證。因為無法撤銷自我簽署憑證，所以沒有指定任何 CRL 資訊。

RootCACertificate/V1

X509v3 參數	Value
主體替代名稱	【從 CSR 傳遞】
主旨	【從 CSR 傳遞】
基本限制條件	關鍵，CA:TRUE
主旨金鑰識別符	【衍生自 CSR】
金鑰用途	關鍵、數位簽章、keyCertSign、CRL 簽章
CRL 分佈點	N/A

RootCACertificate_APIPassthrough/V1 定義

此範本延伸 RootCACertificate/V1 以支援 API 傳遞值。

RootCACertificate_APIPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 API 或 CSR 傳遞】
主旨	【從 API 或 CSR 傳遞】
基本限制條件	關鍵，CA:TRUE
授權金鑰識別符	【從 API 傳遞】
主旨金鑰識別符	【衍生自 CSR】
金鑰用途	關鍵、數位簽章、keyCertSign、CRL 簽章
CRL 分佈點*	N/A

BlankRootCACertificate_APIPassthrough/V1 定義

使用空白根憑證範本，您可以發行根憑證，其中僅存在 X.509 基本限制條件。這是 AWS 私有 CA 可以發行的最簡單根憑證，但可以使用 API 結構進行自訂。基本限制延伸定義憑證是否為 CA 憑證。空白根憑證範本TRUE會針對基本限制強制執行 的值，以確保發出根 CA 憑證。

您可以使用空白傳遞根範本來發行需要金鑰用量 (KU) 特定值的根憑證。例如，金鑰用量可能需要 keyCertSign和 cRLSign，但不需要 digitalSignature。與其他非空白根傳遞憑證範本不同，空白根憑證範本允許 KU 延伸的組態，其中 KU 可以是九個支援值 (digitalSignature、、nonRepudiationkeyEncipherment、dataEnciphermentkeyAgreement、encipherOnly和) 中的任何一個decipherOnly。

BlankRootCACertificate_APIPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 API 或 CSR 傳遞】
主旨	【從 API 或 CSR 傳遞】

X509v3 參數	Value
基本限制條件	關鍵，CA:TRUE
主旨金鑰識別符	【衍生自 CSR】

BlankRootCACertificate_PathLen0_APIPassthrough/V1 定義

如需空白根 CA 範本的一般資訊，請參閱 ???。

BlankRootCACertificate_PathLen0_APIPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 API 或 CSR 傳遞】
主旨	【從 API 或 CSR 傳遞】
基本限制條件	關鍵、CA:TRUE、pathlen: 0
主旨金鑰識別符	【衍生自 CSR】

BlankRootCACertificate_PathLen1_APIPassthrough/V1 定義

如需空白根 CA 範本的一般資訊，請參閱 ???。

BlankRootCACertificate_PathLen1_APIPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 API 或 CSR 傳遞】
主旨	【從 API 或 CSR 傳遞】
基本限制條件	關鍵、CA:TRUE、pathlen: 1
主旨金鑰識別符	【衍生自 CSR】

BlankRootCACertificate_PathLen2_APIPassthrough/V1 定義

如需空白根 CA 範本的一般資訊，請參閱 ???。

BlankRootCACertificate_PathLen2_APIPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 API 或 CSR 傳遞】
主旨	【從 API 或 CSR 傳遞】
基本限制條件	關鍵、CA:TRUE、pathlen: 2
主旨金鑰識別符	【衍生自 CSR】

BlankRootCACertificate_PathLen3_APIPassthrough/V1 定義

如需空白根 CA 範本的一般資訊，請參閱 ???。

BlankRootCACertificate_PathLen3_APIPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 API 或 CSR 傳遞】
主旨	【從 API 或 CSR 傳遞】
基本限制條件	關鍵、CA:TRUE、pathlen: 3
主旨金鑰識別符	【衍生自 CSR】

SubordinateCACertificate_PathLen0/V1 定義

此範本用於發行路徑長度為 的次級 CA 憑證0。CA 憑證包含關鍵的基本限制條件延伸，其中會將 CA 欄位設為 TRUE，指定憑證可以用來發行 CA 憑證。其中並未包含延伸金鑰使用方式，該金鑰使用方式會防止將 CA 憑證做為 TLS 用戶端或伺服器憑證使用。

如需認證路徑的詳細資訊，請參閱對認證路徑設定長度限制條件。

SubordinateCACertificate_PathLen0/V1

X509v3 參數	Value
主體替代名稱	【從 CSR 傳遞】
主旨	【從 CSR 傳遞】
基本限制條件	關鍵、CA:TRUE、pathlen: 0
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】
金鑰用途	關鍵、數位簽章、keyCertSign 、CRL 符號
CRL 分佈點*	【從 CA 組態傳遞】

*只有在 CA 設定為啟用 CRL 產生時，CRL 分佈點才會包含在使用此範本發行的憑證中。

SubordinateCACertificate_PathLen0_APICSRPassthrough/V1 定義

此範本延伸 SubordinateCACertificate_PathLen0/V1，以支援 API 和 CSR 傳遞值。

SubordinateCACertificate_PathLen0_APICSRPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 API 或 CSR 傳遞】
主旨	【從 API 或 CSR 傳遞】
基本限制條件	關鍵、CA:TRUE、pathlen: 0
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】
金鑰用途	關鍵、數位簽章、keyCertSign 、CRL 符號
CRL 分佈點*	【從 CA 組態或 CSR 傳遞】

* 只有在 CA 設定為啟用 CRL 產生時，CRL 分佈點才會包含在範本中。

SubordinateCACertificate_PathLen0_APIPassthrough/V1 定義

此範本延伸 SubordinateCACertificate_PathLen0/V1，以支援 API 傳遞值。

SubordinateCACertificate_PathLen0_APIPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 API 或 CSR 傳遞】
主旨	【從 API 或 CSR 傳遞】
基本限制條件	關鍵、CA:TRUE、pathlen: 0
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】
金鑰用途	關鍵、數位簽章、keyCertSign 、CRL 符號
CRL 分佈點*	【從 CA 組態傳遞】

* 只有在 CA 設定為啟用 CRL 產生時，才會在範本中包含 CRL 分佈點。

SubordinateCACertificate_PathLen0_CSRPassthrough/V1 定義

此範本與具有一個差異的SubordinateCACertificate_PathLen0範本相同：在此範本中，如果未在範本中指定延伸，則 會將憑證簽署請求 (CSR) 中的其他延伸 AWS 私有 CA 傳遞至憑證。範本中指定的延伸一律會覆寫 CSR 中的延伸。

> **Note**
>
> 包含自訂額外延伸項目的 CSR 必須在 外部建立 AWS 私有 CA。

SubordinateCACertificate_PathLen0_CSRPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 CSR 傳遞】

X509v3 參數	Value
主旨	【從 CSR 傳遞】
基本限制條件	關鍵、CA:TRUE、pathlen: 0
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】
金鑰用途	關鍵、數位簽章、keyCertSign 、CRL 符號
CRL 分佈點*	【從 CA 組態或 CSR 傳遞】

* 只有在 CA 已設為啟用產生 CRL 時，CRL 分佈點才會包含在使用此範本發行的憑證中。

SubordinateCACertificate_PathLen1/V1 定義

此範本用於發行路徑長度為 的次級 CA 憑證1。CA 憑證包含 CA 欄位設定為 的關鍵基本限制延伸TRUE，以指定憑證可用於發行 CA 憑證。其中並未包含延伸金鑰使用方式，該金鑰使用方式會防止將 CA 憑證做為 TLS 用戶端或伺服器憑證使用。

如需認證路徑的詳細資訊，請參閱對認證路徑設定長度限制條件。

SubordinateCACertificate_PathLen1/V1

X509v3 參數	Value
主體替代名稱	【從 CSR 傳遞】
主旨	【從 CSR 傳遞】
基本限制條件	關鍵、CA:TRUE、pathlen: 1
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】
金鑰用途	關鍵、數位簽章、keyCertSign 、CRL 符號
CRL 分佈點*	【從 CA 組態傳遞】

* 只有在 CA 已設為啟用產生 CRL 時，CRL 分佈點才會包含在使用此範本發行的憑證中。

SubordinateCACertificate_PathLen1_APICSRPassthrough/V1 定義

此範本延伸 SubordinateCACertificate_PathLen1/V1，以支援 API 和 CSR 傳遞值。

SubordinateCACertificate_PathLen1_APICSRPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 API 或 CSR 傳遞】
主旨	【從 API 或 CSR 傳遞】
基本限制條件	關鍵、CA:TRUE、pathlen: 1
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】
金鑰用途	關鍵、數位簽章、keyCertSign 、CRL 符號
CRL 分佈點*	【從 CA 組態或 CSR 傳遞】

* 只有在 CA 設定為啟用 CRL 產生時，才會在範本中包含 CRL 分佈點。

SubordinateCACertificate_PathLen1_APIPassthrough/V1 定義

此範本延伸 SubordinateCACertificate_PathLen0/V1 以支援 API 傳遞值。

SubordinateCACertificate_PathLen1_APIPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 API 或 CSR 傳遞】
主旨	【從 API 或 CSR 傳遞】
基本限制條件	關鍵、CA:TRUE、pathlen: 1
授權金鑰識別符	【來自 CA 憑證的 SKI】

X509v3 參數	Value
主旨金鑰識別符	【衍生自 CSR】
金鑰用途	關鍵、數位簽章、keyCertSign 、CRL 符號
CRL 分佈點*	【從 CA 組態傳遞】

* 只有在 CA 設定為啟用 CRL 產生時，才會在範本中包含 CRL 分佈點。

SubordinateCACertificate_PathLen1_CSRPassthrough/V1 定義

此範本與具有一個差異的SubordinateCACertificate_PathLen1範本相同：在此範本中，如果未在範本中指定延伸， 會將憑證簽署請求 (CSR) 中的其他延伸 AWS 私有 CA 傳遞至憑證。範本中指定的延伸一律會覆寫 CSR 中的延伸。

> **Note**
>
> 包含自訂額外延伸項目的 CSR 必須在 外部建立 AWS 私有 CA。

SubordinateCACertificate_PathLen1_CSRPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 CSR 傳遞】
主旨	【從 CSR 傳遞】
基本限制條件	關鍵、CA:TRUE、pathlen: 1
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】
金鑰用途	關鍵、數位簽章、keyCertSign 、CRL 符號
CRL 分佈點*	【從 CA 組態或 CSR 傳遞】

* 只有在 CA 已設為啟用產生 CRL 時，CRL 分佈點才會包含在使用此範本發行的憑證中。

SubordinateCACertificate_PathLen2/V1 定義

此範本用來發行路徑長度為 2 的次級 CA 憑證。CA 憑證包含 CA 欄位設定為 的關鍵基本限制延伸TRUE，以指定憑證可用於發行 CA 憑證。其中並未包含延伸金鑰使用方式，該金鑰使用方式會防止將 CA 憑證做為 TLS 用戶端或伺服器憑證使用。

如需認證路徑的詳細資訊，請參閱對認證路徑設定長度限制條件。

SubordinateCACertificate_PathLen2/V1

X509v3 參數	Value
主體替代名稱	【從 CSR 傳遞】
主旨	【從 CSR 傳遞】
基本限制條件	關鍵、CA:TRUE、pathlen: 2
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】
金鑰用途	關鍵、數位簽章、keyCertSign 、CRL 符號
CRL 分佈點*	【從 CA 組態傳遞】

* 只有在 CA 已設為啟用產生 CRL 時，CRL 分佈點才會包含在使用此範本發行的憑證中。

SubordinateCACertificate_PathLen2_APICSRPassthrough/V1 定義

此範本延伸 SubordinateCACertificate_PathLen2/V1，以支援 API 和 CSR 傳遞值。

SubordinateCACertificate_PathLen2_APICSRPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 API 或 CSR 傳遞】
主旨	【從 API 或 CSR 傳遞】
基本限制條件	關鍵、CA:TRUE、pathlen: 2

X509v3 參數	Value
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】
金鑰用途	關鍵、數位簽章、keyCertSign 、CRL 符號
CRL 分佈點*	【從 CA 組態或 CSR 傳遞】

* 只有在 CA 設定為啟用 CRL 產生時，才會在範本中包含 CRL 分佈點。

SubordinateCACertificate_PathLen2_APIPassthrough/V1 定義

此範本延伸 SubordinateCACertificate_PathLen2/V1，以支援 API 傳遞值。

SubordinateCACertificate_PathLen2_APIPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 API 或 CSR 傳遞】
主旨	【從 API 或 CSR 傳遞】
基本限制條件	關鍵、CA:TRUE、pathlen: 2
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】
金鑰用途	關鍵、數位簽章、keyCertSign 、CRL 符號
CRL 分佈點*	【從 CA 組態傳遞】

* 只有在 CA 設定為啟用 CRL 產生時，才會在範本中包含 CRL 分佈點。

SubordinateCACertificate_PathLen2_CSRPassthrough/V1 定義

此範本與具有一個差異的SubordinateCACertificate_PathLen2範本相同：在此範本中，如果未在範本中指定延伸，則 會將憑證簽署請求 (CSR) 中的其他延伸 AWS 私有 CA 傳遞至憑證。範本中指定的延伸一律會覆寫 CSR 中的延伸。

> **Note**
>
> 包含自訂額外延伸項目的 CSR 必須在 外部建立 AWS 私有 CA。

SubordinateCACertificate_PathLen2_CSRPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 CSR 傳遞】
主旨	【從 CSR 傳遞】
基本限制條件	關鍵、CA:TRUE、pathlen: 2
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】
金鑰用途	關鍵、數位簽章、keyCertSign 、CRL 符號
CRL 分佈點*	【從 CA 組態或 CSR 傳遞】

* 只有在 CA 已設為啟用產生 CRL 時，CRL 分佈點才會包含在使用此範本發行的憑證中。

SubordinateCACertificate_PathLen3/V1 定義

此範本用來發行路徑長度為 3 的次級 CA 憑證。CA 憑證包含 CA 欄位設定為 的關鍵基本限制延伸，TRUE以指定憑證可用於發行 CA 憑證。其中並未包含延伸金鑰使用方式，該金鑰使用方式會防止將 CA 憑證做為 TLS 用戶端或伺服器憑證使用。

如需認證路徑的詳細資訊，請參閱對認證路徑設定長度限制條件。

SubordinateCACertificate_PathLen3/V1

X509v3 參數	Value
主體替代名稱	【從 CSR 傳遞】
主旨	【從 CSR 傳遞】

X509v3 參數	Value
基本限制條件	關鍵、CA:TRUE、pathlen: 3
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】
金鑰用途	關鍵、數位簽章、keyCertSign 、CRL 符號
CRL 分佈點*	【從 CA 組態傳遞】

* 只有在 CA 已設為啟用產生 CRL 時，CRL 分佈點才會包含在使用此範本發行的憑證中。

SubordinateCACertificate_PathLen3_APICSRPassthrough/V1 定義

此範本延伸 SubordinateCACertificate_PathLen3/V1，以支援 API 和 CSR 傳遞值。

SubordinateCACertificate_PathLen3_APICSRPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 API 或 CSR 傳遞】
主旨	【從 API 或 CSR 傳遞】
基本限制條件	關鍵、CA:TRUE、pathlen: 3
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】
金鑰用途	關鍵、數位簽章、keyCertSign 、CRL 符號
CRL 分佈點*	【從 CA 組態或 CSR 傳遞】

* 只有在 CA 設定為啟用 CRL 產生時，才會在範本中包含 CRL 分佈點。

SubordinateCACertificate_PathLen3_APIPassthrough/V1 定義

此範本延伸 SubordinateCACertificate_PathLen3/V1，以支援 API 傳遞值。

SubordinateCACertificate_PathLen3_APIPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 API 或 CSR 傳遞】
主旨	【從 API 或 CSR 傳遞】
基本限制條件	關鍵、CA:TRUE、pathlen: 3
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】
金鑰用途	關鍵、數位簽章、keyCertSign 、CRL 符號
CRL 分佈點*	【從 CA 組態傳遞】

* 只有在 CA 設定為啟用 CRL 產生時，才會在範本中包含 CRL 分佈點。

SubordinateCACertificate_PathLen3_CSRPassthrough/V1 定義

此範本與具有一個差異的SubordinateCACertificate_PathLen3範本相同：在此範本中，如果未在範本中指定延伸， 會將憑證簽署請求 (CSR) 中的其他延伸 AWS 私有 CA 傳遞至憑證。範本中指定的延伸一律會覆寫 CSR 中的延伸。

> **Note**
>
> 包含自訂額外延伸項目的 CSR 必須在 外部建立 AWS 私有 CA。

SubordinateCACertificate_PathLen3_CSRPassthrough/V1

X509v3 參數	Value
主體替代名稱	【從 CSR 傳遞】
主旨	【從 CSR 傳遞】
基本限制條件	關鍵、CA:TRUE、pathlen: 3

X509v3 參數	Value
授權金鑰識別符	【來自 CA 憑證的 SKI】
主旨金鑰識別符	【衍生自 CSR】
金鑰用途	關鍵、數位簽章、keyCertSign 、CRL 符號
CRL 分佈點*	【從 CA 組態或 CSR 傳遞】

* 只有在 CA 已設為啟用產生 CRL 時，CRL 分佈點才會包含在使用此範本發行的憑證中。

中的安全性 AWS 私有憑證授權單位

的雲端安全 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。共同責任模式將其描述為雲端的安全性，和雲端中的安全性：

- 雲端的安全性 – AWS 負責保護在 Cloud AWS 中執行 AWS 服務的基礎設施。 AWS 也為您提供可安全使用的服務。在AWS 合規計劃中，第三方稽核人員會定期測試和驗證我們安全的有效性。若要了解適用的合規計劃 AWS 私有憑證授權單位，請參閱AWS 服務 合規計劃範圍內。
- 雲端的安全性 – 您的責任取決於您使用 AWS 的服務。您也必須對其他因素負責，包括資料的機密性、您公司的要求和適用法律和法規。

本文件可協助您了解如何在使用 時套用共同責任模型 AWS 私有 CA。下列主題說明如何設定 AWS 私有 CA 以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 AWS 私有 CA 資源。

主題

- 的 Identity and Access Management (IAM) AWS 私有憑證授權單位
- 跨帳戶存取私有 CAs的安全最佳實務
- 中的資料保護 AWS 私有憑證授權單位
- AWS 私有憑證授權單位的法規遵循驗證
- 中的基礎設施安全 AWS 私有憑證授權單位
- AWS 私有憑證授權單位 客戶 CP/CPS 架構

的 Identity and Access Management (IAM) AWS 私有憑證授權單位

存取 AWS 私有 CA 需要 AWS 登入資料，可用來驗證您的請求。以下主題提供如何使用 AWS Identity and Access Management (IAM) 的詳細資訊，藉由控制可存取的人員，協助確保私有憑證授權機構 (CA) 的安全。

在 中 AWS 私有 CA，您使用的主要資源是憑證授權單位 (CA)。您擁有或控制的每個私有 CA 皆是以 Amazon Resource Name (ARN) 識別，其格式如下。

```
arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566
```

資源擁有者是建立 AWS 資源之 AWS 帳戶的主體實體。下列範例說明其如何運作。

- 如果您使用 的登入 AWS 帳戶根使用者 資料來建立私有 CA， AWS 您的帳戶會擁有 CA。

> **Important**
>
> - 我們不建議使用 AWS 帳戶根使用者 來建立 CAs。
> - 強烈建議您隨時存取時使用多重驗證 (MFA) AWS 私有 CA。

- 如果您在 AWS 帳戶中建立 IAM 使用者，您可以授予該使用者建立私有 CA 的許可。不過，該 CA 歸使用者所屬的帳戶所有。
- 如果您在 AWS 帳戶中建立 IAM 角色，並授予其建立私有 CA 的許可，則任何可以擔任該角色的人都可以建立 CA。不過，該私有 CA 歸角色所屬的帳戶所有。

許可政策描述誰可以存取哪些資源。以下討論會說明可用來建立許可政策的選項。

> **Note**
>
> 本文件討論在 內容中使用 IAM AWS 私有 CA。它不提供 IAM 服務的詳細資訊。如需完整的 IAM 文件，請參閱 IAM 使用者指南。如需 IAM 政策語法和說明的詳細資訊，請參閱 AWS IAM 政策參考資料。

AWS 私有 CA API 操作和許可

當您設定要連接到 IAM 身分的存取控制和許可政策 （以身分為基礎的政策） 時，請使用下表做為參考。表格中的第一欄會列出每個 AWS 私有 CA API 操作。您可以在政策的 `Action` 元素中指定動作。其餘欄位提供其他資訊。

AWS 私有 CA API 操作	所需的許可	Resources
CreateCertificateAuthority	`acm-pca:CreateCertificateAuthority`	`arn:aws:acm-pca: us-east-1 :111122223333 :certificate-`

AWS 私有 CA API 操作	所需的許可	Resources
	acm-pca:TagCertificateAuthority （只有在建立具有標籤的 CA 時才需要。）	authority/ *11223344-1234-1122-2233-112233445566*
CreateCertificateAuthorityAuditReport	acm-pca:CreateCertificateAuthorityAuditReport	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*
CreatePermission	acm-pca:CreatePermission	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*
DeleteCertificateAuthority	acm-pca:DeleteCertificateAuthority	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*
DeletePermission	acm-pca:DeletePermission	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*

AWS 私有 CA API 操作	所需的許可	Resources
DeletePolicy	acm-pca:DeletePolicy	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*
DescribeCertificateAuthority	acm-pca:DescribeCertificateAuthority	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*
DescribeCertificateAuthorityAuditReport	acm-pca:DescribeCertificateAuthorityAuditReport	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*
GetCertificate	acm-pca:GetCertificate	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*
GetCertificateAuthorityCertificate	acm-pca:GetCertificateAuthorityCertificate	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*

AWS 私有 CA API 操作	所需的許可	Resources
GetCertificateAuthorityCsr	acm-pca:GetCertificateAuthorityCsr	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*
GetPolicy	acm-pca:GetPolicy	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*
ImportCertificateAuthorityCertificate	acm-pca:ImportCertificateAuthorityCertificate	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*
IssueCertificate	acm-pca:IssueCertificate	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*
ListCertificateAuthorities	acm-pca:ListCertificateAuthorities	N/A

AWS 私有 CA API 操作	所需的許可	Resources
ListPermissions	acm-pca:ListPermissions	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*
ListTags	acm-pca:ListTags	N/A
PutPolicy	acm-pca:PutPolicy	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*
RevokeCertificate	acm-pca:RevokeCertificate	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*
TagCertificateAuthority	acm-pca:TagCertificateAuthority	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*

AWS 私有 CA API 操作	所需的許可	Resources
UntagCertificateAuthority	acm-pca:UntagCertificateAuthority	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*
UpdateCertificateAuthority	acm-pca:UpdateCertificateAuthority	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*

若要提供存取權，請新增權限至您的使用者、群組或角色：

- 中的使用者和群組 AWS IAM Identity Center：

  建立權限合集。請按照《AWS IAM Identity Center 使用者指南》中的建立權限合集說明進行操作。

- 透過身分提供者在 IAM 中管理的使用者：

  建立聯合身分的角色。遵循《IAM 使用者指南》的為第三方身分提供者 (聯合) 建立角色中的指示。

- IAM 使用者：
  - 建立您的使用者可擔任的角色。請按照《IAM 使用者指南》的為 IAM 使用者建立角色中的指示。
  - (不建議) 將政策直接附加至使用者，或將使用者新增至使用者群組。請遵循《IAM 使用者指南》的新增許可到使用者 (主控台) 中的指示。

AWS 受管政策

AWS 私有 CA 包含一組適用於 AWS 私有 CA 管理員、使用者和稽核人員的預先定義 AWS 受管政策。了解這些政策有助於您實作 客戶管理政策。

選擇下列任何政策，以查看詳細資訊和範例政策程式碼。

AWSPrivateCAFullAccess

授予不受限制的管理控制。

如需政策詳細資訊的 JSON 清單，請參閱 AWSPrivateCAFullAccess。

AWSPrivateCAReadOnly

授予僅限唯讀 API 操作的存取權。

如需政策詳細資訊的 JSON 清單，請參閱 AWSPrivateCAReadOnly。

AWSPrivateCAPrivilegedUser

授予發行和撤銷 CA 憑證的能力。這項政策沒有其他管理功能，且無法發行終端實體憑證。其許可與 User 政策互斥。

如需政策詳細資訊的 JSON 清單，請參閱 AWSPrivateCAPrivilegedUser。

AWSPrivateCAUser

授予發行和撤銷終端實體憑證的能力。這項政策沒有管理功能，且無法發行 CA 憑證。其許可與 PrivilegedUser 政策互斥。

如需政策詳細資訊的 JSON 清單，請參閱 AWSPrivateCAUser。

AWSPrivateCAAuditor

授予唯讀 API 操作的存取權，以及產生 CA 稽核報告的許可。

如需政策詳細資訊的 JSON 清單，請參閱 AWSPrivateCAAuditor。

AWSPrivateCAConnectorForKubernetesPolicy

授予 AWS 私有 CA Connector for Kubernetes 的基本許可。

如需政策詳細資訊的 JSON 清單，請參閱 AWSPrivateCAConnectorForKubernetesPolicy。

的 AWS 受管政策更新 AWS 私有 CA

在下表中，檢視自服務開始追蹤這些變更 AWS 私有 CA 以來 AWS 受管政策更新的詳細資訊。如需所有變更的自動提醒 AWS 私有 CA，請訂閱 文件歷史記錄 頁面上的 RSS 摘要。

受管政策變更

變更	描述	Date
新政策： AWS PrivateCA ConnectorForKubernetesPolicy	推出可與 AWS 私有 CA Connector for Kubernetes 搭配使用的新受管政策。	2025 年 5 月 19 日
AWS PrivateCAPrivilegedUser 和 AWS PrivateCAUser - 已更新政策	`StringLike` 將取代為 `ArnLike`，將取代`StringNotLike` 為 `ArnNotLike` 。 更新範本 arn，`arn:aws:acm-pca:::template` 將萬用字元納入 `arn:aws:acm-pca:*:*:template` 。	2025 年 1 月 22 日
新政策名稱： • `AWS PrivateCAFullAccess` • `AWS PrivateCAReadOnly` • `AWS PrivateCAPrivilegedUser` • `AWS PrivateCAAuditor` • `AWS PrivateCAUser`	政策名稱字首已從 變更為 `AWS CertificateManagerPrivateCA` `AWS PrivateCA` 。 功能保持不變。	2023 年 2 月 13 日

客戶管理政策

根據最佳實務，請勿使用 AWS 帳戶根使用者 與 互動 AWS，包括 AWS 私有 CA。而是使用 AWS Identity and Access Management (IAM) 來建立 IAM 使用者、IAM 角色或聯合身分使用者。建立管理員群組，並將自己新增至該群組。然後，以管理員身分登入。視需要將其他使用者新增至該群組。

另一個最佳實務是建立客戶受管 IAM 政策，您可以將其指派給使用者。客戶受管政策是您建立的獨立身分識別型政策，您可以將政策連接到 AWS 帳戶中的多個使用者、群組或角色。這類政策會限制使用者，使其只能執行您指定的 AWS 私有 CA 動作。

以下範例客戶受管政策會允許使用者建立 CA 稽核報告。以下僅為範例。您可以選擇任何您想要 AWS 私有 CA 的操作。如需更多範例，請參閱內嵌政策。

若要建立客戶受管政策

1. 使用 管理員的登入資料登入 AWS IAM 主控台。
2. 在主控台的導覽窗格中，選擇 Policies (政策)。
3. 選擇建立政策。
4. 請選擇 JSON 標籤。
5. 請複製以下政策，然後在編輯器中貼上。

   JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":"acm-pca:CreateCertificateAuthorityAuditReport",
         "Resource":"*"
      }
   ]
}
```

6. 選擇檢閱政策。
7. 在 Name (名稱) 輸入 `PcaListPolicy`。
8. (選用) 輸入描述。
9. 選擇建立政策。

管理員可以將政策連接至任何 IAM 使用者，以限制使用者可執行的動作 AWS 私有 CA 。如需套用許可政策的方法，請參閱《IAM 使用者指南》中的變更 IAM 使用者的許可。

內嵌政策

內嵌政策是您建立及管理的政策，且直接內嵌至單一使用者、群組或角色。下列政策範例示範如何指派執行 AWS 私有 CA 動作的許可。如需內嵌政策的一般資訊，請參閱《IAM 使用者指南》中的使用內嵌政策。您可以使用 AWS 管理主控台、 AWS Command Line Interface (AWS CLI) 或 IAM API 來建立和內嵌內嵌政策。

> **Important**
>
> 強烈建議您隨時存取時使用多重驗證 (MFA) AWS 私有 CA。

主題

- 列出私有 CAs
- 擷取私有 CA 憑證
- 匯入私有 CA 憑證
- 刪除私有 CA
- Tag-on-create：在建立時將標籤連接至 CA
- Tag-on-create：限制標記
- 使用標籤控制對 Private CA 的存取
- 對 的唯讀存取權 AWS 私有 CA
- 完整存取 AWS 私有 CA

列出私有 CAs

以下政策可讓使用者列出帳戶中的所有私有 CA。

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":"acm-pca:ListCertificateAuthorities",
         "Resource":"*"
```

```
      }
   ]
}
```

擷取私有 CA 憑證

以下政策可讓使用者擷取特定私有 CA 憑證。

JSON

```
{
   "Version":"2012-10-17",
   "Statement":{
      "Effect":"Allow",
      "Action":"acm-pca:GetCertificateAuthorityCertificate",
      "Resource":"arn:aws:acm-pca:us-east-1:123456789012:certificate-
authority/CA_ID/certificate/certificate_ID"
   }
}
```

匯入私有 CA 憑證

以下政策可讓使用者匯入私有 CA 憑證。

JSON

```
{
   "Version":"2012-10-17",
   "Statement":{
      "Effect":"Allow",
      "Action":"acm-pca:ImportCertificateAuthorityCertificate",
      "Resource":"arn:aws:acm-pca:us-east-1:123456789012:certificate-
authority/CA_ID/certificate/certificate_ID"
   }
}
```

刪除私有 CA

以下政策可讓使用者刪除特定私有 CA。

JSON

```
{
   "Version":"2012-10-17",
   "Statement":{
      "Effect":"Allow",
      "Action":"acm-pca:DeleteCertificateAuthority",
      "Resource":"arn:aws:acm-pca:us-east-1:123456789012:certificate-
authority/CA_ID/certificate/certificate_ID"   }
}
```

Tag-on-create：在建立時將標籤連接至 CA

下列政策允許使用者在 CA 建立期間套用標籤。

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Action": [
            "acm-pca:CreateCertificateAuthority",
            "acm-pca:TagCertificateAuthority"
         ],
         "Effect": "Allow",
         "Resource": "*"
      }
   ]
}
```

Tag-on-create：限制標記

下列tag-on-create政策可防止在 CA 建立期間使用鍵值對環境=Prod。允許使用其他鍵/值對進行標記。

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":"acm-pca:*",
         "Resource":"*"
      },
      {
         "Effect":"Deny",
         "Action":"acm-pca:TagCertificateAuthority",
         "Resource":"*",
         "Condition":{
            "StringEquals":{
               "aws:ResourceTag/Environment":[
                  "Prod"
               ]
            }
         }
      }
   ]
}
```

使用標籤控制對 Private CA 的存取

下列政策僅允許存取鍵/值對環境=PreProd CAs。它還需要新的 CAs包含此標籤。

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "acm-pca:*"
         ],
         "Resource":"*",
         "Condition":{
```

```
            "StringEquals":{
               "aws:ResourceTag/Environment":[
                  "PreProd"
               ]
            }
         }
      }
   ]
}
```

對 的唯讀存取權 AWS 私有 CA

以下政策可讓使用者描述及列出私有憑證授權機構，並擷取私有 CA 憑證和憑證鏈。

JSON

```
{
    "Version":"2012-10-17",
    "Statement":{
       "Effect":"Allow",
       "Action":[
          "acm-pca:DescribeCertificateAuthority",
          "acm-pca:DescribeCertificateAuthorityAuditReport",
          "acm-pca:ListCertificateAuthorities",
          "acm-pca:ListTags",
          "acm-pca:GetCertificateAuthorityCertificate",
          "acm-pca:GetCertificateAuthorityCsr",
          "acm-pca:GetCertificate"
       ],
       "Resource":"*"
    }
}
```

完整存取 AWS 私有 CA

下列政策允許使用者執行任何 AWS 私有 CA 動作。

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "acm-pca:*"
         ],
         "Resource":"*"
      }
   ]
}
```

跨帳戶存取私有 CAs的安全最佳實務

AWS 私有 CA 管理員可以與另一個主體（使用者、角色等）共用 CA AWS 帳戶。收到並接受共享時，委託人可以使用 CA 使用 AWS 私有 CA 或 AWS Certificate Manager 資源發行終端實體憑證。委託人可以使用 CA 來發行次級 CA 憑證 AWS 私有 CA。

> **Important**
>
> 與跨帳戶案例中發行之憑證相關聯的費用，會向發行憑證 AWS 的帳戶收取費用。

若要共用 CA 的存取權， AWS 私有 CA 管理員可以選擇下列其中一種方法：

- 使用 AWS Resource Access Manager (RAM) 將 CA 做為資源與另一個帳戶中的委託人共用 AWS Organizations。RAM 是跨帳戶共用 AWS 資源的標準方法。如需 RAM 的詳細資訊，請參閱AWS RAM 《使用者指南》。如需 AWS Organizations的相關資訊，請參閱《使用者指南》AWS Organizations。
- 使用 AWS 私有 CA API 或 CLI 將資源型政策連接至 CA，藉此授予另一個帳戶中的委託人存取權。如需詳細資訊，請參閱資源型政策。

本指南的 控制私有 CA 的存取區段提供在單一帳戶和跨帳戶案例中授予 CAs 存取權的工作流程。

資源型政策

以資源為基礎的政策是您建立並手動連接至資源 （在此情況下為私有 CA) 而非使用者身分或角色的許可政策。或者，您可以使用 的 AWS 受管政策，而不是建立自己的政策 AWS 私有 CA。使用 AWS RAM 套用以資源為基礎的政策， AWS 私有 CA 管理員可以直接或透過 與不同 AWS 帳戶中的使用者共用 CA 的存取權 AWS Organizations。或者， AWS 私有 CA 管理員可以使用 PCA APIsPutPolicy、GetPolicy 和 DeletePolicy，或對應的 AWS CLI 命令 put-policy、get-policy 和 delete-policy，來套用和管理以資源為基礎的政策。

如需以資源為基礎的政策的一般資訊，請參閱以身分為基礎的政策和以資源為基礎的政策，以及使用政策控制存取。

若要檢視 的 AWS 受管資源型政策清單 AWS 私有 CA，請導覽至 AWS Resource Access Manager 主控台中的受管許可程式庫，然後搜尋 CertificateAuthority。如同任何政策，在您套用之前，建議您在測試環境中套用政策，以確保其符合您的需求。

AWS Certificate Manager 具有私有 CA 跨帳戶共用存取權的 (ACM) 使用者可以發行由 CA 簽署的受管憑證。跨帳戶發行者受到資源型政策的限制，只能存取下列終端實體憑證範本：

- EndEntityCertificate/V1
- EndEntityClientAuthCertificate/V1
- EndEntityServerAuthCertificate/V1
- BlankEndEntityCertificate_APIPassthrough/V1
- BlankEndEntityCertificate_APICSRPassthrough/V1
- SubordinateCACertificate_PathLen0/V1

政策範例

本節提供適用於各種需求的範例跨帳戶政策。在所有情況下，都會使用下列命令模式來套用政策：

```
$ aws acm-pca put-policy \
   --region region \
   --resource-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566 \
   --policy file:///[path]/policyN.json
```

除了指定 CA 的 ARN 之外，管理員還提供帳戶 AWS ID 或將授予 CA 存取權的 AWS Organizations ID。下列每個政策的 JSON 會格式化為 檔案以供讀取，但也可以做為內嵌 CLI 引數提供。

> **Note**
>
> 以下顯示的 JSON 資源型政策結構必須精確遵循。客戶只能設定委託人的 ID 欄位 (AWS 帳戶號碼或 AWS 組織 ID) 和 CA ARNs。

1. 檔案： policy1.json – 與不同帳戶中的使用者共用 CA 的存取權

   將 *555555555555* 取代為共用 CA AWS 的帳戶 ID。

   針對資源 ARN，請以您自己的值取代下列項目：

   - *aws* - AWS 分割區。例如，aws、aws-cn、 aws-us-gov等。
   - *us-east-1* - 資源 AWS 可用的區域，例如 us-west-1。
   - *111122223333* - 資源擁有者 AWS 的帳戶 ID。
   - *11223344-1234-1122-2233-112233445566* - 憑證授權單位的資源 ID。

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [{
            "Sid": "ExampleStatementID",
            "Effect": "Allow",
            "Principal": {
                "AWS": "555555555555"
            },
            "Action": [
                "acm-pca:DescribeCertificateAuthority",
                "acm-pca:GetCertificate",
                "acm-pca:GetCertificateAuthorityCertificate",
                "acm-pca:ListPermissions",
                "acm-pca:ListTags"
            ],
            "Resource": "arn:aws:acm-pca:us-east-1:123456789012:certificate-
authority/CA_ID"
        },
        {
            "Sid": "ExampleStatementID2",
            "Effect": "Allow",
            "Principal": {
                "AWS": "555555555555"
```

```
            },
            "Action": [
                "acm-pca:IssueCertificate"
            ],
            "Resource": "arn:aws:acm-pca:us-east-1:123456789012:certificate-
authority/CA_ID",
            "Condition": {
                "StringEquals": {
                    "acm-pca:TemplateArn": "arn:aws:acm-pca:::template/
EndEntityCertificate/V1"
                }
            }
        }
    ]
}
```

2. 檔案： policy2.json – 透過 共用 CA 的存取權 AWS Organizations

   將 AWS Organizations *o-a1b2c3d4z5* 取代為 ID。

   針對資源 ARN，請以您自己的值取代下列項目：

   - *aws* - AWS 分割區。例如，aws、aws-cn、 aws-us-gov等。
   - *us-east-1* - 資源 AWS 可用的區域，例如 us-west-1。
   - *111122223333* - 資源擁有者 AWS 的帳戶 ID。
   - *11223344-1234-1122-2233-112233445566* - 憑證授權單位的資源 ID。

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ExampleStatementID3",
            "Effect": "Allow",
            "Principal": "*",
            "Action": "acm-pca:IssueCertificate",
            "Resource":"arn:aws:acm-pca:us-east-1:123456789012:certificate-
authority/CA_ID",
            "Condition": {
                "StringEquals": {
```

```
                    "acm-pca:TemplateArn": "arn:aws:acm-pca:::template/
EndEntityCertificate/V1",
                    "aws:PrincipalOrgID": "o-a1b2c3d4z5"
                },
                "StringNotEquals": {
                    "aws:PrincipalAccount": "111122223333"
                }
            }
        },
        {
            "Sid": "ExampleStatementID4",
            "Effect": "Allow",
            "Principal": "*",
            "Action": [
                "acm-pca:DescribeCertificateAuthority",
                "acm-pca:GetCertificate",
                "acm-pca:GetCertificateAuthorityCertificate",
                "acm-pca:ListPermissions",
                "acm-pca:ListTags"
            ],
            "Resource":"arn:aws:acm-pca:us-east-1:123456789012:certificate-
authority/CA_ID",
            "Condition": {
                "StringEquals": {
                    "aws:PrincipalOrgID": "o-a1b2c3d4z5"
                },
                "StringNotEquals": {
                    "aws:PrincipalAccount": "111122223333"
                }
            }
        }
    ]
}
```

中的資料保護 AWS 私有憑證授權單位

AWS 共同責任模型適用於 中的資料保護 AWS 私有憑證授權單位。如此模型所述， AWS 負責保護執行所有 的全域基礎設施 AWS 雲端。您負責維護在此基礎設施上託管內容的控制權。您也同時負責所使用 AWS 服務 的安全組態和管理任務。如需資料隱私權的詳細資訊，請參閱資料隱私權常見問答集。如需有關歐洲資料保護的相關資訊，請參閱AWS 安全性部落格上的AWS 共同責任模型和 GDPR 部落格文章。

基於資料保護目的，我們建議您保護 AWS 帳戶 登入資料，並使用 AWS IAM Identity Center 或 AWS Identity and Access Management (IAM) 設定個別使用者。如此一來，每個使用者都只會獲得授與完成其任務所必須的許可。我們也建議您採用下列方式保護資料：

- 每個帳戶均要使用多重要素驗證 (MFA)。
- 使用 SSL/TLS 與 AWS 資源通訊。我們需要 TLS 1.2 並建議使用 TLS 1.3。
- 使用 設定 API 和使用者活動記錄 AWS CloudTrail。如需有關使用 CloudTrail 追蹤擷取 AWS 活動的資訊，請參閱AWS CloudTrail 《 使用者指南》中的使用 CloudTrail 追蹤。
- 使用 AWS 加密解決方案，以及其中的所有預設安全控制 AWS 服務。
- 使用進階的受管安全服務 (例如 Amazon Macie)，協助探索和保護儲存在 Amazon S3 的敏感資料。
- 如果您在 AWS 透過命令列界面或 API 存取 時需要 FIPS 140-3 驗證的密碼編譯模組，請使用 FIPS 端點。如需有關 FIPS 和 FIPS 端點的更多相關資訊，請參閱聯邦資訊處理標準 (FIPS) 140-3。

我們強烈建議您絕對不要將客戶的電子郵件地址等機密或敏感資訊，放在標籤或自由格式的文字欄位中，例如名稱欄位。這包括當您使用 AWS 私有 CA 或使用主控台、API AWS CLI或其他 AWS 服務 AWS SDKs 時。您在標籤或自由格式文字欄位中輸入的任何資料都可能用於計費或診斷日誌。如果您提供外部伺服器的 URL，我們強烈建議請勿在驗證您對該伺服器請求的 URL 中包含憑證資訊。

AWS 私有 CA 私有金鑰的儲存和安全性合規

私有 CAs 的私有金鑰存放在 AWS 受管硬體安全模組 (HSMs中。HSMs 符合密碼編譯模組的 FIPS PUB 140-2 第 3 級安全要求。

AWS 私有 CA Connector for Active Directory 中的資料加密

AWS 私有 CA Connector for AD 會儲存有關連接器、範本、目錄註冊、服務主體名稱和範本群組存取控制項目的客戶組態資料。此資料會在傳輸中和靜態時加密。您可以使用 AWS 私有 CA API 中的 GetCertificate 動作來探索透過 Connector for AD 發行之憑證的相關資訊。不會存放有關發行的憑證，或有關請求憑證的用戶端或機器的資訊 AWS。

AWS 私有憑證授權單位的法規遵循驗證

在多個合規計畫 AWS 私有憑證授權單位 中，第三方稽核人員會評估 的安全與 AWS 合規。這些計劃包括 SOC、PCI、FedRAMP、HIPAA 等等。

如需特定合規計劃範圍內 AWS 的服務清單，請參閱AWS 合規計劃範圍內的服務的服務。如需一般資訊，請參閱 AWS Compliance ProgramsAssurance。

您可以使用 下載第三方稽核報告 AWS Artifact。如需詳細資訊，請參閱在 中下載報告 AWS Artifact。

您使用 時的合規責任 AWS 私有 CA 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：

- 對於需要加密其 Amazon S3 儲存貯體的組織，下列主題說明如何設定加密以容納 AWS 私有 CA 資產：
  - 加密您的稽核報告
  - 加密您的 CRL
- 安全與合規快速入門指南 — 這些部署指南討論架構考量，並提供在 上部署以安全與合規為中心之基準環境的步驟 AWS。
- HIPAA 安全與合規架構白皮書 — 此白皮書說明公司如何使用 AWS 來建立符合 HIPAA 規範的應用程式。
- 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據點。
- 《 AWS Config 開發人員指南》中的使用規則評估資源 — AWS Config 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 CSPM — AWS 此服務提供 內安全狀態的完整檢視 AWS ，協助您檢查是否符合安全產業標準和最佳實務。

將稽核報告與私有 CA 搭配使用

您可以建立稽核報告，以列出私有 CA 已發行或撤銷的所有憑證。報告會儲存在新的或輸入時指定的現有 S3 儲存貯體。

如需為您稽核報告新增加密保護的資訊，請參閱 加密您的稽核報告。

稽核報告檔案具有下列路徑和檔案名稱。Amazon S3 儲存貯體的 ARN 是 的值`amzn-s3-demo-bucket`。 `CA_ID` 是發出 CA 的唯一識別符。 `UUID`是稽核報告的唯一識別符。

```
amzn-s3-demo-bucket/audit-report/CA_ID/UUID.[json|csv]
```

每 30 分鐘可從您的儲存貯體產生及下載新的報告。以下範例顯示以 CSV 分隔的報告。

```
awsAccountId,requestedByServicePrincipal,certificateArn,serial,subject,notBefore,notAfter,issue
123456789012,,arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/
```

```
certificate_ID,00:11:22:33:44:55:66:77:88:99:aa:bb:cc:dd:ee:ff,"2.5.4.5=#012345678901,2.5.4.44=
 Company,L=Seattle,ST=Washington,C=US",2020-03-02T21:43:57+0000,2020-04-07T22:43:57+0000,2020-0
pca:::template/EndEntityCertificate/V1
123456789012,acm.amazonaws.com,arn:aws:acm-pca:region:account:certificate-
authority/CA_ID/
certificate/
certificate_ID,ff:ee:dd:cc:bb:aa:99:88:77:66:55:44:33:22:11:00,"2.5.4.5=#012345678901,2.5.4.44=
 Company,L=Seattle,ST=Washington,C=US",2020-03-02T20:53:39+0000,2020-04-07T21:53:39+0000,2020-0
pca:::template/EndEntityCertificate/V1
```

JSON 格式的報告如下列範例所示。

```
[
   {
      "awsAccountId":"123456789012",
      "certificateArn":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID",
      "serial":"00:11:22:33:44:55:66:77:88:99:aa:bb:cc:dd:ee:ff",

 "subject":"2.5.4.5=#012345678901,2.5.4.44=#0a1b3c4d,2.5.4.65=#0a1b3c4d5e6f,2.5.4.43=#0a1b3c4d5
 Company,L=Seattle,ST=Washington,C=US",
      "notBefore":"2020-02-26T18:39:57+0000",
      "notAfter":"2021-02-26T19:39:57+0000",
      "issuedAt":"2020-02-26T19:39:58+0000",
      "revokedAt":"2020-02-26T20:00:36+0000",
      "revocationReason":"UNSPECIFIED",
      "templateArn":"arn:aws:acm-pca:::template/EndEntityCertificate/V1"
   },
   {
      "awsAccountId":"123456789012",
      "requestedByServicePrincipal":"acm.amazonaws.com",
      "certificateArn":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID",
      "serial":"ff:ee:dd:cc:bb:aa:99:88:77:66:55:44:33:22:11:00",

 "subject":"2.5.4.5=#012345678901,2.5.4.44=#0a1b3c4d,2.5.4.65=#0a1b3c4d5e6f,2.5.4.43=#0a1b3c4d5
 Company,L=Seattle,ST=Washington,C=US",
      "notBefore":"2020-01-22T20:10:49+0000",
      "notAfter":"2021-01-17T21:10:49+0000",
      "issuedAt":"2020-01-22T21:10:49+0000",
      "templateArn":"arn:aws:acm-pca:::template/EndEntityCertificate/V1"
   }
]
```

> **Note**
>
> 當 AWS Certificate Manager 續約憑證時，私有 CA 稽核報告會以 填入 `requestedByServicePrincipal` 欄位`acm.amazonaws.com`。這表示 AWS Certificate Manager 服務代表客戶呼叫 AWS 私有 CA API `IssueCertificate`的動作來續約憑證。

為稽核報告準備 Amazon S3 儲存貯體

> **Important**
>
> AWS 私有 CA 不支援使用 Amazon S3 物件鎖定。如果您在儲存貯體上啟用物件鎖定， AWS 私有 CA 無法將稽核報告寫入儲存貯體。

若要存放稽核報告，您需要準備 Amazon S3 儲存貯體。如需詳細資訊，請參閱如何建立 S3 儲存貯體？

您的 S3 儲存貯體必須受到允許 AWS 私有 CA 存取和寫入您指定之 S3 儲存貯體的許可政策所保護。授權使用者和服務主體需要`Put`允許 AWS 私有 CA 將物件放入儲存貯體的許可，以及擷取物件的`Get`許可。我們建議您套用以下顯示的政策，這會限制存取私有 CA AWS 的帳戶和 ARN。或者，您可以使用 aws：SourceOrgID 條件金鑰來限制對 中特定組織的存取 AWS Organizations。如需儲存貯體政策的詳細資訊，請參閱 Amazon Simple Storage Service 的儲存貯體政策。

建立稽核報告

您可以從 主控台或 建立稽核報告 AWS CLI。

建立稽核報告 (主控台)

1. 登入 AWS 您的帳戶，並在 https：//https://console.aws.amazon.com/acm-pca/home 開啟 AWS 私有 CA 主控台。
2. 在私有憑證授權頁面上，從清單中選擇私有 CA。
3. 從 Actions (動作) 選單中，選擇 Generate audit report (產生稽核報告)。
4. 在稽核報告目的地下，對於建立新的 S3 儲存貯體？，選擇是並輸入唯一的儲存貯體名稱，或選擇否，然後從清單中選擇現有的儲存貯體。

如果您選擇是， 會 AWS 私有 CA 建立預設政策並將其連接至您的儲存貯體。預設政策包含 `aws:SourceAccount`條件金鑰，這會限制對特定 AWS 帳戶的存取。如果您想要進一步限制存取，您可以將其他條件金鑰新增至政策，例如在上述範例中。

如果您選擇否，您必須先將政策連接到儲存貯體，才能產生稽核報告。使用中所述的政策模式為稽核報告準備 Amazon S3 儲存貯體。如需連接政策的資訊，請參閱使用 Amazon S3 主控台新增儲存貯體政策。

5. 在輸出格式下，針對 JavaScript 物件標記選擇 JSON，或針對逗號分隔值選擇 CSV。
6. 選擇 Generate audit report (產生稽核報告)。

建立稽核報告 (AWS CLI)

1. 如果您還沒有要使用的 S3 儲存貯體，請建立一個。
2. 將政策連接至您的儲存貯體。使用中所述的政策模式為稽核報告準備 Amazon S3 儲存貯體。如需連接政策的資訊，請參閱使用 Amazon S3 主控台新增儲存貯體政策
3. 使用 create-certificate-authority-audit-report 命令來建立稽核報告，並將其放在預備的 S3 儲存貯體中。

```
$ aws acm-pca create-certificate-authority-audit-report \
--certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566 \
--s3-bucket-name amzn-s3-demo-bucket \
--audit-report-response-format JSON
```

擷取稽核報告

若要擷取稽核報告以供檢查，請使用 Amazon S3 主控台、API、CLI 或 SDK。如需詳細資訊，請參閱《Amazon Simple Storage Service 使用者指南》中的下載物件。

加密您的稽核報告

您可以選擇性地在包含稽核報告的 Amazon S3 儲存貯體上設定加密。 AWS 私有 CA 支援 S3 中資產的兩種加密模式：

- 使用 Amazon S3-managed AES-256 金鑰自動進行伺服器端加密。
- 使用 AWS Key Management Service 和根據您的規格 AWS KMS key 設定的 進行客戶受管加密。

> **Note**
>
> AWS 私有 CA 不支援使用 S3 自動產生的預設 KMS 金鑰。

下列程序說明如何設定每個加密選項。

設定自動加密

完成以下步驟以啟用 S3 伺服器端加密。

1. 開啟位於 https://console.aws.amazon.com/s3/ 的 Amazon S3 主控台。
2. 在儲存貯體表格中，選擇將存放 AWS 私有 CA 資產的儲存貯體。
3. 在您的儲存貯體頁面上，選擇屬性索引標籤。
4. 選擇預設加密卡片。
5. 選擇啟用。
6. 選擇 Amazon S3 金鑰 (SSE-S3)。
7. 選擇 Save Changes (儲存變更)。

設定自訂加密

完成下列步驟，以使用自訂金鑰啟用加密。

1. 開啟位於 https://console.aws.amazon.com/s3/ 的 Amazon S3 主控台。
2. 在儲存貯體表格中，選擇將存放 AWS 私有 CA 資產的儲存貯體。
3. 在您的儲存貯體頁面上，選擇屬性索引標籤。
4. 選擇預設加密卡片。
5. 選擇啟用。
6. 選擇AWS Key Management Service 金鑰 (SSE-KMS)。
7. 選擇從 AWS KMS 金鑰中選擇或輸入 AWS KMS key ARN。
8. 選擇 Save Changes (儲存變更)。
9. （選用）如果您還沒有 KMS 金鑰，請使用下列 AWS CLI create-key 命令建立一個：

```
$ aws kms create-key
```

輸出包含 KMS 金鑰的金鑰 ID 和 Amazon Resource Name (ARN)。以下是輸出範例：

```
{
    "KeyMetadata": {
        "KeyId": "01234567-89ab-cdef-0123-456789abcdef",
        "Description": "",
        "Enabled": true,
        "KeyUsage": "ENCRYPT_DECRYPT",
        "KeyState": "Enabled",
        "CreationDate": 1478910250.94,
        "Arn": "arn:aws:kms:us-west-2:123456789012:key/01234567-89ab-
cdef-0123-456789abcdef",
        "AWSAccountId": "123456789012"
    }
}
```

10. 使用下列步驟，您可以授予 AWS 私有 CA 服務主體使用 KMS 金鑰的許可。根據預設，所有 KMS 金鑰都是私有的；只有資源擁有者可以使用 KMS 金鑰來加密和解密資料。然而，資源擁有者可以授與其他使用者和資源存取 KMS 金鑰的許可。服務主體必須位於與存放 KMS 金鑰相同的區域。

    a. 首先，使用以下 get-key-policy 命令，將 KMS 金鑰的預設政策儲存為 `policy.json`：

    ```
    $ aws kms get-key-policy --key-id key-id --policy-name default --output text
     > ./policy.json
    ```

    b. 在文字編輯器中開啟 `policy.json` 檔案。選取下列其中一個政策陳述式，並將其新增至現有政策。

    如果您的 Amazon S3 儲存貯體金鑰已啟用，請使用下列陳述式：

    ```
    {
       "Sid":"Allow ACM-PCA use of the key",
       "Effect":"Allow",
       "Principal":{
          "Service":"acm-pca.amazonaws.com"
       },
       "Action":[
          "kms:GenerateDataKey",
          "kms:Decrypt"
       ],
       "Resource":"*",
       "Condition":{
    ```

```
      "StringLike":{
         "kms:EncryptionContext:aws:s3:arn":"arn:aws:s3:::bucket-name"
      }
   }
}
```

如果您的 Amazon S3 儲存貯體金鑰已停用，請使用下列陳述式：

```
{
   "Sid":"Allow ACM-PCA use of the key",
   "Effect":"Allow",
   "Principal":{
      "Service":"acm-pca.amazonaws.com"
   },
   "Action":[
      "kms:GenerateDataKey",
      "kms:Decrypt"
   ],
   "Resource":"*",
   "Condition":{
      "StringLike":{
         "kms:EncryptionContext:aws:s3:arn":[
            "arn:aws:s3:::bucket-name/acm-pca-permission-test-key",
            "arn:aws:s3:::bucket-name/acm-pca-permission-test-key-private",
            "arn:aws:s3:::bucket-name/audit-report/*",
            "arn:aws:s3:::bucket-name/crl/*"
         ]
      }
   }
}
```

c. 最後，使用以下 put-key-policy 命令套用更新的政策：

```
$ aws kms put-key-policy --key-id key_id --policy-name default --policy file://
policy.json
```

中的基礎設施安全 AWS 私有憑證授權單位

作為受管服務， AWS 私有憑證授權單位 受到 AWS 全球網路安全的保護。如需 AWS 安全服務和 如何 AWS 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的最佳實務設計您的 AWS 環境，請參閱安全支柱 AWS Well-Architected Framework 中的基礎設施保護。

您可以使用 AWS 發佈的 API 呼叫， AWS 私有 CA 透過網路存取 。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。

AWS 私有 CA VPC 端點 (AWS PrivateLink)

您可以設定界面 VPC 端點， AWS 私有 CA 在 VPC 和 之間建立私有連線。介面端點採用 AWS PrivateLink技術，可私下存取 AWS 私有 CA API 操作。 會 AWS PrivateLink 路由 VPC 與 AWS 私有 CA Amazon 網路之間的所有網路流量，避免公開網際網路。每個 VPC 端點皆會由一個或多個具私有 IP 地址彈性網路界面來表示，而該界面位於 VPC 子網路中。

介面 VPC 端點會直接將您的 VPC 連接到 ， AWS 私有 CA 無需網際網路閘道、NAT 裝置、VPN 連接或 Direct Connect 連線。VPC 中的執行個體不需要公有 IP 地址，就能與 AWS 私有 CA API 進行通訊。

若要 AWS 私有 CA 透過 VPC 使用 ，您必須從 VPC 內的執行個體連線。或者，您可以使用 AWS Virtual Private Network (Site-to-Site VPN) 或 將私有網路連接到 VPC Direct Connect。如需 的詳細資訊 Site-to-Site VPN，請參閱《Amazon VPC 使用者指南》中的 VPN 連線。如需有關 Direct Connect 的資訊，請參閱《Direct Connect 使用者指南》中的建立連線。

AWS 私有 CA 不需要使用 AWS PrivateLink，但我們建議將其作為額外的安全層。如需 AWS PrivateLink 和 VPC 端點的詳細資訊，請參閱透過 存取服務 AWS PrivateLink。

AWS 私有 CA VPC 端點的考量事項

設定介面 VPC 端點之前 AWS 私有 CA，請注意下列考量：

- AWS 私有 CA 在某些可用區域中可能不支援 VPC 端點。當您建立 VPC 端點時，請先檢查 管理主控台中的支援。不支援的可用區域會標記為「此可用區域不支援 服務」。
- VPC 端點不支援跨區域請求。請確實在計劃發出 AWS 私有 CA API 呼叫的相同區域中建立端點。
- VPC 端點僅支援 Amazon 透過 Amazon Route 53 提供的 DNS。如果您想要使用自己的 DNS，您可以使用條件式 DNS 轉送。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的 DHCP 選項集。
- 連接到 VPC 端點的安全群組，必須允許從 VPC 的私有子網路，透過 443 埠傳入的連線。

AWS 私有 CA API 目前支援下列 中的 VPC 端點 AWS 區域：

- 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)
- 非洲 (開普敦)
- 亞太地區 (香港)
- 亞太區域 (海德拉巴)
- 亞太地區 (雅加達)
- 亞太地區 (墨爾本)
- 亞太地區 (孟買)
- 亞太區域 (大阪)
- 亞太區域 (首爾)
- 亞太區域 (新加坡)
- 亞太地區 (雪梨)
- 亞太區域 (東京)
- 加拿大 (中部)
- 加拿大西部 (卡加利)
- 歐洲 (法蘭克福)
- 歐洲 (愛爾蘭)
- 歐洲 (倫敦)
- 歐洲 (米蘭)
- Europe (Paris)
- 歐洲 (西班牙)
- 歐洲 (斯德哥爾摩)
- 歐洲 (蘇黎世)
- 以色列 (特拉維夫)
- Middle East (Bahrain)
- 中東 (阿拉伯聯合大公國)
- 南美洲 (聖保羅)

建立 的 VPC 端點 AWS 私有 CA

您可以使用位於 https：//https://console.aws.amazon.com/vpc/ 的 VPC 主控台或 來建立 AWS 私有 CA 服務的 VPC 端點 AWS Command Line Interface。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的建立界面端點程序。 AWS 私有 CA 支援呼叫 VPC 內的所有 API 操作。

如果您已啟用端點的私有 DNS 主機名稱，則預設 AWS 私有 CA 端點現在會解析為您的 VPC 端點。如需預設服務端點的完整清單，請參閱服務端點和配額。

如果您尚未啟用私有 DNS 主機名稱，Amazon VPC 會提供 DNS 端點名稱，您可以使用下列格式：

```
vpc-endpoint-id.acm-pca.region.vpce.amazonaws.com
```

> **Note**
>
> 值##代表 支援的 AWS 區域的區域識別符 AWS 私有 CA，例如us-east-2美國東部（俄亥俄）區域。如需 的清單 AWS 私有 CA，請參閱 AWS Certificate Manager Private Certificate Authority Endpoints and Quotas。

如需詳細資訊，請參閱《Amazon AWS 私有 CA VPC 使用者指南》中的 VPC 端點 (AWS PrivateLink)。

建立 AWS 私有 CA的 VPC 端點政策

您可以為 的 Amazon VPC 端點建立政策 AWS 私有 CA ，以指定下列項目：

- 可執行動作的委託人
- 可執行的動作
- 可在其中執行動作的資源

如需詳細資訊，請參閱《Amazon VPC 指南》中的使用 VPC 端點控制對 服務的存取。

範例 – AWS 私有 CA 動作的 VPC 端點政策

連接至端點時，下列政策會將所有主體的存取權授予 AWS 私有 CA 動作 IssueCertificate、DescribeCertificateAuthority、GetCertificate、GetCertificateAut ListPermissions和 ListTags。每一節中的資源都是私有 CA。第一節會授權使用指定的私有 CA

和憑證範本建立終端實體憑證。如果您不想要控制使用的範本，則不需要 Condition 區段。但是，移除此區段會允許所有委託人建立 CA 憑證及終端實體憑證。

```
{
      "Statement":[
         {
            "Principal":"*",
            "Effect":"Allow",
            "Action":[
               "acm-pca:IssueCertificate"
            ],
            "Resource":[
               "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
            ],
            "Condition":{
               "StringEquals":{
                  "acm-pca:TemplateArn":"arn:aws:acm-pca:::template/
EndEntityCertificate/V1"
               }
            }
         },
         {
            "Principal":"*",
            "Effect":"Allow",
            "Action":[
               "acm-pca:DescribeCertificateAuthority",
               "acm-pca:GetCertificate",
               "acm-pca:GetCertificateAuthorityCertificate",
               "acm-pca:ListPermissions",
               "acm-pca:ListTags"
            ],
            "Resource":[
               "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
            ]
         }
      ]
   }
```

雙堆疊端點支援

AWS 私有憑證授權單位 提供支援 IPv4 和 IPv6 用戶端的雙堆疊公有端點。雙堆疊端點可讓用戶端 AWS 私有 CA 使用 IPv4 或 IPv6 地址與 通訊。 AWS 私有 CA 適用於 Active Directory 的 和適用於 SCEP 的 AWS 私有 CA Connector 也支援雙堆疊端點。

的 AWS 私有 CA 雙堆疊公有端點同時`https://acm-pca.your-region.api.aws`支援 IPv4 和 IPv6 用戶端。 AWS 私有 CA 也可以使用 透過 IPv4 和 IPv6 從虛擬私有雲端 (VPC) 私下存取 AWS PrivateLink。如需為 建立私有介面 VPC 端點的詳細資訊 AWS 私有 CA，請參閱 AWS 私有 CA VPC 端點 (AWS PrivateLink)。

如需詳細資訊，請參閱下列資源：

- 您 VPC 和子網路的 IP 定址
- VPC 的 IPv6 支援

在 IAM 和 中使用 IPv6 地址 AWS 私有 CA

嘗試 AWS 私有憑證授權單位 透過 IPv6 存取 之前，請確定任何包含 IP 地址限制的 IAM 政策已更新為包含 IPv6 地址範圍。未更新以處理 IPv6 地址的 IP 型政策可能會導致用戶端在開始使用 IPv6 時錯誤遺失或取得存取權。若要進一步了解 AWS 私有 CA 和雙堆疊支援，請參閱 雙堆疊端點支援。

> **Important**
>
> 這些陳述式不允許任何動作。將這些陳述式與允許特定動作的其他陳述式結合使用。

下列陳述式明確拒絕存取來自 IPv4 地址`192.0.2.*`範圍之請求的所有 AWS 私有 CA 許可。任何超出此範圍的 IP 地址都不會明確拒絕 AWS 私有 CA 許可。由於所有 IPv6 地址都超出拒絕範圍，因此此陳述式不會明確拒絕任何 IPv6 地址的 AWS 私有 CA 許可。

```
{
    "Sid": "DenyPrivateCAPermissions",
    "Effect": "Deny",
    "Action": [
        "acm-pca:*"
    ],
    "Resource": "*",
```

```
    "Condition": {
        "NotIpAddress": {
            "aws:SourceIp": [
                "192.0.2.0/24"
            ]
        }
    }
}
```

您可以修改 Condition元素以拒絕 IPv4 (192.0.2.0/24) 和 IPv6 (2001:db8::/32) 地址範圍，如下列範例所示：

```
{
    "Sid": "DenyPrivateCAPermissions",
    "Effect": "Deny",
    "Action": [
        "acm-pca:*"
    ],
    "Resource": "*",
    "Condition": {
        "NotIpAddress": {
            "aws:SourceIp": [
                "192.0.2.0/24",
                "2001:db8::/32"
            ]
        }
    }
}
```

AWS 私有憑證授權單位 客戶 CP/CPS 架構

AWS 私有憑證授權單位 提供基礎設施服務，可讓您建立憑證授權機構 (CA) 階層，包括根 CA 和次級 CAs，而不需要操作內部部署 CA 的投資和維護成本。當您使用 AWS 私有 CA 建立 CA 階層時，您與之間會有共同的責任 AWS 私有 CA。共同的責任模型可協助您減輕營運負擔，因為 會 AWS 操作、管理和控制服務營運所在設施的實體安全性。您承擔憑證授權機構的責任和管理 （包括建立和刪除 CA 資源；分發信任錨點；PKI 階層建立；憑證政策和實務；允許或拒絕跨 CA 共用的組態 AWS 帳戶；範本使用的政策；稽核；存取控制，包括職責分離；以及其他 CA 組態和政策）。您應該仔細考慮您選擇的服務，因為您的責任會根據所使用的服務、這些服務與 IT 環境的整合，以及適用的法律和法規而有所不同。如需詳細資訊，請參閱AWS 雲端 安全共同責任模型。

為您的私有憑證授權機構建立憑證政策 (CP) 或憑證實務陳述式 (CPS)，是管理公有金鑰基礎設施 (PKI) 的關鍵部分。CP 會定義 PKI 的所有要求/規則，而 CPS 會說明您如何符合 CP 要求。您負責建立 CP 和 CPS 做為 PKI 的憑證授權單位。 AWS 私有 CA 為您提供 AWS 控制和合規文件，例如AWS 系統和組織控制 (SOC) 2 報告，您可以使用 協助建立 CP 和 CPS，並視需要執行控制評估和驗證程序。 AWS SOC 報告是獨立的第三方檢查報告，示範 如何 AWS 實現關鍵合規控制和目標。報告的目的是協助您和稽核人員了解為支援操作和合規而建立的 AWS 控制項。

本文件提供符合 RFC 3647 的架構，協助您撰寫 CP 和 CPS，並識別您與 之間的共同責任 AWS 私有 CA。 AWS 私有 CA 具有合規責任的 CP/CPS 要求章節會以「共享」或「AWS 私有憑證授權單位」識別，並提供對應的「補充資訊」，以協助您了解 如何 AWS 私有 CA 符合相關聯的 CP/CPS 要求。例如，要求 5 (4.5.1) 是 AWS 私有 CA 責任，您可以在 AWS SOC 2 報告的第 D.6 節中找到對應的控制語言，以協助完成您的 CP/CPS。如需 AWS SOC 報告以及如何請求存取 SOC 報告的詳細資訊，請造訪我們的 SOC FAQs頁面。

CP/CPS 要求和責任

CP/CPS 要求	責任	補充資訊
1. Introduction (All)	You	您負責記錄與 PKI 相關的概觀、文件名稱和識別、PKI 參與者、憑證使用、政策管理，以及定義和縮寫。
2. Publication and Repository Responsibilities (All)	You	您有責任記錄與 PKI 相關的定義。
3. Identification and Authentication (All)	You	您有責任在憑證發行之前，將驗證最終使用者憑證申請人身分和/或其他屬性的程序記錄給 CA 或註冊機構 (RA)。
4. Certificate Life-Cycle Operational Requirements (4.4.1 — 4.4.6, 4.4.9 — 4.4.11)	Shared	您負責指定發行 CA、主體 CAs、RAs、訂閱者或其他參與者在憑證生命週期方面的要求。 AWS 私有 CA 為您提供兩種全受管機制，以協助支援撤銷狀態檢查：線上憑證狀態通訊

CP/CPS 要求	責任	補充資訊
		協定 (OCSP) 和憑證撤銷清單 (CRLs)，協助您符合 4.4.9 和 4.4.10。
4. Certificate Life-Cycle Operational Requirements (4.4.7, 4.4.8, 4.4.12)	N/A	AWS 私有 CA 不支援憑證重新金鑰、憑證修改或金鑰託管和復原。
5. Facility, Management, and Operational Controls (4.5.1)	AWS 私有 CA	您可以繼承存取控制，協助您符合本節中 AWS 私有 CA SOC 2 類型 2 報告範圍內的要求（請參閱 D.6 實體安全和環境保護一節）。 **Note** 您要負責將 CA 資料匯出或傳出 AWS 環境的實體安全性和資料分類，但不負責存放於其中的 CA 資料的實體安全性 AWS。
5. Facility, Management, and Operational Controls (4.5.2)	Shared	您負責滿足本節中針對 PKI 環境操作定義受信任角色的要求。 AWS 私有 CA 維護專屬於密碼編譯模組實體存取的受信任角色。

CP/CPS 要求	責任	補充資訊
5. Facility, Management, and Operational Controls (4.5.3)	Shared	您有責任滿足本節中針對信任人員的背景檢查、訓練和紀律行動程序的要求。 您可以繼承與 AWS 私有 CA SOC 2 類型 2 報告範圍內 AWS 員工背景檢查、訓練和紀律行動程序相關的控制（請參閱 A. 政策、A.1 控制環境、B. 通訊和 D.1 安全組織和 D.2 員工使用者存取）。
5. Facility, Management, and Operational Controls (4.5.4)	Shared	您負責啟用、設定保留，以及保護 CloudTrail 和稽核報告日誌和 CloudWatch 提醒。此外，您需負責建立日誌處理程序，並對使用滿足本節要求的 AWS 私有 CA 服務執行漏洞評估。 您可以繼承與日誌可用性相關的控制項， 實體存取/站台安全性、 CA/RA 組態管理、 AWS 基礎設施日誌的安全性、 和 SOC AWS 私有 CA 2 類型 2 報告範圍內 AWS 基礎設施的漏洞評估（請參閱 A.1 控制環境、 第 C.1 節 服務承諾、 D.2 員工使用者存取、 D.3 邏輯安全、 D.6 實體安全與環境保護、 D.7 變更管理、 D.8 資料完整性、 可用性、 和 備援、 和 E.1 監控活動）。

CP/CPS 要求	責任	補充資訊
5. Facility, Management, and Operational Controls (4.5.5)	Shared	您有責任設定符合本節要求的備份和保留期間。 您可以繼承與 AWS 私有 CA SOC 2 類型 2 報告範圍內之日誌可用性（設定時）相關的控制項（請參閱 D.8 資料完整性、可用性和備援）。
5. Facility, Management, and Operational Controls (4.5.6)	N/A	AWS 私有 CA 不支援金鑰轉換。
5. Facility, Management, and Operational Controls (4.5.7)	Shared	您負責實作 的特定事件和入侵處理程序 AWS 私有 CA ，以滿足本節中的要求。 您會繼承事件、入侵處理程序、業務持續性，以及特定於實體網站外殼和基礎設施操作的災難復原程序，協助您符合本節中 AWS 私有 CA SOC 2 類型 2 隱私權報告範圍內的要求（請參閱 D.8 資料完整性、可用性和備援以及 D.10 隱私權）。
5. Facility, Management, and Operational Controls (4.5.8)	You	您需要記錄與 CA 或 RA 終止和終止程序相關的要求，包括 CA 和 RA 封存記錄的託管人身分。

CP/CPS 要求	責任	補充資訊
6. Technical Controls (4.6.1)	Shared	您負責記錄 PKI 的金鑰產生和安裝需求。 AWS 私有 CA 為您提供經過 FIPS 140-3 第 3 級認證可產生 CA 金鑰的密碼編譯模組。
6. Technical Controls (4.6.2)	Shared	您有責任記錄私有金鑰保護和密碼編譯模組工程控制，例如密碼編譯標準要求和多人控制。 AWS 私有 CA 為您提供經過 FIPS 140-3 第 3 級認證的密碼編譯模組，可用於產生 CA 金鑰，以及對 HSMs兩方實體存取控制。
6. Technical Controls (4.6.3)	You	您有責任記錄金鑰對管理的其他層面，例如公有金鑰的封存和憑證的操作期間。
6. Technical Controls (4.6.4)	N/A	AWS AWS 私有 CA HSMs一律在線上，沒有「啟用資料」的概念。 **Note** 您有責任對 Private CA 實作使用者存取控制，以適當限制建立 CA 和發行憑證的能力。

CP/CPS 要求	責任	補充資訊
6. Technical Controls (4.6.5)	Shared	您負責記錄使用 Private CA 的電腦安全控制。 您可以繼承與 AWS 員工邏輯存取相關的控制、 AWS 基礎設施的網路和電腦安全控制，以及 AWS 私有 CA SOC 2 類型 2 報告範圍內 AWS 員工帳戶的密碼參數控制（請參閱 D.2 員工使用者存取、D.3 邏輯安全，以及 D.6 實體安全與環境保護）。
6. Technical Controls (4.6.6)	Shared	您有責任記錄與使用 Private CA 相關的安全管理控制。 您可以繼承與 AWS 私有 CA SOC 2 類型 2 報告範圍內之 AWS 私有 CA 服務系統開發控制相關的控制（請參閱 D.7 變更管理一節）。
6. Technical Controls (4.6.7)	Shared	如果適用於您的 PKI 環境，您需負責記錄使用 Private CA 的網路安全控制。 您可以繼承與 AWS 私有 CA SOC 2 類型 2 報告範圍內 AWS 基礎設施網路安全控制相關的控制（請參閱 C.1 服務承諾、D.3 邏輯安全和 E.1 監控活動一節）。
6. Technical Controls (4.6.8)	AWS 私有 CA	AWS 私有 CA 使用信任的時間來源來時間戳記 CA 資料。

CP/CPS 要求	責任	補充資訊
7. Certificate, CRL, and OCSP Profiles (All)	Shared	您有責任記錄符合 PKI 環境需求的設定檔要求和憑證輸入。 AWS 私有 CA 為您提供設定檔範本，以協助滿足您的設定檔需求。
8. Compliance Audit and Other Assessment (All)	Shared	您負責記錄合規稽核和其他評估。 AWS 私有 CA 為您提供 SOC 2 報告，協助您和稽核人員了解為支援操作和合規而建立的 AWS 控制項。
9. Other Business and Legal Matters	You	您負責記錄涵蓋您 Private CA 的一般業務和法律事宜。

監控 AWS 私有 CA 資源

監控是維護 AWS 私有 CA 和其他 AWS 解決方案的可靠性、可用性和效能的重要部分。 AWS 提供下列監控工具，讓您監看 AWS 私有 CA、回報錯誤，並適時採取自動動作：

- Amazon CloudWatch AWS 會即時監控您的 AWS 資源和您在 上執行的應用程式。您可以收集和追蹤指標、建立自訂儀板表，以及設定警示，在特定指標達到您指定的閾值時通知您或採取動作。例如，您可以讓 CloudWatch 追蹤 CPU 使用量或其他 Amazon EC2 執行個體指標，並在需要時自動啟動新的執行個體。如需詳細資訊，請參閱 Amazon CloudWatch 使用者指南。
- Amazon CloudWatch Logs 可讓您監控、存放和存取來自 Amazon EC2 執行個體、CloudTrail 及其他來源的日誌檔案。CloudWatch Logs 可監控日誌檔案中的資訊，並在達到特定閾值時通知您。您也可以將日誌資料存檔在高耐用性的儲存空間。如需詳細資訊，請參閱 Amazon CloudWatch Logs 使用者指南。
- AWS CloudTrail 擷取您 AWS 帳戶 發出或代表發出的 API 呼叫和相關事件，並傳送日誌檔案至您指定的 Amazon S3 儲存貯體。您可以找出哪些使用者和帳戶呼叫 AWS、發出呼叫的來源 IP 位址，以及呼叫的發生時間。如需詳細資訊，請參閱 AWS CloudTrail 使用者指南。
- Amazon EventBridge 為無伺服器事件匯流排服務，可讓您輕鬆將應用程式與來自各種來源的資料互相連線。EventBridge 可從您自己的應用程式、Software-as-a-Service(SaaS) 應用程式 AWS 和服務提供即時資料串流，並將該資料路由到 Lambda 等目標。這可讓您監控在服務中發生的事件，並建置事件導向的架構。如需詳細資訊，請參閱 Amazon EventBridge 使用者指南。

下列主題說明可與 搭配使用的 AWS 雲端監控工具 AWS 私有 CA。

AWS 私有 CA CloudWatch 指標

Amazon CloudWatch 是 AWS 資源的監控服務。您可以使用 CloudWatch 來收集和追蹤指標、設定警示，並自動回應 AWS 資源的變更。CloudWatch 指標至少發佈一次。

AWS 私有 CA 支援下列 CloudWatch 指標。

指標	Description
`CRLGenerated`	已產生憑證撤銷清單 (CRL)。此指標只適用於私有 CA。

指標	Description
MisconfiguredCRLBucket	未正確設定為 CRL 指定的 S3 儲存貯體。請檢查儲存貯體政策。此指標只適用於私有 CA。
Time	發出請求與發出完成（或失敗）之間的時間，以毫秒為單位。此指標僅適用於 IssueCertificate 操作。
Success	憑證已成功發行。此指標僅適用於 IssueCertificate 操作。
Failure	操作失敗。此指標僅適用於 IssueCertificate 操作。

如需 CloudWatch 指標的詳細資訊，請參閱下列主題：

- 使用 Amazon CloudWatch 指標
- 建立 Amazon CloudWatch 警示

AWS 私有 CA 使用 CloudWatch Events 監控

您可以使用 Amazon CloudWatch Events 自動化您的 AWS 服務，並自動回應系統事件，例如應用程式可用性問題或資源變更。 AWS 服務的事件會以接近即時的方式傳送到 CloudWatch Events。您可撰寫簡單的規則，指出您在意的事件，以及當事件符合規則時所要自動執行的動作。CloudWatch Events 至少發佈一次。如需詳細資訊，請參閱建立由事件觸發的 CloudWatch Events 規則.

使用 Amazon EventBridge 將 CloudWatch Events 轉換成動作。使用 EventBridge，您可以使用事件來觸發目標，包括 AWS Lambda 函數、 AWS Batch 工作、Amazon SNS 主題等。如需詳細資訊，請參閱什麼是 Amazon EventBridge？

建立私有 CA 時成功或失敗

這些事件是由 CreateCertificateAuthority 操作觸發的。

成功

成功時，操作會傳回新 CA 的 ARN。

```
{
   "version":"0",
   "id":"event_ID",
   "detail-type":"ACM Private CA Creation",
   "source":"aws.acm-pca",
   "account":"account",
   "time":"2019-11-04T19:14:56Z",
   "region":"region",
   "resources":[
      "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
   ],
   "detail":{
      "result":"success"
   }
}
```

失敗

失敗時，操作會傳回 CA 的 ARN。使用 ARN，您可以呼叫 DescribeCertificateAuthority 來判斷 CA 的狀態。

```
{
   "version":"0",
   "id":"event_ID",
   "detail-type":"ACM Private CA Creation",
   "source":"aws.acm-pca",
   "account":"account",
   "time":"2019-11-04T19:14:56Z",
   "region":"region",
   "resources":[
      "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
   ],
   "detail":{
      "result":"failure"
   }
}
```

發出憑證時成功或失敗

這些事件是由 IssueCertificate 操作觸發的。

成功

成功時，操作會傳回 CA 和新憑證的 ARN。

```
{
   "version":"0",
   "id":"event_ID",
   "detail-type":"ACM Private CA Certificate Issuance",
   "source":"aws.acm-pca",
   "account":"account",
   "time":"2019-11-04T19:57:46Z",
   "region":"region",
   "resources":[
      "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
      "arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID"
   ],
   "detail":{
      "result":"success"
   }
}
```

失敗

失敗時，操作會傳回憑證 ARN 和 CA 的 ARN。使用憑證 ARN，您可以呼叫 GetCertificate 來檢視失敗的原因。

```
{
   "version":"0",
   "id":"event_ID",
   "detail-type":"ACM Private CA Certificate Issuance",
   "source":"aws.acm-pca",
   "account":"account",
   "time":"2019-11-04T19:57:46Z",
   "region":"region",
   "resources":[
      "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
      "arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID"
   ],
   "detail":{
```

```
      "result":"failure"
   }
}
```

撤銷憑證時成功

這些事件是由 RevokeCertificate 操作觸發的。

如果撤銷失敗或憑證已遭撤銷，則不會傳送任何事件。

Success

成功時，操作會傳回 CA 和撤銷憑證的 ARN。

```
{
   "version":"0",
   "id":"event_ID",
   "detail-type":"ACM Private CA Certificate Revocation",
   "source":"aws.acm-pca",
   "account":"account",
   "time":"2019-11-05T20:25:19Z",
   "region":"region",
   "resources":[
      "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
      "arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID"
   ],
   "detail":{
      "result":"success"
   }
}
```

產生 CRL 時成功或失敗

這些事件是由 RevokeCerticate 操作觸發的，其會造成建立憑證撤銷清單 (CRL)。

成功

成功時，操作會傳回與 CRL 相關聯 CA 的 ARN。

```
{
   "version":"0",
```

```
   "id":"event_ID",
   "detail-type":"ACM Private CA CRL Generation",
   "source":"aws.acm-pca",
   "account":"account",
   "time":"2019-11-04T21:07:08Z",
   "region":"region",
   "resources":[
      "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
   ],
   "detail":{
      "result":"success"
   }
}
```

失敗 1 – 由於許可錯誤，CRL 無法儲存至 Amazon S3

如果發生此錯誤，請檢查您的 Amazon S3 儲存貯體許可。

```
{
   "version":"0",
   "id":"event_ID",
   "detail-type":"ACM Private CA CRL Generation",
   "source":"aws.acm-pca",
   "account":"account",
   "time":"2019-11-07T23:01:25Z",
   "region":"region",
   "resources":[
      "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
   ],
   "detail":{
      "result":"failure",
      "reason":"Failed to write CRL to S3. Check your S3 bucket permissions."
   }
}
```

失敗 2 – 由於內部錯誤，CRL 無法儲存至 Amazon S3

如果發生此錯誤，請重試操作。

```
{
   "version":"0",
```

```
   "id":"event_ID",
   "detail-type":"ACM Private CA CRL Generation",
   "source":"aws.acm-pca",
   "account":"account",
   "time":"2019-11-07T23:01:25Z",
   "region":"region",
   "resources":[
      "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
   ],
   "detail":{
      "result":"failure",
      "reason":"Failed to write CRL to S3. Internal failure."
   }
}
```

失敗 3 – AWS 私有 CA 無法建立 CRL

若要疑難排解此錯誤，請檢查您的 CloudWatch 指標。

```
{
   "version":"0",
   "id":"event_ID",
   "detail-type":"ACM Private CA CRL Generation",
   "source":"aws.acm-pca",
   "account":"account",
   "time":"2019-11-07T23:01:25Z",
   "region":"region",
   "resources":[
      "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
   ],
   "detail":{
      "result":"failure",
      "reason":"Failed to generate CRL. Internal failure."
   }
}
```

建立 CA 稽核報告時成功或失敗

這些事件是由 CreateCertificateAuthorityAuditReport 操作觸發的。

成功

成功時，操作會傳回 CA 的 ARN 和稽核報告的 ID。

```
{
   "version":"0",
   "id":"event_ID",
   "detail-type":"ACM Private CA Audit Report Generation",
   "source":"aws.acm-pca",
   "account":"account",
   "time":"2019-11-04T21:54:20Z",
   "region":"region",
   "resources":[
      "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
      "audit_report_ID"
   ],
   "detail":{
      "result":"success"
   }
}
```

失敗

當 Amazon S3 儲存貯體上 AWS 私有 CA 缺少PUT許可、儲存貯體上啟用加密或其他原因時，稽核報告可能會失敗。

```
{
   "version":"0",
   "id":"event_ID",
   "detail-type":"ACM Private CA Audit Report Generation",
   "source":"aws.acm-pca",
   "account":"account",
   "time":"2019-11-04T21:54:20Z",
   "region":"region",
   "resources":[
      "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
      "audit_report_ID"
   ],
   "detail":{
      "result":"failure"
   }
}
```

使用 記錄 AWS 私有憑證授權單位 API 呼叫 AWS CloudTrail

AWS 私有憑證授權單位 已與 服務整合 AWS CloudTrail，此服務提供由使用者、角色或 AWS 服務在其中採取之動作的記錄 AWS 私有 CA。CloudTrail 會將 的 API 呼叫和簽署操作擷取 AWS 私有 CA 為事件。擷取的呼叫包括從 AWS 私有 CA 主控台進行的呼叫，以及對 AWS 私有 CA API 操作的程式碼呼叫。如果您建立線索，則可以將 CloudTrail 事件持續交付至 Amazon S3 儲存貯體，包括 的事件 AWS 私有 CA。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 所收集的資訊，您可以判斷提出的請求 AWS 私有 CA、提出請求的 IP 地址、提出請求的人員、提出請求的時間，以及其他詳細資訊。

若要進一步了解 CloudTrail，請參閱AWS CloudTrail 《使用者指南》。

AWS 私有 CA CloudTrail 中的資訊

當您建立帳戶 AWS 帳戶 時，您的 上會啟用 CloudTrail。當活動在 中發生時 AWS 私有 CA，該活動會記錄在 CloudTrail 事件中，以及事件歷史記錄中的其他 AWS 服務事件。您可以在 中檢視、搜尋和下載最近的事件 AWS 帳戶。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 中的事件 AWS 帳戶，包括 的事件 AWS 私有 CA，請建立追蹤。線索能讓 CloudTrail 將日誌檔案交付至 Amazon S3 儲存貯體。依預設，當您在主控台中建立追蹤時，該追蹤會套用至所有的 AWS 區域。線索會記錄 AWS 分割區中所有區域的事件，並將日誌檔案交付至您指定的 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 日誌中收集的事件資料。如需詳細資訊，請參閱下列內容：

- 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 AWS 私有 CA 動作，並記錄在 AWS 私有 CA API 參考中。例如，對 `ImportCACertificate`、`IssueCertificate` 和 `CreateAuditReport` 動作發出的呼叫會在 CloudTrail 記錄檔案中產生專案。

每一筆事件或日誌專案都會包含產生請求者的資訊。身分資訊可協助您判斷下列事項：

- 請求是使用根還是 AWS Identity and Access Management (IAM) 使用者登入資料提出。
- 提出該請求時，是否使用了特定角色或聯合身分使用者的暫時安全憑證。

- 請求是否由其他 AWS 服務提出。

如需詳細資訊，請參閱 CloudTrail userIdentity 元素。

AWS 私有 CA 管理事件

AWS 私有 CA 與 CloudTrail 整合，以記錄使用者、角色或服務 AWS 在其中所做的 API 動作 AWS 私有 CA。您可以使用 CloudTrail 即時監控 AWS 私有 CA API 請求，並將日誌存放在 Amazon Simple Storage Service、Amazon CloudWatch Logs 和 Amazon CloudWatch Events 中。 AWS 私有 CA 支援將下列動作和操作記錄為 CloudTrail 日誌檔案中的事件：

- CreateCertificateAuthority
- CreateCertificateAuthorityAuditReport
- CreatePermission
- DeleteCertificateAuthority
- DeletePermission
- DeletePolicy
- DescribeCertificateAuthority
- DescribeCertificateAuthorityReport
- GetCertificate
- GetCertificateAuthorityCertificate
- GetCertificateAuthorityCsr
- GetPolicy
- ImportCertificateAuthorityCertificate
- IssueCertificate
- ListCertificateAuthorities
- ListPermissions
- ListTags
- PutPolicy
- RestoreCertificateAuthority
- RevokeCertificate
- TagCertificateAuthority

- UntagCertificateAuthority
- UpdateCertificateAuthority
- GenerateOCSPResponse - 當 AWS 私有 CA 產生 OCSP 回應時觸發。
- SignCertificate - 當您的用戶端呼叫 IssueCertificate 時產生。
- SignOCSPResponse - AWS 私有 CA 簽署 OCSP 回應時產生。
- GenerateCRL - 在 AWS 私有 CA 產生憑證撤銷清單 (CRL) 時產生。
- SignCACSR - AWS 私有 CA 簽署憑證授權機構 (CA) 憑證簽署請求 (CSR) 時產生。
- SignCRL - AWS 私有 CA 簽署 CRL 時產生。

範例 AWS 私有 CA 事件

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。一個事件為任何來源提出的單一請求，並包含請求動作、請求的日期和時間、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。

以下是 AWS 私有 CA CloudTrail 事件的範例。

範例 1：管理事件、 **IssueCertificate**

以下範例顯示的是展示 IssueCertificate 動作的 CloudTrail 日誌項目。

```
{
   "version":"0",
   "id":"event_ID",
   "detail-type":"ACM Private CA Certificate Issuance",
   "source":"aws.acm-pca",
   "account":"account",
   "time":"2019-11-04T19:57:46Z",
   "region":"region",
   "resources":[
      "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
      "arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID"
   ],
   "detail":{
      "result":"success"
   }
```

```
}
```

範例 2：管理事件、 **ImportCertificateAuthorityCertificate**

以下範例顯示的是展示 ImportCertificateAuthorityCertificate 動作的 CloudTrail 日誌項目。

```
{
   "eventVersion":"1.05",
   "userIdentity":{
      "type":"IAMUser",
      "principalId":"account",
      "arn":"arn:aws:iam::account:user/name",
      "accountId":"account",
      "accessKeyId":"key_ID"
   },
   "eventTime":"2018-01-26T21:53:28Z",
   "eventSource":"acm-pca.amazonaws.com",
   "eventName":"ImportCertificateAuthorityCertificate",
   "awsRegion":"region",
   "sourceIPAddress":"IP_address",
   "userAgent":"agent",
   "requestParameters":{
      "certificateAuthorityArn":"arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
      "certificate":{
         "hb":[
            45,
            45,
            ...10
         ],
         "offset":0,
         "isReadOnly":false,
         "bigEndian":true,
         "nativeByteOrder":false,
         "mark":-1,
         "position":1257,
         "limit":1257,
         "capacity":1257,
         "address":0
      },
      "certificateChain":{
         "hb":[
```

```
            45,
            45,
            ...10
         ],
         "offset":0,
         "isReadOnly":false,
         "bigEndian":true,
         "nativeByteOrder":false,
         "mark":-1,
         "position":1139,
         "limit":1139,
         "capacity":1139,
         "address":0
      }
   },
   "responseElements":null,
   "requestID":"request_ID",
   "eventID":"event_ID",
   "eventType":"AwsApiCall",
   "recipientAccountId":"account"
}
```

對 的問題進行故障診斷 AWS 私有憑證授權單位

如果您在使用 AWS 私有憑證授權單位時遇到問題，請參閱下列主題。

主題

- 對 AWS 私有 CA 憑證撤銷問題進行故障診斷
- 對 AWS 私有憑證授權單位 例外狀況訊息進行故障診斷
- 故障診斷符合 AWS 私有 CA 事項的憑證錯誤

對 AWS 私有 CA 憑證撤銷問題進行故障診斷

OCSP 回應延遲

如果發起人與區域節點快取或發行 CA 的區域有地理距離，則 OCSP 回應速度可能會較慢。如需區域節點快取可用性的詳細資訊，請參閱 Global Edge Network。建議您在附近使用憑證的區域發行憑證。

撤銷自我簽署憑證

您無法撤銷自我簽署的 CA 憑證。若要在功能上撤銷憑證，請刪除 CA。

對 AWS 私有憑證授權單位 例外狀況訊息進行故障診斷

AWS 私有 CA 命令可能會因為多種原因而失敗。如需每個例外狀況及解決這些例外狀況的建議相關資訊，請參閱下表。

AWS 私有 CA 例外狀況

傳回的例外狀況 AWS 私有 CA	Description	修補
`AccessDeniedException`	使用指定命令所需的許可，尚未由私有 CA 委派給呼叫帳戶。	如需在 中委派許可的資訊 AWS 私有 CA，請參閱 將憑證續約許可指派給 ACM。
`InvalidArgsException`	使用了無效參數來發出建立或更新憑證的請求。	請檢查命令的個別文件，以確認您的輸入參數有效。如果您正在建立新憑證，請確認請求

傳回的例外狀況 AWS 私有 CA	Description	修補
		的簽署演算法可與 CA 的金鑰類型搭配使用。
InvalidStateException	因為相關聯的私有 CA 未處於 ACTIVE 狀態，所以無法更新憑證。	嘗試還原私有 CA。如果私有 CA 不在其還原期間內，則無法還原 CA，也無法更新憑證。
LimitExceededException	每個憑證授權機構 (CA) 都有可發行的憑證配額。與指定憑證相關聯的私有 CA 已達到其配額上限。如需詳細資訊，請參閱《 AWS 一般參考 指南》中的 Service Quotas。	請聯絡 AWS 支援 中心以請求提高配額。
MalformedCSRException	AWS 私有 CA 無法確認或驗證提交的憑證簽署請求 (CSR)。	請確認您的 CSR 已正確產生及設定。
OtherException	發生內部錯誤而導致請求失敗。	請嘗試重新執行命令。如果問題仍然存在，請聯絡 AWS 支援 中心。
RequestFailedException	您 AWS 環境中的聯網問題導致請求失敗。	重試 請求。如果故障仍然存在，請檢查您的 Amazon VPC (VPC) 組態。
ResourceNotFoundException	發出憑證的私有 CA 已刪除，且已不存在。	向另一個有效的 CA 請求新憑證。

傳回的例外狀況 AWS 私有 CA	Description	修補
`ThrottlingException`	請求的 API 動作因超過配額而失敗。	確認您發出的呼叫次數不超過允許的次數 AWS 私有 CA。 您遇到暫時性狀況 (而非超過配額) 時，也可能導致發生 `ThrottlingException` 錯誤。如果您發生該錯誤，且發出的呼叫數未超過配額，請重試請求。 如果您即將達到配額上限，您可能可以請求提高配額。如需詳細資訊，請參閱《 AWS 一般參考 指南》中的 Service Quotas。
`ValidationException`	請求的輸入參數格式不正確，或根憑證有效期的結束時間早於請求的憑證有效期結束時間。	請檢查命令的輸入參數語法要求，以及 CA 根憑證的有效期。如需如何變更有效期的資訊，請參閱在 中更新私有 CA AWS 私有憑證授權單位。

故障診斷符合 AWS 私有 CA 事項的憑證錯誤

事項連線標準指定憑證組態，以提高物聯網 (IoT) 裝置的安全性和一致性。用於建立符合事項的根 CA、中繼 CA 和終端實體憑證的 Java 範例可在 找到使用 AWS 私有 CA 實作事項憑證。

為了協助故障診斷，Matter 開發人員提供名為 chip-cert 的憑證驗證工具。工具報告的錯誤會列在下表中並搭配修補。

錯誤碼	意義	修補
0x0000035	`BasicConstraints` 、`KeyUsage`和 `Extension`	請確定您已為使用案例選取正確的範本。

錯誤碼	意義	修補
	KeyUsage 延伸必須標記為關鍵。	
0x00000000	授權單位金鑰識別符延伸必須存在。	AWS 私有 CA 不會在根憑證上設定授權金鑰識別符延伸。您必須 CSR 產生 Base64-encoded的 AuthorityKeyIdentifier 值，然後將 CustomExtension。如需詳細資訊，請參閱為節點操作憑證 (NO CA。及啟用產品鑑證授權機構 (PAA)。
0x000000E	憑證已過期。	確保您使用的憑證未過期。

錯誤碼	意義	修補

0x00000004	憑證鏈驗證失敗。	如果您嘗試建立符合事項的終端實體憑證而不使用提供的 Java 範用 AWS 私有 CA API 傳遞正確設定的 KeyUsage，則可能會發生。
		
		根據預設， AWS 私有 CA 會產生九位元 KeyUsage 延伸值，第產生額外的位元組。在格式轉換期間，Matter 會忽略額外的位元鏈結驗證失敗。不過，APIPassthrough 範本中的 CustomExt可用來設定KeyUsage值中的確切位元組數。如需範例，請參閱操作憑證 (NOC)。
		
		如果您修改範例程式碼或使用替代 X.509 公用程式，例如 OpenS需要執行手動驗證，以避免鏈驗證錯誤。
		
		驗證轉換是否無失真
		
		1. 使用 openssl 來驗證節點（終端實體）憑證是否包含有效的在此分支中， rcac.pem是根 CA 憑證、 icac.pem是中繼證， noc.pem是節點憑證。
		
		`openssl verify -verbose -CAfile <(cat rcac.pem icac noc.pem`
		
		2. 使用 chip-cert 將 PEM 格式節點憑證轉換為 TLV（標籤、長值）格式，然後再次返回。
		
		`./chip-cert convert-cert noc.pem noc.chip -c`
		`./chip-cert convert-cert noc.chip noc_converted.pem`
		
		檔案 noc.pem和 noc_converted.pem 應該與字串比較工認的完全相同。

使用 保護 Kubernetes AWS 私有憑證授權單位

您可以使用 AWS 私有憑證授權單位 提供透過 TLS 和 mTLS 進行安全身分驗證和加密的憑證。 AWS 私有 CA 提供開放原始碼外掛程式 AWS 私有 CA Connector for Kubernetes，(`aws-privateca-issuer`) 給 Kubernetes 廣泛採用的 cert-manager 附加元件，以請求憑證、將憑證分發至 Kubernetes 秘密，並自動化憑證續約。

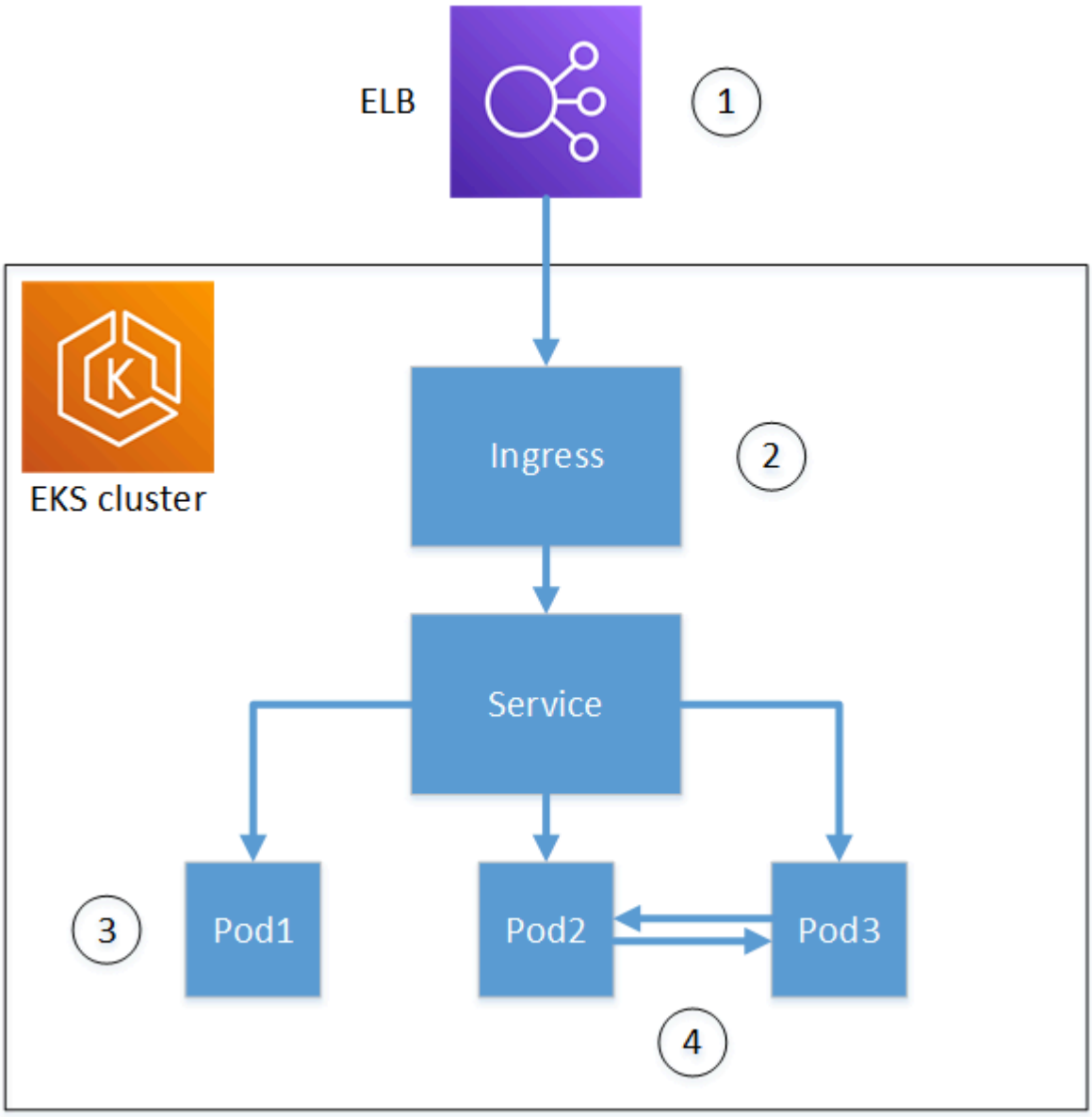
`aws-privateca-issuer` 外掛程式可讓您透過 發行 AWS 私有 CA 憑證`cert-manager`。您可以搭配 Amazon Elastic Kubernetes Service (Amazon EKS) AWS、 上的自我管理 Kubernetes 叢集，或在內部部署 Kubernetes 叢集中使用外掛程式。外掛程式適用於 x86 和 ARM 架構。

AWS 私有 CA 具有無法匯出的 HSM 後端金鑰。如果您有控制存取和稽核 CA 操作的法規要求，您可以使用 AWS 私有 CA 來改善稽核能力並支援合規。

> **Note**
>
> 如果您在 Amazon EKS 上執行 ，我們建議您使用 `cert-manager`和 `aws-privateca-connector-for-kubernetes` 附加元件以獲得受管安裝體驗。如需詳細資訊，請參閱AWS 附加元件。

概念

下圖顯示可在 Amazon EKS 叢集中使用 TLS 的一些選項。範例叢集位於負載平衡器後方。這些數字可識別 TLS 安全通訊的可能端點。



1. 負載平衡器的終止

   Elastic Load Balancing (Elastic Load Balancing) 已與 AWS Certificate Manager 服務整合。您不需要在負載平衡器`cert-manager`上安裝 。您可以使用私有 CA 佈建 ACM、使用私有 CA 簽署憑證，以及使用 Elastic Load Balancing 主控台安裝憑證。 AWS 私有 CA 憑證會自動續約。

   或者，您可以提供私有憑證給非AWS 負載平衡器來終止 TLS。

   這可提供遠端用戶端與負載平衡器之間的加密通訊。將未加密的負載平衡器傳遞至 Amazon EKS 叢集之後的資料。

2. 在 Kubernetes 輸入控制器終止

輸入控制器位於 Amazon EKS 叢集內，可作為負載平衡器和路由器。若要使用輸入控制器做為叢集的端點進行外部通訊，您必須：

- 安裝 `cert-manager`和 `aws-privateca-issuer`
- 使用來自 的 TLS 私有憑證佈建控制器 AWS 私有 CA。

負載平衡器和輸入控制器之間的通訊會加密，資料會未加密傳遞至叢集的資源。

3. 在 Pod 終止

   每個 Pod 是共用儲存和網路資源的一或多個容器群組。如果您同時安裝 `cert-manager` 和 ，`aws-privateca-issuer`並使用私有 CA 佈建叢集，Kubernetes 可以視需要在 Pod 上安裝已簽署的 TLS 私有憑證。根據預設，在 Pod 終止的 TLS 連線不適用於叢集中的其他 Pod。

4. Pod 之間的安全通訊。

   您可以使用憑證佈建多個 Pod，以允許它們彼此通訊。可能的情況如下：

   - 使用 Kubernetes 佈建 產生的自我簽署憑證。這可保護 Pod 之間的通訊，但自我簽署憑證不符合 HIPAA 或 FIPS 要求。
   - 使用由 簽署的憑證進行佈建 AWS 私有 CA。這需要同時安裝 `cert-manager`和 `aws-privateca-issuer`。然後，Kubernetes 可以視需要在 Pod 上安裝已簽署的 mTLS 憑證。

考量

AWS 私有憑證授權單位 搭配 Kubernetes 使用 時，請謹記下列考量。

cert-manager 的跨帳戶使用

具有 CA 跨帳戶存取權的管理員可以使用 for Kubernetes 的 `cert-manager` 附加元件，來為使用共用 CA 的叢集佈建憑證。如需詳細資訊，請參閱 跨帳戶存取私有 CAs的安全最佳實務。

在跨帳戶案例中，您只能使用特定 AWS 私有 CA 憑證範本。

下表列出您可以搭配 cert-manager 用來佈建 Kubernetes 叢集的 AWS 私有 CA 範本。

Kubernetes 支援的範本	支援跨帳戶使用
BlankEndEntityCertificate_CSRPassthrough/V1 定義	否

Kubernetes 支援的範本	支援跨帳戶使用
CodeSigningCertificate/V1 定義	否
EndEntityCertificate/V1 定義	是
EndEntityClientAuthCertificate/V1 定義	是
EndEntityServerAuthCertificate/V1 定義	是
OCSPSigningCertificate/V1 定義	否

Connector AWS 私有 CA for Kubernetes 入門。

下列主題說明如何使用 AWS 私有 CA 來保護 Kubernetes 叢集中的通訊。如需另一個範例，請參閱 GitHub 上 Kubernetes 的傳輸中加密。

您可以使用私有憑證授權機構來保護與 Amazon EKS 叢集的通訊。開始之前，請務必備妥下列項目：

- 具有適用於您安全政策範圍之適當許可 AWS 的帳戶。

  Amazon EKS clusters

  JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "IAM",
      "Effect": "Allow",
      "Action": [
        "iam:CreateRole",
        "iam:AttachRolePolicy",
        "iam:GetRole"
      ],
      "Resource": "*"
    },
    {
      "Sid": "EKS",
      "Effect": "Allow",
      "Action": [
```

```
        "eks:CreateAddon",
        "eks:DescribeAddon",
        "eks:CreatePodIdentityAssociation",
        "eks:DescribeCluster"
      ],
      "Resource": "*"
    },
    {
      "Sid": "IAMPassRole",
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": "arn:aws:iam::*:role/CertManagerPrivateCARole"
    }
  ]
}
```

Other clusters

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "GetAndIssuePCACertificates",
      "Effect": "Allow",
      "Action": [
        "acm-pca:GetCertificate",
        "acm-pca:IssueCertificate"
      ],
      "Resource": "*"
    },
    {
      "Sid": "RolesAnywhere",
      "Effect": "Allow",
      "Action": [
        "rolesanywhere:CreateProfile"
      ],
      "Resource": "*"
    },
    {
      "Sid": "IAM",
```

```
      "Effect": "Allow",
      "Action": [
        "iam:CreateRole",
        "iam:AttachRolePolicy"
      ],
      "Resource": "*"
    },
    {
      "Sid": "IAMPassRole",
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": "arn:aws:iam::*:role/CertManagerPrivateCARole"
    }
  ]
}
```

- Kubernetes 叢集。若要建立 Amazon Elastic Kubernetes Service 叢集，請參閱 Amazon EKS 快速入門指南。為了簡化，請建立環境變數以保留叢集名稱：

```
export CLUSTER=aws-privateca-demo
```

- CA AWS 區域 和 Amazon EKS 叢集所在的 。為了簡化，請建立 環境變數以保留 區域：

```
export REGION=aws-region
```

- AWS 私有 CA 私有憑證授權單位的 Amazon Resource Name (ARN)。為了簡化，請建立環境變數以保留私有 CA ARN：

```
export CA_ARN="arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID"
```

  若要建立私有 CA，請參閱在 中https://docs.aws.amazon.com/privateca/latest/userguide/create-CA.html建立私有 CA AWS 私有 CA

- 已安裝下列軟體的電腦：
  - AWS CLI v2 已設定
  - kubectl 1.13 版以上
  - 對於非 Amazon EKS 叢集，Helm v3

安裝 cert-manager

若要使用私有 CA，您必須安裝請求憑證、分發憑證和自動化憑證續約的cert-manager>附加元件。您也必須安裝可讓您從中發行私有憑證的aws-private-ca-issuer外掛程式 AWS 私有 CA。使用下列步驟來安裝附加元件和外掛程式。

Amazon EKS clusters

安裝 cert-manager 做為 Amazon EKS 附加元件：

```
aws eks create-addon \
  --cluster-name $CLUSTER \
  --addon-name cert-manager \
  --region $REGION
```

Other clusters

cert-manager 使用 Helm 安裝：

```
helm repo add jetstack https://charts.jetstack.io
helm repo update

helm install cert-manager jetstack/cert-manager \
  --namespace cert-manager \
  --create-namespace \
  --set crds.enabled=true
```

設定 IAM 許可

aws-privateca-issuer 外掛程式需要 與 互動的許可 AWS 私有 CA。對於 Amazon EKS 叢集，您可以使用 Pod 身分。對於其他叢集，您可以使用 AWS Identity and Access Management Roles Anywhere。

Fist，建立 IAM 政策。此政策使用 AWSPrivateCAConnectorForKubernetesPolicy受管政策。如需政策的詳細資訊，請參閱《 AWS 受管政策參考指南》中的 AWSPrivateCAConnectorForKubernetesPolicy。

Amazon EKS clusters

1. 建立名為 的檔案，trust-policy.json其中包含下列信任政策：

JSON

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

2. 執行下列命令來建立 IAM 角色：

```
ROLE_ARN=$(aws iam create-role \
  --role-name CertManagerPrivateCARole \
  --assume-role-policy-document file://trust-policy.json \
  --region $REGION \
  --output text \
  --query "Role.Arn")

 aws iam attach-role-policy \
  --role-name CertManagerPrivateCARole \
  --policy-arn arn:aws:iam::aws:policy/AWSPrivateCAConnectorForKubernetesPolicy
```

Other clusters

1. 建立信任錨點，信任存放在 中的私有 CACA_ARN。如需說明，請參閱 入門 IAM Roles Anywhere。建立環境變數以存放信任錨點 ARN：

```
export TRUST_ANCHOR_ARN=trustAnchorArn
```

2. 建立名為 的檔案，`trust-policy.json`其中包含下列信任政策：

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "TrustPolicyForSelfManagedOrOnPremiseClusters",
            "Effect": "Allow",
            "Principal": {
                "Service": "rolesanywhere.amazonaws.com"
            },
            "Action": [
                "sts:AssumeRole",
                "sts:SetSourceIdentity",
                "sts:TagSession"
            ],
            "Condition": {
                "ArnEquals": {
                    "aws:SourceArn": [
                        "arn:aws:rolesanywhere:us-
east-1:123456789012:trust-anchor/TRUST_ANCHOR_ARN"
                    ]
                },
                "StringEquals": {
                    "aws:PrincipalTag/x509Subject/CN": "aws-privateca-
issuer"
                }
            }
        }
    ]
}
```

3. 執行下列命令來建立 IAM 角色：

```
ROLE_ARN=$(aws iam create-role \
  --role-name CertManagerPrivateCARole \
  --assume-role-policy-document file://trust-policy.json \
  --query "Role.Arn" \
  --region $REGION \
  --output text)

 aws iam attach-role-policy \
```

```
  --role-name CertManagerPrivateCARole \
  --region $REGION \
  --policy-arn arn:aws:iam::aws:policy/AWSPrivateCAConnectorForKubernetesPolicy
```

安裝和設定 AWS 私有 CA 叢集發行者

若要安裝 附加aws-privateca-connector-for-kubernetes元件，請使用下列命令：

Amazon EKS clusters

建立附加元件：

```
aws eks create-addon --region $REGION \
  --cluster-name $CLUSTER \
  --addon-name aws-privateca-connector-for-kubernetes \
  --pod-identity-associations "[{
    \"serviceAccount\": \"aws-privateca-issuer\",
    \"roleArn\": \"$ROLE_ARN\"
  }]"
```

然後等待附加元件處於作用中狀態：

```
aws eks describe-addon \
  --cluster-name $CLUSTER \
  --addon-name aws-privateca-connector-for-kubernetes \
  --region $REGION \
  --query 'addon.status'
```

Other clusters

1. 在 中建立設定檔 IAM Roles Anywhere：

```
PROFILE_ARN=$(aws rolesanywhere create-profile \
  --name "privateca-profile" \
  --role-arns "$ROLE_ARN" \
  --region "$REGION" \
  --query 'profile.profileArn' \
  --enabled \
  --output text)
```

2. 產生用戶端憑證以搭配 Connector for Kubernetes 使用 IAM Roles Anywhere ，並驗證 AWS 私有 CA：

   a. 產生用戶端憑證的私有金鑰：

   ```
   openssl genrsa -out client.key 2048
   ```

   b. 產生用戶端憑證的憑證簽署請求 (CSR)：

   ```
   openssl req -new \
     -key client.key \
     -out client.csr \
     -subj "/CN=aws-privateca-issuer"
   ```

   c. 從以下位置發出用戶端憑證 AWS 私有 CA：

   ```
   CERT_ARN=$(aws acm-pca issue-certificate \
     --signing-algorithm SHA256WITHRSA \
     --csr fileb://client.csr \
     --validity Value=1,Type=DAYS \
     --certificate-authority-arn "$CA_ARN" \
     --region "$REGION" \
     --query 'CertificateArn' \
     --output text)
   ```

   d. 將用戶端憑證存放在本機：

   ```
   aws acm-pca get-certificate \
     --certificate-authority-arn $CA_ARN \
     --certificate-arn $CERT_ARN \
     --region $REGION \
     --query 'Certificate'
     --output text > pca-issuer-client-cert.pem
   ```

3. 使用用戶端憑證在叢集中安裝 AWS 私有 CA 發行者：

   a. 新增 awspca Helm 儲存庫:

   ```
   helm repo add awspca https://cert-manager.github.io/aws-privateca-issuer
   helm repo update
   ```

   b. 建立命名空間：

```
kubectl create namespace aws-privateca-issuer
```

c. 將先前建立的憑證放入秘密中：

```
kubectl create secret tls aws-privateca-credentials \
  -n aws-privateca-issuer \
  --cert=pca-issuer-client-cert.pem \
  --key=client.key
```

4. 使用下列命令安裝 AWS 私有 CA 發行者 IAM Roles Anywhere：

a. 建立名為 的檔案`values.yaml`，將 AWS 私有 CA 發行者外掛程式設定為與下列項目搭配使用 IAM Roles Anywhere：

```
cat > values.yaml <<EOF
env:
  AWS_EC2_METADATA_SERVICE_ENDPOINT: "http://127.0.0.1:9911"

extraContainers:
  - name: "rolesanywhere-credential-helper"
    image: "public.ecr.aws/rolesanywhere/credential-helper:latest"
    command: ["aws_signing_helper"]
    args:
      - "serve"
      - "--private-key"
      - "/etc/cert/tls.key"
      - "--certificate"
      - "/etc/cert/tls.crt"
      - "--role-arn"
      - "$ROLE_ARN"
      - "--profile-arn"
      - "$PROFILE_ARN"
      - "--trust-anchor-arn"
      - "$TRUST_ANCHOR_ARN"
    volumeMounts:
      - name: cert
        mountPath: /etc/cert/
        readOnly: true

volumes:
  - name: cert
    secret:
```

```
      secretName: aws-privateca-credentials
EOF
```

b. 使用下列命令安裝 AWS 私有 CA 發行者 IAM Roles Anywhere：

```
helm install aws-privateca-issuer awspca/aws-privateca-issuer \
  -n aws-privateca-issuer \
  -f values.yaml
```

等待發行者準備就緒。使用下列命令：

```
kubectl wait --for=condition=ready pods --all -n aws-privateca-issuer --timeout=120s
```

然後驗證安裝，以確保所有 Pod 都已達到 READY 狀態：

```
kubectl -n aws-privateca-issuer get all
```

若要設定 aws-private-ca-cluster-issuer，請建立名為 的 YAML 檔案，cluster-issuer.yaml其中包含發行者的組態：

```
cat > cluster-issuer.yaml <<EOF
apiVersion: awspca.cert-manager.io/v1beta1
kind: AWSPCAClusterIssuer
metadata:
  name: aws-privateca-cluster-issuer
spec:
  arn: "$CA_ARN"
  region: "$REGION"
EOF
```

接著，套用叢集組態：

```
kubectl apply -f cluster-issuer.yaml
```

檢查發行者的狀態：

```
kubectl describe awspcaclusterissuer aws-privateca-cluster-issuer
```

您應該會看到類似以下的回應：

```
Status:
  Conditions:
    Last Transition Time:  2025-08-13T21:00:00Z
    Message:               AWS PCA Issuer is ready
    Reason:                Verified
    Status:                True
    Type:                  Ready
```

使用 cert-manager 管理 AWS 私有 CA 用戶端憑證

如果您不是使用 Amazon EKS 叢集，則在 中手動引導信任的憑證之後aws-privateca-issuer，您可以轉換到由 管理的用戶端身分驗證憑證cert-manager。這可讓 cert-manager自動續約用戶端身分驗證憑證。

1. 建立名為 的檔案pca-auth-cert.yaml：

```
cat > pca-auth-cert.yaml <<EOF
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: aws-privateca-client-cert
  namespace: aws-privateca-issuer
spec:
  secretName: aws-privateca-credentials
  duration: 168h
  renewBefore: 48h
  commonName: aws-privateca-issuer
  privateKey:
    algorithm: ECDSA
    size: 256
    rotationPolicy: Always
  usages:
    - client auth
  issuerRef:
    name: aws-privateca-cluster-issuer
    kind: AWSPCAClusterIssuer
    group: awspca.cert-manager.io
EOF
```

2. 建立新的受管用戶端身分驗證憑證：

```
kubectl apply -f pca-auth-cert.yaml
```

3. 驗證憑證是否已建立：

```
kubectl get certificate aws-privateca-client-cert -n aws-privateca-issuer
```

您應該會看到類似以下的回應：

```
NAME                        READY   SECRET                        AGE
aws-privateca-client-cert   True    aws-privateca-credentials   19m
```

發行您的第一個 TLS 憑證

現在aws-privateca-issuer已安裝 cert-manager和 ，您可以發行憑證。

建立名為 的 YAML 檔案，certificate.yaml其中包含憑證資源：

```
cat > certificate.yaml <<EOF
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: example-certificate
  namespace: default
spec:
  secretName: example-certificate-tls
  issuerRef:
    name: aws-privateca-cluster-issuer
    kind: AWSPCAClusterIssuer
    group: awspca.cert-manager.io
  commonName: example.internal
  dnsNames:
    - example.internal
    - api.example.internal
  duration: 2160h # 90 days
  renewBefore: 360h # 15 days
  usages:
    - digital signature
    - key encipherment
    - server auth
EOF
```

使用下列命令套用憑證：

```
kubectl apply -f certificate.yaml
```

然後，您可以使用下列命令檢查憑證的狀態：

```
kubectl get certificate example-certificate
kubectl describe certificate example-certificate
```

您應該會看到類似以下的回應：

```
NAME                  READY   SECRET                     AGE
example-certificate   True    example-certificate-tls    30s
```

您可以使用下列命令來檢查發行的憑證：

```
kubectl get secret example-certificate-tls -o yaml
```

您也可以使用下列命令解碼和檢查憑證：

```
kubectl get secret example-certificate-tls -o jsonpath='{.data.tls\.crt}' | base64 -d |
 openssl x509 -text -noout
```

範例

下列範例示範 AWS 私有 CA 如何與 Kubernetes 叢集搭配使用。

- 範例：Kubernetes 的傳輸中加密
- 使用 AWS 私有 CA 和 Amazon EKS 啟用 TLS 的 Kubernetes 叢集
- 使用 new AWS Load Balancer 控制器在 Amazon EKS 上設定end-to-end TLS 加密

使用 監控 Kubernetes AWS 私有 CA

若要監控 Kubernetes 叢集私有 CA，請使用 中概述的技術監控 AWS 私有 CA 資源。您可以使用下列項目來監控私有 CA：

- AWS 私有 CA CloudWatch 指標

- AWS 私有 CA 使用 CloudWatch Events 監控
- 使用 記錄 AWS 私有憑證授權單位 API 呼叫 AWS CloudTrail

使用 對 Kubernetes 進行故障診斷 AWS 私有 CA

您可以使用下列程序取得 aws-private-ca-issuer的日誌：

1. 取得 Pod 的名稱：

```
kubectl get pods -A
```

2. 若要檢視發行者日誌，請使用下列命令：

```
kubectl logs -n aws-privateca-issuer <pod-name> aws-privateca-issuer
```

3. 若要檢視 IAM Roles Anywhere 日誌，請使用下列命令：

```
kubectl logs -n aws-privateca-issuer <pod-name> rolesanywhere-credentials-helper
```

若要檢查 AWS 私有 CA 發行者的狀態，請使用下列其中一項：

若要檢查您的發行者是否已準備就緒，請使用下列命令：

```
kubectl get AWSPCAClusterIssuers -o json | jq '.items[].status
```

回應應類似於以下內容：

```
{
  "conditions": [
    {
      "lastTransitionTime": "2024-07-03T13:56:37Z",
      "message": "Issuer verified",
      "reason": "Verified",
      "status": "True",
      "type": "Ready"
    }
  ]
}
```

如果發行者未處於 Ready 狀態，message 欄位會提供發行者為何無法達到 Ready 狀態的資訊。

若要檢查您的憑證是否已就緒，請使用下列命令：

```
kubectl get certificates -o json | jq '.items[].status'
```

回應應類似於以下內容：

```
{
  "conditions": [
    {
      "lastTransitionTime": "2024-07-03T13:58:13Z",
      "message": "Certificate is up to date and has not expired",
      "observedGeneration": 1,
      "reason": "Ready",
      "status": "True",
      "type": "Ready"
    }
  ],
  "notAfter": "2024-10-01T13:58:12Z",
  "notBefore": "2024-07-03T12:58:12Z",
  "renewalTime": "2024-09-16T13:58:12Z",
  "revision": 1
}
```

如果憑證未處於 Ready 狀態，message 欄位會提供憑證無法達到 Ready 狀態的原因資訊。

AWS 私有 CA 適用於 Active Directory 的連接器

AWS 私有 CA 可以發行和管理 所需的憑證 AWS Managed Microsoft AD。使用 AWS 私有 CA Connector for Active Directory (Connector for AD)，您可以將內部部署企業或其他第三方 CAs 取代為您擁有的受管私有 CA，為 AD 管理的使用者、群組和機器提供憑證註冊。

您可以使用 Connector for AD 搭配 AWS Managed Microsoft AD 來消除內部部署基礎設施，方法是將 AD 和公有金鑰基礎設施遷移至雲端。對於希望 AWS 私有 CA 搭配現場部署 AD 使用 的客戶，此功能也會與 AWS Managed Microsoft AD Connector 整合。

主題

- 您是 AD 使用者的第一次連接器嗎？
- 設定適用於 AD 的 連接器
- Connector AWS 私有 CA for Active Directory 入門
- AWS 私有 CA 適用於 Active Directory 的連接器
- 使用 Amazon EventBridge 將 Connector for AD 整合到事件驅動型應用程式
- Connector AWS 私有 CA for Active Directory 的問題故障診斷

您是 AD 使用者的第一次連接器嗎？

如果您是第一次使用 Connector for AD，建議您先閱讀以下章節：

- 什麼是 AWS 私有 CA？
- 什麼是 Directory Service？

適用於 AD 的 Access Connector

您可以透過主控台 AWS CLI和 APIs 存取 Connector for AD。您可以從主控台 AWS 私有 CA 、主控台或在搜尋 AWS 管理主控台 列中搜尋 Connector for AD 來存取 Directory Service 主控台中的連接器。

定價

Connector for AD 是 的一項功能 AWS 私有 CA ，無需額外費用。您只需為私有憑證授權單位和您透過私有憑證授權單位發行的憑證付費。

如需最新的 AWS 私有 CA 定價資訊，請參閱 AWS 私有憑證授權單位 定價。您也可以使用AWS 定價計算器來預估成本。

設定適用於 AD 的 連接器

本節中的步驟是使用 Connector for AD 的先決條件。其假設您已建立 AWS 帳戶。完成此頁面上的步驟後，您就可以開始為 AD 建立連接器。

步驟 1：使用 建立私有 CA AWS 私有 CA

設定私有憑證授權機構 (CA)，以將憑證發行到您的目錄物件。如需詳細資訊，請參閱中的憑證授權單位 AWS 私有 CA。

私有 CA 必須處於 `Active` 狀態，才能建立 Connector for AD。私有 CA 的主旨名稱必須包含通用名稱。如果您嘗試使用沒有通用名稱的私有 CA 建立連接器，連接器建立將會失敗。

步驟 2：設定 Active Directory

除了私有 CA 之外，您需要虛擬私有雲端 (VPC) 中的作用中目錄。Connector for AD 支援下列 提供的目錄類型 Directory Service：

- AWS 受管 Microsoft Active Directory：使用 Directory Service ，您可以執行 Microsoft Active Directory (AD) 做為受管服務。 AWS Directory Service for Microsoft Active Directory 也稱為 AWS Managed Microsoft AD，採用 Windows Server 2019 技術。透過 AWS Managed Microsoft AD，您可以在 中執行目錄感知工作負載 AWS 雲端，包括 Microsoft Sharepoint 和自訂 .Net 和 SQL Server 型應用程式。
- Active Directory Connector：AD Connector 是一種目錄閘道，可將目錄請求重新導向至您的內部部署 Microsoft Active Directory，而無需快取雲端中的任何資訊。AD Connector 支援連線至 Amazon EC2 上託管的網域

(僅限 Active Directory Connector) 步驟 3：將許可委派給服務帳戶

> **Note**
>
> 如果您使用額外的許可 AWS Managed Microsoft AD ，當您使用目錄授權 Connector for AD 服務時，會自動委派。您可以略過此先決條件步驟。

使用 Directory Service AD Connector 時，您需要將其他許可委派給服務帳戶。在服務帳戶上設定存取控制清單 (ACL)，以允許 功能：

- 新增和移除服務主體名稱 (SPN) 至其本身
- 在以下容器中建立並更新憑證授權機構：

```
#containers
CN=Public Key Services,CN=Services,CN=Configuration
CN=AIA,CN=Public Key Services,CN=Services,CN=Configuration
CN=Certification Authorities,CN=Public Key Services,CN=Services,CN=Configuration
```

- 建立和更新 NTAuthCertificates 憑證授權機構 (CA) 物件。注意：如果 NTAuthCertificates CA 物件存在，則必須委派其許可。如果物件不存在，則必須委派在公有金鑰服務容器上建立子物件的能力。

```
#objects
CN=NTAuthCertificates,CN=Public Key Services,CN=Services,CN=Configuration
```

官方 Connector for Active Directory 儲存庫中可用的 PowerShell 指令碼可用來委派 Directory Service AD Connector 服務帳戶所需的其他許可。

此指令碼會建立 NTAuthCertificates 憑證授權單位物件。

如需最新版本的指令碼和用量詳細資訊，請參閱 GitHub 儲存庫中的 README。

步驟 4：建立 IAM 政策

若要建立 AD 連接器，您需要 IAM 政策，可讓您建立連接器資源、與 Connector for AD 服務共用私有 CA，以及使用目錄授權 Connector for AD 服務。

這是使用者受管政策的範例：

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Action": "pca-connector-ad:*",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "acm-pca:DescribeCertificateAuthority",
                "acm-pca:GetCertificate",
                "acm-pca:GetCertificateAuthorityCertificate",
                "acm-pca:ListCertificateAuthorities",
                "acm-pca:ListTags",
                "acm-pca:PutPolicy"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "acm-pca:IssueCertificate",
            "Resource": "*",
            "Condition": {
                "ArnLike": {
                    "acm-pca:TemplateArn": "arn:aws:acm-pca:::template/
BlankEndEntityCertificate_APIPassthrough/V*"
                },
                "ForAnyValue:StringEquals": {
                    "aws:CalledVia": "pca-connector-ad.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "ds:AuthorizeApplication",
                "ds:DescribeDirectories",
                "ds:ListTagsForResource",
                "ds:UnauthorizeApplication",
                "ds:UpdateAuthorizedApplication"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateVpcEndpoint",
```

```
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcEndpoints",
                "ec2:DescribeVpcs",
                "ec2:DeleteVpcEndpoints"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeTags",
                "ec2:DeleteTags",
                "ec2:CreateTags"
            ],
            "Resource": "arn:*:ec2:*:*:vpc-endpoint/*"
        }
    ]
}
```

Connector for AD 需要額外的 AWS RAM 許可，才能使用主控台和命令列。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "ram:CreateResourceShare",
            "Resource": "*",
            "Condition": {
                "StringEqualsIfExists": {
                    "ram:Principal": "pca-connector-ad.amazonaws.com",
                    "ram:RequestedResourceType": "acm-pca:CertificateAuthority"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "ram:GetResourcePolicies",
```

```
                "ram:GetResourceShareAssociations",
                "ram:GetResourceShares",
                "ram:ListPrincipals",
                "ram:ListResources",
                "ram:ListResourceSharePermissions",
                "ram:ListResourceTypes"
            ],
            "Resource": "*"
        }
    ]
}
```

步驟 5：與 Connector for AD 共用您的私有 CA

您需要使用 AWS Resource Access Manager 服務主體共用，與連接器服務共用私有 CA。

當您在 AWS 主控台中建立連接器時，會自動為您建立資源共享。

當您使用 建立資源共享時 AWS CLI，您將使用 AWS RAM create-resource-share命令。

下列命令會建立資源共享：

```
$  aws ram create-resource-share \
    --region us-east-1 \
    --name MyPcaConnectorAdResourceShare \
    --permission-arns arn:aws:ram::aws:permission/
AWSRAMBlankEndEntityCertificateAPIPassthroughIssuanceCertificateAuthority \
    --resource-arns arn:aws:acm-pca:region:account:certificate-authority/CA_ID \
    --principals pca-connector-ad.amazonaws.com \
    --sources account
```

呼叫 CreateConnector 的服務主體在 PCA 上具有憑證發行許可。若要防止使用 Connector for AD 的服務主體擁有對 AWS 私有 CA 資源的一般存取權，請使用 限制其許可`CalledVia`。

步驟 6：建立目錄註冊

您可以使用您的目錄授權 Connector for AD 服務，以便連接器可以與您的目錄通訊。若要授權 Connector for AD 服務，您可以建立目錄註冊。如需建立目錄註冊的詳細資訊，請參閱 管理目錄註冊

步驟 7：設定安全群組

您的 VPC 與 Connector for AD 連接器之間的通訊是透過（需要一個安全群組） AWS PrivateLink，其中包含在您的 VPC 上開啟連接埠 443 TCP 的傳入規則。當您建立連接器時，系統會要求您輸入此安全群組。您可以將來源指定為自訂，然後選取 VPC 的 CIDR 區塊。您可以選擇進一步限制（即 IP、CIDR 和安全群組 ID)。

步驟 8：設定目錄物件的網路存取

目錄物件需要公有網際網路存取權，才能從下列網域驗證線上憑證狀態通訊協定 (OCSP) 和憑證撤銷清單 (CRLs)：

```
*.windowsupdate.com
*.amazontrust.com
```

最低必要存取規則：

- OCSP 和 CRL 通訊的必要項目：

  ```
  TCP 80: (HTTP) to 0.0.0.0/0
  ```

- Connector for AD 的必要項目：

  ```
  TCP 443: (HTTPS) to 0.0.0.0/0
  ```

- Active Directory 的必要項目：

  ```
  TCP 88: (Kerberos) to Domain Controller IP range
  TCP/UDP 389/636: (LDAP/LDAPS) to Domain Controller IP range, depending on Domain
   Controller configuration
  TCP/UDP 53: (DNS) to 0.0.0.0/0
  ```

如果裝置沒有公有網際網路存取，憑證發行會間歇性失敗並顯示錯誤碼 `WS_E_OPERATION_TIMED_OUT`.

> **Note**
>
> 如果您要為 Amazon EC2 執行個體設定安全群組，則不需要與步驟 7 中的安全群組相同。

Connector AWS 私有 CA for Active Directory 入門

使用 AWS 私有 CA Connector for Active Directory，您可以從私有 CA 向 Active Directory 物件發行憑證，以進行身分驗證和加密。當您建立連接器時， 會在 VPC 中為您 AWS 私有憑證授權單位 建立端點，讓目錄物件請求憑證。

若要發行憑證，您可以為連接器建立連接器和 AD 相容範本。建立範本時，您可以設定 AD 群組的註冊許可。

主題

- 開始之前
- 步驟 1：建立連接器
- 步驟 2：設定 Microsoft Active Directory 政策
- 步驟 3：建立範本
- 步驟 4：設定 Microsoft 群組許可

開始之前

下列教學課程會引導您完成建立 AD 連接器和連接器範本的程序。若要遵循本教學課程，您必須先滿足區段中列出的先決條件。

步驟 1：建立連接器

若要建立連接器，請參閱 為 Active Directory 建立連接器。

步驟 2：設定 Microsoft Active Directory 政策

Connector for AD 無法檢視或管理客戶的群組政策物件 (GPO) 組態。GPO 控制 AD 請求路由到客戶的 AWS 私有 CA 或其他身分驗證或憑證販賣伺服器。無效的 GPO 組態可能會導致您的請求路由不正確。客戶可以自行設定和測試 Connector for AD 組態。

群組政策與 Connector 相關聯，您可以選擇為單一 AD 建立多個 Connector。如果每個連接器的群組政策組態不同，您可以自行管理其存取控制。

資料平面呼叫的安全性取決於 Kerberos 和 VPC 組態。只要對對應的 AD 進行身分驗證，任何有權存取 VPC 的人都可以進行資料平面呼叫。這存在於 AWSAuth 的界限之外，而管理授權和身分驗證取決於您，也就是客戶。

使用 時 AWS Managed Microsoft AD，請使用 Directory Service enable-ca-enrollment-policy命令在 AWS Managed Microsoft AD 執行個體的網域控制器上設定 GPOs。

下列命令可啟用註冊網域控制站：

```
$  aws ds enable-ca-enrollment-policy \
    --pca-connector-arn MyPcaConnectorAdArn \
    --directory-id MyDirectoryId
```

使用 AD Connector 時，請使用下列步驟來建立指向建立連接器時所產生 URI 的 GPO。從主控台或命令列使用 Connector for AD 需要此步驟。

設定 GPOs。

1. 在 DC 上開啟 Server Manager
2. 前往工具，然後選擇主控台右上角的群組政策管理。
3. 移至樹系 > 網域。選取您的網域名稱，並在您的網域上按一下滑鼠右鍵。選取在此網域中建立 GPO，並將其連結到此處...，然後輸入 PCA GPO 做為名稱。
4. 新建立的 GPO 現在會列在您的網域名稱下。
5. 選擇 PCA GPO，然後選擇編輯。如果對話方塊開啟並顯示提醒訊息 這是連結，而該變更將全域傳播，請確認訊息以繼續。群組政策管理編輯器應開啟。
6. 在群組政策管理編輯器中，前往電腦組態 > 政策 > Windows 設定 > 安全設定 > 公有金鑰政策（選擇 資料夾）。
7. 前往物件類型，然後選擇憑證服務用戶端 - 憑證註冊政策
8. 在 選項中，將組態模型變更為已啟用。
9. 確認已勾選 Active Directory 註冊政策並已啟用。選擇新增。
10. Certificate Enrollment Policy Server 視窗應該會開啟。
11. 在輸入註冊伺服器政策 URI 欄位中輸入您建立連接器時產生的憑證註冊政策伺服器端點。
12. 將身分驗證類型保持為 Windows 整合狀態。
13. 選擇驗證。驗證成功後，選取新增。對話方塊會關閉。
14. 返回 Certificate Services 用戶端 - 憑證註冊政策，並勾選新建立連接器旁的核取方塊，以確保連接器是預設註冊政策
15. 選擇 Active Directory 註冊政策，然後選取移除。
16. 在確認對話方塊中，選擇是以刪除 LDAP 型身分驗證。

17. 在憑證服務用戶端 > 憑證註冊政策視窗中選擇套用並確定，然後關閉它。
18. 前往公有金鑰政策資料夾，然後選擇憑證服務用戶端 - 自動註冊。
19. 將組態模型選項變更為已啟用。
20. 確認已檢查續約過期憑證和更新憑證。讓其他設定保持不變。
21. 選擇套用，然後選擇確定，然後關閉對話方塊。

接下來設定使用者組態的公有金鑰政策。移至使用者組態 > 政策 > Windows 設定 > 安全設定 > 公有金鑰政策。請依照步驟 6 到步驟 21 概述的程序，設定使用者組態的公有金鑰政策。

完成設定 GPOs和公有金鑰政策後，網域中的物件會從 AWS 私有 CA Connector for AD 請求憑證，並取得 發行的憑證 AWS 私有 CA。

步驟 3：建立範本

若要建立範本，請參閱 建立連接器範本。

步驟 4：設定 Microsoft 群組許可

若要設定 Microsoft 群組許可，請參閱 Manage Connector for AD 範本存取控制項目。

AWS 私有 CA 適用於 Active Directory 的連接器

本節中的程序說明如何建立 Active Directory (AD) 連接器、設定範本，以及與 AWS 私有 CA 和 Active Directory 整合。您可以從 AWS 私有 CA Connector for AD 主控台執行這些操作，方法是使用 的 Connector for AD 區段 AWS CLI，或使用 AWS 私有 CA Connector for AD API。

> **Note**
>
> 雖然 AWS 私有 CA Connector for AD 與 緊密整合 AWS 私有 CA，但這兩個服務具有不同的 APIs。如需詳細資訊，請參閱 AWS 私有憑證授權單位 API 參考和 AWS 私有 CA Connector for Active Directory API 參考。

為 Active Directory 建立連接器

使用下列程序，使用主控台、命令列或 API for Connector for Active Directory 建立 AWS 私有 CA 連接器。

Console

使用主控台建立連接器

登入 AWS 您的帳戶，並在 開啟 AWS 私有 CA Connector for Active Directory 主控台**https://console.aws.amazon.com/pca-connector-ad/home**。

1. 在第一次服務登陸頁面上或適用於 Active Directory 的連接器頁面上，選擇建立連接器。
2. 在建立 Active Directory 的私有 CA 連接器頁面上，提供 Active Directory 區段中的資訊。
   - 在選取 Active Directory 類型下，選擇兩種可用類型之一：
     - AWS Directory Service for Microsoft Active Directory – 指定由 管理的 Active Directory Directory Service。
     - 具有 AWS AD Connector 的現場部署 Active Directory – 使用 AD Connector 存取您內部部署託管的 Active Directory。
   - 在選取您的目錄下，從清單中選擇您的目錄。

     或者，您可以選擇建立目錄，這會在新視窗中開啟 Directory Service 主控台。當您完成建立新目錄時，請返回 AWS 私有 CA Connector for Active Directory 主控台並重新整理目錄清單。您的新目錄應該可供選取。

     > **Note**
     >
     > 建立目錄時，請注意 Connector for AD 僅支援主控台中 Directory Service 提供的下列目錄類型：
     >
     > - AWS Managed Microsoft AD
     > - AD Connector

   - 在選取 VPC 端點的安全群組下，從清單中選擇安全群組。

     或者，您可以選擇建立安全群組，在新視窗中將 Amazon EC2 主控台開啟至建立安全群組頁面。當您完成建立安全群組時，請返回 AWS 私有 CA Connector for Active Directory 主控台並重新整理安全群組清單。您的新安全群組應該可供選取。
3. 在 IP 地址類型區段中，從下列選項中選擇：
   - IPv4 - 啟用對服務的 IPv4 連線。只有在託管目錄的所有子網路都具有 IPv4 地址範圍時，才選擇此選項。

- Dualstack - 啟用對服務的 IPv4 和 IPv6 連線。只有在託管目錄的所有子網路同時具有 IPv4 和 IPv6 地址範圍時，才選擇此選項。

4. 在私有憑證授權單位區段中，從清單中選擇私有 CA。

   或者，您可以選擇建立私有 CA，在新視窗中將 AWS 私有 CA 主控台開啟私有憑證授權單位頁面。當您完成建立 CA 時，請返回 AWS 私有 CA Connector for Active Directory 主控台並重新整理 CAs清單。您的新 CA 應可供選取。

5. 在標籤 – 選用窗格中，您可以在 AD 資源上套用和移除中繼資料。標籤是索引鍵/值字串對，其中索引鍵對資源必須是唯一的，而值是選用的。此窗格會顯示資料表中資源的任何現有標籤。支援以下動作。

   - 選擇管理標籤以開啟管理標籤頁面。
   - 選擇新增標籤以建立標籤。填寫金鑰欄位，並選擇性地填寫值欄位。選擇儲存變更以套用標籤。
   - 選擇標籤旁的移除按鈕以將其標記為刪除，然後選擇儲存變更以確認。

6. 在提供必要資訊並檢閱您的選擇之後，請選擇建立連接器。這會開啟 Connectors for Active Directory 詳細資訊頁面，您可以在其中檢視連接器建立時的進度。

   建立連接器的程序完成後，請為其指派服務主體名稱。

API

使用 API 建立連接器

若要使用 API 建立 Active Directory 的連接器，請使用 Connector for Active Directory API 中的 CreateConnector AWS 私有 CA 動作。

CLI

使用 建立連接器 AWS CLI

若要使用 CLI 建立 Active Directory 的連接器，請使用 的 AWS 私有 CA Connector for Active Directory 區段中的 create-connector 命令 AWS CLI。

建立連接器範本

範本是憑證應如何發出，以及用戶端應如何處理憑證的組態清單。下列程序說明如何建立範本。

Console

使用主控台建立範本

1. 登入 AWS 您的帳戶，並在 開啟 AWS 私有 CA Connector for Active Directory 主控台https://console.aws.amazon.com/pca-connector-ad/home 。
2. 從 Connectors for Active Directory 清單中選擇連接器，然後選擇檢視詳細資訊。
3. 在連接器的詳細資訊頁面上，尋找範本區段，然後選擇建立範本。
4. 在建立範本頁面上的範本建立方法區段中，選擇其中一個方法選項。
   - 從預先定義的範本開始 （預設） – 從 AD 應用程式的預先定義範本清單中選擇：
     - 程式碼簽署
     - 電腦
     - 網域控制站身分驗證
     - EFS 復原代理程式
     - 註冊代理程式
     - 註冊代理程式 （電腦）
     - IPSec
     - Kerberos 身分驗證
     - RAS 和 IAS 伺服器
     - 智慧卡登入
     - 信任清單簽署
     - 使用者簽章
     - 工作站身分驗證
   - 從您建立的現有範本開始 – 從您先前建立的自訂範本清單中選擇。
   - 從空白範本開始 – 選擇此選項以開始建立全新的範本。
5. 在憑證設定區段中，根據此範本定義憑證的下列設定。
   - 憑證類型 – 指定要建立使用者或電腦憑證。
   - 自動註冊 – 選擇是否根據此範本啟用憑證的自動註冊。
   - 有效期間 – 將憑證有效期間指定為小時、天、週、月或年的整數值。最小值為 2 小時。
   - 續約期間 – 將憑證續約期間指定為小時、天、週、月或年的整數值。續約期間不得超過有效期間的 75%。

- 主旨名稱 – 根據 Active Directory 中包含的資訊，選擇要包含在主旨名稱中的一或多個選項。

  > **Note**
  >
  > 至少必須指定一個主體名稱或主體替代名稱選項。

  - 一般名稱
  - DNS 做為一般名稱
  - 目錄路徑
  - 電子郵件
- 主體替代名稱 – 根據 Active Directory 中包含的資訊，選擇要包含在主體替代名稱中的一或多個選項。

  > **Note**
  >
  > 至少必須指定一個主體名稱或主體替代名稱選項。

  - 目錄 GUID
  - DNS 名稱
  - 網域 DNS
  - 電子郵件
  - 服務主體名稱 (SPN)
  - 使用者主體名稱 (UPN)

6. 在憑證請求處理和註冊選項區段中，根據範本指定憑證的用途，選擇下列其中一個選項。

   - Signature
   - 加密
   - 簽章和加密
   - 簽章和智慧卡登入

   接著，選擇要啟用下列哪些功能。選項會根據憑證用途而有所不同。

- 刪除無效的憑證（請勿封存）
- 包含對稱演算法
- 可匯出的私有金鑰

最後，選擇憑證註冊選項。選項會根據憑證用途而有所不同。

- 不需要使用者輸入
- 註冊期間提示使用者
- 在註冊期間提示使用者，並要求使用者輸入

7. 在應用程式政策區段中，選擇所有適用的應用程式政策。可用的政策會跨多個頁面列出。某些政策可能會因為先前的設定而預先選取。
8. 在自訂應用程式政策區段中，您可以將自訂 OIDs 新增至範本，並指定應用程式政策延伸是否重要。
9. 在密碼編譯設定區段中，根據此範本選擇憑證的下列密碼編譯設定類別。
10. 在群組和許可區段中，您可以檢視範本現有的群組和註冊許可，也可以選擇新增群組和許可按鈕來新增群組和許可。按鈕會開啟需要下列資訊的表單：

    - 顯示名稱
    - 安全識別符 (SID)
    - 使用允許選項註冊 | 拒絕 | 未設定
    - 自動註冊，含允許選項 | 拒絕 | 未設定

11. 在取代範本區段中，您可以通知 Active Directory 目前的範本會取代 AD 中建立的一或多個範本。套用疊代範本，方法是從 Active Directory 選擇新增範本以取代並指定疊代範本的通用名稱。
12. 在標籤 – 選用窗格中，您可以在 AD 資源上套用和移除中繼資料。標籤是索引鍵/值字串對，其中索引鍵對資源必須是唯一的，而值是選用的。此窗格會顯示資料表中資源的任何現有標籤。支援以下動作。

    - 選擇管理標籤以開啟管理標籤頁面。
    - 選擇新增標籤以建立標籤。填寫金鑰欄位，並選擇性地填寫值欄位。選擇儲存變更以套用標籤。
    - 選擇標籤旁的移除按鈕以將其標記為刪除，然後選擇儲存變更以確認。

13. 在提供必要資訊並檢閱您的選擇之後，請選擇建立範本。這會開啟範本詳細資訊，您可以在其中檢閱新範本的設定、編輯或刪除範本、管理群組和許可、管理取代的範本、管理標籤，以及為憑證持有者設定自動重新註冊。

API

使用 API 建立連接器範本

使用 AWS 私有 CA Connector for Active Directory API 中的 CreateTemplate 動作。

CLI

使用 建立連接器範本 AWS CLI

在 的 AWS 私有 CA Connector for Active Directory 區段中使用 create-template 命令 AWS CLI。

更新 Active Directory 的範本

使用下列程序，使用主控台、命令列或 AWS 私有 CA 適用於 Connector for Active Directory 的 API 來更新範本。

Console

使用主控台更新範本

登入 AWS 您的帳戶，並在 開啟 AWS 私有 CA Connector for Active Directory 主控台**https://console.aws.amazon.com/pca-connector-ad/home**。

1. 在 Connectors for Active Directory 清單中，選取您要更新其範本的連接器。選擇編輯以檢視和修改連接器的範本。
2. 在連接器的範本詳細資訊頁面中，選擇編輯。依照提示進行更新。編輯區域完成後，請選擇儲存以儲存變更。

API

使用 API 更新範本

若要使用 API 更新 Active Directory 的範本，請使用 AWS 私有 CA Connector for Active Directory API 中的 UpdateTemplate 動作。

CLI

使用 更新範本 AWS CLI

若要使用 CLI 更新 Active Directory 的連接器，請使用 的 AWS 私有 CA Connector for Active Directory 區段中的 update-template 命令 AWS CLI。

Connector for Active Directory 如何傳播您的範本變更

AWS 私有 CA 當您的用戶端重新整理政策快取時， 會將範本套用至您的政策，也就是每 8 小時一次。這包括範本群組存取控制項目的變更。當您的用戶端重新整理快取時，它會查詢連接器是否有可用的範本。在自動註冊重新整理的情況下，用戶端會發出符合下列其中一個或兩個條件的憑證：

- 憑證在續約期間內。
- 憑證不存在於用戶端裝置上。

若要手動重新整理，用戶端會查詢連接器，而且您必須將範本設定為發行。

如果您要偵錯，您可以手動清除政策快取，以立即查看範本變更。若要這樣做，請在用戶端上執行下列 Powershell 命令。

```
certutil -f -user -policyserver * -policycache delete
```

列出 Active Directory 的連接器

您可以使用 AWS 私有 CA Connector for Active Directory 主控台或 AWS CLI 列出您擁有的連接器。

Console

使用主控台列出您的連接器

1. 登入 AWS 您的帳戶，並在 開啟 AWS 私有 CA Connector for Active Directory 主控台**https://console.aws.amazon.com/pca-connector-ad/home**。
2. 檢閱 Connectors for Active Directory 清單中的資訊。您可以使用右上角的頁碼瀏覽多個頁面的連接器。根據預設，每個連接器都會佔用顯示下列資料欄的資料列。

- 連接器 ID – 連接器的唯一 ID。

- 目錄名稱 – 與連接器相關聯的 Active Directory 資源。
- 連接器狀態 – 連接器狀態。可能的值為：建立 | 作用中 | 刪除 | 失敗。
- 服務主體名稱狀態 – 與連接器相關聯的服務主體名稱 (SPN) 狀態。可能的值為：建立 | 作用中 | 刪除 | 失敗。
- 目錄註冊狀態 – 關聯主管的註冊狀態。可能的值為：建立 | 作用中 | 刪除 | 失敗。
- 建立時間：連接器建立時的時間戳記。

透過選擇主控台右上角的齒輪圖示，您可以使用頁面大小偏好設定來自訂頁面上顯示的連接器數量。

API

使用 API 列出連接器

使用 AWS 私有 CA Connector for Active Directory API 中的 ListConnectors 動作。

CLI

使用 列出您的連接器 AWS CLI

使用 list-connectors 命令列出您的連接器。

列出連接器範本

您可以使用 AWS 私有 CA Connector for Active Directory 主控台或 AWS CLI 列出您擁有之連接器的範本。連接器範本是以 AWS 私有 CA BlankEndEntityCertificate_APIPassthrough/V1 範本為基礎。

Console

使用主控台列出您的範本

1. 登入 AWS 您的帳戶，並在 開啟 AWS 私有 CA Connector for Active Directory 主控台**https://console.aws.amazon.com/pca-connector-ad/home**。
2. 從 Connectors for Active Directory 清單中選擇連接器，然後選擇檢視詳細資訊。
3. 在連接器詳細資訊頁面上，檢閱範本區段中的資訊。您可以使用右上角的頁碼瀏覽多頁範本。每個範本都會佔用一列，顯示下列資料欄的資訊。

- 範本名稱 – 範本的人類可讀取名稱。

- 範本狀態 – 範本的狀態。可能的值為：作用中 | 正在刪除。
- 範本 ID – 範本的唯一識別符。

API

使用 API 列出您的連接器

使用 AWS 私有 CA Connector for Active Directory API 中的 ListTemplates 動作來列出指定連接器的範本。

CLI

使用 列出您的連接器 AWS CLI

使用 list-templates 命令列出指定連接器的範本。

檢視連接器詳細資訊

使用下列程序在主控台、命令列或 API for Connector for Active Directory 中檢視 AWS 私有 CA 連接器的組態詳細資訊。

Console

使用主控台檢視連接器的詳細資訊

1. 登入 AWS 您的帳戶，並在 開啟 AWS 私有 CA Connector for Active Directory 主控台**https://console.aws.amazon.com/pca-connector-ad/home**。
2. 從 Connectors for Active Directory 清單中選擇連接器，然後選擇檢視詳細資訊。
3. 在連接器詳細資訊頁面上，檢閱連接器詳細資訊窗格中的資訊，其中包含下列項目：
   - 連接器 ID
   - 連接器狀態
   - 其他狀態詳細資訊
   - 連接器 ARN
   - 憑證註冊政策伺服器端點
   - 目錄名稱
   - 目錄 ID

- AWS 私有 CA 主旨
- AWS 私有 CA status
- IP 地址類型
- VPC 端點和安全群組

4. 在範本窗格中，您可以建立或管理與連接器相關聯的範本。
5. 從服務主體名稱 (SPN) 窗格中，您可以檢視與連接器相關聯的服務原則名稱。
6. 從目錄註冊窗格中，您可以檢視或變更與連接器相關聯的目錄註冊。
7. 從標籤 — 選用窗格中，您可以建立或管理與連接器相關聯的標籤。

API

使用 API 列出連接器

使用 AWS 私有 CA Connector for Active Directory API 中的 GetConnector 動作。

CLI

使用 列出您的連接器 AWS CLI

在 的 AWS 私有 CA Connector for Active Directory 區段中使用 get-connector 命令 AWS CLI。

檢視連接器範本詳細資訊

使用以下程序，使用主控台、命令列或適用於 Connector for Active Directory 的 API 檢視 AWS 私有 CA 連接器範本的組態詳細資訊

Console

使用主控台檢視連接器範本的詳細資訊

1. 登入 AWS 您的帳戶，並在 開啟 AWS 私有 CA Connector for Active Directory 主控台**https://console.aws.amazon.com/pca-connector-ad/home**。
2. 從 Connectors for Active Directory 清單中選擇連接器，然後選擇檢視詳細資訊。
3. 在連接器詳細資訊頁面上，檢閱範本區段中的資訊，然後選取您要檢查的範本。然後選擇檢視詳細資訊。
4. 在詳細資訊頁面上，範本詳細資訊窗格會顯示範本的下列相關資訊：

- 範本名稱
- 範本 ID
- 範本狀態
- 範本結構描述版本
- 範本版本
- 範本 ARN
- 憑證類型
- 自動註冊已開啟
- 有效期間
- 續約期間
- 主旨名稱要求
- 主體替代名稱要求
- 憑證請求和註冊設定
- 密碼編譯提供者類別
- 金鑰演算法
- 金鑰大小下限（位元）
- 雜湊演算法
- 密碼編譯供應商
- 金鑰用量延伸設定

在此窗格中，您也可以使用編輯、刪除和動作按鈕執行下列動作。

- 編輯
- 刪除
- 管理群組和許可 – 如需詳細資訊，請參閱設定群組和許可。
- 管理取代的範本 – 如需詳細資訊，請參閱檢閱和建立。
- 管理標籤 – 如需詳細資訊，請參閱 標記 Connector for AD 資源。
- 重新註冊所有憑證持有者 – 此設定允許自動增加範本的主要版本。允許使用範本註冊的 Active Directory 群組的所有成員都會收到使用該範本發行的新憑證。如需詳細資訊，請參閱 UpdateTemplate API。

5. 下方窗格會顯示一列標籤，允許變更範本的組態。

- 群組和許可 – 檢視和管理 Active Directory 群組使用此範本註冊憑證的許可。如需詳細資訊，請參閱設定群組和許可
- 應用程式政策 – 檢視和管理範本應用程式政策。如需詳細資訊，請參閱指派應用程式政策。
- 取代的範本 – 檢視和管理取代的範本。如需詳細資訊，請參閱檢閱和建立。
- 標籤選用 – 檢視和管理此範本上的標記。如需詳細資訊，請參閱標記 Connector for AD 資源。

API

使用 API 列出您的連接器

使用 AWS 私有 CA Connector for Active Directory API 中的 GetTemplate 動作。

CLI

使用 列出您的連接器 AWS CLI

在 的 AWS 私有 CA Connector for Active Directory 區段中使用 get-template 命令 AWS CLI。

管理目錄註冊

Console

使用主控台管理目錄註冊

連接器的目錄註冊可以從 AWS 私有 CA Connector for Active Directory 主控台的最上層進行管理。本主題會逐步解說可用的管理選項。

1. 登入 AWS 您的帳戶，並在 開啟 AWS 私有 CA Connector for Active Directory 主控台`https://console.aws.amazon.com/pca-connector-ad/home`。
2. 在左側導覽區域中，選擇目錄註冊。
3. 目錄註冊頁面會顯示已註冊目錄的資料表，其中包含下列欄位：
   - 目錄 ID – 目錄的唯一 ID
   - 目錄名稱 – 目錄網域網站名稱
   - 目錄類型
   - 已註冊 – 註冊的狀態。支援的值正在建立 | 作用中 | 刪除 | 失敗。

- 目錄狀態 – 目錄的狀態

使用 可以使用註冊目錄來建立新的註冊。

4. 您可以選取其中一個列出的註冊來管理它。這會啟用檢視註冊詳細資訊和取消註冊目錄按鈕。檢視註冊詳細資訊按鈕會開啟註冊的詳細資訊頁面。
5. 目錄註冊詳細資訊窗格會顯示下列資訊：
   - 目錄網域網站名稱
   - 目錄 ID – 目錄的唯一 ID。選擇連結會帶您前往 AWS Directory Service 主控台。
   - 目錄類型
   - 狀態 – 目錄的狀態
   - 目錄註冊 ARN – 目錄註冊的 Amazon 資源名稱
   - 其他狀態資訊
6. 在連接器和服務主體名稱 (SPNs窗格中，您可以管理連接器SPNs。如需詳細資訊，請參閱檢視連接器詳細資訊。
7. 在標籤 – 選用窗格中，您可以在 AD 資源上套用和移除中繼資料。標籤是索引鍵/值字串對，其中索引鍵對資源必須是唯一的，而值是選用的。窗格會顯示資料表中資源的任何現有標籤。支援以下動作。
   - 選擇管理標籤以開啟管理標籤頁面。
   - 選擇新增標籤以建立標籤。填寫金鑰欄位，並選擇性地填寫值欄位。選擇儲存變更以套用標籤。
   - 選擇標籤旁的移除按鈕以將其標記為刪除，然後選擇儲存變更以確認。

API

使用 API 管理目錄註冊

在 AWS 私有 CA Connector for Active Directory API 中建立：CreateDirectoryRegistration 動作。

擷取： AWS 私有 CA Connector for Active Directory API 中的 GetDirectoryRegistration 動作。

清單： AWS 私有 CA Connector for Active Directory API 中的 ListDirectoryRegistrations 動作。

刪除： AWS 私有 CA Connector for Active Directory API 中的 DeleteDirectoryRegistration 動作。

CLI

使用 CLI 管理目錄註冊

建立：使用 的 AWS 私有 CA Connector for Active Directory 區段中的 create-directory-registration 命令 AWS CLI。

擷取： 的 AWS 私有 CA Connector for Active Directory 區段中的 get-directory-registration 命令 AWS CLI。

清單： 的 AWS 私有 CA Connector for Active Directory 區段中的 list-directory-registrations 命令 AWS CLI。

刪除： 的 AWS 私有 CA Connector for Active Directory 區段中的 delete-directory-registration 命令 AWS CLI。

Manage Connector for AD 範本存取控制項目

存取控制項目會授予控制哪些 Active Directory 群組可以或不能註冊特定 Connector for AD 範本的憑證。當您可以在 Connector for AD 中建立或管理群組和許可時，您必須從 Active Directory 提供群組物件的安全識別符 (SID)。您可以使用下列 PowerShell 命令來取得 SID。如需 SIDs的相關資訊，請參閱 Microsoft Directory Domain Services 文件中的安全識別符運作方式。

```
$ Get-ADGroup -Identity "my_active_directory_group_name"
```

下列程序說明如何建立和管理 Connector for AD 範本存取群組項目。

Console

使用主控台管理範本群組許可

您可以從範本的詳細資訊頁面管理現有範本的群組和許可。如需詳細資訊，請參閱檢視連接器範本詳細資訊。

設定哪些群組可以或不能為特定範本註冊憑證的許可。您提供 群組的安全識別符 (SID)。然後，您可以設定 群組的註冊和自動註冊許可。對於自動註冊，註冊和自動註冊都必須設定為「允許」。

API

使用 API 管理範本群組許可

建立： AWS 私有 CA Connector for Active Directory API 中的 CreateTemplateGroupAccessControlEntry 動作。

更新： AWS 私有 CA Connector for Active Directory API 中的 UpdateTemplateGroupAccessControlEntry 動作。

擷取： AWS 私有 CA Connector for Active Directory API 中的 GetTemplateGroupAccessControlEntry 動作。

清單： AWS 私有 CA Connector for Active Directory API 中的 ListTemplateGroupAccessControlEntries 動作。

刪除： AWS 私有 CA Connector for Active Directory API 中的 DeleteTemplateGroupAccessControlEntry 動作。

CLI

使用 CLI 管理範本群組許可

在 的 AWS 私有 CA Connector for Active Directory 區段中建立：create-template-group-access-control-entry 命令 AWS CLI。

更新：在 的 AWS 私有 CA Connector for Active Directory 區段中的 update-template-group-access-control-entry 命令 AWS CLI。

擷取： 的 AWS 私有 CA Connector for Active Directory 區段中的 get-template-group-access-control-entry 命令 AWS CLI。

清單： 的 AWS 私有 CA Connector for Active Directory 區段中的 list-template-group-access-control-entries 命令 AWS CLI。

刪除：在 的 AWS 私有 CA Connector for Active Directory 區段中的 delete-template-group-access-control-entries 命令 AWS CLI。

設定服務主體名稱

了解如何設定連接器的服務主體名稱。

Console

使用主控台管理管理服務主體名稱

現有 AD 連接器的服務主體名稱 (SPN) 可以從連接器的詳細資訊頁面進行管理。如需詳細資訊，請參閱管理目錄註冊 檢視連接器詳細資訊

API

使用 API 管理服務主體名稱

在 AWS 私有 CA Connector for Active Directory API 中建立：CreateServicePrincipalName 動作。

擷取： AWS 私有 CA Connector for Active Directory API 中的 GetServicePrincipalName 動作。

清單： AWS 私有 CA Connector for Active Directory API 中的 ListServicePrincipalNames 動作。

刪除： AWS 私有 CA Connector for Active Directory API 中的 DeleteServicePrincipalName 動作。

CLI

使用 CLI 管理服務主體名稱

在 的 AWS 私有 CA Connector for Active Directory 區段中建立：create-service-principal-name 命令 AWS CLI。

擷取：《》中 AWS 私有 CA Connector for Active Directory》區段中的 get-service-principal-name 命令 AWS CLI。

清單： 的 AWS 私有 CA Connector for Active Directory 區段中的 list-service-principal-names 命令 AWS CLI。

刪除： 的 AWS 私有 CA Connector for Active Directory 區段中的 delete-service-principal-name 命令 AWS CLI。

標記 Connector for AD 資源

您可以將標籤套用至連接器、範本和目錄註冊。標記會將中繼資料新增至可協助組織和管理的資源。

Console

使用主控台管理資源標記

現有資源的標記會在資源的詳細資訊頁面上進行管理。如需詳細資訊，請參閱下列程序：

- 檢視連接器範本詳細資訊
- 管理目錄註冊

API

使用 API 管理資源標記

標籤： AWS 私有 CA Connector for Active Directory API 中的 TagResource 動作。

列出標籤： AWS 私有 CA Connector for Active Directory API 中的 ListTagsForResource 動作。

Untag： AWS 私有 CA Connector for Active Directory API 中的 UntagResource 動作。

重要 - 可以使用標籤來標記包含機密資料的物件。不過，標籤本身不應包含任何個人身分識別資訊 (PII)、敏感或機密資訊。

CLI

使用 CLI 管理資源標記

標籤： 的 AWS 私有 CA Connector for Active Directory 區段中的 tag-resource 命令 AWS CLI。

列出標籤： 的 AWS 私有 CA Connector for Active Directory 區段中的 list-tags-for-resource 命令 AWS CLI。

Untag： 的 AWS 私有 CA Connector for Active Directory 區段中的 untag-resource 命令 AWS CLI。

使用 Amazon EventBridge 將 Connector for AD 整合到事件驅動型應用程式

您可以將 Connector for AD 納入使用 Connector for AD 中發生的事件的事件驅動應用程式 (EDAs)，以在應用程式元件之間通訊並啟動下游程序。

例如，當您的帳戶發生下列 Connector for AD 事件時，您可以叫用其他 AWS 服務或自訂元件：

- 建立憑證或建立失敗時建立憑證。
- 憑證已註冊，或註冊失敗。

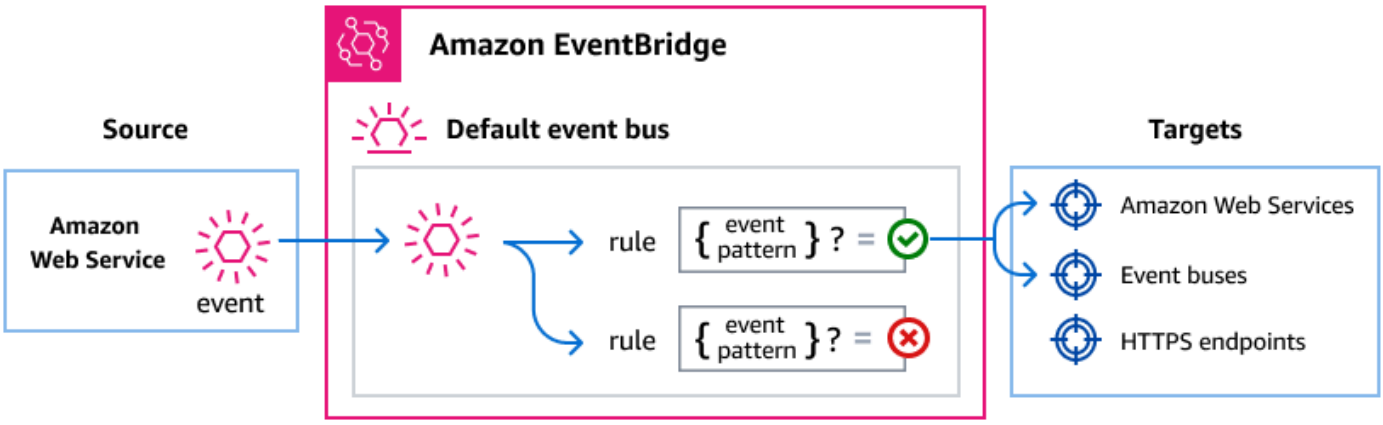
您可以使用 Amazon EventBridge 將事件從 Connector for AD 路由到其他軟體元件來執行此操作。Amazon EventBridge 是一種無伺服器服務，使用事件將應用程式元件連接在一起，讓您更輕鬆地將 Connector for AD 等 AWS 服務整合到事件驅動型架構中，而無需額外的程式碼和操作。

EventBridge 如何路由 Connector for AD 事件

以下是 EventBridge 與 Connector for AD 事件搭配使用的方式：

如同許多 AWS 服務，Connector for AD 會產生事件並傳送至 EventBridge 預設事件匯流排。事件匯流排是接收事件並將其路由到您指定的目的地或目標的路由器。目標可以包含其他服務 AWS 、自訂應用程式和 SaaS 合作夥伴應用程式。

EventBridge 會根據您在事件匯流排上建立的規則路由事件。對於每個規則，您可以指定篩選條件或事件模式，以僅選取您想要的事件。每當事件傳送到事件匯流排時，EventBridge 都會將其與每個規則進行比較。如果事件符合規則，EventBridge 會將事件路由到指定的目標 (s)。



適用於 AD 事件的連接器

如需傳送至 EventBridge 的 Connector for AD 事件清單，請參閱 EventBridge Events Reference 中的 Connector for AD 主題。

事件結構

AWS 服務的所有事件都包含兩種類型的資料：

- 包含事件中繼資料的常見欄位集，例如事件來源 AWS 的服務、產生事件的時間、事件發生的帳戶和區域，以及其他。如需這些一般欄位的定義，請參閱《Amazon EventBridge 事件參考》中的事件結構。

- 包含該特定服務事件特定資料`detail`的欄位。

建立符合 Connector for AD 事件的事件模式

事件模式是篩選條件，其中指定您要選取的事件應包含哪些資料。

每個事件模式都是 JSON 物件，它包含：

- 識別傳送事件之服務的 `source` 屬性。對於 Connector for AD 事件，來源為 `aws.pca-connector-ad`。
- （選用）：包含要比對之事件名稱陣列的`detail-type`屬性。
- (選擇性)：包含要比對的任何其他事件資料的 `detail` 屬性。

例如，下列事件模式會從 Connector for AD 選取所有憑證政策註冊成功事件：

```
{
  "source": ["aws.pca-connector-ad"],
  "detail-type": ["Certificate Policy Enrollment Succeeded"]
}
```

如需撰寫事件模式的詳細資訊，請參閱《EventBridge 使用者指南》中的事件模式。 EventBridge

從 EventBridge 接收事件

您可以指定 Connector for AD 憑證做為規則的目標。這可讓 Connector for AD 接收來自各種來源的事件，包括其他服務、 AWS 自訂應用程式和 SaaS 合作夥伴。如需詳細資訊，請參閱《EventBridge 使用者指南》中的建立對事件做出反應的規則。

如需可指定為目標 AWS 的服務完整清單，請參閱 EventBridge 事件參考中的目標類型。

Connector AWS 私有 CA for Active Directory 的問題故障診斷

使用此處的資訊來協助您診斷和修正 AWS 私有憑證授權單位 Connector for AD 問題。

主題

- Connector for AD 錯誤代碼故障診斷
- 針對適用於 AD 連接器建立失敗的 Connector 進行故障診斷
- 故障診斷 Connector for AD SPN 建立失敗

- 故障診斷 Connector for AD 範本更新問題

Connector for AD 錯誤代碼故障診斷

Connector for AD 會基於多種原因傳送錯誤訊息。如需每個錯誤的資訊以及解決這些錯誤的建議，請參閱下表。您可以透過訂閱 Amazon EventBridge 排程器事件 （事件來源：`aws.pca-connector-ad`) 或使用 Windows 中的手動註冊來接收這些錯誤。

錯誤碼	根本原因	修補
0x8FFFA000	Kerberos 身分驗證失敗。	請確定您的目錄是可存取的，且用戶端是使用者或電腦。如果您使用自動註冊，請修正您的 AWS 資源服務主體。如果您使用 Active Directory UI 取得憑證，請執行 `gpupdate /force`。
0x8FFFA001	SOAP 訊息必須包含動作標頭。	新增動作標頭。
0x8FFFA002	連接器無法存取其連線的私有 CA。	透過建立 AWS 資源存取管理員 (RAM)，在私有 CA 與 Connector for AD 服務之間共用，與連接器共用私有 CA。
0x8FFFA003	此連接器的私有 CA 未處於作用中狀態。	將私有 CA 移至作用中狀態。如果您的私有 CA 處於待定憑證狀態，請安裝 CA 憑證。
0x8FFFA004	此連接器的私有 CA 不存在。	如果憑證授權機構處於已刪除狀態，請將憑證授權機構移至作用中狀態。如果您的私有 CA 已永久刪除，請使用不同的 CA 建立新的連接器。
0x8FFFA005		

錯誤碼	根本原因	修補
	範本指定了憑證主體的directory Guid 屬性或主體替代名稱，但在請求者的 AD 物件中找不到屬性。	Active Directory 未directory Guid 為您的目錄產生 。Active Directory 中的故障診斷。
0x8FFFA006	範本指定了憑證主體的dnsHostName 屬性或主體替代名稱，但在請求者的 AD 物件中找不到屬性。	將 dnsHostName 屬性新增至 AD 物件。
0x8FFFA007	範本指定要包含在憑證主體或主體替代名稱中的電子郵件屬性，但在請求者的 AD 物件中找不到屬性。	將電子郵件屬性新增至 AD 物件
0x8FFFA008	SOAP 訊息必須具有 http://schemas.microsoft.com/windows/pki/2009/01/enrollmentpolicy/IPolicy/GetPolicies 或 的動作標頭http://schemas.microsoft.com/windows/pki/2009/01/enrollment/RST/wstep 。	更新動作標頭以使用其中一個指定的值。
0x8FFFA009	BinarySecurityToken 必須在 中編碼http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd#base64binary 。	更新二進位安全字符類型。
0x8FFFA00A	BinarySecurityToken 無效。	檢查 CSR 是否正確產生。

錯誤碼	根本原因	修補
0x8FFFA00B	BinarySecurityToken 的值類型必須為 `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd#PKCS7` 或 `http://schemas.microsoft.com/windows/pki/2009/01/enrollment#PKCS10` 。	將二進位安全字符值類型更新為有效值。
0x8FFFA00C	BinarySecurityToken 包含無效的 CMS。	Base64 有效，但密碼編譯訊息語法 (CMS) 無效。檢閱 CMS 語法。
0x8FFFA00D	BinarySecurityToken 包含無效的 CSR。	檢查 CSR 是否正確產生。
0x8FFFA00E	私有 CA 無法使用特定範本發行憑證。	從 檢閱驗證例外狀況 AWS 私有 CA。您可以在 Amazon EventBridge 或 中檢視驗證例外狀況 AWS CloudTrail。
0x8FFFA00F	SOAP 訊息的請求類型必須為 `http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue` 。	將請求類型設定為 `http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue` 。
0x8FFFA010	SOAP 訊息必須具有連接器 `CertificateEnrollmentPolicyServerEndpoint` 欄位或 XCEP 回應中 URI 欄位的 到 標頭。	將請求安全字符的標頭設定為 XCEP 回應中的 `CertificateEnrollmentPolicyServerEndpoint` 欄位或 URI 欄位。

錯誤碼	根本原因	修補
0x8FFFA011	SOAP 訊息只能有一個動作標頭。	檢閱請求安全字符的 SOAP 訊息標頭，並正確設定標頭。
0x8FFFA012	SOAP 訊息只能有一個messageId 標頭。	檢閱請求安全字符的 SOAP 訊息標頭，並正確設定標頭。
0x8FFFA013	SOAP 訊息只能有一個 到 標頭。	檢閱請求安全字符的 SOAP 訊息標頭，並正確設定標頭。
0x8FFFA014	請求者無法存取請求的範本。	透過建立存取控制項目，允許申請者的群組使用請求的範本註冊。
0x8FFFA015	CertificateTemplateInformation 或 CertificateTemplateName 延伸模組必須存在於 BinarySecurityToken 中。	將安全擴充功能新增至您的 CSR。
0x8FFFA016	找不到指定連接器的請求範本。	範本是每個連接器的子資源。使用 建立連接器的範本createTemplate 。
0x8FFFA017	由於請求調節，因此請求遭到拒絕。	降低請求率。
0x8FFFA018	SOAP 訊息必須包含 to 標頭。	檢閱 SOAP 訊息的標頭。
0x8FFFA019	由於無法辨識的標頭，無法處理 SOAP 訊息。	檢閱 SOAP 訊息的標頭。

錯誤碼	根本原因	修補
0x8FFFA01A	範本指定要包含在憑證主體或主體替代名稱中的 UPN 屬性，但在請求者的 AD 物件中找不到屬性。	將 UPN 新增至 Active Directory 物件。

針對適用於 AD 連接器建立失敗的 Connector 進行故障診斷

建立 AD 連接器的 連接器可能會因為各種原因而失敗。當連接器建立失敗時，您會在 API 回應中收到失敗原因。如果您使用的是 主控台，則失敗原因會顯示在 Connector 詳細資訊容器中 內的其他狀態詳細資訊欄位下的 Connector 詳細資訊頁面中。下表說明故障原因和建議的解決步驟。

失敗狀態	Description	修補
`CA_CERTIFICATE_REGISTRATION_FAILED`	Connector for AD 無法將 CA 憑證匯入您的目錄。	檢閱先決條件頁面，並檢查您的服務帳戶是否具有適當的許可。將正確的許可委派給您的服務帳戶後，請刪除失敗的連接器並建立新的連接器。如需有關委派許可的資訊，請參閱《AWS Directory Service 管理指南》中的將權限委派給您的服務帳戶。
`DIRECTORY_ACCESS_DENIED`	Connector for AD 無法存取您的目錄。	您必須授予 Connector for AD 存取您的目錄。檢閱 步驟 4：建立 IAM 政策區段，以確保您與 AWS 帳戶相關聯的 IAM 政策可讓您存取和描述目錄。將正確的許可授予您的 AWS 角色之後，請刪除失敗的連接器並建立新的連接器。 如果將 Connector for AD 與 AWS Directory Service AD

失敗狀態	Description	修補
		Connector 搭配使用，請確定 AD Connector 服務帳戶的密碼未過期且有效。如需 AD Connector 服務帳戶的相關資訊，請參閱 AD Connector 管理指南中的 AD Connector 入門。
INTERNAL_FAILURE	Connector for AD 發生內部故障。	請稍後再試。刪除失敗的連接器並建立新的連接器。
INSUFFICIENT_FREE_ADDRESSES	VPC 子網路必須至少有一個可用的私有 IP 地址。	確保子網路中有可用的私有 IP 地址。刪除失敗的連接器並建立新的連接器。
INVALID_SUBNET_IP_PROTOCOL	Connector for AD 無法在您的 VPC 上建立端點，因為與您的目錄相關聯的子網路不支援指定的 IP 地址類型。	確保託管目錄的 VPC 和子網路支援您選擇的 IP 地址類型。如需詳細資訊，請參閱 IP 位址類型。刪除失敗的連接器，並使用支援的 IP 地址類型建立新的連接器。

失敗狀態	Description	修補
PRIVATECA_ACCESS_DENIED	Connector for AD 無法存取您的私有 CA。	檢閱先決條件頁面，並檢查您是否具有建立連接器的許可。如需相關資訊，請參閱步驟 4：建立 IAM 政策。 如果您要透過 AWS CLI 或 API 建立連接器，請檢閱先決條件頁面，並檢查您是否已使用 Connector for AD 共用私有 CA AWS Resource Access Manager。 檢查並修正 IAM 許可 AWS RAM 和資源共用之後，請刪除失敗的連接器並建立新的連接器。
PRIVATECA_RESOURCE_NOT_FOUND	Connector for AD 找不到指定的私有 CA。	請確定您指定正確的私有 CA Amazon Resource Name (ARN)，然後刪除故障的連接器，並使用預期的私有 CA ARN 建立新的連接器。
SECURITY_GROUP_NOT_IN_VPC	安全群組不在託管目錄的 VPC 中。	使用託管目錄的 VPC 中的安全群組。如需詳細資訊，請參閱步驟 7：設定安全群組。刪除失敗的連接器，並使用 VPC 中的安全群組建立新的連接器。

失敗狀態	Description	修補
VPC_ACCESS_DENIED	Connector for AD 無法存取託管目錄的 Amazon VPC。	檢查您的 IAM 許可。刪除失敗的連接器並建立新的連接器。如需包含存取許可的範例 IAM 政策，請參閱 步驟 4：建立 IAM 政策
VPC_ENDPOINT_LIMIT_EXCEEDED	Connector for AD 無法在 Amazon VPC 中建立端點。您已達到您可以為帳戶建立的 VPC 端點限制。	刪除 Amazon VPC 端點，或請求提高限制。完成兩個步驟的其中一個後，請刪除失敗的連接器並建立新的連接器。如需配額的相關資訊，請參閱 Amazon Virtual Private Cloud Service 配額。
VPC_RESOURCE_NOT_FOUND	Connector for AD 找不到指定的 VPC。	請確定您已指定正確的 VPC，且 VPC 存在。然後刪除失敗的連接器，並使用正確的 VPC ID 建立新的連接器。

故障診斷 Connector for AD SPN 建立失敗

服務主體名稱 (SPN) 建立可能會因為各種原因而失敗。當 SPN 建立失敗時，您會在 API 回應中收到失敗原因。如果您使用的是 主控台，則失敗原因會顯示在服務主體名稱 (SPN) 容器內其他狀態詳細資訊欄位下的連接器詳細資訊頁面中。下表說明故障原因和建議的解決步驟。

失敗狀態	Description	修補
DIRECTORY_ACCESS_DENIED	Connector for AD 無法存取您的目錄。	授予 Connector for AD 存取您的目錄。如需包含授予目錄存取許可的 IAM 政策範例，請參閱 步驟 4：建立 IAM 政策。

失敗狀態	Description	修補
DIRECTORY_NOT_REACHABLE	Connector for AD 無法存取您的目錄。	檢查 AWS 和 目錄之間的網路，然後嘗試再次建立 SPN。
DIRECTORY_RESOURCE_NOT_FOUND	Connector for AD 找不到指定的目錄。	請務必指定正確的目錄 ID，然後刪除故障的連接器，並使用預期的目錄 ID 建立新的連接器。
INTERNAL_FAILURE	Connector for AD 發生內部故障。	請稍後再試。
SPN_EXISTS_ON_DIFFERENT_AD_OBJECT	服務主體名稱 (SPN) 存在於不同的 Active Directory 物件上。	從 Active Directory 物件刪除 SPN，然後再次嘗試建立 SPN。
SPN_LIMIT_EXCEEDED	Connector for AD 無法建立 SPN，因為您已達到每個目錄的 SPNs限制。每個目錄SPNs數目上限為 10。	從您的帳戶刪除一或多個 SPNs，然後再次嘗試建立 SPN。

故障診斷 Connector for AD 範本更新問題

如果您變更範本或群組存取控制項目，但未看到變更，這可能是由於政策快取。當用戶端重新整理政策快取時， 會將範本 AWS 私有 CA 套用至您的政策，也就是每 8 小時一次。當您的用戶端重新整理快取時，它會查詢連接器是否有可用的範本。在自動註冊重新整理的情況下，用戶端會發出符合下列其中一個或兩個條件的憑證：

- 憑證在續約期間內。
- 用戶端裝置上不存在憑證。

若要手動重新整理，用戶端會查詢連接器，而且您必須將範本設定為發行。

如果您要偵錯，您可以手動清除政策快取，以立即查看範本變更。若要這樣做，請在用戶端上執行下列 Powershell 命令。

```
certutil -f -user -policyserver * -policycache delete
```

AWS 私有 CA 適用於 SCEP 的連接器

Connector for Simple Certificate Enrollment Protocol (SCEP) AWS 私有憑證授權單位 連結至已啟用 SCEP 的行動裝置和聯網設備。使用 Connector for SCEP，您可以使用 AWS 私有 CA 來發行憑證並註冊 SCEP 裝置。Connector for SCEP 可與熱門的行動裝置管理 (MDM) 系統搭配使用，旨在與支援 SCEP 的用戶端或端點搭配使用。

主題

- 功能
- 如何開始使用 Connector for SCEP
- 相關服務
- 適用於 SCEP 的 Access Connector
- 定價
- 適用於 SCEP 概念的連接器
- 了解 Connector for SCEP 的考量和限制
- 設定適用於 SCEP 的 Connector
- Connector for SCEP 入門
- 設定 Connector for SCEP 的 MDM 系統
- Monitor Connector for SCEP
- 針對適用於 SCEP 問題的 AWS 私有憑證授權單位 Connector 進行故障診斷

功能

支援 SCEP 通訊協定 - SCEP 是一種廣泛採用的通訊協定，用於從憑證授權單位 (CA) 取得數位身分憑證，並將其分發到行動裝置和聯網裝置。您可以使用 Connector for SCEP 來協助您使用 SCEP 註冊端點。

行動裝置註冊 - 您可以將 Connector for SCEP 與熱門 MDM 系統搭配使用，包括 Microsoft Intune 和 Jamf Pro。

大規模發行憑證 - 在您設定已啟用 SCEP 的裝置透過連接器的 SCEP 端點請求憑證之後，用戶端可以自動從中請求憑證 AWS 私有 CA。

如何開始使用 Connector for SCEP

若要開始使用，請從 Connector for SCEP 管理主控台啟動引導式精靈，協助您建立連接器，並指定要與連接器搭配使用的私有 CA。完成這些步驟後，Connector for SCEP 會提供端點和其他組態參數，供您輸入 MDM 系統或聯網設備。設定 MDM 系統或聯網設備之後，您的用戶端會自動向 請求憑證 AWS 私有 CA。若要進一步了解如何開始使用 Connector for SCEP，請參閱 Connector for SCEP 入門。

相關服務

Connector for SCEP 與下列 AWS 服務相關。

- AWS 私有憑證授權單位 - AWS 私有 CA 為您提供高可用性的私有 CA 服務，無需預付投資和持續的維護成本來操作您自己的私有 CA。
- AWS 私有 CA Connector for Active Directory - Connector for AD 會連結您的 Active Directory (AD) AWS 私有 CA。連接器會代理將憑證從 交換 AWS 私有 CA 至 AD 管理的使用者和機器。

適用於 SCEP 的 Access Connector

您可以使用下列任一界面來建立、存取和管理 Connector for SCEP 連接器：

- AWS 管理主控台 - 提供可用來存取 Connector for SCEP 的 Web 界面。請參閱 Connector for SCEP 管理主控台。
- AWS Command Line Interface - 為廣泛的 AWS 服務提供命令，包括 Connector for SCEP。Windows、macOS 和 Linux AWS CLI 支援 。如需詳細資訊，請參閱AWS Command Line Interface。
- AWS SDKs - 提供特定語言APIs，並處理許多連線詳細資訊，例如計算簽章、處理請求重試和錯誤處理。如需詳細資訊，請參閱AWS Command Line Interface。
- Connector for SCEP API - 提供您使用 HTTPS 請求呼叫的低階 API 動作。使用 Connector for SCEP API 是存取服務最直接的方式。不過，Connector for SCEP API 需要您的應用程式處理低階詳細資訊，例如產生雜湊來簽署請求，以及錯誤處理。如需詳細資訊，請參閱 Connector for SCEP API 參考。

定價

Connector for SCEP 是 的一項功能 AWS 私有 CA ，無需額外費用。您只需支付用於建立和更新連接器 AWS 私有憑證授權單位 的操作和憑證。

如需最新的 AWS 私有 CA 定價資訊，請參閱 AWS 私有憑證授權單位 定價。您也可以使用AWS 定價計算器來預估成本。

適用於 SCEP 概念的連接器

Connector for SCEP 是 的附加元件功能 AWS 私有憑證授權單位。

以下是 Connector for SCEP 的重要概念：

憑證簽署請求 (CSR)

提供給 CA 的必要資訊，以便發行數位憑證。此資訊包含公有金鑰和身分。

挑戰密碼

SCEP 通訊協定使用挑戰密碼來驗證請求，然後再從 CA 發出憑證。Connector for SCEP 會根據連接器類型處理 SCEP 挑戰密碼。如需詳細資訊，請參閱設定 Connector for SCEP 的 MDM 系統。

憑證撤銷

憑證撤銷是在其過期日期之前撤銷已發行憑證的程序。您可以在 API AWS Command Line Interface、 AWS SDK 或 中呼叫 RevokeCertificate，撤銷與連接器相關聯的私有 CA 憑證 AWS CloudFormation。

適用於 SCEP 的連接器

SCEP 的連接器 AWS 私有 CA 會連結到已啟用 SCEP 的裝置。

行動裝置管理

行動裝置管理 (MDM) 允許 IT 管理員控制、保護和強制執行智慧型手機、平板電腦和其他端點或裝置上的政策。許多 MDM 系統為 SCEP 型憑證註冊提供內建整合。

SCEP

SCEP 是自動分發憑證的標準化通訊協定 (RFC 8894)。通訊協定為裝置提供端點，以向 CA 請求憑證。SCEP 使用挑戰密碼來授權向裝置發行憑證。SCEP 通常適用於行動裝置管理 (MDM) 系統和聯網設備。MDM 解決方案可讓 IT 管理員控制、保護和強制執行智慧型手機、平板電腦和其他實體上的政策，例如 Apple 工作站。大多數 MDM 解決方案支援 SCEP，例如 Microsoft Intune、Apple MDM 和 Jamf Pro。大多數聯網設備，例如路由器、負載平衡器、Wi-Fi 中樞、VPN 裝置和防火牆，都會使用 SCEP 進行自動憑證註冊。

SCEP 設定檔

SCEP 設定檔包含用於定義憑證設定檔的組態參數。這包括憑證有效期間、金鑰大小、SCEP 組態名稱、挑戰密碼、失敗嘗試重試次數和重試間隔，以及其他與憑證發行相關的資訊。MDM 系統和憑證管理平台通常會將 SCEP 設定檔傳送至請求憑證以進行身分驗證的用戶端。

了解 Connector for SCEP 的考量和限制

使用 Connector for SCEP 時，請記住下列考量和限制。

考量事項

CA 操作模式

您只能將 Connector for SCEP 與使用一般用途操作模式的私有 CAs 搭配使用。Connector for SCEP 預設為發行有效期間為一年的憑證。使用短期憑證模式的私有 CA 不支援發行有效期間超過七天的憑證。如需操作模式的相關資訊，請參閱 了解 AWS 私有 CA CA 模式。

挑戰密碼

- 請謹慎分發您的挑戰密碼，並僅與高度信任的個人和用戶端共用。單一挑戰密碼可用於向任何主體和 SANs 發行任何憑證，這會造成安全風險。
- 如果使用一般用途連接器，建議您經常手動輪換挑戰密碼。

RFC 8894 的一致性

Connector for SCEP 透過提供 HTTPS 端點而非 HTTP 端點，偏離 RFC 8894 通訊協定。

CSRs

- 如果傳送至 Connector for SCEP 的憑證簽署請求 (CSR) 不包含擴充金鑰用量 (EKU) 延伸，我們會將 EKU 值設定為 `clientAuthentication`。如需詳細資訊，請參閱 4.2.1.12。RFC 5280 中的擴充金鑰用量。
- 我們在 CSRs 中支援 `ValidityPeriod`和 `ValidityPeriodUnits`自訂屬性。如果您的 CSR 不包含 `ValidityPeriod`，我們會發行有效期為一年的憑證。請記住，您可能無法在 MDM 系統中設定這些屬性。但是，如果您可以設定它們，我們會支援它們。如需這些屬性的詳細資訊，請參閱 szENROLLMENT_NAME_VALUE_PAIR。

端點共用

僅將連接器的端點分發給信任的對象。將端點視為秘密，因為任何可以找到您唯一完整網域名稱和路徑的人都可以擷取您的 CA 憑證。

限制

下列限制適用於 Connector for SCEP。

動態挑戰密碼

您只能使用一般用途連接器建立靜態挑戰密碼。若要搭配一般用途連接器使用動態密碼，您必須建置自己的輪換機制，以使用連接器的靜態密碼。Connector for SCEP for Microsoft Intune 連接器類型支援您使用 Microsoft Intune 管理的動態密碼。

HTTP

Connector for SCEP 僅支援 HTTPS，並建立 HTTP 呼叫的重新導向。如果您的系統依賴 HTTP，請確定它可以容納 Connector for SCEP 提供的 HTTP 重新導向。

共用私有 CAs

您只能將 Connector for SCEP 與您作為擁有者的私有 CAs搭配使用。

設定適用於 SCEP 的 Connector

本節中的程序可協助您開始使用 Connector for SCEP。其假設您已建立 AWS 帳戶。完成此頁面的步驟後，您可以繼續為 SCEP 建立連接器。

主題

- 步驟 1：建立 AWS Identity and Access Management 政策
- 步驟 2：建立私有 CA
- 步驟 3：使用 建立資源共享 AWS Resource Access Manager

步驟 1：建立 AWS Identity and Access Management 政策

若要建立 SCEP 的連接器，您需要建立 IAM 政策，授予 Connector for SCEP 建立和管理連接器所需資源的能力，並代表您發行憑證。如需 IAM 的詳細資訊，請參閱《IAM 使用者指南》中的什麼是 IAM？。

下列範例是客戶受管政策，可用於 Connector for SCEP。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "pca-connector-scep:*",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "acm-pca:DescribeCertificateAuthority",
                "acm-pca:GetCertificate",
                "acm-pca:GetCertificateAuthorityCertificate",
                "acm-pca:ListCertificateAuthorities",
                "acm-pca:ListTags",
                "acm-pca:PutPolicy"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "acm-pca:IssueCertificate",
            "Resource": "*",
            "Condition": {
                "ArnLike": {
                    "acm-pca:TemplateArn": "arn:aws:acm-pca:::template/
BlankEndEntityCertificate_APICSRPassthrough/V*"
                },
                "ForAnyValue:StringEquals": {
                    "aws:CalledVia": "pca-connector-scep.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "ram:CreateResourceShare",
                "ram:GetResourcePolicies",
                "ram:GetResourceShareAssociations",
                "ram:GetResourceShares",
```

```
                "ram:ListPrincipals",
                "ram:ListResources",
                "ram:ListResourceSharePermissions",
                "ram:ListResourceTypes"
            ],
            "Resource": "*"
        }
    ]
}
```

步驟 2：建立私有 CA

若要使用 Connector for SCEP，您需要將來自 的私有 CA AWS 私有憑證授權單位 與連接器建立關聯。由於 SCEP 通訊協定中存在固有的安全漏洞，我們建議您使用僅適用於連接器的私有 CA。

私有 CA 必須符合下列要求：

- 它必須處於作用中狀態，並使用一般用途操作模式。
- 您必須擁有私有 CA。您無法使用透過跨帳戶共用與您共用的私有 CA。

設定私有 CA 以搭配 Connector for SCEP 使用時，請注意下列考量：

- DNS 名稱限制 – 考慮使用 DNS 名稱限制來控制針對 SCEP 裝置發行的憑證中允許或禁止哪些網域。如需詳細資訊，請參閱如何在 中強制執行 DNS 名稱限制 AWS 私有憑證授權單位。
- 撤銷 – 在您的私有 CA 上啟用 OCSP 或 CRLs 以允許撤銷。如需詳細資訊，請參閱規劃您的 AWS 私有 CA 憑證撤銷方法。
- PII – 我們建議您不要在 CA 憑證中新增個人身分識別資訊 (PII) 或其他機密或敏感資訊。如果發生安全漏洞，這有助於限制敏感資訊的暴露。
- 將根憑證存放在信任存放區 – 將根 CA 憑證存放在裝置信任存放區中，以便您可以驗證憑證和 GetCertificateAuthorityCertificate 的傳回值。如需與信任存放區相關的資訊 AWS 私有 CA，請參閱根 CA。

如需如何建立私有 CA 的資訊，請參閱 在 中建立私有 CA AWS 私有 CA。

步驟 3：使用 建立資源共享 AWS Resource Access Manager

如果您使用 Connector for SCEP API 以程式設計方式使用 AWS Command Line Interface AWS Connector for SCEP，則需要使用 AWS Resource Access Manager 服務主體共用，將私有 CA 與

Connector for SCEP 共用。這可讓 Connector for SCEP 共用存取您的私有 CA。當您在 AWS 主控台中建立連接器時，我們會自動為您建立資源共享。如需資源共用的資訊，請參閱AWS RAM 《使用者指南》中的建立資源共用。

若要使用 建立資源共享 AWS CLI，您可以使用 AWS RAM create-resource-share命令。下列命令會建立資源共享。指定您要共享之私有 CA 的 ARN，做為 *resource-arns* 的值。

```
$  aws ram create-resource-share \
--region us-east-1 \
--name MyPcaConnectorScepResourceShare \
--permission-arns arn:aws:ram::aws:permission/
AWSRAMBlankEndEntityCertificateAPICSRPassthroughIssuanceCertificateAuthority \
--resource-arns arn:aws:acm-pca:Region:account:certificate-authority/CA_ID \
--principals pca-connector-scep.amazonaws.com \
--sources account
```

呼叫 的服務主體`CreateConnector`具有私有 CA 的憑證發行許可。若要防止使用 Connector for SCEP 的服務主體能夠一般存取您的 AWS 私有 CA 資源，請使用 限制其許可`CalledVia`。

Connector for SCEP 入門

使用 AWS 私有憑證授權單位 Connector for SCEP，您可以從私有 CA 向啟用 SCEP 的裝置和行動裝置管理 (MDM) 系統發行憑證。當您建立連接器時， 會為您 AWS 私有憑證授權單位 建立公有 SCEP URL 來請求憑證，並為您提供可用來整合到 MDM 系統的資訊。

若要發行憑證，您必須建立 AWS 私有憑證授權單位 私有 CA、建立連接器，然後設定已啟用 SCEP 的 MDM 系統和裝置，以從連接器請求憑證。

主題

- 開始之前
- 步驟 1：建立連接器
- 步驟 2：將連接器詳細資訊複製到 MDM 系統

開始之前

下列教學課程會引導您完成建立 SCEP 連接器的程序。

若要遵循本教學課程，您需要私有 CA 和啟用 SCEP 的裝置。您也必須先滿足 設定適用於 SCEP 的 Connector區段中列出的先決條件。

下列程序說明如何使用 AWS 主控台建立連接器。

任務

- 步驟 1：建立連接器
- 步驟 2：將連接器詳細資訊複製到 MDM 系統

步驟 1：建立連接器

您將建立一般用途的連接器或適用於 Microsoft Intune 的 Connector for SCEP。一般用途連接器專為與啟用 SCEP 的端點搭配使用而設計，您可以管理 SCEP 挑戰密碼。Connector for SCEP for Microsoft Intune 適用於 Microsoft Intune，您可以使用 Microsoft Intune 管理挑戰密碼。

General-purpose

建立用於一般用途的連接器

登入 AWS 您的帳戶，並在 開啟 Connector for SCEP 主控台**https://console.aws.amazon.com/pca-connector-scep/home**。

1. 選擇 Create connector (建立連接器)。
2. 在建立連接器頁面中，選擇性地在名稱標籤欄位中為連接器提供易記的名稱。名稱將顯示在您的連接器清單中。如果需要，您可以選取新增更多標籤，將更多標籤新增至連接器。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。您可以使用標籤來搜尋和篩選資源或追蹤 AWS 成本。
3. 在連接器類型下，選擇一般用途。
4. 在私有 CA 下，選擇要與此連接器搭配使用的私有 CA。或者，選取建立私有 CA 來建立新的 CA。由於 SCEP 通訊協定中的固有漏洞，我們建議使用此連接器專用的私有 CA。如果您建立了新的 CA，當您完成在其中建立 CA 時 AWS 私有 CA，請返回 Connector for SCEP 主控台並重新整理私有 CAs清單。您的新私有 CA 應該可供選取。
5. 在挑戰密碼下，選取自動產生挑戰密碼。建立此連接器時，我們會為您產生靜態挑戰密碼。
6. 選取建立連接器。

Microsoft Intune

建立 Connector for SCEP for Microsoft Intune

登入 AWS 您的帳戶，並在 開啟 Connector for SCEP 主控台**https://console.aws.amazon.com/pca-connector-scep/home**。

1. 選擇 Create connector (建立連接器)。
2. 在建立連接器頁面上，選擇性地在名稱標籤欄位中為連接器提供易記的名稱。名稱將顯示在您的連接器清單中。如果需要，您可以選取新增更多標籤，將更多標籤新增至連接器。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。您可以使用標籤來搜尋和篩選資源或追蹤 AWS 成本。
3. 在連接器類型下，選擇 Microsoft Intune。
   a. 針對應用程式（用戶端）ID，輸入您 Microsoft Entra ID 應用程式註冊中的應用程式（用戶端）ID。如需搭配 Connector for SCEP 使用 Microsoft Intune 的詳細資訊，請參閱 設定 Connector for SCEP 的 MDM 系統。
   b. 針對目錄（租戶）ID 或主要網域，輸入您 Microsoft Entra ID 應用程式註冊中的目錄（租戶）ID 或主要網域。
4. 在私有 CA 下，選擇要與此連接器搭配使用的私有 CA。或者，選取建立私有 CA 來建立新的 CA。由於 SCEP 通訊協定中的固有漏洞，我們建議使用此連接器專用的私有 CA。如果您建立了新的 CA，當您完成在其中建立 CA 時 AWS 私有 CA，請返回 Connector for SCEP 主控台並重新整理私有 CAs清單。您的新私有 CA 應該可供選取。
5. 選取建立連接器。

步驟 2：將連接器詳細資訊複製到 MDM 系統

建立連接器後，您需要將下列詳細資訊從連接器複製到 MDM 系統。若要使用主控台檢視連接器的詳細資訊，請從 Connectors for SCEP 主控台頁面的清單中選擇連接器。

- 公有 SCEP URL - 這是連接器的端點，SCEP 用戶端將從中請求憑證。請注意，僅將此端點提供給信任的實體。
- （一般用途）挑戰密碼 - 在挑戰密碼下，選取您在上述程序中自動產生的密碼，然後選取檢視密碼以檢視密碼。若要建立額外的密碼，請選取建立密碼。請小心將密碼分發給高度信任的個人和用戶端。單一挑戰密碼可用於向任何主體和 SANs 發行任何憑證，因此應謹慎處理。

- (Microsoft Intune) Open ID 值 - 如果您要與 Microsoft Intune 整合，您必須將 Open ID 發行者、Open ID 主體和 Open ID 對象複製到您 Microsoft Entra 應用程式註冊的 OpenID Connect (OIDC) 憑證。如需詳細資訊，請參閱設定 Connector for SCEP 的 MDM 系統。

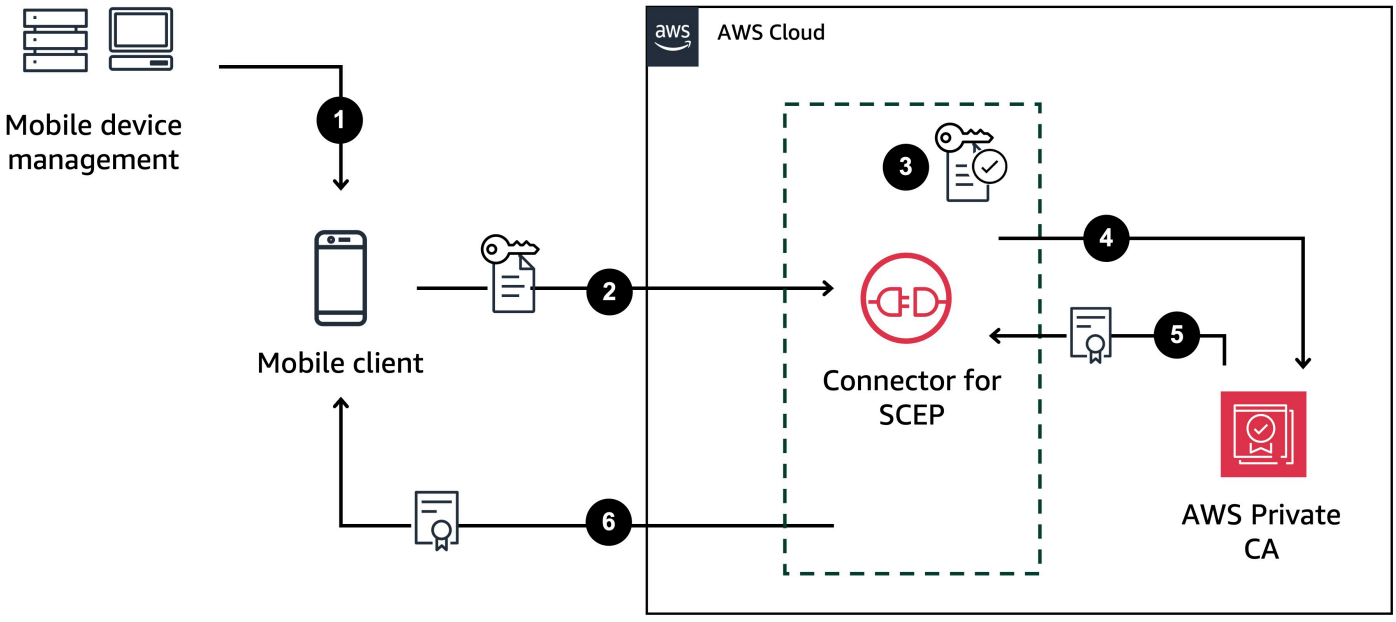
設定 Connector for SCEP 的 MDM 系統

Simple Certificate Enrollment Protocol (SCEP) 是用於憑證註冊和續約的標準通訊協定。Connector for SCEP 是以 RFC 8894 為基礎的 SCEP 伺服器，會自動從 將憑證發行 AWS 私有憑證授權單位到您的 SCEP 用戶端。當您建立連接器時，Connector for SCEP 會提供 HTTPS 端點，讓 SCEP 用戶端從中請求憑證。用戶端會使用挑戰密碼進行身分驗證，該密碼包含在對服務的憑證簽署請求 (CSR) 中。您可以使用 Connector for SCEP 搭配熱門的行動裝置管理 (MDM) 系統，包括 Microsoft Intune、Omanssa Workspace ONE 和 Jamf Pro，來註冊行動裝置。其設計用於支援 SCEP 的任何用戶端或端點。

Connector for SCEP 提供兩種類型的連接器：一般用途和 Connector for SCEP for Microsoft Intune。下列各節說明它們的運作方式，以及如何設定 MDM 系統來使用它們。

一般用途連接器

一般用途連接器旨在使用支援 SCEP 的行動裝置端點，但具有專用連接器的 Microsoft Intune 除外。使用 Jamf Pro 或 Omnissa Workspace ONE 等一般用途連接器，您可以管理 SCEP 挑戰密碼。下圖使用行動裝置管理 (MDM) 系統做為範例，但相同的功能適用於其他已啟用 SCEP 的系統或裝置。

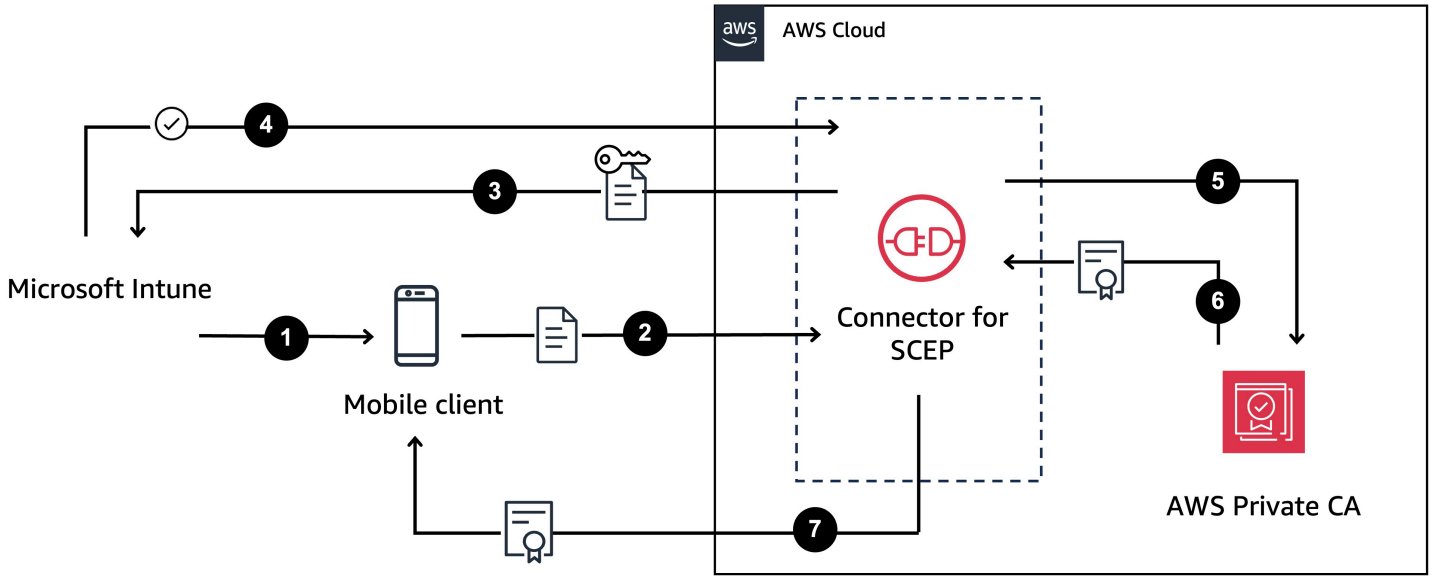


1. MDM 系統（或其他裝置或系統）會將 SCEP 設定檔傳送至行動用戶端。SCEP 設定檔包含定義憑證設定檔的組態參數，例如憑證有效期間、挑戰密碼，以及與憑證發行相關的其他資訊。
2. 行動用戶端會請求憑證，也會傳送包含挑戰密碼的憑證簽署請求 (CSR)。
3. Connector for SCEP 驗證挑戰密碼。如果有效，則服務 AWS 私有 CA 會代表行動用戶端向 請求憑證。
4. AWS 私有 CA 會發出憑證並將其傳送至 Connector for SCEP。
5. Connector for SCEP 會將發行的憑證傳送至行動用戶端。

AWS 私有 CA Connector for SCEP for Microsoft Intune

AWS 私有 CA Connector for SCEP for Microsoft Intune 專為與 Microsoft Intune 搭配使用而設計。使用 Connector for SCEP for Microsoft Intune 連接器類型，您將使用 Microsoft Intune 來管理您的 SCEP 挑戰密碼。如需搭配 Microsoft Intune 使用 Connector for SCEP 的詳細資訊，請參閱 設定適用於 SCEP 連接器的 Microsoft Intune。

若要搭配 Microsoft Intune 使用 Connector for SCEP，您必須使用 Microsoft Intune API 啟用特定功能，並擁有有效的 Microsoft Intune 授權。您也應該檢閱 Microsoft Intune® 應用程式保護政策。



1. Microsoft Intune 會將 SCEP 設定檔傳送至行動用戶端。設定檔包含行動用戶端放入 CSR 的加密挑戰密碼。
2. 行動用戶端會請求憑證，並將 CSR 傳送至 Connector for SCEP。
3. Connector for SCEP 會將 CSR 傳送給 Microsoft Intune 進行授權。

4. Microsoft Intune 會解密 CSR 中的挑戰密碼。如果有效，Microsoft Intune 會將核准傳送至 Connector for SCEP，以向行動用戶端發出憑證。
5. Connector for SCEP AWS 私有 CA 代表行動用戶端向 請求憑證。
6. AWS 私有 CA 會發出憑證並將其傳送至 Connector for SCEP。
7. Connector for SCEP 會將發行的憑證傳送至行動用戶端。

主題

- 設定適用於 SCEP 連接器的 Jamf Pro
- 設定適用於 SCEP 連接器的 Microsoft Intune
- 設定適用於 SCEP 連接器的 Omnissa Workspace ONE

設定適用於 SCEP 連接器的 Jamf Pro

您可以使用 AWS 私有 CA 做為 Jamf Pro 行動裝置管理 (MDM) 系統的外部憑證授權機構 (CA)。本指南說明如何在建立一般用途連接器後設定 Jamf Pro。

設定適用於 SCEP 連接器的 Jamf Pro

本指南提供如何設定 Jamf Pro 以搭配 Connector for SCEP 使用的指示。成功設定適用於 SCEP 的 Jamf Pro 和 Connector 後，您將能夠向受管裝置發行 AWS 私有 CA 憑證。

Jamf Pro 需求

您的 Jamf Pro 實作必須符合下列要求。

- 您必須在 Jamf Pro 中啟用啟用憑證型身分驗證設定。您可以在 Jamf Pro 文件的 Jamf Pro 安全設定頁面上找到此設定的詳細資訊。

步驟 1：（選用 - 建議）取得私有 CA 的指紋

指紋是私有 CA 的唯一識別符，可用於在與其他系統或應用程式建立信任時驗證 CA 的身分。整合憑證授權機構 (CA) 指紋可讓受管裝置驗證其連線的 CA，並僅從預期的 CA 請求憑證。我們建議搭配 Jamf Pro 使用 CA 指紋。

為您的私有 CA 產生指紋

1. 從 AWS 私有 CA 主控台或使用 GetCertificateAuthorityCertificate 取得私有 CA 憑證。將其儲存為 `ca.pem` 檔案。

2. 安裝 OpenSSL Command Line 公用程式。
3. 在 OpenSSL 中，執行下列命令來產生指紋：

```
openssl x509 -in ca.pem -sha256 -fingerprint
```

步驟 2：在 Jamf Pro 中將 設定為 AWS 私有 CA 外部 CA

建立 SCEP 連接器後，您必須在 Jamf Pro 中將 AWS 私有 CA 設定為外部憑證授權機構 (CA)。您可以將 AWS 私有 CA 設定為全域外部 CA。或者，您可以使用 Jamf Pro 組態描述檔，針對不同的使用案例從 發行 AWS 私有 CA 不同的憑證，例如將憑證發行到組織中的裝置子集。實作 Jamf Pro 組態描述檔的指引超出本文件的範圍。

在 Jamf Pro 中將 AWS 私有 CA 設定為外部憑證授權單位 (CA)

1. 在 Jamf Pro 主控台中，前往設定 > 全域 > PKI 憑證，前往 PKI 憑證設定頁面。
2. 選取管理憑證範本索引標籤。
3. 選取外部 CA。
4. 選擇 Edit (編輯)。
5. （選用）選取啟用 Jamf Pro 做為組態設定檔的 SCEP Proxy。您可以使用 Jamf Pro 組態描述檔來發行針對特定使用案例量身打造的不同憑證。如需如何在 Jamf Pro 中使用組態設定檔的指引，請參閱 Jamf Pro 文件中的將 Jamf Pro 啟用為組態設定檔的 SCEP Proxy。
6. 選取使用已啟用 SCEP 的外部 CA 進行電腦和行動裝置註冊。
7. （選用）選取使用 Jamf Pro 做為電腦和行動裝置註冊的 SCEP Proxy。如果您遇到設定檔安裝失敗，請參閱 對設定檔安裝失敗進行故障診斷。
8. 將 Connector for SCEP 公有 SCEP URL 從連接器的詳細資訊複製到 Jamf Pro 中的 URL 欄位。若要檢視連接器的詳細資訊，請從適用於 SCEP 的連接器清單中選擇連接器。或者，您可以呼叫 GetConnector 並從回應複製`Endpoint`值來取得 URL。
9. （選用）在名稱欄位中輸入執行個體的名稱。例如，您可以將其命名為 AWS 私有 CA。
10. 針對挑戰類型選取靜態。
11. 從連接器複製挑戰密碼，並將其貼到挑戰欄位中。連接器可以有多個挑戰密碼。若要檢視連接器的挑戰密碼，請導覽至 AWS 主控台中連接器的詳細資訊頁面，然後選取檢視密碼按鈕。或者，您可以呼叫 GetChallengePassword (GetChallengePassword) 並從回應複製`Password`值，以取得連接器的挑戰密碼。如需使用挑戰密碼的詳細資訊，請參閱 了解 Connector for SCEP 的考量和限制。

12. 將挑戰密碼貼到驗證挑戰欄位中。
13. 選擇金鑰大小。我們建議金鑰大小為 2048 或更高。
14. （選用）選取使用 做為數位簽章。選取此選項以進行身分驗證，以授予裝置安全存取 Wi-Fi 和 VPN 等資源的權限。
15. （選用）選取用於金鑰加密。
16. （選用 - 建議）在指紋欄位中輸入十六進位字串。我們建議您新增 CA 指紋，以允許受管裝置驗證 CA，並僅向 CA 請求憑證。如需如何為私有 CA 產生指紋的說明，請參閱 步驟 1：（選用 - 建議）取得私有 CA 的指紋。
17. 選取儲存。

步驟 3：設定組態設定檔簽署憑證

若要使用 Jamf Pro 搭配 Connector for SCEP，您必須為與連接器相關聯的私有 CA 提供簽署和 CA 憑證。您可以透過將設定檔簽署憑證金鑰存放區上傳到包含兩個憑證的 Jamf Pro 來執行此操作。

以下是建立憑證金鑰存放區並將其上傳至 Jamf Pro 的步驟：

- 使用內部程序產生憑證簽署請求 (CSR)。
- 取得與連接器相關聯之私有 CA 簽署的 CSR。
- 建立包含設定檔簽署和 CA 憑證的設定檔簽署憑證金鑰存放區。
- 將憑證金鑰存放區上傳至 Jamf Pro。

透過遵循這些步驟，您可以確保裝置可以驗證和驗證由私有 CA 簽署的組態設定檔，以使用 Connector for SCEP 搭配 Jamf Pro。

1. 下列範例使用 OpenSSL 和 AWS Certificate Manager，但您可以使用您偏好的方法產生憑證簽署請求。

   AWS Certificate Manager console

   使用 ACM 主控台建立設定檔簽署憑證

   1. 使用 ACM 請求私有 PKI 憑證。包含下列項目：

      - 類型 - 使用做為 MDM 系統 SCEP 憑證授權單位的相同私有 CA 類型。
      - 在憑證授權機構詳細資訊區段中，選取憑證授權機構選單，然後選擇做為 Jamf Pro CA 的私有 CA。

- 網域名稱 - 提供要內嵌至憑證的網域名稱。您可以使用完整網域名稱 (FQDN)，例如 `www.example.com`，或是裸機或頂點網域名稱，例如 `example.com`（不包括 `www.`）。

2. 使用 ACM 匯出您在上述步驟中建立的私有憑證。選擇匯出憑證、憑證鏈和加密金鑰的檔案。請備妥密碼短語，因為在下一個步驟中會需要它。
3. 在終端機中，在包含匯出檔案的資料夾中執行下列命令，將 PKCS#12 套件寫入您在上一個步驟中建立的密碼短語編碼的`output.p12`檔案中。

```
openssl pkcs12 -export \
  -in "Exported Certificate.txt" \
  -certfile "Certificate Chain.txt" \
  -inkey "Exported Certificate Private Key.txt" \
  -name example \
  -out output.p12 \
  -passin pass:your-passphrase \
  -passout pass:your-passphrase
```

AWS Certificate Manager CLI

使用 ACM CLI 建立設定檔簽署憑證

- 下列命令說明如何在 ACM 中建立憑證，然後將檔案匯出為 PKCS#12 套件。

```
PCA=<Enter your Private CA ARN>

CERTIFICATE=$(aws acm request-certificate \
    --certificate-authority-arn $PCA \
    --domain-name <any valid domain name, such as test.name> \
    | jq -r '.CertificateArn')

while [[ $(aws acm describe-certificate \
  --certificate-arn $CERTIFICATE \
  | jq -r '.Certificate.Status') != "ISSUED" ]] do sleep 1; done

aws acm export-certificate \
  --certificate-arn $CERTIFICATE \
  --passphrase password | jq -r '.Certificate' > Certificate.pem
aws acm export-certificate \
  --certificate-arn $CERTIFICATE \
  --passphrase password | jq -r '.CertificateChain' > CertificateChain.pem
```

```
aws acm export-certificate \
  --certificate-arn $CERTIFICATE \
  --passphrase password | jq -r '.PrivateKey' > PrivateKey.pem

openssl pkcs12 -export \
  -in "Certificate.pem" \
  -certfile "CertificateChain.pem" \
  -inkey "PrivateKey.pem" \
  -name example \
  -out output.p12 \
  -passin pass:passphrase \
  -passout pass:passphrase
```

OpenSSL CLI

使用 OpenSSL CLI 建立設定檔簽署憑證

1. 使用 OpenSSL，執行下列命令來產生私有金鑰。

```
openssl genrsa -out local.key 2048
```

2. 產生憑證簽署請求 (CSR)：

```
openssl req -new -key local.key -sha512 -out local.csr -
subj "/CN=MySigningCertificate/O=MyOrganization" -addext
 keyUsage=critical,digitalSignature,nonRepudiation
```

3. 使用 AWS CLI，使用您在上一個步驟中產生的 CSR 發行簽署憑證。執行下列命令，並在回應中記下憑證 ARN。

```
aws acm-pca issue-certificate --certificate-authority-arn <SAME CA AS
 USED ABOVE, SO IT'S TRUSTED> --csr fileb://local.csr --signing-algorithm
 SHA512WITHRSA --validity Value=365,Type=DAYS
```

4. 執行下列命令以取得簽署憑證。指定上一個步驟的憑證 ARN。

```
aws acm-pca get-certificate --certificate-authority-arn <SAME CA AS USED
 ABOVE, SO IT'S TRUSTED> --certificate-arn <ARN OF NEW CERTIFICATE> | jq -r
 '.Certificate' >local.crt
```

5. 執行下列命令以取得 CA 憑證。

```
aws acm-pca get-certificate-authority-certificate --certificate-authority-
arn <SAME CA AS USED ABOVE, SO IT'S TRUSTED> | jq -r '.Certificate' > ca.crt
```

6. 使用 OpenSSL，以 p12 格式輸出簽署憑證金鑰存放區。使用您在步驟 4 和步驟 5 中產生的 CRT 檔案。

```
openssl pkcs12 -export -in local.crt -inkey local.key -certfile ca.crt -name
 "CA Chain" -out local.p12
```

7. 出現提示時，請輸入匯出密碼。此密碼是您提供給 Jamf Pro 的金鑰存放區密碼。

2. 在 Jamf Pro 中，導覽至管理憑證範本，然後前往外部 CA 窗格。
3. 在外部 CA 窗格底部，選取變更簽署和 CA 憑證。
4. 遵循畫面上的指示，上傳外部 CA 的簽署和 CA 憑證。

步驟 4：（選用）在使用者起始的註冊期間安裝憑證

若要在用戶端裝置與私有 CA 之間建立信任，您必須確保裝置信任 Jamf Pro 發行的憑證。您可以使用 Jamf Pro 的使用者起始註冊設定，在用戶端裝置上在註冊過程中請求憑證時，自動在用戶端裝置上安裝 AWS 私有 CA的 CA 憑證。

對設定檔安裝失敗進行故障診斷

如果您在啟用使用 Jamf Pro 做為電腦和行動裝置註冊的 SCEP Proxy 後遇到設定檔安裝失敗，請參閱您的裝置日誌並嘗試下列動作。

裝置日誌錯誤訊息	緩解
`Profile installation failed. Unable to obtain certificate from SCEP server at "<your-jamf-endpoint>.jamfcloud.com". <MDM-SCEP:15001>`	如果您在嘗試註冊時收到此錯誤訊息，請重試註冊。註冊成功之前，可能需要多次嘗試。
`Profile installation failed. Unable to obtain certificate from SCEP server at "<your-jamf-`	您的挑戰密碼可能設定錯誤。驗證 Jamf Pro 中的挑戰密碼是否符合連接器的挑戰密碼。

裝置日誌錯誤訊息	緩解
`endpoint>.jamfcloud.com". <MDM-SCEP:14006>`	

設定適用於 SCEP 連接器的 Microsoft Intune

您可以使用 AWS 私有 CA 做為外部憑證授權機構 (CA) 與 Microsoft Intune 行動裝置管理 (MDM) 系統。本指南說明如何在建立 Connector for SCEP for Microsoft Intune 之後設定 Microsoft Intune。

先決條件

建立 Connector for SCEP for Microsoft Intune 之前，您必須完成下列先決條件。

- 建立 Entra ID。
- 建立 Microsoft Intune 租用戶。
- 在 Microsoft Entra ID 中建立應用程式註冊。如需如何管理應用程式註冊之應用程式層級許可的相關資訊，請參閱 Microsoft Entra 文件中的在 Microsoft Entra ID 中更新應用程式請求的許可。應用程式註冊必須具有下列許可：
  - 在 Intune 設定 scep_challenge_provider 下。
  - 對於 Microsoft 圖形集 Application.Read.All 和 User.Read。
- 您必須在應用程式註冊管理員同意中授予應用程式。如需詳細資訊，請參閱 Microsoft Entra 文件中的授予租用戶整體管理員對應用程式的同意。

> **Tip**
>
> 當您建立應用程式註冊時，請記下應用程式（用戶端）ID 和目錄（租戶）ID 或主要網域。當您建立 Connector for SCEP for Microsoft Intune 時，您將會輸入這些值。如需如何取得這些值的資訊，請參閱 Microsoft Entra 文件中的建立可存取資源的 Microsoft Entra 應用程式和服務主體。

步驟 1：授予 AWS 私有 CA 許可以使用您的 Microsoft Entra ID 應用程式

建立 Connector for SCEP for Microsoft Intune 之後，您必須在 Microsoft 應用程式註冊下建立聯合憑證，以便 Connector for SCEP 可以與 Microsoft Intune 通訊。

在 Microsoft Intune 中將 AWS 私有 CA 設定為外部 CA

1. 在 Microsoft Entra ID 主控台中，導覽至應用程式註冊。
2. 選擇您建立以搭配 Connector for SCEP 使用的應用程式。您按一下的應用程式（用戶端）ID 必須符合您在建立連接器時指定的 ID。
3. 從受管下拉式功能表中選取憑證和秘密。
4. 選取聯合登入資料索引標籤。
5. 選取新增登入資料。
6. 從聯合登入資料案例下拉式選單中，選擇其他發行者。
7. 將 Connector for SCEP for Microsoft Intune 詳細資訊中的 OpenID 發行者值複製並貼到發行者欄位中。若要檢視連接器的詳細資訊，請從 AWS 主控台的 Connectors for SCEP 清單中選擇連接器。或者，您可以呼叫 GetConnector 來取得 URL，然後從回應複製該`Issuer`值。
8. 針對類型，選取明確主旨識別符。
9. 將連接器中的 OpenID 主體值複製並貼到值欄位中。您可以在 AWS 主控台的連接器詳細資訊頁面中檢視 OpenID 發行者值。或者，您可以呼叫 GetConnector 來取得 URL，然後從回應複製該`Audience`值。
10. （選用）在名稱欄位中輸入執行個體的名稱。例如，您可以將其命名為 AWS 私有 CA。
11. （選用）在描述欄位中輸入描述。
12. 將 OpenID Audience 值從 Connector for SCEP for Microsoft Intune 詳細資訊複製並貼到 Audience 欄位。若要檢視連接器的詳細資訊，請從 AWS 主控台的 Connectors for SCEP 清單中選擇連接器。或者，您可以呼叫 GetConnector 來取得 URL，然後從回應複製該`Subject`值。
13. 選取新增。

步驟 2：設定 Microsoft Intune 組態描述檔

授予呼叫 Microsoft Intune 的 AWS 私有 CA 許可後，您必須使用 Microsoft Intune 建立 Microsoft Intune 組態描述檔，指示裝置聯絡 Connector for SCEP 進行憑證發行。

1. 建立信任的憑證組態設定檔。您必須將與 Connector for SCEP 搭配使用的鏈結根 CA 憑證上傳至 Microsoft Intune 以建立信任。如需有關如何建立信任憑證組態設定檔的資訊，請參閱 Microsoft Intune 文件中的 Microsoft Intune 的信任根憑證設定檔。
2. 建立 SCEP 憑證組態描述檔，在您的裝置需要新憑證時，將裝置指向連接器。組態設定檔的設定檔類型應該是 SCEP 憑證。對於組態設定檔的根憑證，請確定您使用在上一個步驟中建立的信任憑證。

對於 SCEP 伺服器 URLs，請從連接器的詳細資訊複製公有 SCEP URL 並貼到 SCEP 伺服器 URLs。若要檢視連接器的詳細資訊，請從適用於 SCEP 的連接器清單中選擇連接器。或者，您可以呼叫 ListConnectors 來取得 URL，然後從回應複製該`Endpoint`值。如需在 Microsoft Intune 中建立組態設定檔的指引，請參閱 Microsoft Intune 文件中的在 Microsoft Intune 中建立和指派 SCEP 憑證設定檔。

> **Note**
>
> 對於非 Mac 作業系統和 iOS 裝置，如果您未在組態設定檔中設定有效期間，Connector for SCEP 會發出有效時間為一年的憑證。如果您未在組態描述檔中設定擴充金鑰用量 (EKU) 值，Connector for SCEP 會使用 設定的 EKU 發出憑證`Client Authentication (Object Identifier: 1.3.6.1.5.5.7.3.2)`。對於 macOS 或 iOS 裝置，Microsoft Intune 不遵守組態設定檔中的 `ExtendedKeyUsage`或 `Validity` 參數。對於這些裝置，Connector for SCEP 透過用戶端身分驗證向這些裝置發出有效期為一年的憑證。

步驟 3：驗證與 Connector for SCEP 的連線

在您建立指向 Connector for SCEP 端點的 Microsoft Intune 組態描述檔之後，請確認已註冊的裝置可以請求憑證。若要確認，請確定沒有任何政策指派失敗。若要確認，請在 Intune 入口網站中導覽至**裝置 > 管理裝置 > 組態**，並確認組態政策指派失敗下沒有列出任何內容。如果有，請使用上述程序的資訊確認您的設定。如果您的設定正確且仍然失敗，請參閱從行動裝置收集可用資料。

如需有關裝置註冊的資訊，請參閱 Microsoft Intune 文件中的什麼是裝置註冊？。

設定適用於 SCEP 連接器的 Omnissa Workspace ONE

您可以使用 AWS 私有 CA 做為 Omnissa Workspace ONE UEM（統一端點管理）系統的外部憑證授權機構 (CA)。本指南說明如何在建立 SCEP 連接器後設定 Omnissa Workspace ONE AWS。

先決條件

為 Omnissa Workspace ONE 建立 SCEP 連接器之前，您必須完成下列先決條件：

- 在 AWS 主控台中建立私有 CA。如需詳細資訊，請參閱在 中建立私有 CA AWS 私有 CA。
- 建立一般用途 SCEP 連接器。如需詳細資訊，請參閱建立連接器。
- 擁有具有組織群組 ID 的作用中 Omnissa Workspace ONE 環境管理員帳戶。

- 如果您要註冊 Apple 裝置，請為 MDM 設定 Apple 推播通知服務 APNs)。如需詳細資訊，請參閱 Omnissa 文件中的 APNs 憑證。

步驟 1：在 Omnissa Workspace ONE 中定義憑證授權單位和範本

在 AWS 主控台中建立私有 CA 和 SCEP 連接器之後，請在 Omnissa Workspace ONE 中定義憑證授權單位和範本。

新增 AWS 私有 CA 做為憑證授權單位

1. 從系統功能表中，選擇企業整合，然後選擇憑證授權機構。
2. 選擇 + ADD 並提供下列資訊：
   - 名稱： AWS-Private-CA。
   - 描述： AWS 私有 CA 裝置憑證發行。
   - 授權類型：選取一般 SCEP。
   - SCEP URL：輸入 SCEP URL AWS 私有 CA。
   - 挑戰類型：選取 STATIC。
   - 靜態挑戰：從 AWS 主控台的 Connector for SCEP 組態輸入 SCEP 靜態挑戰密碼。
   - 輸入重試逾時和重試次數上限值。
3. 儲存組態。

建立憑證範本

1. 從系統功能表中，選擇企業整合，選擇憑證授權機構，然後選擇範本。
2. 選擇新增範本並提供下列資訊：
   - 範本名稱：Device-Cert-Template。
   - 憑證授權單位：選擇 AWS-Private-CA。
   - 主旨名稱：這是可自訂的欄位。您可以從屬性清單中選擇變數值。例如，CN={DeviceReportedName}、O={DevicePlatform}、OU={CustomAttribute1}
   - 私有金鑰長度：2048 位元。
   - 私有金鑰類型：視需要選取簽署和加密
   - 自動續約：已啟用/已停用 （根據您的需求）。
3. 儲存範本。

步驟 2：設定 Omnissa Workspace ONE UEM 設定檔組態

在 Omnissa Workspace ONE UEM 中建立設定檔，將裝置導向 Connector for SCEP 以發出憑證。

建立憑證分佈的 SCEP 裝置設定檔

1. 從資源功能表中，選擇設定檔和基準，然後選擇設定檔。
2. 選擇新增，然後選擇新增設定檔
3. 選取裝置平台 (Android、iOS、macOS、Windows)。
4. 視需要設定 管理類型和內容。
5. 設定名稱：Device-Cert-Profile。
6. 捲動至 SCEP 承載。
7. 選取 SCEP，然後選擇 +新增。
8. 使用下列組態：
   - SCEP：
     - 針對登入資料來源，選取定義的憑證授權單位 （預設）。
     - 針對憑證授權單位，選取 AWS-Private-CA
     - 針對憑證範本，選取步驟 1 中定義的 Device-Cert-Template。
9. 選擇下一步，然後在指派區段中，從清單中選擇正確的智慧群組 （裝置的指派群組）。
10. 選取指派類型為自動以啟用自動續約。
11. 儲存和發佈設定檔。

> **Note**
>
> 如需詳細資訊，請參閱 Omnissa 文件中的 SCEP。

步驟 3：在 Omnissa Workspace ONE 中註冊裝置

建立或驗證智慧群組

1. 從群組和設定中選擇群組，然後選擇指派群組。
2. 建立或編輯 POC-Devices 智慧群組：

- 名稱：POC-Devices。
- 裝置類型：選取全部或特定平台（例如 Android 或 iOS)。
- 條件：使用 UserGroup、平台和作業系統、OEM 和模型來指定將目標裝置分組的條件。
- 擁有權：為個人或公司裝置選取任何。

3. 儲存並確認目標裝置會出現在預覽索引標籤中。

手動裝置註冊

Android

- 從 Google Play 下載 Workspace ONE Intelligent Hub 應用程式。
- 開啟應用程式並輸入註冊 URL 或掃描 QR 碼。
- 登入並依照提示註冊為 MDM 受管裝置。

iOS/macOS

- 在裝置上，開啟 Safari 並導覽至註冊 URL（例如 https://<WorkspaceONEUEMHostname>/enroll)。
- 使用使用者登入資料登入。
- 從 App Store 下載並安裝 Workspace ONE Intelligent Hub 應用程式。
- 依照設定 > 一般 > VPN & 裝置管理 > 設定檔 > 安裝中的提示安裝 MDM 設定檔。

Windows

- 從工作區 ONE 伺服器或 Microsoft Store 下載工作區 ONE Intelligent Hub。
- 使用註冊 URL 和登入資料透過 Hub 註冊。

在裝置 > 列出檢視 > 更多動作 > 指派給智慧群組中，將已註冊的裝置指派給 POC-Devices 智慧群組。

如需詳細資訊，請參閱 Omnissa 文件中的自動化裝置註冊。

驗證註冊

1. 在 Omnissa Workspace ONE UEM 主控台中，前往裝置，然後列出檢視。
2. 確認您的已註冊裝置顯示狀態設為已註冊。
3. 確認裝置位於裝置詳細資訊的群組索引標籤中的 POC-Devices 智慧群組中。

步驟 4：發行憑證

發出憑證的觸發程序

1. 在裝置清單檢視中，選取已註冊的裝置。
2. 選擇查詢按鈕以提示簽入。
3. Device-Cert-Profile 應該透過 發出憑證 AWS 私有 CA。

驗證憑證安裝

Android

選擇設定、安全性、信任的登入資料，然後選擇使用者來驗證憑證。

iOS

前往設定，然後選擇一般，然後選擇 VPN 和裝置管理，然後選擇組態設定檔。確認來自 AWS-Private-CA 的憑證存在。

macOS

開啟 Keychain Access，然後開啟 System Keychain 並驗證憑證。

Windows

開啟 certmgr.msc，然後開啟 Personal，然後開啟 Certificates 來驗證憑證。

疑難排解

SCEP 錯誤（例如「22013」-SCEP 伺服器傳回無效的回應」）

- 驗證工作區 ONE 中的 SCEP URL 和靜態挑戰密碼相符 AWS 私有 CA。
- 測試 SCEP 端點連線：curl <SCEP_URL>。
- 檢查 AWS CloudTrail 日誌是否有 AWS 私有 CA 錯誤 (`IssueCertificate` 失敗等）。

APNs問題 (iOS/macOS)

- 確定 APNs憑證有效，並指派給正確的組織群組。
- 測試 APNs連線能力：telnet gateway.push.apple.com 2195。

設定檔安裝失敗

- 確認裝置位於正確的智慧型群組 (裝置、清單檢視，以及群組)。
- 強制設定檔同步：更多動作，然後傳送，然後設定檔清單。

日誌

- Android：使用 Logcat 或工作區 ONE 日誌。
- iOS/macOS：log show --predicate 'process == "mdmclient"' --last 1h （透過 Xcode/Apple Configurator)。
- Windows：事件檢視器，然後應用程式和服務日誌，然後 Microsoft-Windows-DeviceManagement。
- Workspace ONE UEM：監控，然後報告和分析，然後是事件，然後是裝置事件。

如需 中用於 SCEP 監控的詳細 Connector AWS，請參閱 Monitor Connector for SCEP。

安全考量

- 安全地存放 SCEP URLs 和秘密。如需詳細資訊，請參閱 AWS Secrets Manager 服務。
- 將智慧型群組條件限制為僅限目標裝置。
- 定期續約 Apple 推播通知 (APNs) 憑證 （有效期為 1 年）。
- 為概念驗證專案設定短憑證有效期間，以將風險降至最低。
- 對於個人裝置，請確定清除會移除所有設定檔和憑證。

如需有關如何使用 SCEP 連接器設定 Omnissa Workspace ONE UEM 和 CA 整合的資訊，請參閱 Omnissa Workspace ONE 文件中的 SCEP。

Monitor Connector for SCEP

監控是維護 Connector for SCEP 及其他 AWS 解決方案的可靠性、可用性和效能的重要部分。 AWS 提供下列監控工具來監看 Connector for SCEP、在發生錯誤時回報，並適時採取自動動作：

- AWS CloudTrail 擷取您 AWS 帳戶 發出或代表發出的 API 呼叫和相關事件，並傳送日誌檔案至您指定的 Amazon S3 儲存貯體。您可以識別哪些使用者和帳戶稱為 AWS APIs、進行呼叫的來源 IP 地址，以及呼叫的時間。

  如果您監控 CloudTrail 資料事件，日誌會包含來自用戶端裝置的所有最近請求清單。資料事件隨附識別用戶端裝置資訊，例如 IP 地址、執行的操作類型，以及操作導致`failed`狀態時的錯誤碼和詳細訊息。如需詳細資訊，請參閱「AWS CloudTrail 使用者指南」。
- Amazon EventBridge 為無伺服器事件匯流排服務，可讓您輕鬆將應用程式與來自各種來源的資料互相連線。EventBridge 會從您自己的應用程式、Software-as-a-Service(SaaS) 應用程式 AWS 和服務

提供即時資料串流，並將該資料路由至 Lambda 和 CloudWatch Logs 等目標。這可讓您監控在服務中發生的事件，並建置事件導向的架構。如需詳細資訊，請參閱 Amazon EventBridge 使用者指南。

主題

- 使用 EventBridge 自動化適用於 SCEP 的連接器
- 使用 的 Log Connector for SCEP API 呼叫 AWS CloudTrail

使用 EventBridge 自動化適用於 SCEP 的連接器

您可以使用 Amazon EventBridge 自動化您的 AWS 服務，並自動回應系統事件，例如應用程式可用性問題或資源變更。 AWS 服務的事件會以接近即時的方式傳送到 EventBridge。您可撰寫簡單的規則，指出您在意的事件，以及當事件符合規則時所要自動執行的動作。EventBridge 至少發佈一次。如需詳細資訊，請參閱建立對 EventBridge 中的事件做出反應的規則。

CloudWatch Events 會使用 EventBridge 轉換為動作。使用 EventBridge，您可以使用事件來觸發目標。如需詳細資訊，請參閱什麼是 Amazon EventBridge？

Connector for SCEP 事件類型

憑證發行成功

當我們發出憑證以回應`PkiOperationPost`請求時，Connector for SCEP 會將`Certificate Issuance Succeeded`事件傳送至 EventBridge。

以下是事件的範例資料。

```
{
   "version": "0",
   "id": "event_ID",
   "detail-type": "Certificate Issuance Succeeded",
   "source": "aws.pca-connector-scep",
   "account": "account",
   "time": "2024-09-12T19:14:56Z",
   "region": "region",
   "resources":[
       "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
       "arn:aws:pca-connector-scep:us-
east-1:111122223333:connector/11223344-1234-1122-2233-112233445566"
```

```
   ],
   "detail": {
       "result": "success",
       "requestType": "PkiOperationPost",
       "certificateArn": "arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID"
   }
}
```

憑證發行失敗

當我們無法發出憑證以回應PkiOperationPost請求時，Connector for SCEP 會將Certificate Issuance Failed事件傳送至 EventBridge。

以下是事件的範例資料。

```
{
   "version": "0",
   "id": "event_ID",
   "detail-type": "Certificate Issuance Failed",
   "source": "aws.pca-connector-scep",
   "account": "account",
   "time": "2024-09-12T19:14:56Z",
   "region": "region",
   "resources":[
       "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
       "arn:aws:pca-connector-scep:us-
east-1:111122223333:connector/11223344-1234-1122-2233-112233445566"
   ],
   "detail": {
       "result": "failure",
       "requestType": "PkiOperationPost",
       "reason": "The certificate authority is not active."
   }
}
```

憑證授權單位憑證擷取成功

當我們收到GetCACert請求並成功擷取連接器的私有 CA 憑證時，Connector for SCEP 會將Certificate Authority Certificate Retrieval Succeeded事件傳送至 EventBridge。

以下是事件的範例資料。

```
{
   "version": "0",
   "id": "event_ID",
   "detail-type": "Certificate Authority Certificate Retrieval Succeeded",
   "source": "aws.pca-connector-scep",
   "account": "account",
   "time": "2024-09-12T19:14:56Z",
   "region": "region",
   "resources":[
       "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
       "arn:aws:pca-connector-scep:us-
east-1:111122223333:connector/11223344-1234-1122-2233-112233445566"
   ],
   "detail": {
       "result": "success",
       "requestType": "GetCACert"
   }
}
```

憑證授權機構憑證擷取失敗

當我們收到GetCACert請求且無法擷取連接器的私有 CA 憑證時，Connector for SCEP 會將Certificate Authority Certificate Retrieval Failed事件傳送至 EventBridge。事件包含失敗的原因。

以下是事件的範例資料。

```
{
   "version": "0",
   "id": "event_ID",
   "detail-type": "Certificate Authority Certificate Retrieval Failed",
   "source": "aws.pca-connector-scep",
   "account": "account",
   "time": "2024-09-12T19:14:56Z",
   "region": "region",
   "resources":[
       "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
       "arn:aws:pca-connector-scep:us-
east-1:111122223333:connector/11223344-1234-1122-2233-112233445566"
   ],
   "detail": {
```

```
       "result": "failure",
       "requestType": "GetCACert",
       "reason": "The certificate authority certificate validity must be at least one
 year from today."
   }
}
```

憑證授權單位憑證擷取成功

當我們收到GetCACert請求並成功擷取連接器的私有 CA 憑證時，Connector for SCEP 會將Certificate Authority Certificate Retrieval Succeeded事件傳送至 EventBridge。

以下是事件的範例資料。

```
{
   "version": "0",
   "id": "event_ID",
   "detail-type": "Certificate Authority Certificate Retrieval Succeeded",
   "source": "aws.pca-connector-scep",
   "account": "account",
   "time": "2024-09-12T19:14:56Z",
   "region": "region",
   "resources":[
       "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
       "arn:aws:pca-connector-scep:us-
east-1:111122223333:connector/11223344-1234-1122-2233-112233445566"
   ],
   "detail": {
       "result": "success",
       "requestType": "GetCACert"
   }
}
```

憑證授權單位功能擷取成功

當我們收到 SCEP GetCACaps請求並成功擷取 CA 的功能時，Connector for SCEP 會將Certificate Authority Capabilities Retrieval Succeeded事件傳送至 EventBridge。

以下是事件的範例資料。

憑證授權機構功能擷取失敗

當我們收到 SCEP GetCACaps請求且無法擷取 CA 的功能時，Connector for SCEP 會將Certificate Authority Capabilities Retrieval Failed事件傳送至 EventBridge。我們在事件中包含失敗的原因。

以下是事件的範例資料。

```
{
 "resources":
     [
     "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
     "arn:aws:pca-connector-scep:us-
east-1:111122223333:connector11223344-1234-1122-2233-112233445566"
     ],
 "detailType":"Certificate Authority Capabilities Retrieval Failed",
 "detail": {
     "result":"failure",
     "requestType":"GetCACaps",
     "reason":"The request was denied due to request throttling."
 },
 "source":"aws.pca-connector-scep","accountId":"111122223333"
 }
```

叫用不支援的操作

叫用不支援的操作

如果傳送至連接器端點的操作不受支援或未知，Connector for SCEP 會將Unsupported Operation Invoked事件傳送至 EventBridge。

```
{
   "version": "0",
   "id": "event_ID",
   "detail-type": "Unsupported Operation Invoked",
   "source": "aws.pca-connector-scep",
   "account": "account",
   "time": "2024-09-12T19:14:56Z",
   "region": "region",
   "resources":[
       "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
```

```
        "arn:aws:pca-connector-scep:us-
east-1:111122223333:connector/11223344-1234-1122-2233-112233445566"
    ],
    "detail": {}
}
```

建立 EventBridge 規則

在 EventBridge 中，您可以建立回應 CloudTrail 記錄之事件的規則。若要建立包含 Connector for SCEP 記錄的所有事件的規則，請將來源設定為 `aws.pca-connector-scep`。如需規則的詳細資訊，請參閱在 Amazon EventBridge 中建立規則。

使用 的 Log Connector for SCEP API 呼叫 AWS CloudTrail

Connector for Simple Certificate Enrollment Protocol (SCEP) 已與 服務整合 AWS CloudTrail，此服務可提供使用者、角色、用戶端或 AWS 服務所採取動作的記錄。CloudTrail 會將 Connector for SCEP 的所有 API 呼叫擷取為事件。擷取的呼叫包括來自 Connector for SCEP 主控台的呼叫，以及對 Connector for SCEP API 操作的程式碼呼叫。如果您建立線索，您可以將 CloudTrail 事件持續交付至 Amazon S3 儲存貯體，包括 Connector for SCEP 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。您可以使用 CloudTrail 所收集的資訊，判斷對 Connector for SCEP 提出的請求、提出請求的 IP 地址、提出請求的人員、提出請求的時間，以及其他詳細資訊。

若要進一步了解 CloudTrail，請參閱「AWS CloudTrail 使用者指南」。

CloudTrail 中 SCEP 資訊的連接器

當您建立帳戶 AWS 帳戶 時，您的 上會啟用 CloudTrail。在 Connector for SCEP 中發生活動時，該活動會與事件歷史記錄中的其他服務 AWS 事件一起記錄在 CloudTrail 事件中。您可以在 中檢視、搜尋和下載最近的事件 AWS 帳戶。如需詳細資訊，請參閱「使用 CloudTrail 事件歷史記錄檢視事件」。

如需持續記錄 中的事件 AWS 帳戶，包括 Connector for SCEP 的事件，請建立追蹤。線索能讓 CloudTrail 將日誌檔案交付至 Amazon S3 儲存貯體。依預設，當您在主控台中建立追蹤時，該追蹤會套用至所有的 AWS 區域。線索會記錄 AWS 分割區中所有區域的事件，並將日誌檔案傳送到您指定的 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 日誌中收集的事件資料。如需詳細資訊，請參閱下列內容：

- 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 Connector for SCEP 動作，並記錄在 Connector for SCEP API 參考中。例如，對 `CreateConnector`、`GetConnector` 和 `CreateChallenge` 動作發出的呼叫會在 CloudTrail 記錄檔案中產生專案。

每一筆事件或日誌專案都會包含產生請求者的資訊。身分資訊可協助您判斷下列事項：

- 是否使用根或 AWS Identity and Access Management (IAM) 使用者登入資料提出請求。
- 提出該請求時，是否使用了特定角色或聯合身分使用者的暫時安全憑證。
- 請求是否由其他 AWS 服務提出。
- 請求是否由 SCEP 用戶端裝置提出。

如需詳細資訊，請參閱 CloudTrail userIdentity 元素。

適用於 SCEP 管理事件的連接器

Connector for SCEP 與 CloudTrail 整合，以記錄使用者、角色或 Connector for SCEP 中的 AWS 服務所做的 API 動作。您可以使用 CloudTrail 即時監控 Connector for SCEP API 請求，並將日誌存放在 Amazon Simple Storage Service、Amazon CloudWatch Logs 和 Amazon CloudWatch Events 中。Connector for SCEP 支援將下列動作記錄為 CloudTrail 日誌檔案中的事件：

- CreateChallenge
- CreateConnector
- GetConnector
- GetChallengeMetadata
- GetChallengePassword
- DeleteConnector
- DeleteChallenge

CloudTrail 中 SCEP 資料事件的連接器

資料事件提供有關在 資源上執行或在資源中執行的資源操作的資訊，例如，當您的用戶端傳送 SCEP `GetCACaps` 訊息到連接器端點時。這些也稱為資料平面操作。資料事件通常是大量資料的活動。根據預設，CloudTrail 不會記錄任何資料事件，CloudTrail 事件歷史記錄也不會記錄這些資料事件。

資料事件需支付額外的費用。如需 CloudTrail 定價的詳細資訊，請參閱 AWS CloudTrail 定價。

您可以使用 CloudTrail 主控台 AWS CLI或 CloudTrail API 操作來記錄AWS::PCAConnectorSCEP::Connector資源類型的資料事件。如需如何記錄資料事件的詳細資訊，請參閱 AWS CloudTrail 使用者指南中的使用 AWS 管理主控台　記錄資料事件和使用 AWS Command Line Interface記錄資料事件。

下表列出您可以記錄資料事件的 Connector for SCEP 資源類型。資料事件類型 (主控台) 資料行會顯示從 CloudTrail 主控台上的資料事件類型清單中選擇的值。resources.type 值欄會顯示值，您會在使用 AWS CLI 或 CloudTrail APIs 設定進階事件選取器時指定該resources.type值。記錄到 CloudTrail 的資料 API 資料行會針對資源類型顯示記錄到 CloudTrail 的 API 呼叫。

資料事件類型 (主控台)	resources.type 值	記錄到 CloudTrail 的資料 API
連接器	AWS::PCAConnectorSCEP::Connector	• PKIOperationGet - 如果對連接器的資料平面端點提出包含PKCSReq訊息的 HTTP GET SCEP 請求，且該訊息的操作設定為 ，則會產生此請求PKIOperation 。 • PKIOperationPost - 如果對連接器的資料平面端點提出包含PKCSReq訊息的 HTTP POST SCEP 請求，且該訊息的操作設定為 ，則會產生此請求PKIOperation 。 • GetCACaps - 如果向連接器的資料平面端點發出包含GetCACaps 訊息的 SCEP 請求，則產生。 • GetCACert - 如果向連接器的資料平面端點發出包含GetCACert 訊息的 SCEP 請求，則產生。

您可以設定進階事件選取器來篩選 eventName、readOnly 和 resources.ARN 欄位，以僅記錄對您重要的事件。下列範例是資料事件組態的 JSON 檢視，該組態僅記錄特定函數的事件。如需有關這些欄位的詳細資訊，請參閱《AWS CloudTrail API 參考》中的 AdvancedFieldSelector。

```
[
  {
    "name": "connector-scep-events",
    "fieldSelectors": [
      {
        "field": "eventCategory",
        "equals": [
          "Data"
        ]
      },
      {
        "field": "resources.type",
        "equals": [
          "AWS::PCAConnectorSCEP::Connector"
        ]
      },
      {
        "field": "resources.ARN",
        "equals": [
          "arn:aws:pca-connector-scep:US West (N.
 California):111122223333:connector/11223344-1122-2233-3344-cae95a00d2a7"
        ]
      }
    ]
  }
]
```

範例項目

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。一個事件為任何來源提出的單一請求，並包含請求動作、請求的日期和時間、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。

範例 1：管理事件、 **CreateConnector**

以下範例顯示的是展示 CreateConnector 動作的 CloudTrail 日誌項目。

```
{
        "eventVersion": "1.09",
        "userIdentity": {
          "type": "AssumedRole",
          "principalId": "AABB1122CCDD4455HHJJ1:11cc33nn2a97724dc48a89071111111111",
          "arn": "arn:aws:sts::111122223333:assumed-role/Admin",
          "accountId": "111122223333",
          "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
          "sessionContext": {
              "sessionIssuer": {
                  "type": "Role",
                  "principalId": "AABB1122CCDD4455HHJJ1",
                  "arn": "arn:aws:iam::111122223333:role/Admin",
                  "accountId": "111122223333",
                  "userName": "my-user-name"
              },
              "attributes": {
                  "creationDate": "2024-08-16T17:46:41Z",
                  "mfaAuthenticated": "false"
              }
          }
        },
        "eventTime": "2024-08-16T17:48:07Z",
        "eventSource": "pca-connector-scep.amazonaws.com",
        "eventName": "CreateConnector",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "10.0.0.0",
        "userAgent": "Python/3.11.8 Darwin/22.6.0 exe/x86_64",
        "requestParameters": {
          "ClientToken": "11223344-2222-3333-4444-666555444555",
          "CertificateAuthorityArn": "arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/a1b2c3d4-5678-90ab-cdef-EXAMPLE22222"
        },
        "responseElements": {
          "ConnectorArn": "arn:aws:pca-connector-scep:us-east-1:111122223333:connector/
a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
        },
        "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
        "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
        "readOnly": false,
        "eventType": "AwsApiCall",
        "managementEvent": true,
        "recipientAccountId": "111122223333",
```

```
        "eventCategory": "Management"
        }
```

範例 2：管理事件、 **CreateChallenge**

以下範例顯示的是展示 CreateChallenge 動作的 CloudTrail 日誌項目。

```
{
        "eventVersion": "1.09",
        "userIdentity": {
          "type": "AssumedRole",
          "principalId": "AABB1122CCDD4455HHJJ1:11cc33nn2a97724dc48a89071111111111",
          "arn": "arn:aws:sts::111122223333:assumed-role/Admin",
          "accountId": "111122223333",
          "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
          "sessionContext": {
              "sessionIssuer": {
                  "type": "Role",
                  "principalId": "AABB1122CCDD4455HHJJ1",
                  "arn": "arn:aws:iam::111122223333:role/Admin",
                  "accountId": "111122223333",
                  "userName": "user-name"
              },
              "attributes": {
                  "creationDate": "2024-08-16T17:46:41Z",
                  "mfaAuthenticated": "false"
              }
          }
        },
        "eventTime": "2024-08-16T17:47:52Z",
        "eventSource": "pca-connector-scep.amazonaws.com",
        "eventName": "CreateChallenge",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "10.0.0.0",
        "userAgent": "Python/3.11.8 Darwin/22.6.0 exe/x86_64",
        "requestParameters": {
          "ConnectorArn": "arn:aws:pca-connector-scep:us-east-1:111122223333:connector/
a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
          "ClientToken": "11223344-2222-3333-4444-666555444555"
        },
        "responseElements": {
          "Challenge": {
```

```
            "Arn": "arn:aws:pca-connector-scep:us-
east-1:111122223333:connector/9cac40bc-acba-412e-9a24-f255ef2fe79a/a1b2c3d4-5678-90ab-
cdef-EXAMPLE22222",
            "ConnectorArn": "arn:aws:pca-connector-scep:us-
east-1:111122223333:connector/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
            "CreatedAt": 1723830472.942,
            "Password": "***",
            "UpdatedAt": 1723830472.942
          }
        },
        "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
        "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
        "readOnly": false,
        "eventType": "AwsApiCall",
        "managementEvent": true,
        "recipientAccountId": "111122223333",
        "eventCategory": "Management"
        }
```

範例 3：管理事件、 **GetChallengePassword**

以下範例顯示的是展示 GetChallengePassword 動作的 CloudTrail 日誌項目。

```
{
        "eventVersion": "1.09",
        "userIdentity": {
          "type": "AssumedRole",
          "principalId": "AABB1122CCDD4455HHJJ1:11cc33nn2a97724dc48a89071111111111",
          "arn": "arn:aws:sts::111122223333:assumed-role/Admin",
          "accountId": "111122223333",
          "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
          "sessionContext": {
              "sessionIssuer": {
                  "type": "Role",
                  "principalId": "AABB1122CCDD4455HHJJ1",
                  "arn": "arn:aws:iam::111122223333:role/Admin",
                  "accountId": "905418114790",
                  "userName": "111122223333"
              },
              "attributes": {
                  "creationDate": "2024-08-16T17:55:01Z",
                  "mfaAuthenticated": "false"
              }
          },
```

```
          "invokedBy": "signin.amazonaws.com"
        },
        "eventTime": "2024-08-16T17:55:54Z",
        "eventSource": "pca-connector-scep.amazonaws.com",
        "eventName": "GetChallengePassword",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "10.0.0.0",
        "userAgent": "Python/3.11.8 Darwin/22.6.0 exe/x86_64",
        "requestParameters": {
          "ChallengeArn": "arn:aws:pca-connector-scep:us-east-1:111122223333:challenge/
a1b2c3d4-5678-90ab-cdef-EXAMPLE33333"
        },
        "responseElements": null,
        "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
        "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
        "readOnly": true,
        "eventType": "AwsApiCall",
        "managementEvent": true,
        "recipientAccountId": "111122223333",
        "eventCategory": "Management"
        }
```


範例 4：資料事件、 **PkiOperationPost**

下列範例顯示示範失敗PkiOperationPost呼叫的 CloudTrail 日誌項目。日誌包含錯誤代碼和錯誤訊息，以及失敗的說明。


```
{
        "eventVersion": "1.10",
        "userIdentity": {
          "type": "FederatedUser",
          "principalId": "111122223333",
          "accountId": "111122223333"
        },
        "eventTime": "2024-08-16T17:40:09Z",
        "eventSource": "pca-connector-scep.amazonaws.com",
        "eventName": "PkiOperationPost",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "10.0.0.0",
        "userAgent": "Python/3.11.8 Darwin/22.6.0 exe/x86_64",
        "errorCode": "BadRequestException",
```

```
        "errorMessage": "The certificate authority is not in a valid state for issuing
 certificates (Service: AcmPca, Status Code: 400, Request ID: a1b2c3d4-5678-90ab-cdef-
EXAMPLE55555)",
        "requestParameters": null,
        "responseElements": null,
        "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
        "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
        "readOnly": false,
        "resources": [
          {
              "accountId": "111122223333",
              "type": "AWS::PCAConnectorSCEP::Connector",
              "ARN": "arn:aws:pca-connector-scep:us-east-1:111122223333:connector/
a1b2c3d4-5678-90ab-cdef-EXAMPLE33333"
          }
        ],
        "eventType": "AwsApiCall",
        "managementEvent": false,
        "recipientAccountId": "905418114790",
        "eventCategory": "Data",
        "tlsDetails": {
          "clientProvidedHostHeader": "111122223333-a1b2c3d4-5678-90ab-cdef-
EXAMPLE33333.enroll.pca-connector-scep.us-east-1.api.aws"
        }
        }
```


針對適用於 SCEP 問題的 AWS 私有憑證授權單位 Connector 進行故障診斷

您可能需要疑難排解與 Connector for SCEP 實作相關的問題。本章提供有關服務傳送的 HTTP 和用戶端錯誤的詳細資訊。

主題

- 從 Connector for SCEP 疑難排解 HTTP 錯誤
- Connector for SCEP 用戶端錯誤故障診斷

從 Connector for SCEP 疑難排解 HTTP 錯誤

當您的用戶端觸發 Connector for SCEP 資料平面 API 動作並導致錯誤時，Connector for SCEP 會傳送 HTTP 回應碼給請求用戶端，其中包含錯誤的相關資訊。

除了直接提供給用戶端的服務回應之外，您還可以使用 Monitor Connector for SCEP節所述的監控工具來檢視和偵錯導致 HTTP 錯誤的錯誤。

以下是 服務傳回給 SCEP 用戶端的錯誤訊息、潛在原因，以及您可以採取的步驟來解決問題。

HTTP 400 錯誤請求

HTTP 400 回應碼表示 Connector for SCEP 因明顯的用戶端錯誤而無法處理請求，例如請求中的資料遺失或無效。如果錯誤是由 SCEP-protocol 特定錯誤造成，Connector for SCEP 會將 SCEP 回應作為二進位包含在訊息中。Connector for SCEP APIs 可以傳回 400 個回應，原因如下。

回應標頭 (x-amzn-ErrorType)	錯誤訊息 (x-amzn-ErrorMessage)	根本原因	修補	包含 SCEP 回應？
LimitExceededException	超過憑證授權單位發行限制。	與連接器相關聯的私有憑證授權機構 (CA) 已超過其可發行的憑證數量配額。	SCEP 連接器在生命週期內只能連接到一個私有 CA。如果您已用盡私有 CA 的限制，請建立新的連接器或請求增加配額。如需私有 CA 配額的詳細資訊，請參閱AWS 私有憑證授權單位 配額。	否
ValidationException	請求必須包含 base64。	Connector for SCEP 無法處理 HTTP GET 請求，因為內文不是有效的 Base64。	如果可能，請將您的用戶端設定為使用 HTTP POST 訊息，而非 HTTP GET 訊息。如果您必須使用 HTTP GET，訊息必須使用 Base64 格式。如果您的用戶端與這些要求不相容，請聯絡 AWS 支援 尋求協助。	否

回應標頭 (x-amzn-ErrorType)	錯誤訊息 (x-amzn-ErrorMessage)	根本原因	修補	包含 SCEP 回應？
ValidationException	憑證授權機構未處於作用中狀態。	與連接器相關聯的私有 CA 處於非作用中狀態。	重新啟用私有 CA。如需相關資訊，請參閱在 中更新私有 CA AWS 私有憑證授權單位。	否
ValidationException	憑證授權單位憑證有效性必須至少為從今天起算一年。	與一般用途連接器相關聯的私有 CA 必須自今天起有一年的有效期間。	重新發行有效期間大於自今天起算一年的憑證。如需管理憑證的資訊，請參閱 管理私有 CA 生命週期。	否
ValidationException	請求中包含的憑證已過期。	服務接收時，用戶端裝置在每個交易上產生的暫時性憑證已過期。	您的用戶端裝置很可能未正確設定時間設定，而且正在建立日期晚於即時的憑證。如果您無法解決此問題，請聯絡 AWS 支援 尋求協助。	否
ValidationException	請求包含無效的密碼編譯訊息語法。	服務無法解碼 SCEP 請求訊息。	檢查您的 SCEP 訊息是否符合 SCEP RFC 8894 中定義的密碼編譯訊息語法。如果您無法解決此問題，請聯絡 AWS 支援 尋求協助。	否

回應標頭 (x-amzn-ErrorType)	錯誤訊息 (x-amzn-ErrorMessage)	根本原因	修補	包含 SCEP 回應？
ValidationException	連接器未處於作用中狀態。	連接器的狀態為非作用中。	您可以在 主控台或 API 中的狀態欄位中找到連接器的狀態。連接器的狀態可以是建立、作用中、刪除或失敗。如果正在建立狀態，請稍後嘗試您的請求。如果狀態失敗，請檢視故障診斷問題的狀態原因，然後建立新的連接器。	否
ValidationException	請求中必須包含有效的憑證。	來自用戶端的請求訊息中包含的暫時性憑證遺失或無效。	與 SCEP 相容的用戶端必須提供自我簽署的憑證來驗證自己。如果您的用戶端無法提供所需的自我簽署憑證，請聯絡 AWS 支援 尋求協助。	否
ValidationException	請求 URI 無效。	Connector for SCEP 無法剖析請求，因為請求的 URI 路徑或查詢無效。	管理員應驗證用戶端裝置的組態設定，通常透過行動裝置管理 (MDM) 系統進行管理。如需詳細資訊，請參閱步驟 2：將連接器詳細資訊複製到 MDM 系統。	否

回應標頭 (x-amzn-ErrorType)	錯誤訊息 (x-amzn-ErrorMessage)	根本原因	修補	包含 SCEP 回應？
ValidationException	請求中需要一個主機標頭。	用戶端未在請求中提供有效的 HTTP 主機標頭，這是處理請求的必要項目。	需要 HTTP 主機標頭才能區分來自不同連接器的請求。如果您的用戶端無法提供所需的 HTTP 主機標頭，請聯絡 AWS 支援尋求協助。	否
ValidationException	無法解碼請求。請傳送有效的 SCEP 請求。	服務無法解碼和處理用戶端傳送的加密訊息語法 (CMS) 請求。	如果您的用戶端無法實作 SCEP，請記下回應中的請求 ID (`x-amzn-requestid`)，並聯絡 AWS 支援。	否

回應標頭 (x-amzn-ErrorType)	錯誤訊息 (x-amzn-ErrorMessage)	根本原因	修補	包含 SCEP 回應？
ValidationException	回應無法使用衍生自請求的值進行編碼。請傳送有效的 SCEP 請求。	服務無法編碼 SCEP 回應。	當服務無法使用提供的請求者憑證來正確編碼 SCEP 回應訊息時，通常會發生此問題。例如，如果請求者憑證具有 SCEP 連接器不支援的橢圓曲線數位簽章演算法 (ECDSA) 金鑰，就可能會發生這種情況。 如果您遇到此問題，請先將 MDM 或 SCEP 用戶端設定為使用 RSA。如果您仍然無法解決問題，請記下回應中的請求 ID (`x-amzn-requestid`)，並聯絡 AWS 支援 尋求協助。	否

回應標頭 (x-amzn-ErrorType)	錯誤訊息 (x-amzn-ErrorMessage)	根本原因	修補	包含 SCEP 回應？
ValidationException	不支援的演算法：<OID>	請求已由不支援的密碼編譯演算法簽署或加密。	我們的服務不支援特定過時且較弱的密碼編譯演算法。此資訊會透過 `GetCACaps` 請求傳達給用戶端。不過，有些用戶端可能不會使用此方法來檢查支援的演算法。 如果您的用戶端似乎與我們的服務支援的密碼編譯演算法不相容，請聯絡 AWS 支援 尋求協助。	否

回應標頭 (x-amzn-ErrorType)	錯誤訊息 (x-amzn-ErrorMessage)	根本原因	修補	包含 SCEP 回應？
ValidationException	不支援的 PkiOperation messageType。	請求訊息包含無效的 `PkiOperation` 訊息類型，且服務無法處理。	我們的服務僅支援 RFC 8894 中定義的 SCEP 通訊協定訊息類型子集。具體而言，我們辨識並處理下列訊息類型：CertRep、PKCSReq、GetCert、GetCRL 和 CertPoll。 我們透過 GetCACaps 方法將支援的訊息類型傳達給用戶端。不幸的是，有些用戶端可能並未使用此方法，而且可能不符合我們服務的功能。 如果您的用戶端似乎與我們的服務支援的 SCEP 訊息類型不相容，請聯絡 AWS 支援。	否

回應標頭 (x-amzn-ErrorType)	錯誤訊息 (x-amzn-ErrorMessage)	根本原因	修補	包含 SCEP 回應？
BadRequestException	挑戰密碼無效。	用戶端提供的挑戰密碼對已聯絡的服務端點及其相關聯的連接器無效。挑戰密碼是 SCEP 通訊協定中定義的必要安全措施，以確保只有授權的用戶端才能存取服務。	請確定您的用戶端在請求中提供正確的挑戰密碼。您可以在主控台或 GetChallengePassword API 的連接器詳細資訊中找到 。如需詳細資訊，請參閱步驟 2：將連接器詳細資訊複製到 MDM 系統。	是
BadRequestException	憑證簽署請求中需要一個挑戰密碼。	用戶端在請求中提供零或多個挑戰密碼。	請確定您的用戶端在其請求中提供了一個挑戰密碼。您可以在 主控台或透過 GetChallengePassword API 在連接器的詳細資訊中找到挑戰密碼。如需詳細資訊，請參閱步驟 2：將連接器詳細資訊複製到 MDM 系統。	是
BadRequestException	連接器無法存取 Azure。	Connector for Microsoft Intune 透過 Microsoft Intune 授權用戶端請求。這需要您授予 Connector for SCEP 存取 Azure 資源的許可。	設定 中詳述的許可步驟 1：授予 AWS 私有 CA 許可以使用您的 Microsoft Entra ID 應用程式。	是

回應標頭 (x-amzn-ErrorType)	錯誤訊息 (x-amzn-ErrorMessage)	根本原因	修補	包含 SCEP 回應？
BadRequestException	Azure 應用程式無法存取 來執行 <action>。	Connector for Microsoft Intune 透過 Microsoft Intune 授權用戶端請求。這需要您授予 Connector for SCEP 存取 Azure 資源的許可。	設定 中詳述的許可步驟 1：授予 AWS 私有 CA 許可以使用您的 Microsoft Entra ID 應用程式。	是
BadRequestException	找不到 Azure 應用程式。	Connector for Microsoft Intune 透過 Microsoft Intune 授權用戶端請求。此錯誤表示您的 Microsoft Entra ID 中沒有應用程式註冊，或連接器的 Intune 詳細資訊設定錯誤。	請遵循 設定適用於 SCEP 連接器的 Microsoft Intune主題中的指引。	是
BadRequestException	Intune 憑證簽署請求驗證失敗。原因：<reason>	Connector for Microsoft Intune 透過 Microsoft Intune 授權用戶端請求。此錯誤訊息表示 Intune 驗證程序失敗，並提供對應的 Intune 錯誤碼。	請遵循 設定適用於 SCEP 連接器的 Microsoft Intune主題中的指引。如果您的問題仍然存在，請聯絡 Microsoft Support。	是

回應標頭 (x-amzn-ErrorType)	錯誤訊息 (x-amzn-ErrorMessage)	根本原因	修補	包含 SCEP 回應？
BadRequestException	不支援的 PkiOperation messageType：<message type>。	請求訊息包含無效的訊息類型，且服務無法處理。	我們的服務僅支援 RFC 8894 中定義的 SCEP 通訊協定訊息類型子集。具體而言，我們辨識並處理下列訊息類型：CertRep、PKCSReq、GetCert、GetCRL 和 CertPoll。 我們透過 GetCACaps 方法將支援的訊息類型傳達給用戶端。遺憾的是，有些用戶端可能並未使用此方法，而且可能不符合我們服務的功能。 如果您的用戶端似乎與我們的服務支援的 SCEP 訊息類型不相容，請聯絡 AWS 支援。	是

回應標頭 (x-amzn-ErrorType)	錯誤訊息 (x-amzn-ErrorMessage)	根本原因	修補	包含 SCEP 回應？
BadRequestException	不支援金鑰演算法或長度。	服務不支援憑證簽署請求中包含的公有金鑰。	我們的服務僅支援高達 16，384 位元的標準 RSA 金鑰，以及高達 521 位元的 ECDSA 金鑰。如果您的用戶端需要使用目前不支援的演算法，請聯絡 AWS 支援 尋求協助。	是

HTTP 401 未授權

401 未經授權的回應狀態碼表示用戶端請求尚未完成，因為其缺少所請求資源的有效身分驗證憑證。

回應標頭 (x-amzn-ErrorType)	錯誤訊息 (x-amzn-ErrorMessage)	根本原因	修補	包含 SCEP 回應？
AccessDeniedException	連接器無法存取憑證授權單位。	Connector for SCEP 無法存取連接器相關聯的私有 CA。	使用 與 Connector for SCEP 共用您的私有 CA AWS Resource Access Manager。	否
AccountDoesNotExistException	AWS 帳戶不存在。	Connector for SCEP 資源不再存在。	擁有目標資源的帳戶已刪除。如果錯誤地這麼做，AWS 支援請在關閉後 90 天內聯絡 。	否

找不到 HTTP 404

HTTP 404 回應碼通常表示找不到您要尋找的資源。

回應標頭 (x-amzn-ErrorType	錯誤訊息 (x-amzn-ErrorMessage)	根本原因	修補	包含 SCEP 回應？
ResourceNotFoundException	憑證授權單位不存在。	連接器的關聯私有 CA 已刪除。	有一個寬限期，在此期間，如果私有憑證授權機構 (CA) 被錯誤刪除，則可以將其還原。如需詳細資訊，請參閱還原私有 CA。	否
ResourceNotFoundException	具有端點 <URL> 的連接器不存在。	用戶端裝置已嘗試連線到不屬於任何現有連接器的 URL。	請確定您的用戶端為連接器提供正確的端點。若要檢視連接器的 `Endpoint`，請呼叫 GetConnector API，或在主控台的連接器詳細資訊頁面中檢視它。	否

HTTP 409 衝突

HTTP 409 衝突回應表示與連接器相關聯的私有 CA 自啟動請求以來已變更。

回應標頭 (x-amzn-ErrorType)	錯誤訊息 (x-amzn-ErrorMessage)	根本原因	修補	包含 SCEP 回應？
				否

回應標頭 (x-amzn-ErrorType)	錯誤訊息 (x-amzn-ErrorMessage)	根本原因	修補	包含 SCEP 回應？
ConflictException	連接器自啟動請求以來已變更。	與連接器相關聯的私有 CA 已更新，觸發連接器內部憑證的輪換，用於透過 SCEP 與用戶端裝置通訊。 隨著新憑證的部署，此憑證輪換可能會在更新期間造成暫時性問題。不過，應該及時自動解決此錯誤。	請在幾分鐘後再次嘗試您的請求。如果問題無法解決，請聯絡 AWS 支援 尋求協助。	

HTTP 429 請求過多

Connector for SCEP 具有每個區域的帳戶層級配額。如果您超過對連接器的請求限制，您的請求將因 HTTP 429 錯誤而被拒絕。如果您需要增加配額，請參閱AWS 私有憑證授權單位 端點和配額。

回應標頭 (x-amzn-ErrorType)	錯誤訊息 (x-amzn-ErrorMessage)	根本原因	修補	包含 SCEP 回應？
ThrottlingException	由於請求調節，因此請求遭到拒絕。	已向此連接器發出太多請求，觸發一些請求遭到拒絕。 隨著新憑證的部署，此憑證輪換可能會在更新期間造成暫時性問題。不過，應該及時自動解決此錯誤。	如果您超過對連接器的請求限制，您的請求將被拒絕。如果您需要增加配額，請參閱 Connector for SCEP 端點和配額。	否

Connector for SCEP 用戶端錯誤故障診斷

使用下列指引來疑難排解與 Connector for SCEP 相關的用戶端錯誤。

訊息範例	根本原因	解決方案
不支援 ECDSA 金鑰	連接器連接到使用 ECDSA 金鑰而非 RSA 的私有 CA。雖然此服務支援 ECDSA 金鑰，但並非所有用戶端裝置都與此演算法相容。	請考慮使用 RSA 加密的私有 CA，而非 ECDSA。如果您建立使用 RSA 的私有 CA，則還需要建立新的連接器。連接器在生命週期內只能繫結至一個私有 CA。
加密或簽署憑證不存在	根據 RFC 8894，SCEP 服務會將中繼 CA 憑證傳回給用戶端。用戶端會使用這些憑證來執行加密和簽章驗證操作，做為 SCEP 通訊協定的一部分。 Connector for SCEP 使用相同的憑證進行加密和簽章驗證，這是常見的方法。不過，有些用戶端可能預期會有兩個不同的憑證。	如果您無法使用相容的用戶端，請聯絡 AWS 支援 尋求協助。

AWS 私有 CA 服務配額

AWS 私有 CA 會將配額指派給允許的憑證和憑證授權單位數量。API 動作的請求率也受限於帳戶 AWS 和區域特定的 quotas. AWS 私有 CA quotas。

AWS 私有 CA 會根據 API 操作，以不同的速率調節 API 請求。調節表示 AWS 私有 CA 拒絕其他有效的請求，因為請求超過每秒請求數的操作配額。調節請求時， 會 AWS 私有 CA 傳回 ThrottlingException 錯誤。 AWS 私有 CA 不保證 APIs的最低請求率。

若要查看可以調整的配額，請參閱 中的AWS 私有 CA 配額表AWS 一般參考。

您可以使用 Service Quotas 檢視目前的配額並請求增加 AWS 配額。

若要查看up-to-date AWS 私有 CA 配額清單

1. 登入 AWS 您的帳戶。
2. 開啟 Service Quotas 主控台，網址為 https://console.aws.amazon.com/servicequotas/。
3. 在服務清單中，選擇 AWS Certificate Manager Private Certificate Authority (ACM PCA)。 服務配額清單中的每個配額會顯示您目前套用的配額值、預設配額值，以及配額是否可以調整。選擇配額的名稱，以取得其詳細資訊。

請求提高配額

1. 在服務配額清單中，選擇可調整配額的選項按鈕。
2. 選擇請求配額增加按鈕。
3. 完成並提交請求配額增加表單。

AWS 私有 CA 已與 整合 AWS Certificate Manager。您可以使用 ACM 主控台 AWS CLI或 ACM API，從現有的私有 CA 請求私有憑證。這些由 ACM 管理的私有 PKI 憑證受 PCA 配額和 ACM 放置在公有和匯入憑證上的配額約束。如需 ACM 需求的詳細資訊，請參閱 AWS Certificate Manager 《 使用者指南》中的請求私有憑證和配額。

文件歷史記錄

下表說明此文件自 2018 年 1 月後的重大變更。除了這裡所列的主要變更外，我們也會經常更新文件以改進說明內容和範例，並且反映您傳送給我們的意見回饋。如要接收重大變更的通知，請使用右上角的連結來訂閱 RSS 摘要。

變更	描述	日期
文件更新	使用新的入門程序、範例以及監控和故障診斷主題更新 Secure Kubernetes AWS 私有憑證授權單位。	2025 年 10 月 1 日
雙堆疊支援	AWS 私有憑證授權單位 支援雙堆疊。	2025 年 6 月 23 日
Connector for AD 的子網域支援現已正式推出	您現在可以使用子網域設定 Connector for AD。	2025 年 6 月 2 日
新的 受管政策： AWSPrivateCAConnectoForKubernetesPolicy	推出可與 AWS 私有 CA Connector for Kubernetes 搭配使用的新受管政策。	2025 年 5 月 19 日
已更新 AWSPrivateCAPrivilegedUser 和 AWSPrivateCAUser 受管政策	在 AWSPrivateCAUser 和 ArnLike中StringLike 取代為 AWSPrivateCAPrivilegedUser 。更新範本 ARN，將萬用字元納入 arn:aws:acm-pca::: template arn:aws:acm-pca:*:*:template 。	2025 年 1 月 22 日
Connector for SCEP 現已正式推出	Connector for SCEP 現已正式推出。	2024 年 9 月 16 日
新的故障診斷主題	新增了新主題，可協助您疑難排解與更新 Connector for	2024 年 7 月 31 日

	Active Directory 範本相關的問題。	
新增如何更新 Connector for AD 範本	新增程序，說明如何更新 Connector for AD 範本，以及如何 AWS 私有 CA 傳播這些更新。	2024 年 7 月 31 日
適用於 Omnissa Workspace ONE 的 SCEP 連接器現已正式推出	Connector for SCEP for Omnissa Workspace ONE 現已正式推出。	2024 年 7 月 23 日
已新增稽核報告的限制條件	AWS 私有 CA 不支援搭配用於稽核報告的儲存貯體使用 Amazon S3 物件鎖定。	2024 年 7 月 3 日
現在支援適用於中國區域的 SM2	AWS 私有 CA 現在僅針對中國區域支援 SM2 簽署演算法。	2024 年 6 月 27 日
AWS 私有 CA 現在支援 Connector for SCEP （預覽版）	使用 Connector for SCEP AWS 私有 CA 連結到已啟用 SCEP 的用戶端和裝置。	2024 年 6 月 11 日
新連接器故障診斷指引	新增了有關故障診斷連接器和 SPN 建立失敗的新章節。	2024 年 4 月 4 日
新增適用於事項的 CDP 延伸模組	新增對 Matter 憑證撤銷清單分發點 (CDP) 延伸的支援。	2024 年 1 月 25 日
AWS 私有 CA mDL 的 API 支援	新增 API 支援，以建立符合 ISO/IEC 行動駕照 (mDL) 標準的憑證。	2024 年 1 月 16 日
AWS 私有 CA 適用於 Active Directory 的連接器	Connector for AD 的使用者指南、API 和 CLI 支援。如需詳細資訊，請參閱 Connector for AD 文件。	2023 年 8 月 24 日

變更安全政策名稱以符合新的服務名稱	採用指定標準許可的 AWS 受管 IAM 政策的新名稱 AWS 私有 CA。如需詳細資訊，請參閱 AWS 受管政策。	2023 年 2 月 13 日
新增 AWS 受管政策的變更追蹤器	新增文件以追蹤指定標準許可的 AWS 受管 IAM 政策變更 AWS 私有 CA。如需詳細資訊，請參閱 AWS 受管政策的更新 AWS 私有 CA。	2022 年 11 月 11 日
API 和 CLI 支援發行短期憑證CAs	隨著 CA 使用模式的推出，CA 可以設定為發行一般用途或僅限短期憑證。如需詳細資訊，請參閱憑證授權單位模式。	2022 年 10 月 24 日
服務品牌重塑和主控台更新	服務已重新命名為 AWS 私有憑證授權單位 (AWS 私有 CA)。 AWS 私有 CA 主控台可改善可用性，包括連結至完成文件的整合式說明面板。	2022 年 9 月 27 日
符合事項的憑證支援	三個新憑證範本新增了對符合事項規範 CA 和終端實體憑證的支援。如需詳細資訊，請參閱了解憑證範本。	2022 年 7 月 20 日
新區域支援	新增亞太區域（雅加達）的端點。如需 AWS 私有 CA 端點的完整清單，請參閱 ACM Private Certificate Authority Endpoints and Quotas。	2022 年 5 月 4 日
支援自訂屬性和延伸	使用 CustomAttribute 物件來設定自訂 CAs和憑證，並使用 CustomExtension 物件來設定自訂憑證。	2022 年 3 月 16 日

支援受管 OCSP	請參閱設定撤銷選項的憑證撤銷方法，包括 OCSP。	2021 年 8 月 18 日
支援 CRLs的 S3 封鎖公開存取功能	請參閱啟用 S3 封鎖公開存取功能。	2021 年 5 月 27 日
全新和更新的 Java 實作範例	請參閱使用 ACM Private CA API (Java 範例）。	2020 年 9 月 9 日
新區域支援	針對非洲（開普敦）和歐洲（米蘭）新增的端點。如需 AWS 私有 CA 端點的完整清單，請參閱AWS Certificate Manager 私有憑證授權機構端點和配額。	2020 年 8 月 27 日
支援跨帳戶私有 CA 存取	AWS Certificate Manager 使用者可以獲得授權，使用他們未擁有的私有 CAs發行憑證。如需詳細資訊，請參閱跨帳戶存取私有 CAs。	2020 年 8 月 17 日
VPC 端點 (PrivateLink) 支援	新增了使用 VPC 端點 (AWS PrivateLink) 以提高網路安全的支援。如需詳細資訊，請參閱 ACM Private CA VPC 端點 (AWS PrivateLink)。	2020 年 3 月 26 日
已新增專用安全區段	的安全文件 AWS 已合併至專用安全區段。如需安全性的詳細資訊，請參閱AWS Certificate Manager 私有憑證授權單位中的安全性。	2020 年 3 月 26 日
範本 ARN 已新增至稽核報告。	如需詳細資訊，請參閱為您的私有 CA 建立稽核報告。	2020 年 3 月 6 日

CloudFormation 支援	已新增 的支援 CloudFormation。如需詳細資訊，請參閱《 使用者指南》中的 CloudFormation ACMPCA 資源類型參考。	2020 年 1 月 22 日
CloudWatch Events 整合	針對非同步事件 (包括建立 CA、憑證發行和建立 CRL 等)，與 CloudWatch Events 整合。如需詳細資訊，請參閱使用 CloudWatch Events。	2019 年 12 月 23 日
FIPS 端點	新增 for AWS GovCloud （美國東部） 和 AWS GovCloud （美國西部） 的 FIPS 端點。如需 AWS 私有 CA 端點的完整清單，請參閱AWS Certificate Manager 私有憑證授權機構端點和配額。	2019 年 12 月 13 日
標籤型許可	標籤式許可支援使用 `TagResource` 、`UntagResource` 和 `ListTagsForResource` 等新的 API。如需標籤式控制的一般資訊，請參閱使用 IAM 資源標籤控制 IAM 使用者和角色的存取權。	2019 年 11 月 5 日
名稱限制條件強制執行	新增了對所匯入 CA 憑證實施主體名稱限制條件的支援。如需詳細資訊，請參閱對私有 CA 實施名稱限制條件。	2019 年 10 月 28 日
新的憑證範本	已新增新的憑證範本，包括使用 簽署程式碼的範本 AWS Signer。如需詳細資訊，請參閱使用範本。	2019 年 10 月 1 日

規劃您的 CA	新增了使用 AWS 私有 CA規劃 PKI 的新章節。如需詳細資訊，請參閱規劃您的 ACM 私有 CA 部署。	2019 年 9 月 30 日
已新增的區域支援	新增 AWS 亞太區域（香港）區域支援。如需支援區域的完整清單，請參閱AWS Certificate Manager 私有憑證授權機構端點和配額。	2019 年 7 月 24 日
新增完整的私有 CA 階層支援	支援建立和託管根 CAs，無需外部父系。	2019 年 6 月 20 日
已新增的區域支援	新增 AWS GovCloud（美國西部和美國東部）區域的區域支援。如需支援區域的完整清單，請參閱 AWS Certificate Manager Private Certificate Authority Endpoints and Quotas。	2019 年 5 月 8 日
已新增的區域支援	新增 AWS 亞太區域（孟買和首爾）、美國西部（加利佛尼亞北部）和歐洲（巴黎和斯德哥爾摩）區域的區域支援。如需支援區域的完整清單，請參閱AWS Certificate Manager 私有憑證授權機構端點和配額。	2019 年 4 月 4 日
測試憑證續約工作流程	客戶現在可以手動測試 ACM 受管續約工作流程的組態。如需更多資訊，請參閱測試 ACM 受管續約的組態。	2019 年 3 月 14 日

已新增的區域支援	新增 AWS 歐洲（倫敦）區域的區域支援。如需支援區域的完整清單，請參閱 AWS Certificate Manager Private Certificate Authority Endpoints and Quotas。	2018 年 8 月 1 日
還原已刪除CAs	私有 CA 還原允許客戶在憑證授權單位 (CA) 被刪除之後高達 30 天內，還能夠將其還原。如需詳細資訊，請參閱還原您的私有 CA。	2018 年 6 月 20 日

舊版更新

下表說明 2018 年 6 月 AWS 私有憑證授權單位 之前的文件發行歷史記錄。

變更	描述	日期
新指南	此版本推出 AWS 私有憑證授權單位。	2018 年 4 月 04 日

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。