aws

ユーザーガイド

AWS Private Certificate Authority

Version latest

AWS Private Certificate Authority: ユーザーガイド

Table of Contents

とは AWS Private CA

AWS Private CA では、オンプレミス CA の運用にかかる投資とメンテナンスのコストをかけずに、ルート CA と下位 CAsを含むプライベート認証機関 (CA) 階層を作成できます。プライベート CA は、次のようなシナリオでエンドエンティティ X.509 証明書を発行できます。

- 暗号化された TLS 通信チャネルの作成
- ユーザー、コンピュータ、API エンドポイント、および IoT デバイスの認証
- 暗号署名コード
- 証明書失効ステータスを取得するためのオンライン証明書ステータスプロトコル (OCSP) の実装

AWS Private CA オペレーションには、、 AWS Private CA API AWS Management Console、または を使用して からアクセスできます AWS CLI。

トピック

ニーズに最適な証明書サービスはどれですか?

X.509 証明書の発行とデプロイには 2 つの AWS サービスがあります。ニーズに最適なものを選択してください。考慮事項には、パブリック証明書とプライベート証明書のどちらが必要か、カスタマイズされた証明書、他の AWS サービスにデプロイする証明書、または証明書の自動管理と更新が必要です。

1. AWS Private CA—このサービスは、 AWS クラウド内にパブリックキーインフラストラクチャ (PKI) を構築するエンタープライズのお客様を対象とし、組織内でのプライベートな使用を目的としています。を使用すると AWS Private CA、独自の CA 階層を作成し、内部ユーザー、コンピュータ、アプリケーション、サービス、サーバー、その他のデバイスを認証したり、コン

ピュータコードに署名したりするための証明書を発行できます。プライベート CA によって発行された証明書は、インターネット上ではなく、 の組織内でのみ信頼されます。

プライベート CA を作成したら、証明書を直接発行して (つまり、サードパーティー CA から検証を取得せずに)、組織の内部ニーズに合わせて証明書をカスタマイズできます。たとえば、次のような操作を実行できます。

- 任意のサブジェクト名で証明書を作成します。
- 任意の有効期限で証明書を作成します。
- サポートされている任意のプライベートキーアルゴリズムとキー長を使用できます。
- サポートされている任意の署名アルゴリズムを使用できます。
- テンプレートを使用して証明書の発行を制御します。

お客様は、このサービスの対象となります。開始するには、https://console.aws.amazon.com/acm-pca/ コンソールにサインインします。

2. AWS Certificate Manager (ACM) - このサービスは、TLS を使用して、パブリックに信頼された安全なウェブプレゼンスを必要とするエンタープライズ顧客の証明書を管理します。ACM 証明書は、 AWS Elastic Load Balancing 、Amazon 、Amazon API Gateway CloudFront、およびその他の統合サービス にデプロイできます。 Amazon API Gateway この種の代表的なアプリケーションは、重要なトラフィック要件を持つセキュアなパブリックウェブサイトです。

   このサービスでは、ACM が提供するパブリック証明書 (ACM 証明書) または ACM にインポートする証明書を使用できます。 AWS Private CA を使用して CA を作成する場合、ACM はそのプライベート CA からの証明書の発行を管理し、証明書の更新を自動化できます。

   詳細については、「AWS Certificate Manager ユーザーガイド」を参照してください。

リージョン

ほとんどの AWS リソースと同様に、プライベート認証機関 (CAsはリージョンリソースです。複数のリージョンでプライベート CA を使用するには、それらのリージョンで CA を作成する必要があります。リージョン間でプライベート CA をコピーすることはできません。「AWS 全般のリファレンス」の「AWS リージョンとエンドポイント」、または「AWS リージョン表」で「 AWS Private CA を利用できるリージョン」を参照してください。

> **Note**
>
> ACM は現在、そう AWS Private CA でない一部のリージョンで利用可能です。

と統合されたサービス AWS Private Certificate Authority

AWS Certificate Manager を使用してプライベート証明書をリクエストする場合、その証明書を ACM と統合された任意のサービスに関連付けることができます。これは、 AWS Private CA ルートに連鎖された証明書と外部ルートに連鎖された証明書の両方に適用されます。詳細については、「 ユーザーガイド」の「統合サービス AWS Certificate Manager 」を参照してください。

プライベート CA を Amazon Elastic Kubernetes Service に統合して、Kubernetes クラスター内で証明書を発行することもできます。詳細については、「AWS Private CA での Kubernetes のセキュリティ保護」を参照してください。

> **Note**
>
> Amazon Elastic Kubernetes Service は ACM 統合サービスではありません。

AWS Private CA API または を使用して証明書 AWS CLI を発行したり、ACM からプライベート証明書をエクスポートしたりする場合は、任意の場所に証明書をインストールできます。

サポートされている暗号アルゴリズム

AWS Private CA は、プライベートキーの生成と証明書の署名に次の暗号化アルゴリズムをサポートしています。

サポートされているアルゴリズム

プライベートキーアルゴリズム	署名アルゴリズム
RSA_2048	SHA256WITHECDSA
RSA_4096	SHA384WITHECDSA
EC_prime256v1	SHA512WITHECDSA

プライベートキーアルゴリズム	署名アルゴリズム
EC_secp384r1 SM2 (中国リージョンのみ)	SHA256WITHRSA SHA384WITHRSA SHA512WITHRSA SM3WITHSM2

このリストは、コンソール、API、またはコマンドライン AWS Private CA を介して によって直接発行された証明書にのみ適用されます。から CA を使用して証明書 AWS Certificate Manager を発行する場合 AWS Private CA、これらのアルゴリズムの一部はサポートされますが、すべてはサポートされません。詳細については、「 ユーザーガイド」の「プライベート証明書のリクエスト AWS Certificate Manager 」を参照してください。

> **Note**
>
> どの場合においても、指定された署名アルゴリズムファミリー (RSA または ECDSA) は、CA のプライベートキーのアルゴリズムファミリーと一致する必要があります。

クォータ

AWS Private CA は、許可された数の証明書と認証機関にクォータを割り当てます。API アクションのリクエストレートも、アカウント AWS とリージョンに固有の quotas. AWS Private CA quotas の対象となります。

AWS Private CA は、API オペレーションに応じて異なるレートで API リクエストを調整します。スロットリングとは、リクエストが 1 秒あたりのリクエスト数のオペレーションのクォータを超えているため、 がそれ以外の有効なリクエスト AWS Private CA を拒否することを意味します。リクエストがスロットリングされると、 は ThrottlingException error. AWS Private CA does AWS Private CA を返します。APIsの最小リクエストレートは保証されません。

調整できるクォータを確認するには、「」の「 AWS Private CA クォータ」表を参照してくださいAWS 全般のリファレンス。

AWS Service Quotas を使用すると、現在のクォータを確認したり、クォータの引き上げをリクエストしたりできます。

AWS Private CA クォータ up-to-date のリストを表示するには

1. AWS アカウントにログインします。
2. Service Quotas のコンソールを開きます。https://console.aws.amazon.com/servicequotas/
3. [サービス] リストで、[AWS Certificate Manager プライベート認証機関 (ACM PCA)] を選択します。[サービスクォータ] リストの各クォータには、現在適用されているクォータ値、デフォルトクォータ値、およびクォータが調整可能かどうかが表示されます。クォータの名前をクリックすると、そのクォータに関するより詳細な情報が表示されます。

クォータの引き上げをリクエストするには

1. [サービスクォータ] リストで、クォータを調整できるラジオボタンを選択します。
2. [クォータの引き上げをリクエスト] ボタンを選択します。
3. [クォータの引き上げリクエスト] フォームに必要事項を記入して送信します。

AWS Private CA は と統合されています AWS Certificate Manager。ACM コンソール、または ACM API を使用して AWS CLI、既存のプライベート CA にプライベート証明書をリクエストできます。これらのプライベート PKI 証明書は ACM によって管理され、PCA クォータと ACM がパブリック証明書とインポート証明書に設定するクォータの両方の対象となります。ACM の要件の詳細については、「 AWS Certificate Manager ユーザーガイド」の「プライベート証明書とクォータのリクエストhttps://docs.aws.amazon.com/acm/latest/userguide/acm-limits.html」を参照してください。

RFC 準拠

AWS Private CA は、RFC 5280 で定義されている特定の制約を適用しません。逆も同様で、プライベート CA に適切な追加の制約が強制されます。

強制

- 日付を過ぎると強制されません。RFC 5280 に準拠して、 AWS Private CA は、交付元 CA 認定の交付日よりも後の `Not After` `Not After` 日付を持つ証明書は発行しません。
- 基本的な制約 . AWS Private CA は、インポートされた CA 証明書に基本的な制約とパスの長さを適用します。

  基本的な制約は、証明書によって識別されるリソースが CA であり、証明書を発行できるかどうかを示します。 AWS Private CA にインポートされる CA 認定には、基本的制約の拡張を含め

る必要があり、拡張に `critical` とマークする必要があります。`critical` フラグに加えて、`CA=true`を設定する必要があります。 は、次の理由で検証例外で失敗することで基本的な制約 AWS Private CA を適用します。

- 拡張が CA 認定に含まれていない。
- 拡張が `critical` とマークされていない。

パスの長さ (pathLenConstraint) は、インポートされた CAs 証明書の下流に存在する可能性のある下位 CA の数を決定します。 は、以下の理由で検証例外が発生して失敗することで、パスの長さを AWS Private CA 強制します。

- CA 認定をインポートすると、CA 認定またはチェーン内の任意の CA 認定のパスの長さ制約に違反する。
- 証明書を発行すると、パスの長さの制約に違反する。

- 名前の制約は、証明書パスの後続の証明書のすべてのサブジェクト名を配置する必要がある名前空間を示します。サブジェクト識別名とサブジェクト代替名には制限が適用されます。

強制されない

- 証明書ポリシー 。証明書ポリシーは、CA が証明書を発行する条件を規制します。
- anyPolicy を禁止します。CAs。
- 発行者の代替名 。CA 証明書の発行者との追加の ID の関連付けを許可します。
- ポリシーの制限 これらの制約により、CA の下位 CA 認定を交付する機能が制限されます。
- ポリシーマッピング 。CA 証明書で使用されます。OIDs。各ペアには issuerDomainPolicy と が含まれますsubjectDomainPolicy。
- サブジェクトディレクトリ属性 。サブジェクトの識別属性を伝えるために使用されます。
- サブジェクト情報アクセス 。拡張機能が表示される証明書のサブジェクトの情報とサービスにアクセスする方法。
- サブジェクトキー識別子 (SKI) と 権限キー識別子 (AKI)。RFC では、SKI 拡張を含む CA 認定が必要です。CA によって発行された証明書には、CA 証明書の SKI. AWS does に一致する AKI 拡張機能が含まれている必要があります。これらの要件は適用されません。CA 認定に SKI が含まれていない場合、発行されたエンドエンティティまたは下位 CA 認定 AKI は、発行者パブリックキーの SHA-1 ハッシュになります。
- SubjectPublicKeyInfo およびサブジェクト代替名 (SAN)。証明書を発行すると、 AWS Private CA は検証を実行せずに、提供された CSR から SubjectPublicKeyInfo および SAN 拡張機能をコピーします。

料金

アカウントを作成した時点で、各プライベート CA の月額料金が課金されます。また、発行する証明書ごとに課金されます。この料金には、ACM からエクスポートする証明書と、 AWS Private CA API または AWS Private CA CLI から作成した証明書が含まれます。プライベート CA が削除された後は、それに対して課金されません。ただし、プライベート CA を復元すると、削除と復元の間の料金が課金されます。プライベートキーにアクセスできないプライベート証明書については、料金は発生しません。これには、Elastic Load Balancing 、 CloudFront、API Gateway などの統合サービスで使用される証明書が含まれます。

最新の AWS Private CA 料金情報については、「 の料金AWS Private Certificate Authority」を参照してください。AWS 料金計算ツール でコストを見積もることもできます。

のセキュリティ AWS Private Certificate Authority

のクラウドセキュリティが最優先事項 AWS です。お客様は AWS 、セキュリティを最も重視する組織の要件を満たすように構築されたデータセンターとネットワークアーキテクチャからメリットを得られます。

セキュリティは、 AWS とユーザーの間で共有される責任です。責任共有モデルでは、これをクラウドのセキュリティおよびクラウド内のセキュリティと説明しています。

- クラウドのセキュリティ — クラウドで AWS サービスを実行するインフラストラクチャを保護する責任 AWS は AWS にあります。 AWS は、安全に使用できるサービスも提供します。コンプライアンスAWS プログラムコンプライアンスプログラム の一環として、サードパーティーの監査者は定期的にセキュリティの有効性をテストおよび検証。に適用されるコンプライアンスプログラムの詳細については AWS Private Certificate Authority、「コンプライアンスプログラムAWS のサービス による対象範囲内の」、「コンプライアンスプログラム」を参照してください。
- クラウドのセキュリティ — お客様の責任は、使用する AWS サービスによって決まります。また、お客様は、データの機密性、会社の要件、適用される法律や規制など、その他の要因についても責任を負います。

このドキュメントは、 を使用する際の責任共有モデルの適用方法を理解するのに役立ちます AWS Private CA。以下のトピックでは、セキュリティおよびコンプライアンスの目的を達成するために AWS Private CA を設定する方法を示します。また、 AWS Private CA リソースのモニタリングや保護 AWS のサービス に役立つ他の の使用方法についても説明します。

トピック

- の Identity and Access Management (IAM) AWS Private Certificate Authority
- プライベート CA へのクロスアカウントアクセスに関するセキュリティのベストプラクティス
- でのデータ保護 AWS Private Certificate Authority
- AWS Private Certificate Authorityのコンプライアンス検証
- のインフラストラクチャセキュリティ AWS Private Certificate Authority
- AWS Private Certificate Authorityでのログ記録とモニタリング

の Identity and Access Management (IAM) AWS Private Certificate Authority

へのアクセスには AWS 、 がリクエストの認証に使用できる認証情報 AWS Private CA が必要です。以下のトピックでは、プライベート CA をセキュリティで保護するために AWS Identity and Access Management (IAM) を使用してプライベート認証機関 (CA) にアクセスできるユーザーを管理する方法について詳しく説明します。

では AWS Private CA、使用するプライマリリソースは認証局 (CA) です。所有または管理する各プライベート CA は、Amazon リソースネーム (ARN) によって識別され、その形式は次のとおりです。

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

リソース所有者は、 AWS リソースが作成される AWS アカウントのプリンシパルエンティティです。以下は、この仕組みを説明する例です。

- の認証情報を使用してプライベート CA AWS アカウントのルートユーザー を作成する場合、アカウント AWS は CA を所有します。

  > **Important**
  > - を使用して CAs AWS アカウントのルートユーザー を作成することはお勧めしません。
  > - にアクセスするときはいつでも多要素認証 (MFA) を使用することを強くお勧めします AWS Private CA。

- AWS アカウントに IAM ユーザーを作成する場合、プライベート CA を作成するアクセス許可をそのユーザーに付与できます。ただし、そのユーザーが所属するアカウントが CA を所有しています。
- AWS アカウントに IAM ロールを作成し、プライベート CA を作成するアクセス許可を付与すると、ロールを引き受けることのできるいずれのユーザーも CA を作成できます。ただし、そのロールが所属するアカウントがプライベート CA を所有します。

アクセスポリシーは、誰が何に対するアクセス権を持っているのかを説明します。以下のディスカッションで、アクセス許可のポリシーを作成するために使用可能なオプションについて説明します。

> **Note**
>
> このドキュメントでは、 のコンテキストでの IAM の使用について説明します AWS Private CA。これは、IAM サービスに関する詳細情報を取得できません。完全な IAM ドキュメントについては、「IAM ユーザーガイド」を参照してください。IAM ポリシー構文の詳細および説明については、「AWS IAM ポリシーの参照」を参照してください。

AWS Private CA API オペレーションとアクセス許可

IAM アイデンティティにアタッチするアクセス制御とアクセス許可ポリシー (アイデンティティベースのポリシー) を設定するときは、以下の表をリファレンスとして使用できます。表の最初の列には、各 AWS Private CA API オペレーションが一覧表示されます。ポリシーの `Action` 要素でアクションを指定します。残りの列では、追加情報を指定します。

AWS Private CA API オペレーション	必要な許可	リソース
CreateCertificateAuthority	`acm-pca:CreateCertificateAuthority` `acm-pca:TagCertificateAuthority` (タグ付きの CA を作成する場合にのみ必要です。)	`arn:aws:acm-pca:us-east-1:111122223333:certificate-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-`

AWS Private CA API オペレーション	必要な許可	リソース
		1234-1122-2233-112 233445566
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*
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*

AWS Private CA API オペレーション	必要な許可	リソース
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*
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*

AWS Private CA API オペレーション	必要な許可	リソース
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	該当なし
ListPermissions	acm-pca:ListPermissions	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*
ListTags	acm-pca:ListTags	該当なし
PutPolicy	acm-pca:PutPolicy	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*

AWS Private CA API オペレーション	必要な許可	リソース
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*
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*

アクセス権限を付与するには、ユーザー、グループ、またはロールにアクセス許可を追加します。

- のユーザーとグループ AWS IAM Identity Center :

  アクセス許可セットを作成します。「AWS IAM Identity Center ユーザーガイド」の「権限設定を作成する」の手順に従ってください。

- IAM 内で、ID プロバイダーによって管理されているユーザー:

  ID フェデレーションのロールを作成します。詳細については、「IAM ユーザーガイド」の「サードパーティー ID プロバイダー (フェデレーション) 用のロールの作成」を参照してください。
- IAM ユーザー:
  - ユーザーが担当できるロールを作成します。手順については、「IAM ユーザーガイド」の「IAM ユーザー用ロールの作成」を参照してください。
  - (お奨めできない方法) ポリシーをユーザーに直接アタッチするか、ユーザーをユーザーグループに追加する。詳細については、「IAM ユーザーガイド」の「ユーザー (コンソール) へのアクセス権限の追加」を参照してください。

AWS マネージドポリシー

AWS Private CA には、管理者、ユーザー、および監査人向けの AWS Private CA 事前定義された AWS 管理ポリシーのセットが含まれています。これらのポリシーを理解すると、カスタマー管理ポリシー を実装するのに役立ちます。

以下のポリシーのいずれかを選択すると、詳細とサンプルポリシーコードが表示されます。

AWSPrivateCAFullAccess

無制限の管理制御を付与します。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "acm-pca:*"
         ],
         "Resource":"*"
      }
   ]
}
```

AWSPrivateCAReadOnly

読み取り専用の API オペレーションに限定されたアクセスを付与します。

```
{
   "Version":"2012-10-17",
   "Statement":{
      "Effect":"Allow",
      "Action":[
         "acm-pca:DescribeCertificateAuthority",
         "acm-pca:DescribeCertificateAuthorityAuditReport",
         "acm-pca:ListCertificateAuthorities",
         "acm-pca:GetCertificateAuthorityCsr",
         "acm-pca:GetCertificateAuthorityCertificate",
         "acm-pca:GetCertificate",
         "acm-pca:GetPolicy",
         "acm-pca:ListPermissions",
         "acm-pca:ListTags"
      ],
      "Resource":"*"
   }
}
```

AWSPrivateCAPrivilegedUser

CA 証明書を発行および取り消す機能を付与します。このポリシーに他の管理機能はなく、エンドエンティティ証明書を発行する機能もありません。アクセス許可は、User ポリシーと相互に排他的です。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "acm-pca:IssueCertificate"
         ],
         "Resource":"arn:aws:acm-pca:*:*:certificate-authority/*",
         "Condition":{
            "StringLike":{
               "acm-pca:TemplateArn":[
                  "arn:aws:acm-pca:::template/*CACertificate*/V*"
               ]
            }
         }
      },
      {
```

```
         "Effect":"Deny",
         "Action":[
            "acm-pca:IssueCertificate"
         ],
         "Resource":"arn:aws:acm-pca:*:*:certificate-authority/*",
         "Condition":{
            "StringNotLike":{
               "acm-pca:TemplateArn":[
                  "arn:aws:acm-pca:::template/*CACertificate*/V*"
               ]
            }
         }
      },
      {
         "Effect":"Allow",
         "Action":[
            "acm-pca:RevokeCertificate",
            "acm-pca:GetCertificate",
            "acm-pca:ListPermissions"
         ],
         "Resource":"arn:aws:acm-pca:*:*:certificate-authority/*"
      },
      {
         "Effect":"Allow",
         "Action":[
            "acm-pca:ListCertificateAuthorities"
         ],
         "Resource":"*"
      }
   ]
}
```

AWSPrivateCAUser

エンドエンティティ証明書を発行および取り消す機能を付与します。このポリシーに他の管理機能はなく、CA 認定を交付する機能もありません。アクセス許可はPrivilegedUserポリシーと相互に排他的です。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
```

```
         "Action":[
            "acm-pca:IssueCertificate"
         ],
         "Resource":"arn:aws:acm-pca:*:*:certificate-authority/*",
         "Condition":{
            "StringLike":{
               "acm-pca:TemplateArn":[
                  "arn:aws:acm-pca:::template/EndEntityCertificate/V*"
               ]
            }
         }
      },
      {
         "Effect":"Deny",
         "Action":[
            "acm-pca:IssueCertificate"
         ],
         "Resource":"arn:aws:acm-pca:*:*:certificate-authority/*",
         "Condition":{
            "StringNotLike":{
               "acm-pca:TemplateArn":[
                  "arn:aws:acm-pca:::template/EndEntityCertificate/V*"
               ]
            }
         }
      },
      {
         "Effect":"Allow",
         "Action":[
            "acm-pca:RevokeCertificate",
            "acm-pca:GetCertificate",
            "acm-pca:ListPermissions"
         ],
         "Resource":"arn:aws:acm-pca:*:*:certificate-authority/*"
      },
      {
         "Effect":"Allow",
         "Action":[
            "acm-pca:ListCertificateAuthorities"
         ],
         "Resource":"*"
      }
   ]
```

```
}
```

AWSPrivateCAAuditor

読み取り専用の API オペレーションへのアクセス権と、CA 監査報告書を生成する許可を付与します。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "acm-pca:CreateCertificateAuthorityAuditReport",
            "acm-pca:DescribeCertificateAuthority",
            "acm-pca:DescribeCertificateAuthorityAuditReport",
            "acm-pca:GetCertificateAuthorityCsr",
            "acm-pca:GetCertificateAuthorityCertificate",
            "acm-pca:GetCertificate",
            "acm-pca:GetPolicy",
            "acm-pca:ListPermissions",
            "acm-pca:ListTags"
         ],
         "Resource":"arn:aws:acm-pca:*:*:certificate-authority/*"
      },
      {
         "Effect":"Allow",
         "Action":[
            "acm-pca:ListCertificateAuthorities"
         ],
         "Resource":"*"
      }
   ]
}
```

の AWS マネージドポリシーの更新 AWS Private CA

次の表では、サービスがこれらの変更の追跡を開始 AWS Private CA してからの の AWS マネージドポリシーの更新に関する詳細を表示します。へのすべての変更に関する自動アラートについては AWS Private CA、 ドキュメント履歴ページの RSS フィードにサブスクライブしてください。

マネージドポリシーの変更

変更	説明	日付
新しいポリシー名: • AWSPrivateCAFullAccess • AWSPrivateCAReadOnly • AWSPrivateCAPrivilegedUser • AWSPrivateCAAuditor • AWSPrivateCAUser	ポリシー名のプレフィックスが AWSCertificateManagerPrivateCA から AWSPrivateCA に変更されました。 機能は変わりません。	2023 年 2 月 13 日

カスタマー管理ポリシー

ベストプラクティスとして、 を使用して AWSを含む と AWS アカウントのルートユーザー やり取りしないでください AWS Private CA。代わりに AWS Identity and Access Management (IAM) を使用して、IAM ユーザー、IAM ロール、またはフェデレーティッドユーザーを作成します。管理者グループを作成し、このグループに自分自身を追加します。次に、管理者としてログインします。必要に応じて、追加のユーザーをグループに追加します。

別のベストプラクティスとしては、IAM ポリシーを作成し、これをユーザーに割り当てることができます。カスタマー管理ポリシーは、作成したスタンドアロンの ID ベースのポリシーで、 AWS アカウントの複数のユーザー、グループ、またはロールにアタッチすることができます。このようなポリシーにより、ユーザーは指定した AWS Private CA アクションのみを実行するように制限されます。

次の例のカスタマー管理ポリシーでは、CA 監査報告書を作成することをユーザーに許可します。これはあくまで例に過ぎません。任意の AWS Private CA オペレーションを選択できます。その他の例については、「インラインポリシー」を参照してください。

カスタマー管理ポリシーを作成するには

1. AWS 管理者の認証情報を使用して IAM コンソールにサインインします。
2. コンソールのナビゲーションペインで、[Policies] (ポリシー) を選択します。

3. [Create policy] (ポリシーを作成) を選択します。
4. [JSON] タブを選択します。
5. 以下のポリシーをコピーして、エディタに貼り付けます。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":"acm-pca:CreateCertificateAuthorityAuditReport",
         "Resource":"*"
      }
   ]
}
```

6. [ポリシーの確認] を選択します。
7. [Name] (名前) に、「PcaListPolicy」と入力します。
8. (オプション) 説明を入力します。
9. [ポリシーの作成] を選択します。

管理者は、IAM ユーザーにポリシーをアタッチすることで、ユーザーが実行できる AWS Private CA アクションを制限できます。アクセス許可ポリシーを適用する方法については、「IAM ユーザーガイド」の「IAM ユーザーのアクセス許可の変更」を参照してください。

インラインポリシー

インラインポリシーとは、ユーザー、グループ、または役割りを作成、管理、または直接組み込むポリシーです。次のポリシー例は、 AWS Private CA アクションを実行するアクセス許可を割り当てる方法を示しています。インラインポリシーに関する全般的な情報については、「IAM ユーザーガイド」の「インラインポリシーの使用」を参照してください。、 (AWS CLI) AWS Management Console、 AWS Command Line Interface または IAM API を使用して、インラインポリシーを作成して埋め込むことができます。

> **Important**
>
> にアクセスするときはいつでも多要素認証 (MFA) を使用することを強くお勧めします AWS Private CA。

トピック

- プライベート CA を一覧表示する
- プライベート CA 証明書の取得
- プライベート CA 証明書をインポートする
- プライベート CA の削除
- T ag-on-create: 作成時に CA にタグをアタッチする
- T ag-on-create: タグ付けの制限
- タグを使用したプライベート CA リソースへのアクセスの制御
- への読み取り専用アクセス AWS Private CA
- へのフルアクセス AWS Private CA
- すべての AWS リソースへの管理者アクセス

プライベート CA を一覧表示する

次のポリシーを使用すると、ユーザーはアカウントのすべてのプライベート CA を一覧表示できます。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":"acm-pca:ListCertificateAuthorities",
         "Resource":"*"
      }
   ]
}
```

プライベート CA 証明書の取得

次のポリシーを使用すると、ユーザーは特定のプライベート CA 証明書を取得できます。

```
{
   "Version":"2012-10-17",
   "Statement":{
      "Effect":"Allow",
      "Action":"acm-pca:GetCertificateAuthorityCertificate",
```

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

プライベート CA 証明書をインポートする

次のポリシーを使用すると、ユーザーはプライベート CA 証明書をインポートできます。

```
{
   "Version":"2012-10-17",
   "Statement":{
      "Effect":"Allow",
      "Action":"acm-pca:ImportCertificateAuthorityCertificate",
      "Resource":"arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
   }
}
```

プライベート CA の削除

次のポリシーを使用すると、ユーザーは特定のプライベート CA 証明書を削除できます。

```
{
   "Version":"2012-10-17",
   "Statement":{
      "Effect":"Allow",
      "Action":"acm-pca:DeleteCertificateAuthority",
      "Resource":"arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
   }
}
```

T ag-on-create: 作成時に CA にタグをアタッチする

以下のポリシーでは、CA の作成中にユーザーがタグを適用できます。

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
```

```
         "Action": [
            "acm-pca:CreateCertificateAuthority",
            "acm-pca:TagCertificateAuthority"
         ],
         "Effect": "Allow",
         "Resource": "*"
      }
   ]
}
```

T ag-on-create: タグ付けの制限

次の tag-on-create ポリシーでは、CA の作成中にキーと値のペア Environment=Prod を使用できないようにします。他のキーと値のペアによるタグ付けは可能です。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":"acm-pca:*",
         "Resource":"*"
      },
      {
         "Effect":"Deny",
         "Action":"acm-pca:TagCertificateAuthority",
         "Resource":"*",
         "Condition":{
            "StringEquals":{
               "aws:ResourceTag/Environment":[
                  "Prod"
               ]
            }
         }
      }
   ]
}
```

タグを使用したプライベート CA リソースへのアクセスの制御

次のポリシーでは、キーと値のペア Environment= CAs へのアクセスのみを許可しますPreProd。また、新しい CA にもこのタグを含める必要があります。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "acm-pca:*"
         ],
         "Resource":"*",
         "Condition":{
            "StringEquals":{
               "aws:ResourceTag/Environment":[
                  "PreProd"
               ]
            }
         }
      }
   ]
}
```

への読み取り専用アクセス AWS Private CA

次のポリシーを使用すると、ユーザーはプライベート証明書認証機関を説明、一覧表示し、また、プライベート CA 認定と証明書チェーンを取得できます。

```
{
   "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":"*"
   }
}
```

へのフルアクセス AWS Private CA

次のポリシーでは、ユーザーに任意のアクションを実行することを許可します AWS Private CA 。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "acm-pca:*"
         ],
         "Resource":"*"
      }
   ]
}
```

すべての AWS リソースへの管理者アクセス

次のポリシーでは、ユーザーは任意の AWS リソースに対して任意のアクションを実行できます。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":"*",
         "Resource":"*"
      }
   ]
}
```

プライベート CA へのクロスアカウントアクセスに関するセキュリティのベストプラクティス

AWS Private CA 管理者は、別の AWS アカウント のプリンシパル (ユーザー、ロールなど) と CA を共有できます。共有が受信および承諾されると、プリンシパルは CA を使用して、 AWS Private CA または AWS Certificate Manager リソースを使用してエンドエンティティ証明書を発行できます。プリンシパルは CA を使用して、 を使用して下位 CA 証明書を発行できます AWS Private CA。

> **Important**
>
> クロスアカウントシナリオで発行された証明書に関連する料金は、証明書を発行する AWS アカウントに請求されます。

CA へのアクセスを共有するには、 AWS Private CA 管理者は次のいずれかの方法を選択できます。

- AWS Resource Access Manager (RAM) を使用して、CA をリソースとして別のアカウントのプリンシパルまたは と共有します AWS Organizations。RAM は、アカウント間で AWS リソースを共有するための標準的な方法です。RAM の詳細については、「AWS RAM ユーザーガイド」を参照してください。の詳細については AWS Organizations、「 AWS Organizations ユーザーガイド」を参照してください。
- AWS Private CA API または CLI を使用してリソースベースのポリシーを CA にアタッチし、別のアカウントのプリンシパルにアクセスを許可します。詳細については、「リソースベースのポリシー」を参照してください。

このガイドの プライベート CA へのアクセスの制御 セクションでは、単一アカウントシナリオとクロスアカウントシナリオの両方で CA へのアクセスを許可するワークフローについて説明します。

リソースベースのポリシー

リソースベースのポリシーは、ユーザー ID やロールではなく、ユーザーが作成してリソース (この場合はプライベート CA) に手動でアタッチするアクセス許可ポリシーです。または、独自のポリシーを作成する代わりに、 の AWS 管理ポリシーを使用できます AWS Private CA。 AWS RAM を使用してリソースベースのポリシーを適用すると、 AWS Private CA 管理者は CA へのアクセスを別の AWS アカウントのユーザーと直接または を通じて共有できます AWS Organizations。または、AWS Private CA 管理者は PCA APIs PutPolicy、、 GetPolicyDeletePolicyまたは対応する AWS CLI コマンド put-policy 、get-policy 、delete-policy を使用して、リソースベースのポリシーを適用および管理できます。

リソースベースのポリシーに関する一般的な情報については、「アイデンティティベースのポリシーおよびリソースベースのポリシー」および「ポリシーを使用したアクセスの制御」を参照してください。

の AWS マネージドリソースベースのポリシーのリストを表示するには AWS Private CA、 AWS Resource Access Manager コンソールで マネージドアクセス許可ライブラリに移動し、 を検索しま

すCertificateAuthority。他のポリシーと同様に、ポリシーを適用する前に、テスト環境でポリシーを適用して、要件を満たしていることを確認することをお勧めします。

AWS Certificate Manager プライベート CA へのクロスアカウント共有アクセス権を持つ (ACM) ユーザーは、CA によって署名されたマネージド証明書を発行できます。クロスアカウント発行者はリソースベースのポリシーによって制約され、以下のエンドエンティティ証明書テンプレートにのみアクセスできます。

- EndEntityCertificate/V1
- EndEntityClientAuthCertificate/V1
- EndEntityServerAuthCertificate/V1
- BlankEndEntityCertificate_APIPassthrough /V1
- BlankEndEntityCertificate_APICSRPassthrough /V1
- SubordinateCACertificate _PathLen0/V1

ポリシーの例

このセクションでは、さまざまなニーズに対応するクロスアカウントポリシーの例を示します。いずれの場合も、以下のコマンドパターンを使用してポリシーを適用します。

```
$ 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
```

CA の ARN を指定することに加えて、管理者は CA へのアクセスを許可するアカウント ID または AWS Organizations ID を提供します AWS 。以下の各ポリシーの JSON は、読みやすいようにファイルとしてフォーマットされていますが、インライン CLI 引数として指定することもできます。

> **Note**
>
> 以下に示す JSON リソースベースのポリシーの構造に正確に従う必要があります。お客様が設定できるのは、プリンシパルの ID フィールド (AWS アカウント番号または AWS Organizations ID) と CA ARNs のみです。

1. ファイル: policy1.json — CA へのアクセスを別のアカウントのユーザーと共有する

555555555555 を CA を共有している AWS アカウント ID に置き換えます。

リソース ARN の場合、以下を独自の値に置き換えます。

- *aws* - AWS パーティション。例えば、、aws-cn、 aws aws-us-govなどです。
- *us-east-1* - など、リソースが利用可能な AWS リージョンus-west-1。
- *111122223333* - リソース所有者の AWS アカウント ID。
- *11223344-1234-1122-2233-112233445566* - 認証機関のリソース ID。

```
{
  "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-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
     },
     {
        "Sid":"ExampleStatementID2",
        "Effect":"Allow",
        "Principal":{
           "AWS":"555555555555"
        },
        "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"
            }
         }
      }
   ]
}
```

2. ファイル: policy2.json – を介した CA へのアクセスの共有 AWS Organizations

   o-a1b2c3d4z5 を AWS Organizations ID に置き換えます。

   リソース ARN の場合、以下を独自の値に置き換えます。

   - *aws* - AWS パーティション。例えば、、、aws-cn、 aws aws-us-govなどです。
   - *us-east-1* - など、リソースが利用可能な AWS リージョンus-west-1。
   - *111122223333* - リソース所有者の AWS アカウント ID。
   - *11223344-1234-1122-2233-112233445566* - 認証機関のリソース ID。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"ExampleStatementID3",
         "Effect":"Allow",
         "Principal":"*",
         "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",
               "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-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
            "Condition":{
               "StringEquals":{
                  "aws:PrincipalOrgID":"o-a1b2c3d4z5"
               },
               "StringNotEquals":{
                  "aws:PrincipalAccount":"111122223333"
            }
         }
      ]
}
```

でのデータ保護 AWS Private Certificate Authority

責任 AWS 共有モデル、 でのデータ保護に適用されます AWS Private Certificate Authority。このモデルで説明されているように、 AWS はすべての を実行するグローバルインフラストラクチャを保護する責任があります AWS クラウド。お客様は、このインフラストラクチャでホストされているコンテンツに対する管理を維持する責任があります。また、使用する AWS のサービス のセキュリティ設定と管理タスクもユーザーの責任となります。データプライバシーの詳細については、データプライバシーのよくある質問を参照してください。欧州でのデータ保護の詳細については、AWS セキュリティブログに投稿されたAWS 責任共有モデルおよび GDPRのブログ記事を参照してください。

データ保護の目的で、認証情報を保護し AWS アカウント 、 AWS IAM Identity Center または AWS Identity and Access Management (IAM) を使用して個々のユーザーを設定することをお勧めします。

この方法により、それぞれのジョブを遂行するために必要な権限のみが各ユーザーに付与されます。また、次の方法でデータを保護することもお勧めします:

- 各アカウントで多要素認証 (MFA) を使用します。
- SSL/TLS を使用して AWS リソースと通信します。TLS 1.2 は必須であり TLS 1.3 がお勧めです。
- で API とユーザーアクティビティのログ記録を設定します AWS CloudTrail。
- AWS 暗号化ソリューションと、 内のすべてのデフォルトのセキュリティコントロールを使用します AWS のサービス。
- Amazon Macie などの高度なマネージドセキュリティサービスを使用します。これらは、Amazon S3 に保存されている機密データの検出と保護を支援します。
- コマンドラインインターフェイスまたは API AWS を介して にアクセスするときに FIPS 140-2 検証済みの暗号化モジュールが必要な場合は、FIPS エンドポイントを使用します。利用可能な FIPS エンドポイントの詳細については、「連邦情報処理規格 (FIPS) 140-2」を参照してください。

お客様の E メールアドレスなどの極秘または機密情報は、タグ、または名前フィールドなどの自由形式のテキストフィールドに配置しないことを強くお勧めします。これは、コンソール、API、AWS Private CA または SDK を使用して AWS CLIまたは他の AWS のサービス を操作する場合も同様です。 AWS SDKs 名前に使用する自由記述のテキストフィールドやタグに入力したデータは、課金や診断ログに使用される場合があります。外部サーバーへの URL を提供する場合は、そのサーバーへのリクエストを検証するための認証情報を URL に含めないように強くお勧めします。

AWS Private CA プライベートキーのストレージとセキュリティのコンプライアンス

プライベート CAsは、 AWS マネージドハードウェアセキュリティモジュール (HSMsに保存されます。HSMs暗号化モジュールの FIPS PUB 140-2 Level 3 セキュリティ要件に準拠しています。

AWS Private CA Connector for Active Directory でのデータ暗号化

AWS Private CA Connector for AD は、コネクタ、テンプレート、ディレクトリ登録、サービスプリンシパル名、およびテンプレートグループのアクセスコントロールエントリに関する顧客設定データを保存します。データは転送中および保管時に暗号化されます。Connector for AD を介して発行された証明書に関する情報は、 AWS Private CA API の GetCertificateアクションを使用して検出できます。発行された証明書に関する情報、または証明書をリクエストするクライアントまたはマシンに関する情報は、 によって保存されません AWS。

AWS Private Certificate Authorityのコンプライアンス検証

サードパーティーの監査者は、複数の コンプライアンスプログラム AWS Private Certificate Authority の一環として のセキュリティと AWS コンプライアンスを評価します。これらのプログラムには、SOC、PCI、FedRAMP、HIPAA などがあります。

特定のコンプライアンスプログラムの対象となる AWS サービスのリストについては、「コンプライアンスプログラムAWS による対象範囲内のサービスコンプライアンスプログラム」を参照してください。一般的な情報については、AWS 「コンプライアンスプログラム」を参照してください。

を使用して、サードパーティーの監査レポートをダウンロードできます AWS Artifact。詳細については、「 でのレポートのダウンロード AWS Artifact」の」を参照してください。

を使用する際のお客様のコンプライアンス責任 AWS Private CA は、お客様のデータの機密性、貴社のコンプライアンス目的、適用される法律および規制によって決まります。 AWS では、コンプライアンスに役立つ以下のリソースを提供しています。

- Amazon S3 バケットの暗号化が必要な組織の場合、以下のトピックでは、 AWS Private CA アセットに対応するように暗号化を設定する方法について説明します。
  - 監査報告書の暗号化
  - CRL の暗号化
- セキュリティとコンプライアンスのクイックスタートガイド — これらのデプロイガイドでは、アーキテクチャ上の考慮事項について説明し、セキュリティとコンプライアンスに重点を置いたベースライン環境を にデプロイする手順を説明します AWS。
- 「HIPAA セキュリティとコンプライアンスの設計」ホワイトペーパー — このホワイトペーパーでは、企業が AWS を使用して HIPAA 準拠のアプリケーションを作成する方法について説明します。
- AWS コンプライアンスリソース — このワークブックとガイドのコレクションは、お客様の業界や地域に適用される場合があります。
- AWS Config デベロッパーガイドの ルールでのリソースの評価 — AWS Config サービスでは、自社のプラクティス、業界ガイドライン、および規制に対するリソースの設定の準拠状態を評価します。
- AWS Security Hub — この AWS サービスは、 内のセキュリティ状態を包括的に把握 AWS し、セキュリティ業界標準とベストプラクティスへの準拠を確認するのに役立ちます。

プライベート CA での監査レポートの使用

プライベート CA が発行または取り消したすべての証明書を一覧表示する監査報告書を作成できます。このレポートは、入力時に指定する新しい S3 バケットまたは既存の S3 バケットに保存されます。

監査報告書に暗号化保護を追加する方法については、「監査報告書の暗号化」を参照してください。

監査レポートファイルのパスとファイル名は次のとおりです。Amazon S3 バケットの ARN は bucket-name の値です。CA_ID は発行元の CA の一意の識別子です。UUID は監査報告書の固有識別子です。

```
bucket-name/audit-report/CA_ID/UUID.[json|csv]
```

30 分ごとに新しい報告書を生成し、バケットからダウンロードできます。次の例は、CSV 区切りレポートを示しています。

```
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
```

次の例では、JSON 形式のレポートを示します。

```
[
   {
      "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

が証明書 AWS Certificate Manager を更新すると、プライベート CA 監査レポートによって `requestedByServicePrincipal`フィールドに が入力されます`acm.amazonaws.com`。これは、 AWS Certificate Manager サービスが顧客に代わって AWS Private CA API の `IssueCertificate` アクションを呼び出して証明書を更新したことを示します。

監査レポート用の Amazon S3 バケットを準備する

Important

AWS Private CA は Amazon S3 オブジェクトロック の使用をサポートしていません。バケットでオブジェクトロックを有効にした場合、 AWS Private CA は監査レポートをバケットに書き込むことができません。

監査レポートを保存するには、Amazon S3 バケットを準備する必要があります。詳細については、「S3 バケットを作成するには」を参照してください。

S3 バケットは、アタッチされたアクセス許可ポリシーによって保護されている必要があります。許可されたユーザーとサービスプリンシパルには、 がバケットにオブジェクトを配置 AWS Private CA するためのPutアクセス許可と、オブジェクトを取得するGetアクセス許可が必要です。プライベート CA の AWS アカウントと ARN の両方へのアクセスを制限する以下のポリシーを適用することをお勧めします。詳細については、「Amazon S3 コンソールを使用したバケットポリシーの追加」を参照してください。

> **Note**
>
> 監査レポートを作成するコンソールの手順で、 に新しいバケット AWS Private CA を作成し、デフォルトのアクセス許可ポリシーを適用させることができます。デフォルトポリシーは CA に SourceArn 制限を適用しないため、推奨ポリシーよりも許容範囲が広くなります。デフォルトを選択した場合は、後でいつでも変更できます。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
            "Service":"acm-pca.amazonaws.com"
         },
         "Action":[
            "s3:PutObject",
            "s3:PutObjectAcl",
            "s3:GetBucketAcl",
            "s3:GetBucketLocation"
         ],
         "Resource":[
            "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*",
            "arn:aws:s3:::DOC-EXAMPLE-BUCKET"
         ],
         "Condition":{
            "StringEquals":{
               "aws:SourceAccount":"account",
               "aws:SourceArn":"arn:partition:acm-pca:region:account:certificate-
authority/CA_ID"
```

```
            }
         }
      }
   ]
}
```

監査レポートの作成

監査レポートは、コンソールまたは AWS CLIから作成できます。

監査報告書を作成するには (コンソール)

1. AWS アカウントにサインインし、https://console.aws.amazon.com/acm-pca/home で AWS Private CA コンソールを開きます。
2. [プライベート証明書認証局] ページで、リストからプライベート CA を選択します。
3. [アクション] メニューで、[監査報告書の生成] を選択します。
4. [新しい S3 バケットを作成しますか?] の [監査レポート送信先] で、[はい] を選択して一意のバケット名を入力するか、[いいえ] を選択してリストから既存のバケットを選択します。

   はい を選択した場合、 はデフォルトポリシー AWS Private CA を作成してバケットにアタッチします。[いいえ] を選択した場合は、監査報告書を生成する前にポリシーをバケットにアタッチする必要があります。監査レポート用の Amazon S3 バケットを準備する で説明されているポリシーパターンを使用してください。ポリシーを適用する方法の詳細については、「Amazon S3 コンソールを使用したバケットポリシーの追加」を参照してください
5. 出力形式 で、 JavaScript オブジェクト表記に JSON、カンマ区切り値に CSV を選択します。
6. [監査報告書の生成] を選択します。

監査報告書を作成するには (AWS CLI)

1. 使用する S3 バケットがまだなければ作成してください。
2. バケットにポリシーをアタッチする 監査レポート用の Amazon S3 バケットを準備する で説明されているポリシーパターンを使用してください。ポリシーを適用する方法の詳細については、「Amazon S3 コンソールを使用したバケットポリシーの追加」を参照してください
3. create-certificate-authority-audit-report コマンドを使用して監査レポートを作成し、準備済みの S3 バケットに配置します。

```
$ 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 bucket_name \
--audit-report-response-format JSON
```

監査レポートの取得

監査レポートを取得して検査するには、Amazon S3 コンソール、API、CLI、または SDK を使用します。詳細については、「Amazon Simple Storage Service ユーザーガイド」の「オブジェクトのダウンロード」を参照してください。

監査報告書の暗号化

オプションで、監査レポートを含む Amazon S3 バケットで暗号化を設定できます。 は、S3 のアセットに対して 2 つの暗号化モード AWS Private CA をサポートしています。

- Amazon S3 で管理された AES-256 キーを使用したサーバー側の自動暗号化。
- AWS Key Management Service および仕様に AWS KMS key 設定された を使用したカスタマーマネージド暗号化。

> **Note**
>
> AWS Private CA は、S3 によって自動的に生成されたデフォルトの KMS キーの使用をサポートしていません。

次の手順では、各暗号化オプションを設定する方法について説明します。

自動暗号化を設定するには

S3 サーバー側の暗号化を有効にするには、以下のステップを実行します。

1. https://console.aws.amazon.com/s3/でAmazon S3 コンソールを開きます。
2. バケット テーブルで、 AWS Private CA アセットを保持するバケットを選択します。
3. バケットのページで、[プロパティ] タブを選択します。
4. [デフォルト暗号化] カードを選択します。
5. [Enable (有効化)] を選択します。

6. [Amazon S3 キー (SSE-S3)] を選択します。
7. [変更の保存] をクリックします。

カスタム暗号化を設定するには

カスタムキーを使用して暗号化を有効にするには、以下のステップを実行します。

1. Amazon S3 コンソール (https://console.aws.amazon.com/s3/) を開きます。
2. バケット テーブルで、 AWS Private CA アセットを保持するバケットを選択します。
3. バケットのページで、[プロパティ] タブを選択します。
4. [デフォルト暗号化] カードを選択します。
5. [Enable (有効化)] を選択します。
6. AWS Key Management Service キー (SSE-KMS) を選択します。
7. AWS KMS キーから選択するか、ARN AWS KMS key を入力します。
8. [変更の保存] をクリックします。
9. (オプション) KMS キーがない場合は、次の AWS CLI create-key コマンドを使用して作成します。

```
$ aws kms create-key
```

出力には、KMS キーのキー ID と Amazon リソースネーム (ARN) が含まれます。以下は、その出力例です。

```
{
    "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. 次の手順を使用して、KMS キーを使用するアクセス許可を AWS Private CA サービスプリンシパルに付与します。デフォルトでは、すべての KMS キーがプライベートです。リソース所有者だけが KMS キーを使用してデータを暗号化および復号できます。ただし、リソース所有者は、他のユーザーとリソースに KMS キーへのアクセス許可を付与することができます。サービスプリンシパルは、KMS キーが保存されているのと同じリージョンにある必要があります。

    a. まず、次のget-key-policyコマンドpolicy.jsonを使用して、KMS キーのデフォルトポリシーを として保存します。

    ```
    $ aws kms get-key-policy --key-id key-id --policy-name default --output text
     > ./policy.json
    ```

    b. テキストエディタで policy.json ファイルを開きます。次のポリシーステートメントのいずれかを選択し、既存のポリシーに追加します。

    Amazon S3 バケットキーが有効になっている場合は、次のステートメントを使用してください。

    ```
    {
       "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"
          }
       }
    }
    ```

    Amazon S3 バケットキーが無効になっている場合は、次のステートメントを使用してください。

    ```
    {
       "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. 最後に、次のput-key-policyコマンドを使用して、更新されたポリシーを適用します。

```
$ aws kms put-key-policy --key-id key_id --policy-name default --policy file://
policy.json
```

のインフラストラクチャセキュリティ AWS Private Certificate Authority

マネージドサービスである AWS Private Certificate Authority は、 AWS グローバルネットワークセキュリティで保護されています。 AWS セキュリティサービスと がインフラストラクチャ AWS を保護する方法については、AWS 「 クラウドセキュリティ」を参照してください。インフラストラクチャセキュリティのベストプラクティスを使用して AWS 環境を設計するには、「 Security Pillar AWS Well-Architected Framework」の「Infrastructure Protection」を参照してください。

が AWS 公開している API コールを使用して、ネットワーク AWS Private CA 経由で にアクセスします。クライアントは以下をサポートする必要があります:

- Transport Layer Security (TLS)。TLS 1.2 は必須で TLS 1.3 がお勧めです。

- DHE (楕円ディフィー・ヘルマン鍵共有) や ECDHE (楕円曲線ディフィー・ヘルマン鍵共有) などの完全前方秘匿性 (PFS) による暗号スイート。これらのモードは、Java 7 以降など、ほとんどの最新システムでサポートされています。

また、リクエストには、アクセスキー ID と、IAM プリンシパルに関連付けられているシークレットアクセスキーを使用して署名する必要があります。または、AWS Security Token Service (AWS STS) を使用して、一時的なセキュリティ認証情報を生成し、リクエストに署名することもできます。

AWS Private CA VPC エンドポイント (AWS PrivateLink)

インターフェイス VPC エンドポイントを設定 AWS Private CA することで、VPC と の間にプライベート接続を作成できます。インターフェイスエンドポイントはAWS PrivateLink、 AWS Private CA API オペレーションにプライベートにアクセスするためのテクノロジーである を利用しています。 は、 AWS Private CA VPC と Amazon ネットワーク間のすべてのネットワークトラフィックを AWS PrivateLink ルーティングし、オープンインターネットでの露出を回避します。各 VPC エンドポイントは、VPC サブネット内の 1 つ以上の Elastic Network Interface とプライベート IP アドレスで表されます。

インターフェイス VPC エンドポイントは、インターネットゲートウェイ、NAT デバイス、VPN 接続、または AWS Direct Connect 接続 AWS Private CA なしで VPC を直接 に接続します。VPC 内のインスタンスは、 AWS Private CA API と通信するためにパブリック IP アドレスを必要としません。

VPC AWS Private CA 経由で を使用するには、VPC 内のインスタンスから接続する必要があります。または、 AWS Virtual Private Network （ AWS VPN) または を使用してプライベートネットワークを VPC に接続することもできます AWS Direct Connect。の詳細については AWS VPN、「Amazon VPC ユーザーガイド」の「VPN 接続」を参照してください。の詳細については AWS Direct Connect、「 ユーザーガイド」の「接続の作成AWS Direct Connect 」を参照してください。

AWS Private CA は を使用する必要はありませんが AWS PrivateLink、セキュリティの追加レイヤーとしてお勧めします。 AWS PrivateLink および VPC エンドポイントの詳細については、「 経由でのサービスへのアクセス AWS PrivateLink」を参照してください。

AWS Private CA VPC エンドポイントに関する考慮事項

のインターフェイス VPC エンドポイントを設定する前に AWS Private CA、次の考慮事項に注意してください。

- AWS Private CA は、一部のアベイラビリティーゾーンで VPC エンドポイントをサポートしていない場合があります。VPC エンドポイントを作成するときは、まず管理コンソールでサポートを確認してください。サポートされていないアベイラビリティーゾーンには「このアベイラビリティーゾーンではサポートされていないサービス」とマークされます。
- VPC エンドポイントはクロスリージョンリクエストをサポートしていません。 AWS Private CA に対して API コールを発行するリージョンと同じリージョンにエンドポイントを作成してください。
- VPC エンドポイントでは、Amazon Route 53 を介して Amazon 提供の DNS のみがサポートされています。独自の DNS を使用したい場合は、条件付き DNS 転送を使用できます。詳細については、Amazon VPC ユーザーガイドのDHCP Options Setsを参照してください。
- VPCエンドポイントにアタッチされたセキュリティグループでは、VPCのプライベートサブネットから、ポート 443 で着信接続を許可する必要があります。
- AWS Certificate Manager は VPC エンドポイントをサポートしていません。
- FIPS エンドポイント (およびそのリージョン) は VPC エンドポイントをサポートしていません。

AWS Private CA API は現在、次の で VPC エンドポイントをサポートしています AWS リージョン。

- 米国東部 (オハイオ)
- 米国東部 (バージニア北部)
- 米国西部 (北カリフォルニア)
- 米国西部 (オレゴン)
- アフリカ (ケープタウン)
- アジアパシフィック (香港)
- アジアパシフィック (ムンバイ)
- アジアパシフィック (大阪)
- アジアパシフィック (ソウル)
- アジアパシフィック (シンガポール)
- アジアパシフィック (シドニー)
- アジアパシフィック (東京)
- カナダ (中部)
- 欧州 (フランクフルト)

- 欧州 (アイルランド)
- 欧州 (ロンドン)
- 欧州 (パリ)
- 欧州 (ストックホルム)
- 欧州 (ミラノ)
- イスラエル (テルアビブ)
- 中東 (バーレーン)
- 南米 (サンパウロ)

AWS Private CA用の VPC エンドポイントの作成

AWS Private CA サービスの VPC エンドポイントは、https://console.aws.amazon.com/vpc/ の VPC コンソールまたは を使用して作成できます AWS Command Line Interface。詳細については、「Amazon VPC ユーザーガイド」の「インターフェイスエンドポイントの作成」の手順を参照してください。 は、VPC 内のすべての API オペレーションへの呼び出し AWS Private CA をサポートしています。

エンドポイントのプライベート DNS ホスト名を有効にすると、デフォルトの AWS Private CA エンドポイントはお客様の VPC エンドポイントに解決されます。デフォルトのサービスエンドポイントの詳しい一覧については、「サービスエンドポイントとクォータ」を参照してください。

プライベート DNS ホスト名を有効にしていない場合、Amazon VPC は次の形式で使用できる DNS エンドポイント名を提供します。

```
vpc-endpoint-id.acm-pca.region.vpce.amazonaws.com
```

> **Note**
>
> 値*#####*は、米国東部 (オハイオ) リージョンの など AWS Private CA、 でサポートされている AWS リージョン`us-east-2`のリージョン識別子を表します。のリストについては AWS Private CA、AWS 「 Certificate Manager プライベート認証機関のエンドポイントとクォータ」を参照してください。

詳細については、「Amazon AWS Private CA VPC ユーザーガイド」の「VPC エンドポイント (AWS PrivateLink) 」を参照してください。

AWS Private CA用の VPC エンドポイントポリシーを作成する

の Amazon VPC エンドポイントのポリシーを作成して AWS Private CA 、以下を指定できます。

- アクションを実行できるプリンシパル
- 実行可能なアクション
- アクションを実行できるリソース

詳細については、Amazon VPC ユーザーガイドの「VPC エンドポイントによるサービスのアクセスコントロール」を参照してください。

例 - AWS Private CA アクションの VPC エンドポイントポリシー

エンドポイントにアタッチされると、次のポリシーは、すべてのプリンシパルに AWS Private CA アクション `IssueCertificate`、、`DescribeCertificateAuthority`、`GetCertificate`、`GetCertificateAu` および へのアクセスを許可しますListTags。各スタンザのリソースはプライベート CA です。最初のスタンザは、指定されたプライベート CA と証明書テンプレートを使用して、エンドエンティティ証明書の作成を許可します。使用するテンプレートを制御しない場合、Condition セクションは必要ありません。ただし、これを削除すると、すべてのプリンシパルが CA 認定とエンドエンティティ証明書を交付できるようになります。

```
{
      "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"
            ]
         }
      ]
   }
```

AWS Private Certificate Authorityでのログ記録とモニタリング

モニタリングは、 および AWS ソリューションの信頼性、可用性、パフォーマンスを維持する AWS Private Certificate Authority 上で重要な部分です。マルチポイント障害が発生した場合は、その障害をより簡単にデバッグできるように、 AWS ソリューションのすべての部分からモニタリングデータを収集する必要があります。

以下のトピックでは、 で使用できる AWS クラウドモニタリングツールについて説明します AWS Private CA。

トピック

- サポートされている CloudWatch メトリクス
- CloudWatch イベントの使用
- の使用 CloudTrail

サポートされている CloudWatch メトリクス

Amazon CloudWatch は AWS リソースのモニタリングサービスです。を使用して CloudWatch 、メトリクスの収集と追跡、アラームの設定、 AWS リソースの変更への自動対応を行うことができます。 CloudWatch メトリクスは少なくとも 1 回発行されます。

AWS Private CA では、次の CloudWatch メトリクスがサポートされています。

メトリクス	説明
CRLGenerated	証明書失効リスト (CRL) が生成されました。このメトリクスは、プライベート CA にのみ適用されます。
MisconfiguredCRLBucket	CRL に指定された S3 バケットが正しく設定されていません。バケットポリシーを確認します。このメトリクスは、プライベート CA にのみ適用されます。
Time	発行リクエストから発行の完了 (または失敗) までの時間 (ミリ秒単位)。このメトリクスは IssueCertificateオペレーションにのみ適用されます。
Success	証明書が正常に発行されました。このメトリクスは IssueCertificateオペレーションにのみ適用されます。
Failure	オペレーションに失敗しました。このメトリクスは IssueCertificateオペレーションにのみ適用されます。

CloudWatch メトリクスの詳細については、以下のトピックを参照してください。

- Amazon CloudWatch メトリクスの使用
- Amazon CloudWatch アラームの作成

CloudWatch イベントの使用

Amazon CloudWatch Events を使用して AWS サービスを自動化し、アプリケーションの可用性の問題やリソースの変更などのシステムイベントに自動的に対応できます。 AWS サービスからのイベントは、ほぼリアルタイムで CloudWatch イベントに配信されます。シンプルなルールを作成して、関心のあるイベントと、イベントがルールに一致するときに実行する自動アクションを指定できま

す。 CloudWatch イベントは少なくとも 1 回発行されます。詳細については、CloudWatch 「イベント でトリガーするイベントルールの作成」を参照してください。

CloudWatch イベントは Amazon を使用してアクションに変換されます EventBridge。では EventBridge、イベントを使用して、 AWS Lambda 関数、 AWS Batch ジョブ、Amazon SNS トピックなど、さまざまなターゲットをトリガーできます。詳細については、「Amazon とは」を参照してください EventBridge。

プライベート CA 作成時の成功または失敗

これらのイベントは、 CreateCertificateAuthorityオペレーションによってトリガーされます。

成功

成功すると、オペレーションは新しい CA の ARN を返します。

```
{
   "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

失敗すると、オペレーションは CA の ARN を返します。ARN を使用して DescribeCertificateAuthorityを呼び出し、CA のステータスを判断できます。

```
{
   "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"
   }
}
```

証明書発行時の成功または失敗

これらのイベントは、 IssueCertificateオペレーションによってトリガーされます。

成功

成功すると、オペレーションは CA と新しい証明書の ARN を返します。

```
{
   "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

失敗すると、証明書 ARN と CA の ARN が返されます。証明書 ARN を使用すると、GetCertificateを呼び出して失敗の理由を表示できます。

```
{
   "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"
   }
}
```

証明書取り消しの成功

このイベントは、 RevokeCertificateオペレーションによってトリガーされます。

取り消しに失敗した場合、または証明書がすでに取り消されている場合、イベントは送信されません。

成功

成功すると、CA および取り消された証明書 ARN が返されます。

```
{
   "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"
   }
}
```

CRL 生成時の成功または失敗

これらのイベントは RevokeCertificateオペレーションによってトリガーされ、証明書失効リスト (CRL) が作成されます。

成功

成功すると、オペレーションは CRL に関連付けられた CA の ARN を返します。

```
{
   "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 — アクセス許可エラーのため CRL を Amazon S3 に保存できませんでした

このエラーが発生した場合は、Amazon S3 バケットのアクセス許可を確認してください。

```
{
   "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 — 内部エラーのため CRL を Amazon S3 に保存できませんでした

このエラーが発生した場合は、オペレーションを再試行してください。

```
{
   "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 – CRL の作成 AWS Private CA に失敗しました

このエラーをトラブルシューティングするには、CloudWatch メトリクス を確認してください。

```
{
   "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."
   }
}
```

CA 監査報告書作成時の成功または失敗

これらのイベントは、 CreateCertificateAuthorityAuditReportオペレーションによってトリガーされます。

成功

成功すると、オペレーションは CA の ARN と監査報告書の ID を返します。

```
{
   "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

監査レポートは、 が Amazon S3 バケットに対するPUTアクセス許可 AWS Private CA を持っていない場合、バケットで暗号化が有効になっている場合、またはその他の理由で失敗することがあります。

```
{
   "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"
   }
}
```

の使用 CloudTrail

を使用してAWS CloudTrail、 によって行われた API コールを記録できます AWS Private Certificate Authority。詳細については、以下のトピックを参照してください。

トピック

- ポリシーの作成
- ポリシーの取得
- ポリシーの削除
- 認証機関の作成
- GenerateCRL
- GenerateOCSPResponse
- 監査レポートの作成
- 認証機関の削除
- 証明機関の復元
- 認証機関の説明

- 認証機関証明書の取得
- 証明書認証機関署名リクエストの取得
- 証明書の取得
- 認証機関の証明書のインポート
- 証明書の発行
- 認証機関の一覧表示
- タグを一覧表にする
- 証明書の失効
- プライベート認証機関のタグ付け
- プライベート認証機関によるタグの削除
- 認証機関の更新

ポリシーの作成

次の CloudTrail 例は、 PutPolicyオペレーションの呼び出しの結果を示しています。

```
{
   "eventVersion":"1.08",
   "userIdentity":{
      },
      "invokedBy":"agent"
   },
   "eventTime":"2021-02-26T21:25:36Z",
   "eventSource":"acm-pca.amazonaws.com",
   "eventName":"PutPolicy",
   "awsRegion":"region",
   "sourceIPAddress":"xx.xx.xx.xx",
   "userAgent":"agent",
   "requestParameters":{
      "resourceArn":"arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
      "policy":"{\"Version\":\"2012-10-17\",\"Statement\":[{\"Sid\":
\"01234567-89ab-cdef-0123-456789abcdef4-external-principals\",\"Effect\":\"Allow
\",\"Principal\":{\"AWS\":\"account\"},\"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\"}}},{\"Sid\":\"01234567-89ab-cdef-0123-456789abcdef-external-principals
```

```
\",\"Effect\":\"Allow\",\"Principal\":{\"AWS\":\"account\"},\"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\"}]}"
   },
   "responseElements":null,
   "requestID":"01234567-89ab-cdef-0123-456789abcdef",
   "eventID":"01234567-89ab-cdef-0123-456789abcdef",
   "readOnly":false,
   "eventType":"AwsApiCall",
   "managementEvent":true,
   "eventCategory":"Management",
   "recipientAccountId":"account"
}
```


ポリシーの取得

次の CloudTrail 例は、 GetPolicyオペレーションの呼び出しの結果を示しています。


```
{
   "eventVersion":"1.08",
   "userIdentity":{
      "type":"AssumedRole",
      "principalId":"account",
      "arn":"arn:aws:sts::account:assumed-role/role",
      "accountId":"account",
      "accessKeyId":"key_ID",
      "sessionContext":{
         "sessionIssuer":{
            "type":"Role",
            "principalId":"account",
            "arn":"arn:aws:iam::account:role/role",
            "accountId":"account",
            "userName":"name"
         },
         "webIdFederationData":{

         },
         "attributes":{
            "mfaAuthenticated":"false",
            "creationDate":"2021-02-26T20:49:51Z"
         }
```

```
      }
   },
   "eventTime":"2021-02-26T21:19:14Z",
   "eventSource":"acm-pca.amazonaws.com",
   "eventName":"GetPolicy",
   "awsRegion":"region",
   "sourceIPAddress":"IP_address",
   "userAgent":"agent",
   "errorCode":"ResourceNotFoundException",
   "errorMessage":"Could not find policy for resource arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566.",
   "requestParameters":{
      "resourceArn":"arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
   },
   "responseElements":null,
   "requestID":"request_ID",
   "eventID":"event_ID",
   "readOnly":true,
   "eventType":"AwsApiCall",
   "managementEvent":true,
   "eventCategory":"Management",
   "recipientAccountId":"account"
}
```

ポリシーの削除

次の CloudTrail 例は、 DeletePolicyオペレーションの呼び出しの結果を示しています。

```
{
   "eventVersion":"1.08",
   "userIdentity":{
      "type":"AssumedRole",
      "principalId":"account",
      "arn":"arn:aws:sts::account:assumed-role/role",
      "accountId":"account",
      "accessKeyId":"key_ID",
      "sessionContext":{
         "sessionIssuer":{
            "type":"Role",
            "principalId":"account",
            "arn":"arn:aws:iam::account:role/role",
            "accountId":"account",
```

```
            "userName":"name"
         },
         "webIdFederationData":{

         },
         "attributes":{
            "mfaAuthenticated":"false",
            "creationDate":"2021-02-26T21:23:17Z"
         }
      }
   },
   "eventTime":"2021-02-26T21:23:31Z",
   "eventSource":"acm-pca.amazonaws.com",
   "eventName":"DeletePolicy",
   "awsRegion":"region",
   "sourceIPAddress":"IP_address",
   "userAgent":"agent",
   "requestParameters":{
      "resourceArn":"arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
   },
   "responseElements":null,
   "requestID":"request_ID",
   "eventID":"event_ID",
   "readOnly":false,
   "eventType":"AwsApiCall",
   "managementEvent":true,
   "eventCategory":"Management",
   "recipientAccountId":"account"
}
```

認証機関の作成

次の CloudTrail 例は、 CreateCertificateAuthorityオペレーションの呼び出しの結果を示しています。

```
{
   "eventVersion":"1.05",
   "userIdentity":{
      "type":"IAMUser",
      "principalId":"account",
      "arn":"arn:aws:iam::account:user/name",
      "accountId":"account",
      "accessKeyId":"key_ID"
```

```
    },
    "eventTime":"2018-01-26T21:22:33Z",
    "eventSource":"acm-pca.amazonaws.com",
    "eventName":"CreateCertificateAuthority",
    "awsRegion":"region",
    "sourceIPAddress":"IP_address",
    "userAgent":"agent",
    "requestParameters":{
       "certificateAuthorityConfiguration":{
          "keyType":"RSA2048",
          "signingAlgorithm":"SHA256WITHRSA",
          "subject":{
             "country":"US",
             "organization":"Example Company",
             "organizationalUnit":"Corp",
             "state":"WA",
             "commonName":"www.example.com",
             "locality":"Seattle"
          }
       },
       "revocationConfiguration":{
          "crlConfiguration":{
             "enabled":true,
             "expirationInDays":3650,
             "customCname":"your-custom-name",
             "s3BucketName":"your-bucket-name"
          }
       },
       "certificateAuthorityType":"SUBORDINATE",
       "idempotencyToken":"98256344"
    },
    "responseElements":{
       "certificateAuthorityArn":"arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
    },
    "requestID":"request_ID",
    "eventID":"event_ID",
    "eventType":"AwsApiCall",
    "recipientAccountId":"account"
}
```

GenerateCRL

次の CloudTrail 例は、GenerateCRLイベントのレコードを示しています。

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "accountId": "account",
        "invokedBy": "acm-pca.amazonaws.com"
    },
    "eventTime": "2021-02-09T17:37:45Z",
    "eventSource": "acm-pca.amazonaws.com",
    "eventName": "GenerateCRL",
    "awsRegion": "region",
    "sourceIPAddress": "acm-pca.amazonaws.com",
    "userAgent": "acm-pca.amazonaws.com",
    "requestParameters": null,
    "responseElements": null,
    "eventID": "01234567-89ab-cdef-0123-456789abcdef",
    "readOnly": false,
    "resources": [
        {
            "type": "AWS::ACMPCA::CertificateAuthority",
            "ARN": "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
        }
    ],
    "eventType": "AwsServiceEvent",
    "managementEvent": true,
    "eventCategory": "Management",
    "recipientAccountId": "account"
}
```

GenerateOCSPResponse

次の CloudTrail 例は、GenerateOCSPResponseイベントのレコードを示しています。

```
{
   "eventVersion":"1.08",
   "userIdentity":{
      "accountId":"account",
      "invokedBy":"acm-pca.amazonaws.com"
   },
```

```
   "eventTime":"2021-02-08T23:52:29Z",
   "eventSource":"acm-pca.amazonaws.com",
   "eventName":"GenerateOCSPResponse",
   "awsRegion":"region",
   "sourceIPAddress":"acm-pca.amazonaws.com",
   "userAgent":"acm-pca.amazonaws.com",
   "eventID":"01234567-89ab-cdef-0123-456789abcdef",
   "readOnly":false,
   "resources":[
      {
         "type":"AWS::ACMPCA::Certificate",
         "ARN":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID"
      }
   ]
}
```

監査レポートの作成

次の CloudTrail 例は、 CreateCertificateAuthorityAuditReportオペレーションの呼び出しの結果を示しています。

```
{
   "eventVersion":"1.05",
   "userIdentity":{
      "type":"IAMUser",
      "principalId":"account",
      "arn":"arn:aws:iam::account:user/name",
      "accountId":"account",
      "accessKeyId":"key_ID"
   },
   "eventTime":"2018-01-26T21:56:00Z",
   "eventSource":"acm-pca.amazonaws.com",
   "eventName":"CreateCertificateAuthorityAuditReport",
   "awsRegion":"region",
   "sourceIPAddress":"IP_address",
   "userAgent":"agent",
   "requestParameters":{
      "certificateAuthorityArn":"arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
      "s3BucketName":"bucket_name",
      "auditReportResponseFormat":"JSON"
   },
```

```
   "responseElements":{
      "auditReportId":"report_ID",
      "s3Key":"audit-report/CA_ID/audit_report_ID.json"
   },
   "requestID":"request_ID",
   "eventID":"event_ID",
   "eventType":"AwsApiCall",
   "recipientAccountId":"account"
}
```

認証機関の削除

次の CloudTrail 例は、 DeleteCertificateAuthorityオペレーションの呼び出しの結果を示しています。この例では、認証機関が ACTIVE 状態であるため、その認証機関を削除できません。

```
{
   "eventVersion":"1.05",
   "userIdentity":{
      "type":"IAMUser",
      "principalId":"account",
      "arn":"arn:aws:iam::account:user/name",
      "accountId":"account",
      "accessKeyId":"key_ID"
   },
   "eventTime":"2018-01-26T22:01:11Z",
   "eventSource":"acm-pca.amazonaws.com",
   "eventName":"DeleteCertificateAuthority",
   "awsRegion":"region",
   "sourceIPAddress":"IP_address",
   "userAgent":"agent",
   "errorCode":"InvalidStateException",
   "errorMessage":"The certificate authority is not in a valid state for deletion.",
   "requestParameters":{
      "certificateAuthorityArn":"arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
   },
   "responseElements":null,
   "requestID":"request_ID",
   "eventID":"event_ID",
   "eventType":"AwsApiCall",
   "recipientAccountId":"account"
}
```

証明機関の復元

次の CloudTrail 例は、 RestoreCertificateAuthorityオペレーションの呼び出しの結果を示しています。この例では、認証機関が DELETED 状態ではないため、その認証機関を復元できません。


```
{
   "eventVersion":"1.05",
   "userIdentity":{
      "type":"IAMUser",
      "principalId":"account",
      "arn":"arn:aws:iam::account:user/name",
      "accountId":"account",
      "accessKeyId":"key_ID"
   },
   "eventTime":"2018-01-26T22:01:11Z",
   "eventSource":"acm-pca.amazonaws.com",
   "eventName":"RestoreCertificateAuthority",
   "awsRegion":"region",
   "sourceIPAddress":"xIP_address",
   "userAgent":"agent",
   "errorCode":"InvalidStateException",
   "errorMessage":"The certificate authority is not in a valid state for restoration.",
   "requestParameters":{
      "certificateAuthorityArn":"arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
   },
   "responseElements":null,
   "requestID":"request_ID",
   "eventID":"event_ID",
   "eventType":"AwsApiCall",
   "recipientAccountId":"account"
}
```


認証機関の説明

次の CloudTrail 例は、 DescribeCertificateAuthorityオペレーションの呼び出しの結果を示しています。


```
{
   "eventVersion":"1.05",
   "userIdentity":{
      "type":"IAMUser",
```

```
      "principalId":"account",
      "arn":"arn:aws:iam::account:user/name",
      "accountId":"account",
      "accessKeyId":"key_ID"
   },
   "eventTime":"2018-01-26T21:58:18Z",
   "eventSource":"acm-pca.amazonaws.com",
   "eventName":"DescribeCertificateAuthority",
   "awsRegion":"region",
   "sourceIPAddress":"IP_address",
   "userAgent":"agent",
   "requestParameters":{
      "certificateAuthorityArn":"arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
   },
   "responseElements":null,
   "requestID":"request_ID",
   "eventID":"event_ID",
   "eventType":"AwsApiCall",
   "recipientAccountId":"account"
}
```

認証機関証明書の取得

次の CloudTrail 例は、 GetCertificateAuthorityCertificateオペレーションの呼び出しの結果を示しています。

```
{
   "eventVersion":"1.05",
   "userIdentity":{
      "type":"IAMUser",
      "principalId":"account",
      "arn":"arn:aws:iam::account:user/name",
      "accountId":"account",
      "accessKeyId":"key_ID"
   },
   "eventTime":"2018-01-26T22:03:52Z",
   "eventSource":"acm-pca.amazonaws.com",
   "eventName":"GetCertificateAuthorityCertificate",
   "awsRegion":"region",
   "sourceIPAddress":"IP_address",
   "userAgent":"agent",
   "requestParameters":{
```

```
      "certificateAuthorityArn":"arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
   },
   "responseElements":null,
   "requestID":"request_ID",
   "eventID":"event_ID",
   "eventType":"AwsApiCall",
   "recipientAccountId":"account"
}
```

証明書認証機関署名リクエストの取得

次の CloudTrail 例は、 GetCertificateAuthorityCsrオペレーションの呼び出しの結果を示しています。

```
{
   "eventVersion":"1.05",
   "userIdentity":{
      "type":"IAMUser",
      "principalId":"account",
      "arn":"arn:aws:iam::account:user/name",
      "accountId":"account",
      "accessKeyId":"key_ID"
   },
   "eventTime":"2018-01-26T21:40:33Z",
   "eventSource":"acm-pca.amazonaws.com",
   "eventName":"GetCertificateAuthorityCsr",
   "awsRegion":"region",
   "sourceIPAddress":"IP_address",
   "userAgent":"agent",
   "requestParameters":{
      "certificateAuthorityArn":"arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
   },
   "responseElements":null,
   "requestID":"request_ID",
   "eventID":"event_ID",
   "eventType":"AwsApiCall",
   "recipientAccountId":"account"
}
```

証明書の取得

次の CloudTrail 例は、 GetCertificateオペレーションの呼び出しの結果を示しています。

```
{
   "eventVersion":"1.05",
   "userIdentity":{
      "type":"IAMUser",
      "principalId":"account",
      "arn":"arn:aws:iam::account:user/name",
      "accountId":"account",
      "accessKeyId":"key_ID"
   },
   "eventTime":"2018-01-26T22:22:54Z",
   "eventSource":"acm-pca.amazonaws.com",
   "eventName":"GetCertificate",
   "awsRegion":"region",
   "sourceIPAddress":"IP_address",
   "userAgent":"agent",
   "requestParameters":{
      "certificateAuthorityArn":"arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
      "certificateArn":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID"
   },
   "responseElements":null,
   "requestID":"request_ID",
   "eventID":"event_ID",
   "eventType":"AwsApiCall",
   "recipientAccountId":"account"
}
```

認証機関の証明書のインポート

次の CloudTrail 例は、 ImportCertificateAuthorityCertificateオペレーションの呼び出しの結果を示しています。

```
{
   "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"
}
```

証明書の発行

次の CloudTrail 例は、 IssueCertificateオペレーションの呼び出しの結果を示しています。

```
{
   "eventVersion":"1.05",
   "userIdentity":{
      "type":"IAMUser",
      "principalId":"account",
      "arn":"arn:aws:iam::account:user/name",
      "accountId":"account",
      "accessKeyId":"key_ID"
   },
   "eventTime":"2018-01-26T22:18:43Z",
   "eventSource":"acm-pca.amazonaws.com",
   "eventName":"IssueCertificate",
   "awsRegion":"region",
   "sourceIPAddress":"xIP_address",
   "userAgent":"agent",
   "requestParameters":{
      "certificateAuthorityArn":"arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
      "csr":{
         "hb":[
            45,
            45,
            ...10
         ],
         "offset":0,
         "isReadOnly":false,
         "bigEndian":true,
         "nativeByteOrder":false,
         "mark":-1,
         "position":1090,
         "limit":1090,
         "capacity":1090,
         "address":0
      },
      "signingAlgorithm":"SHA256WITHRSA",
```

```
      "validity":{
         "value":365,
         "type":"DAYS"
      },
      "idempotencyToken":"1234"
   },
   "responseElements":{
      "certificateArn":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID"
   },
   "requestID":"request_ID",
   "eventID":"event_ID",
   "eventType":"AwsApiCall",
   "recipientAccountId":"account"
}
```

認証機関の一覧表示

次の CloudTrail 例は、 ListCertificateAuthoritiesオペレーションの呼び出しの結果を示しています。

```
{
   "eventVersion":"1.05",
   "userIdentity":{
      "type":"IAMUser",
      "principalId":"account",
      "arn":"arn:aws:iam::account:user/name",
      "accountId":"account",
      "accessKeyId":"key_ID"
   },
   "eventTime":"2018-01-26T22:09:43Z",
   "eventSource":"acm-pca.amazonaws.com",
   "eventName":"ListCertificateAuthorities",
   "awsRegion":"region",
   "sourceIPAddress":"IP_address",
   "userAgent":"agent",
   "requestParameters":{
      "maxResults":10
   },
   "responseElements":null,
   "requestID":"request_ID",
   "eventID":"event_ID",
   "eventType":"AwsApiCall",
   "recipientAccountId":"account"
```

```
}
```

タグを一覧表にする

次の CloudTrail 例は、 ListTagsオペレーションの呼び出しの結果を示しています。

```
{
   "eventVersion":"1.05",
   "userIdentity":{
      "type":"IAMUser",
      "principalId":"account",
      "arn":"arn:aws:iam::account:user/name",
      "accountId":"account",
      "accessKeyId":"key_ID"
   },
   "eventTime":"2018-02-02T00:21:56Z",
   "eventSource":"acm-pca.amazonaws.com",
   "eventName":"ListTags",
   "awsRegion":"region",
   "sourceIPAddress":"IP_address",
   "userAgent":"agent",
   "requestParameters":{
      "certificateAuthorityArn":"arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
   },
   "responseElements":{
      "tags":[
         {
            "key":"Admin",
            "value":"Alice"
         },
         {
            "key":"User",
            "value":"Bob"
         }
      ]
   },
   "requestID":"request_ID",
   "eventID":"event_ID",
   "eventType":"AwsApiCall",
   "recipientAccountId":"account"
}
```

証明書の失効

次の CloudTrail 例は、 RevokeCertificateオペレーションの呼び出しの結果を示しています。

```
{
   "eventVersion":"1.05",
   "userIdentity":{
      "type":"IAMUser",
      "principalId":"account",
      "arn":"arn:aws:iam::account:user/name",
      "accessKeyId": "key_ID"
   },
   "eventTime": "2018-01-26T22:35:03Z",
   "eventSource": "acm-pca.amazonaws.com",
   "eventName": "RevokeCertificate",
   "awsRegion": "region",
   "sourceIPAddress": "IP_address",
   "userAgent": "agent",
   "requestParameters": {
      "certificateAuthorityArn": "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
      "certificateSerial": "67:07:44:76:83:a9:b7:f4:05:56:27:ff:d5:5c:eb:cc",
      "revocationReason": "KEY_COMPROMISE"
   },
   "responseElements": null,
   "requestID": "request_ID",
   "eventID": "event_ID",
   "eventType": "AwsApiCall",
   "recipientAccountId": "account"
}
```

プライベート認証機関のタグ付け

次の CloudTrail 例は、 TagCertificateAuthorityオペレーションの呼び出しの結果を示しています。

```
{
   "eventVersion":"1.05",
   "userIdentity":{
      "type":"IAMUser",
      "principalId":"account",
      "arn":"arn:aws:iam::account:user/name",
      "accountId":"account",
      "accessKeyId":"key_ID"
```

```
   },
   "eventTime":"2018-02-02T00:18:48Z",
   "eventSource":"acm-pca.amazonaws.com",
   "eventName":"TagCertificateAuthority",
   "awsRegion":"region",
   "sourceIPAddress":"IP_address",
   "userAgent":"agent",
   "requestParameters":{
      "certificateAuthorityArn":"arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
      "tags":[
         {
            "key":"Admin",
            "value":"Alice"
         }
      ]
   },
   "responseElements":null,
   "requestID":"request_ID",
   "eventID":"event_ID",
   "eventType":"AwsApiCall",
   "recipientAccountId":"account"
}
```

プライベート認証機関によるタグの削除

次の CloudTrail 例は、 UntagCertificateAuthorityオペレーションの呼び出しの結果を示しています。

```
{
   "eventVersion":"1.05",
   "userIdentity":{
      "type":"IAMUser",
      "principalId":"account",
      "arn":"arn:aws:iam::account:user/name",
      "accountId":"account",
      "accessKeyId":"key_ID"
   },
   "eventTime":"2018-02-02T00:21:50Z",
   "eventSource":"acm-pca.amazonaws.com",
   "eventName":"UntagCertificateAuthority",
   "awsRegion":"region",
   "sourceIPAddress":"IP_address",
   "userAgent":"agent",
```

```
   "requestParameters":{
      "certificateAuthorityArn":"arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
      "tags":[
         {
            "key":"Admin",
            "value":"Alice"
         }
      ]
   },
   "responseElements":null,
   "requestID":"request_ID",
   "eventID":"event_ID",
   "eventType":"AwsApiCall",
   "recipientAccountId":"account"
}
```

認証機関の更新

次の CloudTrail 例は、 UpdateCertificateAuthorityオペレーションの呼び出しの結果を示しています。

```
{
   "eventVersion":"1.05",
   "userIdentity":{
      "type":"IAMUser",
      "principalId":"account",
      "arn":"arn:aws:iam::account:user/name",
      "accountId":"account",
      "accessKeyId":"key_ID"
   },
   "eventTime":"2018-01-26T22:08:59Z",
   "eventSource":"acm-pca.amazonaws.com",
   "eventName":"UpdateCertificateAuthority",
   "awsRegion":"region",
   "sourceIPAddress":"IP_address",
   "userAgent":"agent",
   "requestParameters":{
      "certificateAuthorityArn":"arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",

      "revocationConfiguration":{
         "crlConfiguration":{
```

```
            "enabled":true,
            "expirationInDays":3650,
            "customCname":"your-custom-name",
            "s3BucketName":"your-bucket-name"
         }
      },
      "status":"DISABLED"
   },
   "responseElements":null,
   "requestID":"request_ID",
   "eventID":"event_ID",
   "eventType":"AwsApiCall",
   "recipientAccountId":"account"
}
```

AWS Private CA デプロイの計画

AWS Private CA では、ルート認証機関 (CA) から下位 CAsを完全にクラウドベースの制御できます。安全性、保守性、拡張性、組織のニーズに適した PKI には、綿密な計画が不可欠です。このセクションでは、CA 階層の設計、プライベート CA とプライベートエンドエンティティ証明書のライフサイクルの管理、セキュリティのベストプラクティスの適用に関するガイダンスを提供します。

このセクションでは、プライベート認証機関 (CA) を作成する前に、 の使用 AWS Private CA を準備する方法について説明します。オンライン証明書ステータスプロトコル (OCSP) または証明書失効リスト (CRL) を使用して失効サポートを追加するオプションについても説明します。

さらに、組織がプライベートルート CA 認証情報を ではなくオンプレミスでホストするかどうかを決定する必要があります AWS。その場合は、 を使用する前に、セルフマネージドプライベート PKI を設定して保護する必要があります AWS Private CA。このシナリオでは、 の外部にある親 CA によって AWS Private CA バックアップされた下位 CA を に作成します AWS Private CA。詳細については、「外部の親 CA によって署名された下位 CA 証明書のインストール」を参照してください。

トピック

- AWS アカウントと のセットアップ AWS CLI
- CA 階層の設計
- プライベート CA ライフサイクルの管理
- 証明書失効方法の設定
- 認証局モード
- レジリエンスの計画

AWS アカウントと のセットアップ AWS CLI

すでに Amazon Web Services (AWS) カスタマーの場合、 AWS Private CAを使用できるようにサインアップする必要があります。このアカウントでは、利用可能なすべてのサービスにアクセスできますが、使用したサービスに対してのみ課金されます。

> **Note**
>
> AWS Private CA は AWS 無料利用枠 では使用できません。

トピック

- にサインアップする AWS アカウント
- 管理アクセスを持つユーザーを作成する
- のインストール AWS Command Line Interface

にサインアップする AWS アカウント

がない場合は AWS アカウント、次の手順を実行して作成します。

にサインアップするには AWS アカウント

1. https://portal.aws.amazon.com/billing/signup を開きます。
2. オンラインの手順に従います。

   サインアップ手順の一環として、通話呼び出しを受け取り、電話キーパッドで検証コードを入力するように求められます。

   にサインアップすると AWS アカウント、 AWS アカウントのルートユーザーが作成されます。ルートユーザーには、アカウントのすべての AWS のサービス とリソースへのアクセス権があります。セキュリティのベストプラクティスとして、ユーザーに管理アクセスを割り当て、ルートユーザーのみを使用してルートユーザーアクセスが必要なタスクを実行してください。

AWS サインアッププロセスが完了すると、 から確認メールが送信されます。https://aws.amazon.com/ の [マイアカウント] を選んで、いつでもアカウントの現在のアクティビティを表示し、アカウントを管理できます。

管理アクセスを持つユーザーを作成する

にサインアップしたら AWS アカウント、 を保護し AWS アカウントのルートユーザー、 を有効にして AWS IAM Identity Center、日常的なタスクにルートユーザーを使用しないように管理ユーザーを作成します。

のセキュリティ保護 AWS アカウントのルートユーザー

1. ルートユーザーを選択し、 AWS アカウント E メールアドレスを入力して、アカウント所有者AWS Management Consoleとして にサインインします。次のページでパスワードを入力します。

ルートユーザーを使用してサインインする方法については、AWS サインイン ユーザーガイドの「ルートユーザーとしてサインインする」を参照してください。

2. ルートユーザーの多要素認証 (MFA) を有効にします。

   手順については、「IAM ユーザーガイド」の AWS アカウント 「ルートユーザーの仮想 MFA デバイスを有効にする (コンソール)」を参照してください。

管理アクセスを持つユーザーを作成する

1. IAM アイデンティティセンターを有効にします。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「AWS IAM Identity Centerの有効化」を参照してください。

2. IAM アイデンティティセンターで、ユーザーに管理アクセスを付与します。

   を ID ソース IAM アイデンティティセンターディレクトリ として使用する方法のチュートリアルについては、「 ユーザーガイド」の「デフォルト でユーザーアクセスを設定する IAM アイデンティティセンターディレクトリAWS IAM Identity Center 」を参照してください。

管理アクセス権を持つユーザーとしてサインインする

- IAM アイデンティティセンターのユーザーとしてサインインするには、IAM アイデンティティセンターのユーザーの作成時に E メールアドレスに送信されたサインイン URL を使用します。

  IAM Identity Center ユーザーを使用してサインインする方法については、「 AWS サインイン ユーザーガイド」の AWS 「 アクセスポータルへのサインイン」を参照してください。

追加のユーザーにアクセス権を割り当てる

1. IAM アイデンティティセンターで、最小特権のアクセス許可を適用するというベストプラクティスに従ったアクセス許可セットを作成します。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「権限設定を作成する」を参照してください。

2. グループにユーザーを割り当て、そのグループにシングルサインオンアクセス権を割り当てます。

手順については、「AWS IAM Identity Center ユーザーガイド」の「グループの参加」を参照してください。

のインストール AWS Command Line Interface

をインストールしていない AWS CLI が、使用したい場合は、「」の指示に従ってくださいAWS Command Line Interface。このガイドは、エンドポイント、リージョン、認証の詳細が設定されていることを前提としており、これらのパラメータはサンプルコマンドから省略しています。

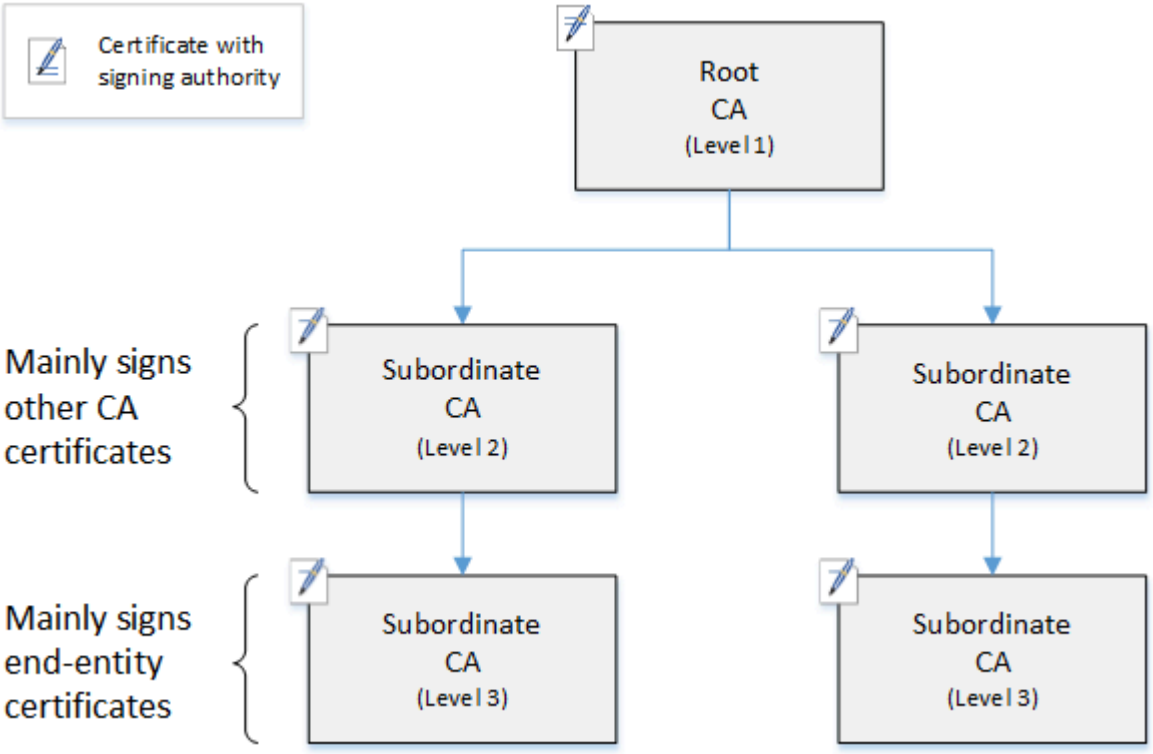
CA 階層の設計

では AWS Private CA、最大 5 つのレベルで認証機関の階層を作成できます。階層ツリーの最上位にあるルート CA は、任意の数のブランチを持つことができます。ルート CA は、各ブランチの上位 CA のレベルを 4 つまで設定できます。また、それぞれ独自のルートを持つ複数の階層を作成することもできます。

適切に設計された CA 階層には、次のような利点があります。

- 各 CA に適したきめ細かなセキュリティ制御
- ロードバランシングとセキュリティを向上するための管理タスクの分割
- 日常業務における信頼が限定され、取り消し可能な CA の使用
- 有効期間と証明書パスの制限

次の図は、単純な 3 レベルの CA 階層を示しています。



ツリー内の各 CA は、署名権限 (pen-and-paper アイコンでシンボル化) を持つ X.509 v3 証明書によってバックアップされます。つまり、CA は、CA として下位にある他の証明書に署名できます。CA が下位レベルの CA の証明書に署名すると、署名付き証明書に対する制限付きの取り消し可能な権限が付与されます。レベル 1 のルート CA は、レベル 2 の高レベル下位 CA 証明書に署名します。次に、これらの CA は、エンドエンティティ証明書を管理する PKI (パブリックキーインフラストラクチャ) 管理者が使用する、レベル 3 の CA の証明書に署名します。

CA 階層内のセキュリティは、ツリーの最上部で最強になるように構成する必要があります。この構成により、ルート CA 証明書とそのプライベートキーが保護されます。ルート CA は、すべての下位 CA とその下のエンドエンティティ証明書に対する信頼をアンカーします。ローカライズされた損傷は、エンドエンティティ証明書の侵害によって生じることがありますが、ルートの侵害は PKI 全体の信頼を破壊します。ルートおよび上位レベルの下位 CA は、まれにしか使用されません (通常は他の CA 証明書に署名するため)。その結果、リスクの低減を保証するために、厳密に管理および監査されます。階層の下位レベルでは、セキュリティは制限が少なくなります。このアプローチにより、ユーザー、コンピュータホスト、ソフトウェアサービスのエンドエンティティ証明書の発行と失効という日常的な管理タスクが可能になります。

> **Note**
>
> ルート CA を使用して下位証明書に署名することは、ごく少数の状況で発生するまれなイベントです。
>
> - PKI が作成されたとき
> - 高レベルの認証機関を置き換える必要がある場合
> - 証明書失効リスト (CRL) またはオンライン証明書ステータスプロトコル (OCSP) 応答側を構成する必要がある場合
>
> ルートおよびその他の高レベル CA には、安全性の高い運用プロセスとアクセス制御プロトコルが必要です。

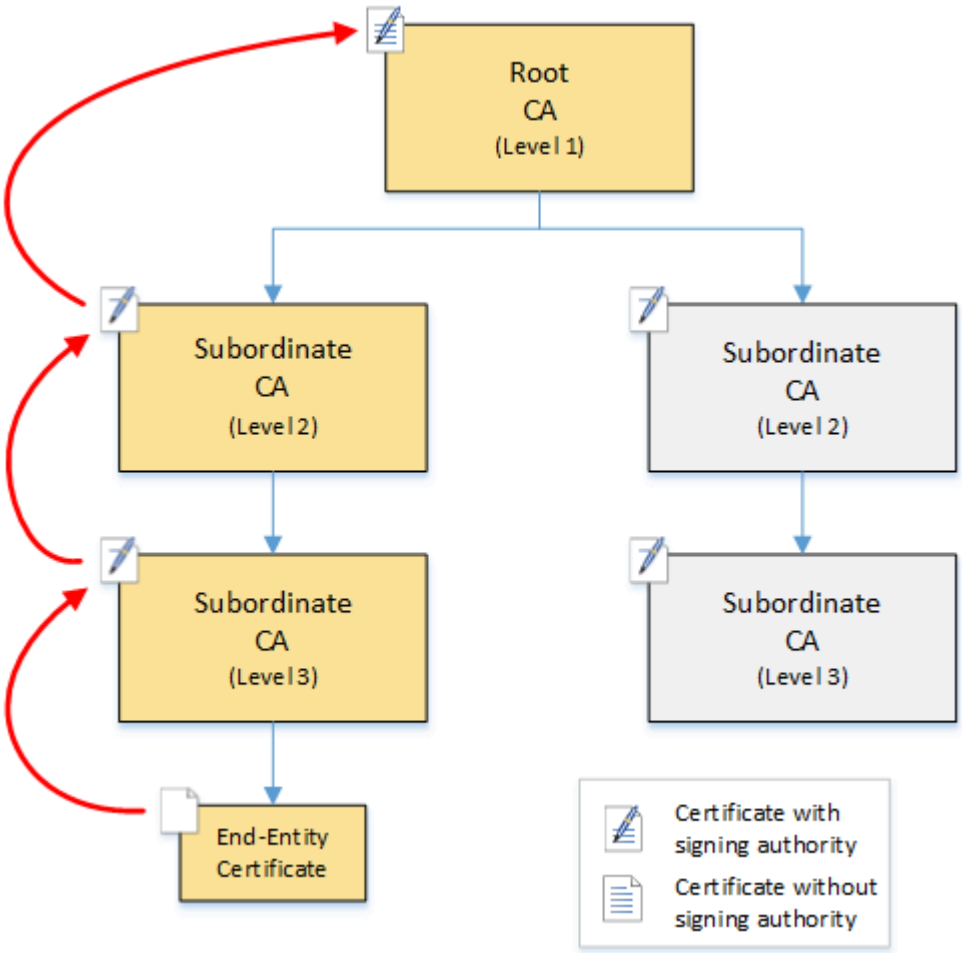
トピック

エンドエンティティ証明書の検証

エンドエンティティ証明書は、下位の CA を経由してルート CA に戻る証明書パスから信頼を取得します。ウェブブラウザまたは他のクライアントにエンドエンティティ証明書が提示されると、信頼チェーンを構築しようとします。たとえば、証明書の発行者識別名とサブジェクト識別名が、発行元の CA 証明書の対応するフィールドと一致しているかどうかを確認する場合があります。一致は、クライアントが信頼ストアに含まれている信頼されたルートに到達するまで、階層の連続した各レベルで継続されます。

信頼ストアは、ブラウザまたはオペレーティングシステムに含まれる信頼された CA のライブラリです。プライベート PKI の場合、組織の IT 担当者は、各ブラウザまたはシステムが以前にプライベートルート CA を信頼ストアに追加していることを確認する必要があります。それ以外の場合、証明書パスを検証できず、クライアントエラーが発生します。

次の図は、エンドエンティティ X.509 証明書が提示されたときにブラウザが従う検証パスを示しています。エンドエンティティ証明書には署名権限がなく、それを所有するエンティティの認証にのみ機能することに注意してください。



ブラウザは、エンドエンティティ証明書を検査します。ブラウザは、証明書が信頼認証情報として下位 CA (レベル 3) からの署名を提供していることを検出します。下位 CA の証明書は、同じ PEM ファイルに含める必要があります。または、信頼チェーンを構成する証明書を含む別のファイルに保存することもできます。これらを見つけると、ブラウザは下位 CA (レベル 3) の証明書を確認し、下位 CA (レベル 2) からの署名を提供していることが検出されます。次に、下位 CA (レベル 2) は、ルート CA (レベル 1) からの署名を信頼認証情報として提供しています。ブラウザが信頼ストアに事前インストールされているプライベートルート CA 証明書のコピーを検出すると、エンドエンティティ証明書が信頼済みとして検証されます。

通常、ブラウザは、証明書失効リスト (CRL) に対して各証明書を確認します。期限切れ、失効、誤って設定された証明書は拒否され、検証は失敗します。

CA 階層構造の計画

一般に、CA 階層は組織の構造を反映する必要があります。管理およびセキュリティの役割を委任するのに必要最低限のパスの長さ (つまり、CA レベルの数) を設定します。CA を階層に追加すると、

証明書パス内の証明書の数が増え、検証時間が長くなります。パスの長さを最小限に抑えることで、エンドエンティティ証明書の検証時にサーバーからクライアントに送信される証明書の数も減ります。

理論上、 pathLenConstraintパラメータを持たないルート CA は、無制限レベルの下位 CAsを許可できます。下位 CA は、内部設定で許可されている数だけ子下位 CAs を持つことができます。 AWS Private CA マネージド階層は、最大 5 レベルの深さの CA 認定パスをサポートします。

適切に設計された CA 構造には、いくつかの利点があります。

- 組織単位ごとに個別の管理コントロール
- 下位 CA へのアクセスを委任する機能
- 追加のセキュリティ制御によって上位レベルの CA を保護する階層構造

これを実現するのは、次の 2 つの共通の CA 構造です。

- 2 つの CA レベル: ルート CA と下位 CA

  これは、ルート CA と下位 CA に対して個別の管理、制御、セキュリティポリシーを許可する、最も単純な CA 構造です。ルート CA に対する制限の厳しい制御とポリシーを維持しながら、下位 CA に対するより許可されたアクセスを許可できます。後者は、エンドエンティティ証明書の一括発行に使用されます。

- 3 つの CA レベル: ルート CA と 2 つの下位 CA のレイヤー

  上記と同様に、この構造はさらに CA レイヤーを追加し、ルート CA を低レベル CA のオペレーションから分離します。中間の CA レイヤーは、エンドエンティティ証明書の発行を実行する下位 CA に署名するためにのみ使用されます。

あまり一般的ではない CA 構造には、次のようなものがあります。

- 4 以上の CA レベル

  3 レベルの階層よりも一般的ではありませんが、4 以上のレベルを持つ CA 階層は可能であり、管理委任を許可するために必要になる場合があります。

- 1 つの CA レベル: ルート CA のみ

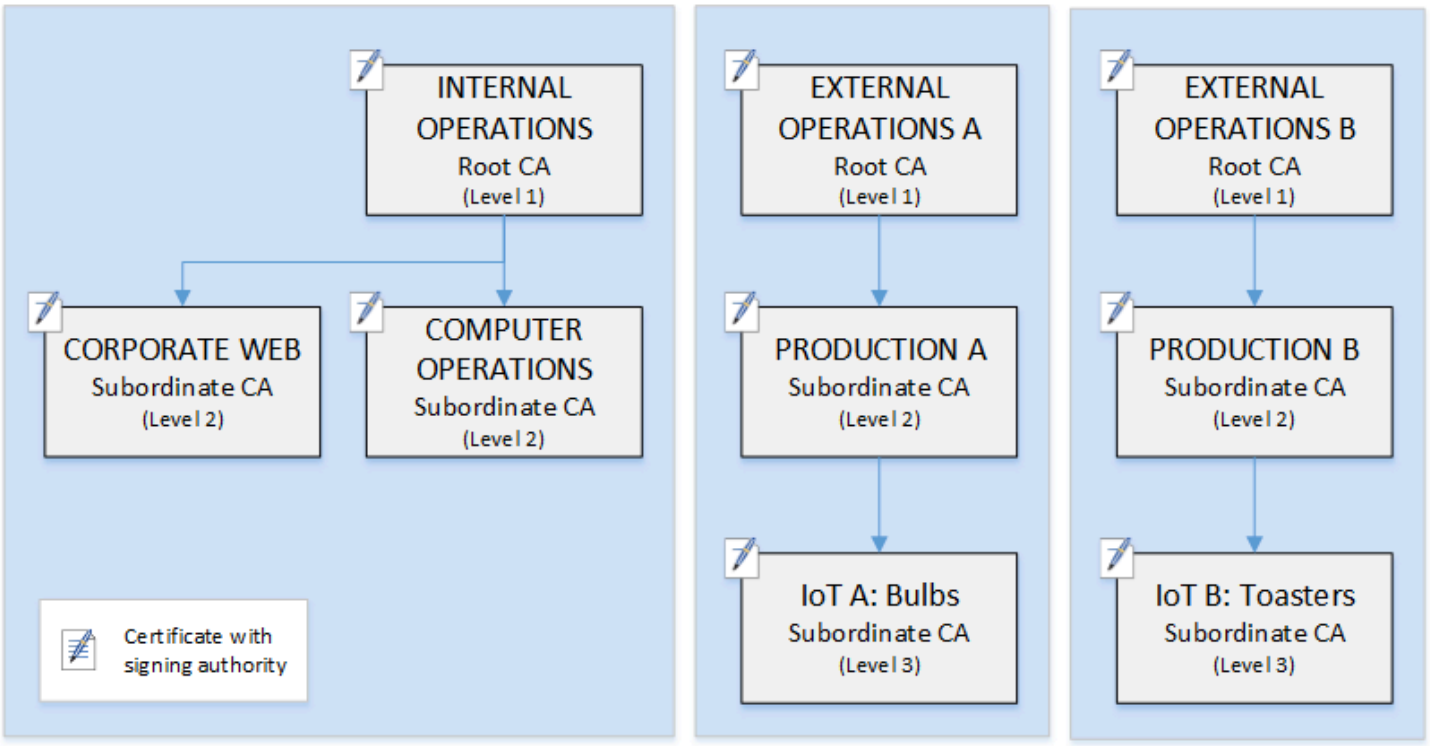
この構造は、完全な信頼チェーンが必要ない場合に開発およびテストによく使用されます。生産に使用され、非定型です。さらに、ルート CA とエンドエンティティ証明書を発行する CA に対して個別のセキュリティポリシーを維持するというベストプラクティスに違反します。

ただし、ルート CA から直接証明書を発行している場合は、 に移行できます AWS Private CA。そうすることで、OpenSSL やその他のソフトウェアで管理されるルート CA を使用するよりもセキュリティと制御の利点が得られます。

製造元のプライベート PKI の例

この例では、ある架空のテクノロジー企業が、スマート電球とスマートトースターの 2 つのモノのインターネット (IoT) 製品を製造しています。本番稼働時には、各デバイスにはエンドエンティティ証明書が発行されるため、インターネット経由で製造元と安全に通信できます。また、会社の PKI は、内部ウェブサイトや財務および事業運営を行うさまざまなセルフホスト型コンピュータサービスなど、自社のコンピュータインフラストラクチャを保護しています。

その結果、CA 階層は、ビジネスのこれらの管理面と運用面を密接にモデル化します。



この階層には、内部オペレーション用に 1 つ、および外部オペレーション用に 2 つ (製品ラインごとに 1 つのルート CA) の 3 つのルートが含まれます。また、内部オペレーション用の 2 つのレベルの

CA と、外部オペレーション用の 3 つのレベルを持つ、複数の証明書のパスの長さについても説明します。

分離されたルート CA の使用と下位 CA レイヤーの外部運用側は、ビジネスとセキュリティのニーズを満たす設計上の決定です。複数の CA ツリーがある場合、PKI は企業の再編、売却、買収に対して将来的に証明されます。変更が発生すると、ルート CA 階層全体が、セキュリティで保護されている分割とともに正常に移動できます。また、下位の CA が 2 レベルの場合、ルート CA は、数千または数百万の製造品目の証明書の一括署名を担当するレベル 3 CA から高い分離レベルを持ちます。

内部では、企業のウェブと内部コンピュータの操作は、2 レベルの階層を完了します。これらのレベルにより、ウェブ管理者と運用エンジニアは、各自のワークドメインで証明書発行を個別に管理できます。PKI を個別の機能ドメインに区分化することは、セキュリティのベストプラクティスであり、他方に影響を及ぼす可能性のある侵害からそれぞれを保護します。ウェブ管理者は、社内のウェブブラウザで使用するためのエンドエンティティ証明書を発行し、社内のウェブサイトで通信を認証および暗号化します。運用エンジニアは、データセンターホストとコンピュータサービスを相互に認証するエンドエンティティ証明書を発行します。このシステムは、LAN 上で暗号化することにより、機密データを安全に保ちます。

証明書パスでの長さの制約の設定

CA 階層の構造は、各証明書に含まれる基本制約拡張によって定義され、適用されます。拡張では、次の 2 つの制約が定義されています。

- cA — 証明書が CA を定義しているかどうか。この値が false (デフォルト) の場合、証明書はエンドエンティティ証明書です。
- `pathLenConstraint` - 有効な信頼チェーン内に存在できる下位レベルの下位 CA の最大数。エンドエンティティ証明書は CA 証明書ではないためカウントされません。

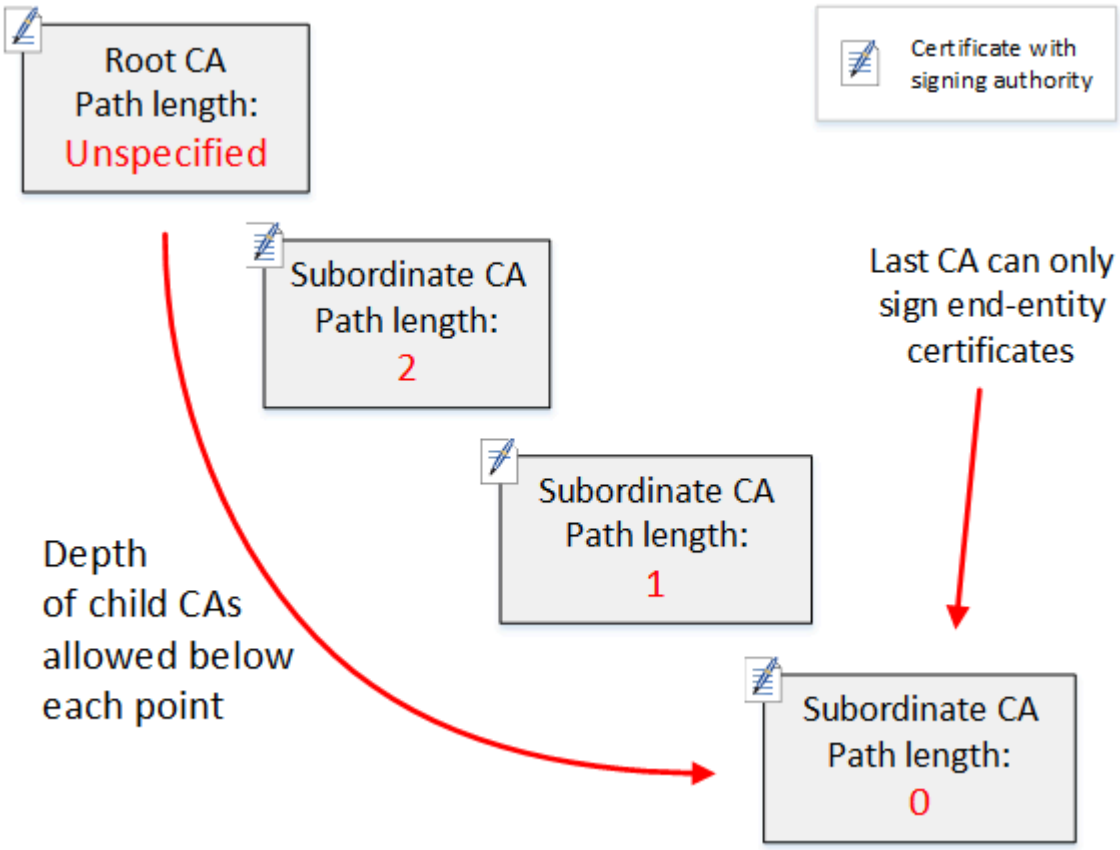
ルート CA 証明書には最大限の柔軟性が必要であり、パス長の制約は含まれません。これにより、ルートは任意の長さの証明書パスを定義できます。

> **Note**
>
> AWS Private CA は、証明書パスを 5 つのレベルに制限します。

下位 CA には、階層の配置と目的の機能の位置に応じて、ゼロ以上の `pathLenConstraint` 値があります。たとえば、3 つの CA を持つ階層では、ルート CA にパスの制約は指定されません。最初の

下位 CA のパス長は 1 であるため、子 CA に署名できます。これらのそれぞれの子 CA は、必ずゼロの pathLenConstraint 値を持つ必要があります。つまり、エンドエンティティ証明書に署名することはできますが、追加の CA 証明書を発行することはできません。新しい CA を作成する能力を制限することは、重要なセキュリティ制御です。

次の図は、階層の下に、制限された権限のこの伝播を示しています。



この 4 レベルの階層では、ルートは拘束されません (いつものように)。しかし、最初の下位 CA が 2 の pathLenConstraint 値で、子 CA が 3 レベル以上深くなることを制限します。したがって、有効な証明書パスの場合、制約値は次の 2 つのレベルでゼロに減少する必要があります。ウェブブラウザで、パス長が 4 を超えるこのブランチからのエンドエンティティ証明書が検出された場合、検証は失敗します。このような証明書は、誤って作成された CA、誤って設定された CA、または不正な発行が原因である可能性があります。

テンプレートによるパスの長さの管理

AWS Private CA には、ルート、下位、エンドエンティティの証明書を発行するためのテンプレートが用意されています。これらのテンプレートは、パス長を含む基本的な制約値のベストプラクティスをカプセル化します。テンプレートには、次のものがあります。

- RootCACertificate/V1
- SubordinateCACertificate _PathLen0/V1
- SubordinateCACertificate _PathLen1/V1
- SubordinateCACertificate _PathLen2/V1
- SubordinateCACertificate _PathLen3/V1
- EndEntityCertificate/V1

発行元の CA 証明書のパス長以上のパス長を持つ CA を作成しようとすると、`IssueCertificate` API はエラーを返します。

証明書テンプレートの詳細については、「 証明書テンプレートについて」を参照してください。

AWS CloudFormationを使用した CA 階層設定の自動化

CA 階層の設計を決定したら、 AWS CloudFormation テンプレートを使用してテストし、本番環境に配置できます。このようなテンプレートの例については、「AWS CloudFormation ユーザーガイド」の「プライベート CA 階層の宣言」を参照してください。

プライベート CA ライフサイクルの管理

CA 証明書には、一定の有効期限または有効期間があります。CA 証明書の有効期限が切れると、CA 階層の下位の CA によって直接または間接的に発行されたすべての証明書が無効になります。事前に計画することで、CA 証明書の有効期限を回避できます。

有効期間の選択

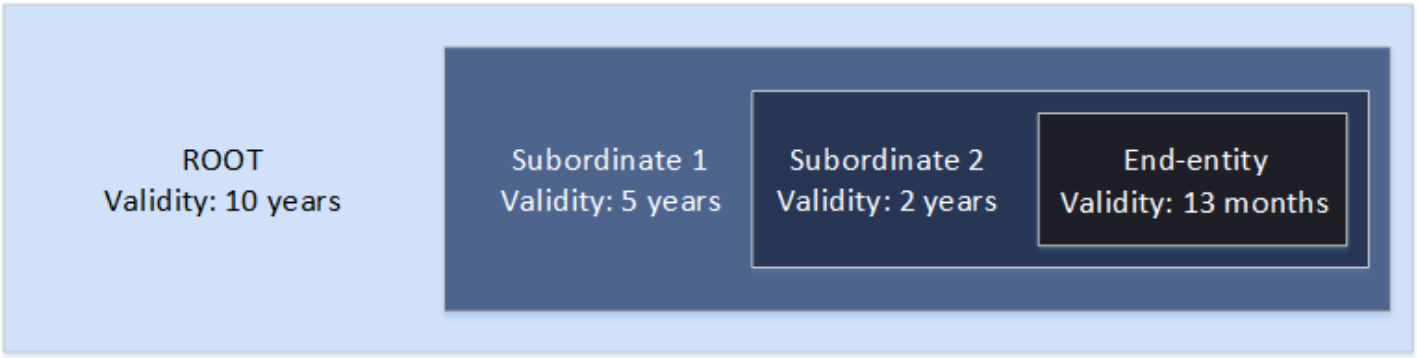
X.509 証明書の有効期間は、必須の基本証明書フィールドです。これにより、発行元の CA が証明書を信頼できることを証明する時間範囲が決定され、取り消しは発生しません。(自己署名されたルート証明書は、独自の有効期間を証明します。)

AWS Private CA と は、以下の制約事項に従って証明書の有効期間の設定 AWS Certificate Manager を支援します。

- によって管理される証明書には、証明書を発行した CA の有効期間以下の有効期間 AWS Private CA が必要です。つまり、子 CA とエンドエンティティ証明書は、親証明書の有効期間を過ぎることはできません。`IssueCertificate` API を使用して、親の CA 以上の有効期間を持つ CA 証明書を発行しようとすると、失敗します。

- によって発行および管理される証明書 AWS Certificate Manager (ACM がプライベートキーを生成する証明書) の有効期間は 13 か月 (395 日) です。ACM はこれらの証明書の更新プロセスを管理します。 AWS Private CA を使用して証明書を直接発行する場合は、任意の有効期間を選択できます。

次の図は、ネストされた有効期間の典型的な構成を示しています。ルート証明書は最も有効期間が長く、エンドエンティティ証明書は比較的短く、下位 CA はこれらの極端な範囲にあります。



CA 階層を計画するときは、CA 証明書の最適な有効期間を決定します。発行するエンドエンティティ証明書の必要な有効期間から逆方向に作業します。

エンドエンティティ証明書

エンドエンティティ証明書には、ユースケースに適した有効期間が必要です。有効期間が短いため、プライベートキーが紛失または盗難された場合に証明書が公開される危険性が最小限に抑えられます。ただし、有効期間が短いということは、頻繁に更新されることを意味します。期限が切れる証明書を更新しないと、ダウンタイムが発生する可能性があります。

エンドエンティティ証明書を分散して使用すると、セキュリティ侵害が発生した場合に物流上の問題が発生する可能性があります。計画では、更新証明書と配布証明書、侵害された証明書の取り消し、および証明書に依存するクライアントに取り消しが伝播する速さについて考慮する必要があります。

ACM から発行されるエンドエンティティ証明書のデフォルトの有効期間は 13 か月 (395 日間) です。では AWS Private CA、 `IssueCertificate` API を使用して、発行元の CA よりも有効期間が短い限り、任意の有効期間を適用できます。

下位 CA 証明書

下位 CA 証明書は、発行する証明書よりもかなり長い有効期間を持つ必要があります。CA 証明書の有効期間は、子 CA 証明書またはエンドエンティティ証明書の発行期間の 2～5 倍です。たとえば、2 レベルの CA 階層 (ルート CA と 1 つの下位 CA) があるとします。エンドエンティティ証明

書を 1 年間の有効期間で発行する場合は、下位の発行 CA の有効期間を 3 年に設定できます。これは、 の下位 CA 証明書のデフォルトの有効期間です AWS Private CA。下位 CA 証明書は、ルート CA 証明書を置き換えることなく変更できます。

ルート証明書

ルート CA 証明書を変更すると、PKI (パブリックキーインフラストラクチャ) 全体に影響するため、依存するすべてのクライアントオペレーティングシステムとブラウザの信頼ストアを更新する必要があります。運用上の影響を最小限に抑えるには、ルート証明書の有効期間を長くする必要があります。ルート証明書の AWS Private CA デフォルトは 10 年です。

CA 承継の管理

CA の承継を管理するには、2 つの方法があります。古い CA を置き換えるか、CA を新しい有効期間で再発行します。

古い CA の置換

古い CA を置き換えるには、新しい CA を作成し、同じ親 CA にチェーンします。その後、新しい CA から証明書を発行します。

新しい CA から発行された証明書には、新しい CA チェーンがあります。新しい CA が確立されたら、古い CA を無効にして、新しい証明書を発行しないようにできます。無効になっている間、古い CA は CA から発行された古い証明書の取り消しをサポートします。そうするよう設定されている場合には、OCSP によって証明書を検証し続けるか。証明書失効リスト (CRL) を検証し続けるか、またはその両方を行います。古い CA から発行された最後の証明書の有効期限が切れたら、古い CA を削除できます。CA から発行されたすべての証明書の監査レポートを生成して、発行されたすべての証明書の有効期限が切れていることを確認できます。古い CA に下位の CA がある場合は、下位の CA が同時に期限切れになるか、親 CA よりも前に期限切れになるため、下位の CA も置き換える必要があります。まず、置換が必要な階層内の最上位の CA を置き換えます。次に、後続の下位レベルごとに新しい置換下位 CA を作成します。

AWS では、必要に応じて CA の名前に CAs 生成識別子を含めることをお勧めします。たとえば、第 1 世代 CA に「Corporate Root CA」という名前を付けます。第 2 世代の CA を作成するときは、「Corporate Root CA G2」という名前を付けます。この単純な命名規則は、両方の CA が期限切れでない場合の混乱を避けるのに役立ちます。

CA のプライベートキーをローテーションするため、CA の継承方式が優先されます。CA キーでは、プライベートキーのローテーションがベストプラクティスです。ローテーションの頻度は、キーの使

用頻度に比例する必要があります。より多くの証明書を発行する CA は、ローテーションの頻度が高くなります。

> **Note**
>
> CA を置き換えると、ACM から発行されたプライベート証明書は更新できません。発行および更新に ACM を使用する場合は、CA 証明書を再発行して、CA の有効期間を延長する必要があります。

古い CA の再発行

CA の有効期限が近づいた場合、有効期限を延長する別の方法として、新しい有効期限を設定して CA 証明書を再発行する方法があります。再発行では、すべての CA メタデータはそのまま残り、既存のシークレットキーとパブリックキーは保持されます。このシナリオでは、CA が発行した既存の証明書チェーンと有効期限が切れていないエンドエンティティ証明書は、有効期限が切れるまで有効です。新しい証明書の発行も中断することなく継続できます。再発行された証明書で CA を更新するには、CA 証明書の作成とインストール で説明されている通常のインストール手順に従ってください。

> **Note**
>
> 新しいキーペアに切り替えることでセキュリティ上の利点が得られるため、証明書を再発行するよりも期限切れの CA を置き換えることをお勧めします。

CA の取り消し

CA の基盤となる証明書を取り消すと、その CA が取り消されます。これにより、CA が発行した証明書もすべて事実上失効します。失効情報は OCSP または CRL によってクライアントに配信されます。すべての発行済みエンドエンティティおよび CA 証明書を失効させる場合にのみ、CA 証明書を取り消してください。

証明書失効方法の設定

でプライベート PKI を計画するときは AWS Private CA、エンドポイントのプライベートキーが公開されるときなど、エンドポイントが発行された証明書を信頼しなくなった場合の処理方法を検討する必要があります。この問題に対する一般的なアプローチは、有効期間の短い証明書を使用するか、証

明書失効を設定することです。有効期間が短い証明書は、数時間または数日という短い期間で期限切れになるため、失効しても意味がありません。エンドポイントに失効を通知するのとほぼ同じ時間で証明書が無効になります。このセクションでは、設定やベストプラクティスなど、 AWS Private CA ユーザー向けの失効オプションについて説明します。

失効方法が必要な場合、オンライン証明書ステータスプロトコル (OCSP)、証明書失効リスト (CRL)、またはその両方を選択できます。

> **Note**
>
> 失効を設定せずに CA を作成した場合は、後からいつでも設定できます。詳細については、「プライベート CA の更新」を参照してください。

- オンライン証明書ステータスプロトコル (OCSP)

  AWS Private CA は、お客様がインフラストラクチャ自体を運用しなくても証明書が取り消されたことをエンドポイントに通知するフルマネージド OCSP ソリューションを提供します。お客様は、 AWS Private CA コンソール、API、CLI、または を使用して、単一オペレーションで新規または既存の CAs で OCSP を有効にできます AWS CloudFormation。CRL はエンドポイントで保存および処理されるため古くなることがありますが、OCSP のストレージと処理の要件はレスポンダーのバックエンドで同期的に処理されます。

  CA で OCSP を有効にすると、 は発行された新しい証明書ごとに、OCSP レスポンダーの URL を Authority Information Access (AIA) 拡張機能に AWS Private CA 含めます。拡張により、ウェブブラウザなどのクライアントは、レスポンダーにクエリを実行して、エンドエンティティおよび下位 CA 証明書が信頼できるかどうかを判断できるようになります。レスポンダーは、本物であることを保証するための暗号的に署名されたステータスメッセージを返します。

  AWS Private CA OCSP レスポンダーは RFC 5019 に準拠しています。

  OCSP に関する考慮事項

  - OCSP ステータスメッセージは、発行元の CA が使用するように設定されているのと同じ署名アルゴリズムを使用して署名されます。 AWS Private CA コンソールで作成された CA は、デフォルトで SHA256WITHRSA 署名アルゴリズムを使用します。サポートされているその他のアルゴリズムは、 CertificateAuthorityConfiguration API ドキュメントに記載されています。
  - OCSP レスポンダーが有効になっていると、APIPassthrough および CSRPassthrough 証明書テンプレートは AIA 拡張では機能しません。

  - マネージド OCSP サービスのエンドポイントには、パブリックインターネットからアクセスできます。OCSP を必要としていても、パブリックエンドポイントが必要ないユーザーは、独自の OCSP インフラストラクチャを運用する必要があります。
- 証明書失効リスト (CRL)

  CRL には、失効した証明書のリストが含まれています。CRLs、 は発行された新しい各証明書に CRL ディストリビューションポイント拡張 AWS Private CA を含めます。この拡張は CRL の URL を提供します。拡張により、ウェブブラウザなどのクライアントは、CRL にクエリを実行して、エンドエンティティおよび下位 CA 証明書が信頼できるかどうかを判断できるようになります。

クライアントは CRL をダウンロードしてローカルで処理する必要があるため、OCSP よりもメモリを大量に消費します。新しい接続を試みるたびに失効状態をチェックする OCSP と比較して、CRL ではリストがダウンロードされてキャッシュされるため、消費するネットワーク帯域幅が少なくなる可能性があります。

> **Note**
>
> OCSP と CRL はどちらも、失効してからステータス変更が可能になるまでに多少の遅延があります。
>
> - 証明書を取り消すと、OCSP レスポンスに新しいステータスが反映されるまでに最大 60 分かかることがあります。一般に、OCSP は失効情報の配信が速い傾向があります。これは、クライアントが数日間キャッシュすることがある CRL とは異なり、OCSP レスポンスは通常クライアントによってキャッシュされないためです。
> - CRL は通常、証明書が取り消された約 30 分後に更新されます。何らかの理由で CRL 更新が失敗した場合、 は 15 分ごとにさらに試行 AWS Private CA を行います。

失効設定の一般要件

すべての失効設定には、次の要件が適用されます。

- CRL または OCSP を無効にする設定には `Enabled=False` パラメータのみを含める必要があり、`CustomCname` や `ExpirationInDays` などの他のパラメータが含まれていると失敗します。
- CRL 構成では、`S3BucketName` パラメータは Amazon Simple Storage Service バケットの命名規則に準拠している必要があります。

- CRL または OCSP 用のカスタムの正規名 (CNAME) パラメータを含む設定は、CNAME での特殊文字の使用に関する RFC7230 の制限に準拠する必要があります。
- CRL や OCSP 構成では、CNAME パラメーターの値には、CNAME パラメーターの値には、「http://」や「https://」などのプロトコルプレフィックスを含めることはできません。

トピック

- 証明書失効リスト (CRL) の計画
- AWS Private CA OCSP 用のカスタム URL の設定

証明書失効リスト (CRL) の計画

CA 作成プロセスの一環として CRL を設定する前に、事前設定が必要な場合があります。このセクションでは、CRL が添付された CA を作成する前に理解しておくべき前提条件とオプションについて説明します。

CRL の代替または補足としてオンライン証明書ステータスプロトコル (OCSP) を使用する方法については、「証明書失効オプション」および「AWS Private CA OCSP 用のカスタム URL の設定」を参照してください。

トピック

- CRL 構造
- Amazon S3 の CRL のアクセスポリシー
- で S3 ブロックパブリックアクセス (BPA) を有効にする CloudFront
- CRL の暗号化
- CRL ディストリビューションポイント (CDP) URI の確認

CRL 構造

各 CRL は、DER でエンコードされたファイルです。ファイルをダウンロードし、OpenSSL を使用して表示するには、次のようなコマンドを使用します。

```
openssl crl -inform DER -in path-to-crl-file -text -noout
```

CRL の形式は次のとおりです。

```
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**
>
> CRL は、それを参照する証明書が発行された後にのみ Amazon S3 に保管されます。それ以前は、Amazon S3 バケットに表示される `acm-pca-permission-test-key` ファイルのみが存在します。

Amazon S3 の CRL のアクセスポリシー

CRL を作成する場合は、Amazon S3 バケットを に保存するように準備する必要があります。指定した Amazon S3 バケットに CRL AWS Private CA を自動的に預金し、定期的に更新します。詳細については、「バケットの作成」を参照してください。

S3 バケットは、アタッチされた IAM アクセス許可ポリシーによって保護されている必要があります。権限のあるユーザーとサービスプリンシパルには、 AWS Private CA にバケットへのオブジェクトの配置を許可する `Put` 権限と、そのオブジェクトを取得する `Get` 権限が必要です。CA を作成するコンソール手順中に、 で新しいバケット AWS Private CA を作成し、デフォルトのアクセス許可ポリシーを適用することを選択できます。

> **Note**
>
> IAM ポリシーの設定は、 AWS リージョン 関係する によって異なります。リージョンは次の 2 つのカテゴリに分類されます。
>
> - デフォルトが有効なリージョン – すべての でデフォルトで有効になっているリージョン AWS アカウント。
> - デフォルト無効リージョン — デフォルトでは無効になっているが、ユーザーが手動で有効にできるリージョン。
>
> デフォルトが無効になっているリージョンの詳細とリストについては、「 の管理 AWS リージョン」を参照してください。IAM のコンテキストにおけるサービスプリンシパルの説明については、「オプトインリージョンのAWS サービスプリンシパル」を参照してください。CRLs を証明書失効メソッドとして設定すると、 は CRL AWS Private CA を作成し、S3 バケットに発行します。S3 バケットには、 AWS Private CA サービスプリンシパルにバケットへの書き込みを許可する IAM ポリシーが必要です。サービスプリンシパルの名前は使用するリージョンによって異なり、すべてのオプションがサポートされているわけではありません。

PCA	S3	サービスプリンシパル
どちらも同じリージョンにあります。		acm-pca.amazonaws.com
有効	有効	acm-pca.amazonaws.com
無効	有効	acm-pca.*Region*.amazonaws.com
有効	無効	サポートされていません

デフォルトポリシーでは CA に SourceArn 制限は適用されません。以下に示す許可の少ないポリシーを手動で適用することをお勧めします。これにより、特定の AWS アカウントと特定のプライベート CA の両方へのアクセスが制限されます。詳細については、「Amazon S3 コンソールを使用したバケットポリシーの追加」を参照してください。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
            "Service":"acm-pca.amazonaws.com"
         },
         "Action":[
            "s3:PutObject",
            "s3:PutObjectAcl",
            "s3:GetBucketAcl",
            "s3:GetBucketLocation"
         ],
         "Resource":[
            "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*",
            "arn:aws:s3:::DOC-EXAMPLE-BUCKET"
         ],
         "Condition":{
            "StringEquals":{
```

```
                "aws:SourceAccount":"account",
                "aws:SourceArn":"arn:partition:acm-pca:region:account:certificate-
authority/CA_ID"
            }
         }
      }
   ]
}
```

デフォルトポリシーを許可することを選択した場合は、後でいつでも変更できます。

で S3 ブロックパブリックアクセス (BPA) を有効にする CloudFront

新しい Amazon S3 バケットは、パブリックアクセスのブロック (BPA) 機能が有効化された状態でデフォルトで設定されます。Amazon S3 のセキュリティベストプラクティスに含まれている BPA は、ユーザーが S3 バケット内のオブジェクトやバケット全体へのアクセスの微調整に使用できるアクセス制御のセットです。BPA がアクティブで正しく設定されている場合、承認および認証された AWS ユーザーのみがバケットとそのコンテンツにアクセスできます。

AWS では、潜在的な攻撃者に機密情報が公開されないように、すべての S3 バケットで BPA を使用することをお勧めします。ただし、PKI クライアントがパブリックインターネット経由で CRLsを取得する場合 (つまり、 AWS アカウントにログインしていない場合) 、追加の計画が必要です。このセクションでは、S3 バケットへの認証されたクライアントアクセスを必要とせずに CRLs を提供するように CloudFront、コンテンツ配信ネットワーク (CDN) である Amazon を使用してプライベート PKI ソリューションを設定する方法について説明します。

> **Note**
>
> CloudFront を使用すると、 AWS アカウントに追加のコストが発生します。詳細については、「Amazon CloudFront の料金」を参照してください。
> BPA が有効になっている S3 バケットに CRL を保存することを選択し、 を使用しない場合は CloudFront、PKI クライアントが CRL にアクセスできるように別の CDN ソリューションを構築する必要があります。

BPA を使用して Amazon S3 を設定する

S3 で、通常どおり CRL 用の新しいバケットを作成し、そのバケットで BPA を有効にします。

CRL へのパブリックアクセスをブロックする Amazon S3 バケットを設定するには

1. 「バケットの作成」の手順を使用して、新しい S3 バケットを作成します。手順の実行時、「すべてのパブリックアクセスをブロック」オプションを選択します。

   詳細については、「Amazon S3 ストレージへのパブリックアクセスのブロック」を参照してください。
2. バケットが作成されたら、リストから名前を選択し、[権限] タブに移動し、[オブジェクト所有権] セクションで [編集] を選択し、[優先バケット所有者] を選択します。
3. [権限] タブでも、Amazon S3 の CRL のアクセスポリシー で説明されているように IAM ポリシーをバケットに追加します。

BPA CloudFront のセットアップ

プライベート S3 バケットにアクセスできるディス CloudFront トリビューションを作成し、認証されていないクライアントに CRLsを提供できます。

CRL の CloudFront ディストリビューションを設定するには

1. CloudFront 「Amazon CloudFront デベロッパーガイド」の「ディストリビューションの作成」の手順を使用して、新しいディストリビューションを作成します。

   手順を完了する際、以下の設定を適用します。

   - [オリジンドメイン名] で S3 バケットを選択します。
   - [バケットアクセスの制限] で、[はい] を選択します。
   - [オリジンアクセスアイデンティティ] で [新しいアイデンティティを作成] を選択します。
   - [バケットへの読み取り権限の付与] で [はい、バケットポリシーを更新します] を選択します。

   > **Note**
   >
   > この手順では、 はバケットポリシー CloudFront を変更して、バケットオブジェクトへのアクセスを許可します。このポリシーを編集して、`crl` フォルダ内のオブジェクトにのみアクセスを許可することも検討してください。
2. ディストリビューションが初期化されたら、 CloudFront コンソールでドメイン名を見つけ、次の手順のために保存します。

> **Note**
>
> S3 バケットが us-east-1 以外のリージョンで新しく作成された場合、 を介して公開されたアプリケーションにアクセスすると、HTTP 307 一時リダイレクトエラーが発生することがあります CloudFront。バケットのアドレスが反映されるまでに数時間かかることがあります。

BPA 用に CA を設定する

新しい CA を設定するときに、ディストリビューションにエイリアスを含めます CloudFront。

の CNAME を使用して CA を設定するには CloudFront

- CA を作成するための手順 (CLI) を使用して CA を作成します。

  手順を実行する場合、失効ファイルには、非パブリック CRL オブジェクトを指定し、のディストリビューションエンドポイントへの URL を提供するために、次の行を含めるrevoke_config.txt必要があります CloudFront。

  ```
  "S3ObjectAcl":"BUCKET_OWNER_FULL_CONTROL",
   "CustomCname":"abcdef012345.cloudfront.net"
  ```

  その後、この CA で証明書を発行すると、次のようなブロックが含まれます。

  ```
  X509v3 CRL Distribution Points:
   Full Name:
   URI:http://abcdef012345.cloudfront.net/crl/01234567-89ab-
  cdef-0123-456789abcdef.crl
  ```

> **Note**
>
> この CA が発行した古い証明書を持っている場合、その証明書は CRL にアクセスできなくなります。

CRL の暗号化

必要に応じて、CRL を含む Amazon S3 バケットで暗号化を設定できます。 は、Amazon S3 のアセットに対して 2 つの暗号化モード AWS Private CA をサポートしています。 CRLs

- Amazon S3 で管理された AES-256 キーを使用したサーバー側の自動暗号化。
- AWS Key Management Service および仕様に AWS KMS key 設定された を使用したカスタマーマネージド暗号化。

> **Note**
>
> AWS Private CA は、S3 によって自動的に生成されたデフォルトの KMS キーの使用をサポートしていません。

次の手順では、各暗号化オプションを設定する方法について説明します。

自動暗号化を設定するには

S3 サーバー側の暗号化を有効にするには、以下のステップを実行します。

1. https://console.aws.amazon.com/s3/でAmazon S3 コンソールを開きます。
2. バケット テーブルで、 AWS Private CA アセットを保持するバケットを選択します。
3. バケットのページで、[プロパティ] タブを選択します。
4. [デフォルト暗号化] カードを選択します。
5. [Enable (有効化)] を選択します。
6. [Amazon S3 キー (SSE-S3)] を選択します。
7. [変更の保存] をクリックします。

カスタム暗号化を設定するには

カスタムキーを使用して暗号化を有効にするには、以下のステップを実行します。

1. Amazon S3 コンソール (https://console.aws.amazon.com/s3/) を開きます。
2. バケット テーブルで、 AWS Private CA アセットを保持するバケットを選択します。
3. バケットのページで、[プロパティ] タブを選択します。

4. [デフォルト暗号化] カードを選択します。
5. [Enable (有効化)] を選択します。
6. AWS Key Management Service キー (SSE-KMS) を選択します。
7. AWS KMS キーから選択するか、ARN AWS KMS key を入力します。
8. [変更の保存] をクリックします。
9. (オプション) KMS キーがない場合は、次の AWS CLI create-key コマンドを使用して作成します。

```
$ aws kms create-key
```

出力には、KMS キーのキー ID と Amazon リソースネーム (ARN) が含まれます。以下は、その出力例です。

```
{
    "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. 次の手順を使用して、KMS キーを使用するアクセス許可を AWS Private CA サービスプリンシパルに付与します。デフォルトでは、すべての KMS キーがプライベートです。リソース所有者だけが KMS キーを使用してデータを暗号化および復号できます。ただし、リソース所有者は、他のユーザーとリソースに KMS キーへのアクセス許可を付与することができます。サービスプリンシパルは、KMS キーが保存されているのと同じリージョンにある必要があります。

    a. まず、次のget-key-policyコマンドpolicy.jsonを使用して、KMS キーのデフォルトポリシーを として保存します。

```
$ aws kms get-key-policy --key-id key-id --policy-name default --output text
 > ./policy.json
```

b. テキストエディタで policy.json ファイルを開きます。次のポリシーステートメントのいずれかを選択し、既存のポリシーに追加します。

Amazon S3 バケットキーが有効になっている場合は、次のステートメントを使用してください。

```
{
   "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"
      }
   }
}
```

Amazon S3 バケットキーが無効になっている場合は、次のステートメントを使用してください。

```
{
   "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. 最後に、次のput-key-policyコマンドを使用して、更新されたポリシーを適用します。

```
$ aws kms put-key-policy --key-id key_id --policy-name default --policy file://
policy.json
```

CRL ディストリビューションポイント (CDP) URI の確認

CA の CDP として S3 バケットを使用する場合、CDP URI は次の形式のいずれかになります。

- `http://DOC-EXAMPLE-BUCKET.s3.region-code.amazonaws.com/crl/CA-ID.crl`
- `http://s3.region-code.amazonaws.com/DOC-EXAMPLE-BUCKET/crl/CA-ID.crl`

CA にカスタム CNAME を設定した場合、CDP URI には CNAME が含まれます。例えば、`http://alternative.example.com/crl/CA-ID.crl`

AWS Private CA OCSP 用のカスタム URL の設定

> **Note**
>
> このトピックは、OCSP レスポンダーエンドポイントのパブリック URL をブランディングやその他の目的でカスタマイズしたいお客様を対象としています。 AWS Private CA マネージド OCSP のデフォルト設定を使用する場合は、このトピックをスキップして、「失効の設定」の設定手順に従ってください。

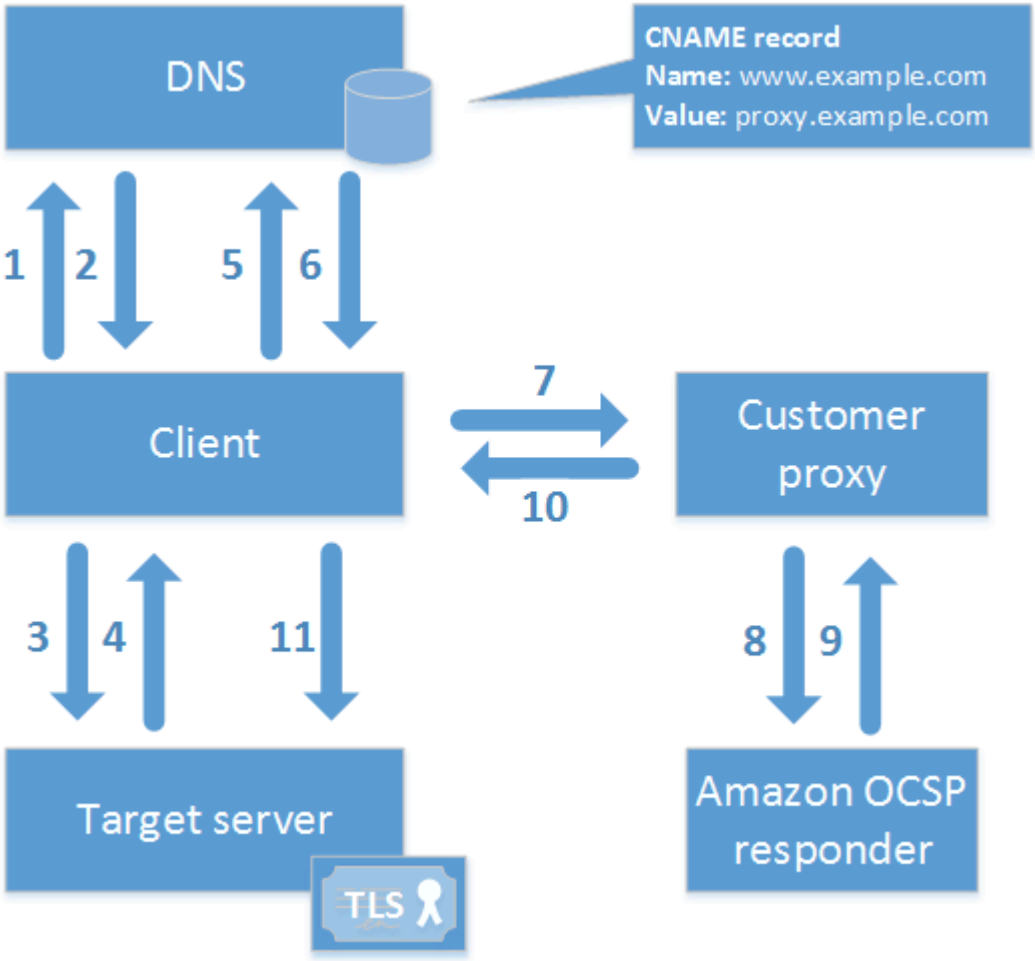
デフォルトでは、 で OCSP を有効にすると AWS Private CA、発行する各証明書には AWS OCSP レスポンダーの URL が含まれます。これにより、暗号的に安全な接続を要求するクライアントは、OCSP 検証クエリを AWSに直接送信できます。ただし、最終的には OCSP クエリを AWSに送信するものの、証明書に別の URL を記載したほうがよい場合もあります。

> **Note**
>
> OCSP の代替または補足として証明書失効リスト (CRL) を使用する方法については、「失効の設定」および「証明書失効リスト (CRL) の計画」を参照してください。

OCSP のカスタム URL の設定には 3 つの要素が含まれます。

- CA 設定 — CA を作成するための手順 (CLI) の 例 2: OCSP とカスタム CNAME が有効な CA を作成する で説明されているように、`RevocationConfiguration` のカスタム OCSP URL をユーザーの CA に指定します。
- DNS — CNAME レコードをドメイン設定に追加して、証明書に記載されている URL をプロキシサーバーの URL にマッピングします。詳細については、「CA を作成するための手順 (CLI) 」の 例 2: OCSP とカスタム CNAME が有効な CA を作成する を参照してください。
- 転送プロキシサーバー — 受信した OCSP トラフィックを AWS OCSP レスポンダーに透過的に転送できるプロキシサーバーを設定します。

次の図は、これらの要素がどのように連携するかを示しています。



図に示すように、カスタマイズされた OCSP 検証プロセスには次の手順が含まれます。

1. クライアントはターゲットドメインの DNS にクエリを実行します。
2. クライアントはターゲット IP を受信します。
3. クライアントはターゲットとの TCP 接続を開きます。
4. クライアントはターゲット TLS 証明書を受け取ります。
5. クライアントは、証明書に記載されている OCSP ドメインの DNS にクエリを実行します。
6. クライアントはプロキシ IP を受信します。
7. クライアントは OCSP クエリをプロキシに送信します。
8. プロキシは OCSP レスポンダにクエリを転送します。
9. レスポンダーは証明書の状態をプロキシに返します。
10. プロキシは証明書の状態をクライアントに転送します。
11. 証明書が有効な場合、クライアントは TLS ハンドシェイクを開始します。

> **Tip**
>
> この例は、上記のように CA を設定した後、Amazon CloudFront と Amazon Route 53 を使用して実装できます。
>
> 1. で CloudFrontディストリビューションを作成し、次のように設定します。
>   - カスタム CNAME と一致する代替名を作成します。
>   - 証明書をそれにバインドします。
>   - ocsp.acm-pca.*<region>*.amazonaws.com をオリジンとします。
>   - `Managed-CachingDisabled` ポリシーを適用します。
>   - [ビューワーのプロトコルポリシー] を [HTTP および HTTPS] に設定します。
>   - [許可される HTTP メソッド] を [GET、HEAD、OPTIONS、PUT、POST、PATCH、DELETE] に設定します。
> 2. Route 53 で、カスタム CNAME を CloudFront ディストリビューションの URL にマッピングする DNS レコードを作成します。

認証局モード

AWS Private CA は、2 つのモードのいずれかで CA の作成をサポートします。GENERAL_PURPOSE モードと SHORT_LIVED_CERTIFICATE モードは、CA が発行する証明書に設定できる有効期間に影響します。

> **Note**
>
> AWS Private CA はルート CA 証明書の有効性チェックを実行しません。

GENERAL_PURPOSE (デフォルト)

このモードでは、CA は任意の有効期間の証明書を発行できます。ほとんどのアプリケーションはこの種類の証明書を使用します。通常、CA は失効メカニズムも規定しています。

SHORT_LIVED_CERTIFICATE

このモードは、有効期間が最大 7 日間の証明書のみを発行する CA を定義します。これらの有効期限の短い証明書はすぐに期限切れになるため、失効メカニズムを導入する必要なくデプロイできま

す。一部のアプリケーションでは、失効によってネットワークや処理のオーバーヘッドを発生させるよりも、有効期間の短い証明書を頻繁にデプロイする方が理にかなっています。

SHORT_LIVED_CERTIFICATE モードの CA は、汎用 CA よりもコストが低くなります。詳細については、「AWS Private Certificate Authority 料金」を参照してください。

有効期間の短い証明書を発行する CA を作成するには、CA を作成するAWS CLI手順を使用して、UsageModeパラメータを SHORT_LIVED_CERTIFICATE に設定します。

> **Note**
>
> AWS Certificate Manager は、有効期間の短いモードでプライベート CA によって署名された証明書を発行できません。

有効期間が短い証明書の使用は、以下の AWS サービスでサポートされています。

- Amazon AppStream
- Amazon WorkSpaces

レジリエンスの計画

AWS グローバルインフラストラクチャは、 AWS リージョンとアベイラビリティーゾーンを中心に構築されています。 AWS リージョンは、低レイテンシー、高スループット、および高度に冗長なネットワークで接続された、物理的に分離および分離された複数のアベイラビリティーゾーンを提供します。アベイラビリティーゾーンでは、ゾーン間で中断することなく自動的にフェイルオーバーするアプリケーションとデータベースを設計および運用することができます。アベイラビリティーゾーンは、従来の単一または複数のデータセンターインフラストラクチャよりも可用性が高く、フォールトトレラントで、スケーラブルです。

AWS リージョンとアベイラビリティーゾーンの詳細については、AWS 「 グローバルインフラストラクチャ」を参照してください。

冗長性とディザスタリカバリ

CA 階層を計画するときは、冗長性と DR を検討してください。 AWS Private CA は複数のリージョン で利用できるため、複数のリージョンで冗長 CAs を作成できます。この AWS Private CA サービスは、99.9% の可用性のサービスレベルアグリーメント (SLA) で動作します。冗長性と災害対策に

は、少なくとも 2 つのアプローチを検討できます。冗長性は、ルート CA または最上位の下位 CA で設定できます。各アプローチには長所と短所があります。

1. 冗長性とディザスタリカバリのために、2 つの異なる AWS リージョンに 2 つのルート CAs を作成できます。この設定では、各ルート CA は AWS リージョンで独立して動作し、単一リージョンの災害が発生した場合にユーザーを保護します。ただし、冗長ルート CA を作成すると、運用が複雑になります。両方のルート CA 証明書を、使用している環境内のブラウザとオペレーティングシステムのトラストストアに配布する必要があります。
2. また、冗長な下位 CA を作成して各 AWS リージョンにデプロイし、それらを 1 つの AWS リージョンの同じ一意のルート CA にチェーンすることもできます。このアプローチの利点は、1 つのルート CA 証明書のみを環境内の信頼ストアに配布する必要があることです。制限は、ルート CA が存在する AWS リージョンに影響する災害が発生した場合に冗長なルート CA がないことです。

AWS Private CA のベストプラクティス

ベストプラクティスは、AWS Private CA をより効果的に使用するのに役立つ推奨事項です。次のベストプラクティスは、現在の AWS Certificate Manager クライアントおよび AWS Private CA 顧客の実際の経験に基づいています。

CA の構造とポリシーのドキュメント化

AWS では、CA の運用に関するすべてのポリシーとプラクティスを文書化することを推奨します。アプローチには以下が含まれます。

- CA 構造に関する意思決定の推論
- CA とその関係を示す図
- CA の有効期間に関するポリシー
- CA 承継の計画
- パスの長さに関するポリシー
- アクセス許可のカタログ
- 管理制御構造の説明
- セキュリティ

この情報は、証明書ポリシー (CP) と証明書プラクティスステートメント (CPS) と呼ばれる 2 つのドキュメントに取り込むことができます。CA オペレーションに関する重要な情報を取得するためのフレームワークについては、「RFC 3647」を参照してください。

可能な場合にはルート CA の使用を最小限に抑える

ルート CA は、通常、中間 CA 認定を交付するためにのみ使用する必要があります。これにより、中間 CA がエンドエンティティ証明書を発行する毎日のタスクを実行しながら、ルート CA を害のない方法で保存することができます。

ただし、組織の現在のプラクティスがルート CA から直接エンドエンティティ証明書を発行する場合、AWS Private CA はこのワークフローをサポートすると同時に、セキュリティと運用制御を向上させることができます。このシナリオでエンドエンティティ証明書を発行するには、ルート CA がエンドエンティティ証明書テンプレートを使用することを許可する IAM アクセス許可ポリシーが必

要です。IAM ポリシーの詳細については、「の Identity and Access Management (IAM) AWS Private Certificate Authority」を参照してください。

> **Note**
>
> この設定では、運用上の問題が発生する可能性がある制限が課されます。たとえば、ルート CA が侵害されたり紛失したりした場合、新しいルート CA を作成し、環境内のすべてのクライアントに配布する必要があります。この回復プロセスが完了するまで、新しい証明書を発行することはできません。また、ルート CA から証明書を直接発行すると、アクセスを制限したり、ルートから発行される証明書の数を制限したりできなくなります。これらの証明書は、ルート CA の管理に関するベストプラクティスと見なされます。

ルート CA を独自のものにする AWS アカウント

2 つの異なる AWS アカウントでルート CA と下位 CA を作成するのが推奨されるベストプラクティスです。これにより、ルート CA に対する追加の保護とアクセス制御が提供されます。そのためには、あるアカウントの下位 CA から CSR をエクスポートして、別のアカウントのルート CA で署名します。このアプローチの利点は、CA の制御をアカウントごとに分けることができることです。欠点は、AWS Management Console ウィザードを使用して、ルート CA から下位 CA の CA 証明書に署名するプロセスを簡素化できないことです。

> **Important**
>
> AWS Private CA にアクセスするときは常に多要素認証 (MFA) を使用することを強くお勧めします。

管理者ロールと発行者ロールを分ける

CA 管理者ロールは、エンドエンティティ証明書の発行のみが必要なユーザーとは別にする必要があります。CA 管理者と証明書発行者が同じ に存在する場合はAWS アカウント、その目的専用の IAM ユーザーを作成して発行者のアクセス許可を制限できます。

証明書のマネージド失効を実装する

マネージド失効は、証明書が取り消されたときに、証明書クライアントに自動的に通知します。暗号情報が侵害されたり、証明書が誤って発行されたりした場合は、証明書の取り消しが必要になること

があります。通常、クライアントは取り消された証明書の受け取りを拒否します。AWS Private CA には、オンライン証明書ステータスプロトコル (OCSP) と証明書失効リスト (CRL) の 2 つのマネージド失効の標準オプションがあります。詳細については、「証明書失効方法の設定」を参照してください。

AWS CloudTrail をオンにする

プライベート CA を作成して運用を開始する前に、 CloudTrail ログ記録を有効にします。を使用すると CloudTrail、アカウントの AWS API コールの履歴を取得して、AWSデプロイをモニタリングできます。この履歴には、AWS Management Console、AWS SDK、AWS Command Line Interface、およびより高レベルの AWS サービスから行われた API 呼び出しが含まれます。また、履歴では、PCA API オペレーションを呼び出したユーザーとアカウント、呼び出し元のソース IP アドレス、および呼び出しの発生日時を特定できます。API を使用してアプリケーション CloudTrail に統合したり、組織の証跡の作成を自動化したり、証跡のステータスを確認したり、管理者が CloudTrail ログ記録のオンとオフを切り替える方法を制御したりできます。詳細については、「証跡の作成」を参照してください。AWS Private CA オペレーションの証跡の例については、「の使用 CloudTrail」を参照してください。

CA プライベートキーを切り替える

ベストプラクティスとして、プライベート CA のプライベートキーを定期的に更新します。キーを更新するには、新しい CA 認定をインポートするか、プライベート CA を新しい CA に置き換えます。

> **Note**
>
> CA 自体を置き換える場合は、CA の ARN が変わることに注意してください。この結果、ハードコーディングされた ARN に依存する自動化が失敗します。

未使用の CA を削除する

プライベート CA を完全に削除できます。この操作は、CA が不要になった場合や、より新しいプライベートキーを持つ CA に置き換える必要がある場合に行います。CA を安全に削除するには、「プライベート CA の削除」で示している手順に従うことをお勧めします。

> **Note**
>
> AWS は、CA が削除されるまで請求を行います。

CRL へのパブリックアクセスをブロックする

AWS Private CA では、CRL を含むバケットで Amazon S3 パブリックアクセスブロック (BPA) 機能を使用することをお勧めします。これにより、プライベート PKI の詳細が潜在的な敵に不必要に公開されるのを防ぐことができます。BPA は S3 のベストプラクティスであり、新しいバケットではデフォルトで有効になっています。追加のセットアップが必要な場合があります。詳細については、「で S3 ブロックパブリックアクセス (BPA) を有効にする CloudFront」を参照してください。

Amazon EKS アプリケーションのベストプラクティス

AWS Private CA を使用して X.509 証明書による Amazon EKS のプロビジョニングを行う場合は、Amazon EKS Best Practices Guides のマルチテナント環境のセキュリティ保護に関する推奨事項に従ってください。AWS Private CA と Kubernetes との統合に関する一般的な情報については、「AWS Private CA での Kubernetes のセキュリティ保護」を参照してください。

プライベート CA 管理

を使用すると AWS Private CA、組織による内部使用のために、ルート認証機関と下位認証機関 (CAsの完全に AWS ホストされた階層を作成できます。証明書の失効を管理するには、オンライン証明書ステータスプロトコル (OCSP)、証明書失効リスト (CRLs)、またはその両方を有効にできます。 は CA 証明書、CRLs、および OCSP レスポンス AWS Private CA を保存および管理し、ルート機関のプライベートキーは によって安全に保存されます AWS。

> **Note**
>
> の OCSP 実装 AWS Private CA は、OCSP リクエスト拡張をサポートしていません。複数の証明書を含む OCSP バッチクエリを送信すると、 AWS OCSP レスポンダーはキュー内の最初の証明書のみを処理し、他の証明書はドロップします。OCSP 応答に失効が表示されるまでに最大 1 時間かかる場合があります。

、 AWS Management Console、および AWS Private CA API AWS Private CA を使用して AWS CLI にアクセスできます。次のトピックでは、コンソールおよび CLI を使用する方法を示します。この API の詳細については、「AWS Private Certificate Authority API リファレンス」を参照してください。API の使用方法を示す Java の例については、「AWS Private CA API の使用 (Java の例)」を参照してください。

トピック

プライベート CA の作成

このセクションの手順を使用して、ルート CA と下位 CA のいずれかを作成し、組織のニーズに合った信頼関係の監査可能な階層を作成できます。CA は、 AWS Management Console、 の PCA 部分 AWS CLI、または を使用して作成できます AWS CloudFormation。

既に作成した CA の設定の更新については、「プライベート CA の更新」を参照してください。

CA を使用してユーザー、デバイス、アプリケーションのエンドエンティティ証明書に署名する方法については、「プライベートエンドエンティティ証明書の発行」を参照してください。

> **Note**
>
> アカウントを作成した時点で、各プライベート CA の月額料金が課金されます。
> 最新の AWS Private CA 料金情報については、「 の料金AWS Private Certificate Authority」を参照してください。AWS 料金計算ツール でコストを見積もることもできます。

トピック

- CA の作成手順 (コンソール)
- CA を作成するための手順 (CLI)
- AWS CloudFormation を使用して CA を作成する

CA の作成手順 (コンソール)

AWS Management Consoleを使用してプライベート CA を作成するには、次のステップを実行します。

コンソールの使用を開始するには

AWS アカウントにサインインし、 で AWS Private CA コンソールを開きます**https://console.aws.amazon.com/acm-pca/home**。

- プライベート CA がないリージョンでコンソールを開くと、導入ページが表示されます。[プライベート CA を作成] を選択します。
- 既に CA を作成しているリージョンでコンソールを開くと、[プライベート認証機関] ページが開き、CA のリストが表示されます。[CA を作成] を選択します。

モードオプション

コンソールの [モードオプション] セクションで、CA が発行する証明書の有効期限モードを選択します。

- [汎用] — 任意の有効期限を設定できる証明書を発行します。これがデフォルトです。
- [有効期間の短い証明書] — 最大有効期間が 7 日間の証明書を発行します。有効期間を短くすることで失効メカニズムの代わりになる場合もあります。

CA タイプオプション

コンソールの [タイプオプション] セクションで、作成するプライベート認証機関のタイプを選択します。

- [ルート] を選択すると、新しい CA 階層が構築されます。この CA は、自己署名証明書によって認証されています。これは、階層内の他の CA およびエンドエンティティ証明書の最終署名機関として機能します。
- [下位] を選択すると、階層のより上位にある親 CA による署名を必要とする CA が作成されます。下位 CA は通常、他の下位 CA を作成したり、ユーザー、コンピュータ、アプリケーションにエンドエンティティ証明書を発行したりするために使用されます。

> **Note**
>
> AWS Private CA は、下位 CA の親 CA も によってホストされている場合に、自動署名プロセスを提供します AWS Private CA。必要なのは使用する親 CA の選択だけです。
> 下位 CA は外部の信頼サービスプロバイダーによる署名が必要な場合があります。その場合は、署名付き CA 証明書をダウンロードして取得するために使用する必要がある証明書署名リクエスト (CSR) が AWS Private CA から提供されます。詳細については、「外部の親 CA が署名した下位 CA 証明書のインストール」を参照してください。

サブジェクトの識別名のオプション

[サブジェクトの識別名のオプション] で、プライベート CA のサブジェクト名を設定します。次のオプションの 1 つ以上に値を入力する必要があります。

- [組織 (O)] — 会社名など
- [組織単位 (OU)] — 会社内の部門など

- [国名 (C)] — 2 文字の国コード
- [州名/都道府県名] — 州または都道府県の正式名称
- [地域名] — 都市の名前
- 共通名 (CN) — CA を識別するための人間が読める文字列。

> **Note**
>
> 発行時に APIPassthrough テンプレートを適用することで、証明書のサブジェクト名をさらにカスタマイズできます。詳細と具体例については、「APIPassthrough テンプレートを使用して、カスタムサブジェクト名で証明書を発行します。」を参照してください。

バッキング証明書は自己署名であるため、プライベート CA に関して入力するサブジェクト情報は、パブリック CA に含まれる情報よりも少ない可能性が高いです。サブジェクト識別名を構成する各値の詳細については、「RFC 5280」を参照してください。

キーアルゴリズムのオプション

[キーアルゴリズムのオプション] で、キーアルゴリズムとキーのビットサイズを選択します。デフォルト値は、2048 ビットのキー長の RSA アルゴリズムです。次のアルゴリズムから選択できます。

- RSA 2048
- RSA 4096
- ECDSA P256
- ECDSA P384

証明書失効オプション

[証明書失効オプション] では、証明書を使用するクライアントと失効ステータスを共有する 2 つの方法から選択できます。

- [CRL のディストリビューションをアクティブ化]
- [OCSP をオンにする]

CA では、これらの失効オプションのいずれかを設定することも、いずれも設定しないことも、両方を設定することもできます。任意ではありますが、ベストプラクティスとしてはマネージド失効が推

奨されます。このステップを完了する前に、各方法の利点、必要になる可能性のある事前設定、その他の失効機能に関する情報について「証明書失効方法の設定」を確認してください。

> **Note**
>
> 失効を設定せずに CA を作成した場合は、後からいつでも設定できます。詳細については、「プライベート CA の更新」を参照してください。

CRL を設定するには

1. [証明書失効オプション] で [CRL ディストリビューションをアクティブ化] を選択します。
2. CRL エントリの Amazon S3 バケットを作成するには、新しい S3 バケットを作成 を選択し、一意のバケット名を入力します。(バケットへのパスを含める必要はありません)。それ以外の場合、[S3 バケット URI] で、リストから既存のバケットを選択します。

   コンソールで新しいバケットを作成すると、 AWS Private CA は 必要なアクセスポリシーをバケットにアタッチし、そのバケットに設定された S3 デフォルトの パブリックアクセスブロック (BPA) を無効にしようとします。代わりに既存のバケットを指定する場合は、アカウントとバケットに対し BPA を無効にする必要があります。そうしないと、CA を作成するオペレーションは失敗します。CA が正常に作成された場合でも、CRL の生成を開始する前に、手動でポリシーをアタッチする必要があります。Amazon S3 の CRL のアクセスポリシー で説明されているポリシーパターンのいずれかを使用してください。詳細については、「Amazon S3 コンソールを使用したバケットポリシーの追加」を参照してください。

   > **Important**
   >
   > 以下の条件がすべて当てはまる場合、 AWS Private CA コンソールを使用して CA を作成しようとすると失敗します。
   >
   > - CRL を設定中です。
   > - S3 バケットを自動的に作成 AWS Private CA するように に依頼します。
   > - S3 で BPA 設定を実施しています。
   >
   > この場合、コンソールはバケットを作成しますが、パブリックにアクセスできるようにしようとして失敗します。このような場合は Amazon S3 の設定を確認し、必要に応

じて BPA を無効にしてから、CA を作成する手順を再度行います。詳細については、「Amazon S3 ストレージへのパブリックアクセスのブロック」を参照してください。

3. 追加の設定オプションを表示するには、[CRL 設定] を展開します。

   - カスタム CRL 名を追加して、Amazon S3 バケットのエイリアスを作成します。この名前は、RFC 5280 で定義されている「CRL ディストリビューションポイント」拡張で CA が発行した証明書に含まれています。
   - CRL が有効になる [有効期限 (日数)] を入力します。デフォルト値は 7 日です。オンライン CRL の場合、有効期間は 2～7 日が一般的です。 AWS Private CA は指定した期間の中間で CRL の再生成を試みます。

4. [バケットのバージョニング] と [バケットアクセスログ記録] のオプション設定を表示するには、[S3 設定] を展開します。

OCSP を設定するには

1. [証明書失効オプション] で [OCSP をオンにする] を選択します。
2. [カスタム OCSP エンドポイント (オプション)] フィールドでは、Amazon OCSP 以外のエンドポイントの完全修飾ドメイン名 (FQDN) を指定できます。

   このフィールドに FQDN を指定すると、 は AWS OCSP レスポンダーのデフォルト URL の代わりに、発行された各証明書の Authority Information Access 拡張機能に FQDN AWS Private CA を挿入します。エンドポイントは、カスタム FQDN を含む証明書を受け取ると、そのアドレスに OCSP レスポンスを問い合わせます。このメカニズムを機能させるには、さらに 2 つのアクションを実行する必要があります。

   - プロキシサーバーを使用して、カスタム FQDN に到着したトラフィックを AWS OCSP レスポンダーに転送します。
   - 対応する CNAME レコードを DNS データベースに追加します。

> **Tip**
>
> カスタム CNAME を使用して完全な OCSP ソリューションを実装する方法の詳細については、「AWS Private CA OCSP 用のカスタム URL の設定」を参照してください。

例えば、Amazon Route 53 に表示されるようにカスタマイズされた OCSP の CNAME レコードを次に示します。

レコード名	タイプ	ルーティングポリシー	差別化要因	値/トラフィックのルーティング先
alternative.example.com	CNAME	低	-	proxy.example.com

> **Note**
>
> CNAME の値には、「http://」や「https://」などのプロトコルプレフィックスを含めることはできません。

タグを追加する

[タグを追加] ページでは、任意で CA にタグを付けることができます。タグとは、 AWS リソースを識別および整理するためのメタデータとして機能するキーと値のペアのことを指します。 AWS Private CA タグパラメータのリストと、作成後に CAs「」を参照してくださいプライベート CA のタグ管理。

> **Note**
>
> 作成プロシージャ中にプライベート CA にタグをアタッチするには、CA 管理者はまずインライン IAM ポリシーを `CreateCertificateAuthority` アクションに関連付けて、タグ付けを明示的に許可する必要があります。詳細については、「T ag-on-create: 作成時に CA にタグをアタッチする」を参照してください。

CA 許可のオプション

CA アクセス許可オプション では、オプションで自動更新アクセス許可を AWS Certificate Manager サービスプリンシパルに委任できます。ACM は、この許可が付与されている場合にのみ、この CA

によって生成されたプライベートエンドエンティティ証明書を自動的に更新できます。API または create-permission CLI コマンドを使用して、 AWS Private CA CreatePermissionいつでも更新許可を割り当てることができます。

デフォルトでは、これらのアクセス許可が有効になっています。

> **Note**
>
> AWS Certificate Manager は、有効期間の短い証明書の自動更新をサポートしていません。

料金

[料金] で、プライベート CA の料金を理解していることを確認します。

> **Note**
>
> 最新の AWS Private CA 料金情報については、「 の料金AWS Private Certificate Authority」を参照してください。AWS 料金計算ツール でコストを見積もることもできます。

CA の作成

入力した情報がすべて正しいことを確認したら、[CA を作成] を選択します。CA の詳細ページが開き、ステータスが [証明書を保留中] と表示されます。

> **Note**
>
> 詳細ページでは、[アクション]、[CA 証明書をインストール] を選択して CA の設定を完了することができます。また、後で [プライベート認証機関] リストに戻って、該当するインストール手順を完了することもできます。
>
> - ルート CA 証明書のインストール
> - によってホストされる下位 CA 証明書のインストール AWS Private CA
> - 外部の親 CA が署名した下位 CA 証明書のインストール

CA を作成するための手順 (CLI)

create-certificate-authority コマンドを使用してプライベート CA を作成します。CA 設定 (アルゴリズムやサブジェクト名の情報など)、失効設定 (OCSP や CRL を使用する予定の場合)、および CA タイプ (ルートまたは下位) を指定する必要があります。設定と失効設定の詳細は、コマンドの引数として指定する 2 つのファイルに含まれています。任意で、CA 使用モード (標準証明書または有効期間の短い証明書の発行用) の設定、タグのアタッチ、および冪等性トークンの提供もできます。

CRL を設定する場合、create-certificate-authority コマンドを発行する前に、セキュリティで保護された Amazon S3 バケットを用意しておく必要があります。詳細については、「Amazon S3 の CRL のアクセスポリシー」を参照してください。

CA 設定ファイルは次の情報を指定します。

- アルゴリズムの名前
- CA プライベートキーの作成に使用されるキーサイズ
- CA が署名に使用する署名アルゴリズムのタイプ
- X.500 件名情報

OCSP の失効設定では、以下の情報を含む `OcspConfiguration` オブジェクトを定義します。

- 「true」に設定された `Enabled` フラグ。
- (任意) `OcspCustomCname` の値として宣言されたカスタム CNAME。

CRL の失効設定は、以下の情報を含む `CrlConfiguration` オブジェクトを定義します。

- 「true」に設定された `Enabled` フラグ。
- CRL の有効期限までの日数 (CRL の有効期間)。
- CRL を含む Amazon S3 バケット。
- (オプション) CRL がパブリックにアクセス可能かどうかを決定する S3ObjectAcl 値。ここに示す例では、パブリックアクセスはブロックされています。詳細については、「で S3 ブロックパブリックアクセス (BPA) を有効にする CloudFront」を参照してください。
- (任意) CA によって発行された証明書に含まれている S3 バケットの CNAME エイリアス。CRL がパブリックにアクセスできない場合、これは Amazon などのディストリビューションメカニズムを指します CloudFront。

- (オプション) 次の情報を含むCrlDistributionPointExtensionConfigurationオブジェクト。
  - OmitExtension フラグは「true」または「false」に設定されています。これは、CDP 拡張機能のデフォルト値を CA によって発行された証明書に書き込むかどうかを制御します。CDP 拡張機能の詳細については、「」を参照してくださいCRL ディストリビューションポイント (CDP) URI の確認 。が「true」の場合 OmitExtension 、 を設定 CustomCname することはできません。

> **Note**
>
> OcspConfiguration オブジェクトと CrlConfiguration オブジェクトの両方を定義することで、同じ CA で両方の失効メカニズムを有効にできます。--revocation-configuration パラメータを指定しない場合、どちらのメカニズムもデフォルトで無効になります。後で失効検証のサポートが必要になった場合は、「CA の更新 (CLI)」を参照してください。

以下の例では、.aws 設定ディレクトリに有効なデフォルトリージョン、エンドポイント、認証情報が設定されていることを前提としています。 AWS CLI 環境の設定については、「設定と認証情報ファイルの設定」を参照してください。読みやすくするために、例のコマンドでは CA の設定と失効の入力を JSON ファイルで行っています。例のファイルは必要に応じて実際の使用に合わせて変更してください。

特に明記されていない限り、すべての例では ca_config.txt 設定ファイルを使用しています。

ファイル: ca_config.txt

```
{
   "KeyAlgorithm":"RSA_2048",
   "SigningAlgorithm":"SHA256WITHRSA",
   "Subject":{
      "Country":"US",
      "Organization":"Example Corp",
      "OrganizationalUnit":"Sales",
      "State":"WA",
      "Locality":"Seattle",
      "CommonName":"www.example.com"
   }
}
```

例 1: OCSP が有効な CA を作成する

この例では、失効ファイルによりデフォルトの OCSP サポートが有効になり、 AWS Private CA レスポンダーを使用して証明書のステータスを確認します。

ファイル: OCSP 用の revoke_config.txt

```
{
   "OcspConfiguration":{
      "Enabled":true
   }
}
```

コマンド

```
$ 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
```

正常に実行されると、このコマンドは新しい CA の Amazon リソースネーム (ARN) を出力します。

```
{
 "CertificateAuthorityArn":"arn:aws:acm-pca:region:account:
       certificate-authority/CA_ID"
}
```

コマンド

```
$ 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
```

作成されると、このコマンドは CA の Amazon リソースネーム (ARN)　を出力します。

```
{
```

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

次のコマンドを使用して CA の設定を検査します。

```
$ 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
```

この記述には、次のセクションが含まれているはずです。

```
"RevocationConfiguration": {
   ...
   "OcspConfiguration": {
      "Enabled": true
   }
   ...
}
```

例 2: OCSP とカスタム CNAME が有効な CA を作成する

この例では、カスタマイズされた OCSP サポートが失効ファイルにより有効になっています。`OcspCustomCname` パラメータは、値として完全修飾ドメイン名 (FQDN) を使用します。

このフィールドに FQDN を指定すると、 は AWS OCSP レスポンダーのデフォルト URL の代わりに、発行された各証明書の Authority Information Access 拡張機能に FQDN AWS Private CA を挿入します。エンドポイントは、カスタム FQDN を含む証明書を受け取ると、そのアドレスに OCSP レスポンスを問い合わせます。このメカニズムを機能させるには、さらに 2 つのアクションを実行する必要があります。

- プロキシサーバーを使用して、カスタム FQDN に到着したトラフィックを AWS OCSP レスポンダーに転送します。
- 対応する CNAME レコードを DNS データベースに追加します。

> **Tip**
>
> カスタム CNAME を使用して完全な OCSP ソリューションを実装する方法の詳細については、「AWS Private CA OCSP 用のカスタム URL の設定」を参照してください。

例えば、Amazon Route 53 に表示されるようにカスタマイズされた OCSP の CNAME レコードを次に示します。

レコード名	タイプ	ルーティングポリシー	差別化要因	値/トラフィックのルーティング先
alternative.example.com	CNAME	低	-	proxy.example.com

> **Note**
>
> CNAME の値には、「http://」や「https://」などのプロトコルプレフィックスを含めることはできません。

ファイル: OCSP 用の revoke_config.txt

```
{
   "OcspConfiguration":{
      "Enabled":true,
      "OcspCustomCname":"alternative.example.com"
   }
}
```

コマンド

```
$ 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
```

作成されると、このコマンドは CA の Amazon リソースネーム (ARN)　を出力します。

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

次のコマンドを使用して CA の設定を検査します。

```
$ 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
```

この記述には、次のセクションが含まれているはずです。

```
"RevocationConfiguration": {
   ...
   "OcspConfiguration": {
      "Enabled": true,
      "OcspCustomCname": "alternative.example.com"
   }
   ...
}
```

例 3: CRL がアタッチされた CA を作成する

この例では、失効設定で CRL パラメータが定義されています。

ファイル: revoke_config.txt

```
{
   "CrlConfiguration":{
      "Enabled":true,
      "ExpirationInDays":7,
      "S3BucketName":"DOC-EXAMPLE-BUCKET"
   }
}
```

コマンド

```
$ 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
```

作成されると、このコマンドは CA の Amazon リソースネーム (ARN)　を出力します。

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

次のコマンドを使用して CA の設定を検査します。

```
$ 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
```

この記述には、次のセクションが含まれているはずです。

```
"RevocationConfiguration": {
   ...
   "CrlConfiguration": {
      "Enabled": true,
      "ExpirationInDays": 7,
      "S3BucketName": "DOC-EXAMPLE-BUCKET"
   },
   ...
}
```

例 4: CRL がアタッチされカスタム CNAME が有効な CA を作成する

この例では、失効設定がカスタム CNAME を含む CRL パラメータを定義しています。

ファイル: revoke_config.txt

```
{
   "CrlConfiguration":{
      "Enabled":true,
```

```
      "ExpirationInDays":7,
      "CustomCname": "alternative.example.com",
      "S3BucketName":"DOC-EXAMPLE-BUCKET"
   }
}
```

コマンド

```
$ 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
```

作成されると、このコマンドは CA の Amazon リソースネーム (ARN)　を出力します。

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

次のコマンドを使用して CA の設定を検査します。

```
$ 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
```

この記述には、次のセクションが含まれているはずです。

```
"RevocationConfiguration": {
   ...
   "CrlConfiguration": {
      "Enabled": true,
      "ExpirationInDays": 7,
      "CustomCname": "alternative.example.com",
      "S3BucketName": "DOC-EXAMPLE-BUCKET",
   ...
   }
}
```

例 5: CA を作成して使用モードを指定する

この例では、CA の作成時に CA 使用モードを指定します。指定しない場合、使用モードパラメータはデフォルトで GENERAL_PURPOSE になります。この例では、パラメータは SHORT_LIVED_CERTIFICATE に設定され、CA は最大有効期間が 7 日間の証明書を発行することになります。失効を設定すると不便な状況では、侵害された有効期間の短い証明書は、通常の操作の一環ですぐに期限切れになります。そのため、この例の CA には失効メカニズムがありません。

> **Note**
>
> AWS Private CA は、ルート CA 証明書の有効性チェックを実行しません。

```
$ 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
```

次のdescribe-certificate-authorityコマンドに示すように AWS CLI 、 の コマンドを使用して、結果の CA の詳細を表示します。

```
$ 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
         }
      },
...
```

例 6: Active Directory ログインの CA を作成する

Microsoft Active Directory (AD) の Enterprise NTAuth ストアでの使用に適したプライベート CA を作成できます。このストアでは、カードログオン証明書またはドメインコントローラー証明書を発行できます。CA 証明書を AD にインポートする方法については、「サード パーティ証明機関 (CA) 証明書を Enterprise NTAuth ストアにインポートする方法」を参照してください。

Microsoft certutil ツールを使用すると、-dspublish オプションを呼び出して CA 証明書を AD に発行できます。certutil を使用して AD に発行された証明書は、フォレスト全体で信頼されます。グループポリシーを使用すると、1 つのドメインやドメイン内のコンピュータのグループなど、フォレスト全体のサブセットに信頼を制限することもできます。ログオンが機能するためには、発行元の CA も NTAuth ストアで発行されている必要があります。詳細については、「グループ ポリシーを使用してクライアントコンピューターに証明書を配布する」を参照してください。

この例では次の `ca_config_AD.txt` 設定ファイルを使用します。

ファイル: 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"
         }
      ]
   }
}
```

コマンド

```
$ aws acm-pca create-certificate-authority \
      --certificate-authority-configuration file://ca_config_AD.txt \
      --certificate-authority-type "ROOT" \
      --tags Key=application,Value=ActiveDirectory
```

作成されると、このコマンドは CA の Amazon リソースネーム (ARN)　を出力します。

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

次のコマンドを使用して CA の設定を検査します。

```
$ 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
```

この記述には、次のセクションが含まれているはずです。

```
...

"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"
      }
   ]
}
...
```

例 7: CRL がアタッチされ、CDP 拡張機能が発行された証明書から省略された Matter CA を作成する

Matter スマートホーム標準の証明書の発行に適したプライベート CA を作成できます。この例では、 の CA 設定は、ベンダー ID (VID) を FFF1 に設定して Matter Product Attestation Authority (PAA) `ca_config_PAA.txt`を定義します。

ファイル: 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"
      }
    ]
  }
}
```

失効設定は CRLs、発行された証明書からデフォルトの CDP URL を省略するように CA を設定します。

ファイル: revoke_config.txt

```
{
   "CrlConfiguration":{
      "Enabled":true,
      "ExpirationInDays":7,
      "S3BucketName":"DOC-EXAMPLE-BUCKET",
   "CrlDistributionPointExtensionConfiguration":{
  "OmitExtension":true
   }
   }
}
```

コマンド

```
$ 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
```

作成されると、このコマンドは CA の Amazon リソースネーム (ARN)　を出力します。

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

次のコマンドを使用して CA の設定を検査します。

```
$ 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
```

この記述には、次のセクションが含まれているはずです。

```
"RevocationConfiguration": {
```

```
   ...
   "CrlConfiguration": {
      "Enabled": true,
      "ExpirationInDays": 7,
      "S3BucketName": "DOC-EXAMPLE-BUCKET",
   "CrlDistributionPointExtensionConfiguration":{
  "OmitExtension":true
   }
   },
   ...
}
...
```

AWS CloudFormation を使用して CA を作成する

を使用してプライベート CA を作成する方法については AWS CloudFormation、「 ユーザーガイド」のAWS Private CA 「リソースタイプのリファレンスAWS CloudFormation 」を参照してください。

CA 証明書の作成とインストール

プライベート CA 証明書を作成してインストールするには、次の手順を実行します。その後、CA を使用できるようになります。

AWS Private CA は、CA 証明書をインストールするための 3 つのシナリオをサポートしています。

- によってホストされるルート CA の証明書のインストール AWS Private CA
- 親機関が AWS Private CAにホストされている下位 CA 証明書のインストール
- 親機関が外部でホストされている下位 CA 証明書のインストール

次のセクションでは、各シナリオの手順について説明します。コンソールの手順は、コンソールページの [プライベート CA] から開始します。

互換性のある署名アルゴリズム

CA 証明書の署名アルゴリズムのサポートは、親 CA の署名アルゴリズムと AWS リージョンによって異なります。コンソールと AWS CLI オペレーションの両方に次の制約が適用されます。

- RSA 署名アルゴリズムを使用する親 CA は、以下のアルゴリズムで証明書を発行できます。
  - SHA256 RSA

- SHA384 RSA
- SHA512 RSA

- レガシー では AWS リージョン、EDCSA 署名アルゴリズムを持つ親 CA は、次のアルゴリズムで証明書を発行できます。
  - SHA256 ECDSA
  - SHA384 ECDSA
  - SHA512 ECDSA

  レガシー AWS リージョン には以下が含まれます。

リージョン名	地理的場所
eu-north-1	欧州 (ストックホルム)
me-south-1	中東 (バーレーン)
ap-south-1	アジアパシフィック (ムンバイ)
eu-west-3	欧州 (パリ)
us-east-2	米国東部 (オハイオ)
af-south-1	アフリカ (ケープタウン)
eu-west-1	欧州 (アイルランド)
eu-central-1	欧州 (フランクフルト)
sa-east-1	南米 (サンパウロ)
ap-east-1	アジアパシフィック (香港)

リージョン名	地理的場所
us-east-1	米国東部 (バージニア北部)
ap-northeast-2	アジアパシフィック (ソウル)
eu-west-2	欧州 (ロンドン)
ap-northeast-1	アジアパシフィック (東京)
us-gov-east-1	AWS GovCloud （米国東部）
us-gov-west-1	AWS GovCloud （米国西部）
us-west-2	米国西部 (オレゴン)
us-west-1	米国西部 (北カリフォルニア)
ap-southeast-1	アジアパシフィック (シンガポール)
ap-southeast-2	アジアパシフィック (シドニー)

- 非レガシー では AWS リージョン、EDCSA に次のルールが適用されます。
  - EC_prime256v1 署名アルゴリズムを使用する親 CA は、ECDSA P256 を使用して証明書を発行できます。
  - EC_secp384r1 署名アルゴリズムを使用する親 CA は ECDSA P384 を使用して証明書を発行できます。

ルート CA 証明書のインストール

ルート CA 証明書は、 AWS Management Console または からインストールできます AWS CLI。

プライベートルート CA の証明書を作成してインストールするには (コンソール)

1. (オプション) CA の詳細ページをまだ開いていない場合は、https://console.aws.amazon.com/acm-pca/home の AWS Private CA コンソールを開きます。[プライベート認証機関] ページで、ステータスが [証明書を保留中] または [アクティブ] のルート CA を選択します。
2. [アクション]、[CA 証明書をインストール] を選択し、[ルート CA 証明書をインストール] ページを開きます。
3. [ルート CA 証明書のパラメータを指定] で、次の証明書パラメータを指定します。

   - [有効性] — CA 証明書の期限が切れる日時を指定します。ルート CA 証明書の AWS Private CA デフォルトの有効期間は 10 年です。
   - [署名アルゴリズム] — ルート CA が新しい証明書を発行するときに使用する署名アルゴリズムを指定します。使用可能なオプションは、CA を作成する AWS リージョン によって異なります。詳細については、「」の互換性のある署名アルゴリズム「」、サポートされている暗号アルゴリズム「」、SigningAlgorithm「」を参照してくださいCertificateAuthorityConfiguration。
     - SHA256 RSA
     - SHA384 RSA
     - SHA512 RSA

   設定が正しいことを確認し、CA の AWS Private CA CSR を確認してインストールし、ルート CA 証明書テンプレート を使用して証明書を生成し、証明書に自己署名します。 AWS Private CA その後、自己署名ルート CA 証明書をインポートします。
4. CA の詳細ページの上部には、インストールのステータス (成功または失敗) が表示されます。インストールが正常に完了すると、新たに完了したルート CA のステータスが、[一般] ペインに [Active] と表示されます。

プライベートルート CA の証明書を作成してインストールするには (AWS CLI)

1. 証明書署名リクエスト (CSR) を生成します。

```
$ 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
```

生成されたファイル ca.csr は base64 形式でエンコードされた PEM ファイルで、次のような外見をしています。

```
-----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-----
```

OpenSSL を使用して CSR の内容を表示および検証できます。

```
openssl req -text -noout -verify -in ca.csr
```

この出力は次のようになります。

```
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. 前のステップの CSR を `--csr` パラメータの引数として使用して、ルート証明書を発行します。

> **Note**
>
> AWS CLI バージョン 1.6.3 以降を使用している場合は、必要な入力ファイルを指定するfileb://ときに プレフィックスを使用します。これにより、 は Base64-encodedされたデータを正しく AWS Private CA 解析します。

```
$ 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. ルート証明書を取得します。

```
$ 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
```

生成されたファイル cert.pem は base64 形式でエンコードされた PEM ファイルで、次のような外見をしています。

```
-----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-----
```

OpenSSL を使用して、証明書の内容を表示および検証できます。

```
openssl x509 -in cert.pem -text -noout
```

この出力は次のようになります。

```
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. ルート CA 証明書をインポートして CA にインストールします。

> **Note**
>
> AWS CLI バージョン 1.6.3 以降を使用している場合は、必要な入力ファイルを指定するfileb://ときに プレフィックスを使用します。これにより、 は Base64-encodedされたデータを正しく AWS Private CA 解析します。

```
$ aws acm-pca import-certificate-authority-certificate \
     --certificate-authority-arn arn:aws:acm-pca:region:account:certificate-
authority/CA_ID \
     --certificate file://cert.pem
```

CA の新しいステータスを検査します。

```
$ 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
```

ステータスは ACTIVE と表示されています。

```
{
    "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": "DOC-EXAMPLE-BUCKET1"
            },
            "OcspConfiguration": {
                "Enabled": false
            }
        }
    }
}
```

によってホストされる下位 CA 証明書のインストール AWS Private CA

を使用して AWS Management Console 、ホストされた下位 CA の証明書を作成してインストールできます AWS Private CA 。

AWS Private CA ホストされた下位 CA の証明書を作成してインストールするには

1. (オプション) CA の詳細ページをまだ開いていない場合は、https://console.aws.amazon.com/acm-pca/home の AWS Private CA コンソールを開きます。[プライベート認証機関] ページで、ステータスが[証明書を保留中] または [有効] の下位 CA を選択します。
2. [アクション]、[CA 証明書をインストール] を選択し、[下位 CA 証明書をインストール] ページを開きます。
3. 「下位 CA 証明書のインストール」ページの「CA タイプを選択AWS Private CA」で、 によって管理される証明書のインストールを選択します AWS Private CA。
4. [親 CA を選択] で、[親プライベート CA] リストから CA を選択します。リストは、次の基準を満たす CA が表示されるようにフィルター処理されます。

   - CA を使用する権限が必要です。
   - CA が自己署名することはありません。
   - CA は ACTIVE の状態にあります。
   - CA モードは GENERAL_PURPOSE です。
5. [下位 CA 証明書パラメータを指定] で、次の証明書パラメータを指定します。

   - [有効性] — CA 証明書の期限が切れる日時を指定します。
   - [署名アルゴリズム] — ルート CA が新しい証明書を発行するときに使用する署名アルゴリズムを指定します。オプション:
     - SHA256 RSA

- SHA384 RSA
- SHA512 RSA

- [パスの長さ] — 新しい証明書に署名するときに下位 CA が追加可能な信頼レイヤーの数を指定します。パスの長さが 0 (デフォルト) の場合、エンドエンティティ証明書のみが作成でき、CA 証明書は作成できません。パスの長さが 1 以上の場合は、下位 CA が証明書を発行して、さらに下位の CA を追加作成できます。
- [テンプレート ARN] — この CA 証明書の設定テンプレートの ARN を表示します。指定した [パスの長さ] を変更すると、テンプレートも変更されます。CLI issue-certificate コマンドまたは API IssueCertificateアクションを使用して証明書を作成する場合は、ARN を手動で指定する必要があります。使用可能な CA 証明書テンプレートの詳細については、「証明書テンプレートについて」を参照してください。

6. 設定が正しいかどうかを確認し、「CSR の確認とインストール AWS Private CA 」を選択し、下位 CA 証明書テンプレート を使用して証明書を生成し、選択した親 CA で証明書に署名します。 AWS Private CA その後、署名された下位 CA 証明書をインポートします。
7. CA の詳細ページの上部には、インストールのステータス (成功または失敗) が表示されます。インストールが正常に完了すると、新たに完了した下位 CA のステータスが、[一般] ペインに [アクティブ] と表示されます。

外部の親 CA が署名した下位 CA 証明書のインストール

CA の作成手順 (コンソール) または CA を作成するための手順 (CLI) の説明に従って下位プライベート CA を作成したら、外部の署名機関によって署名された CA 証明書をインストールして下位プライベート CA をアクティブ化できます。外部 CA を使用して下位 CA 証明書を署名するには、まず外部の信頼サービスプロバイダーを署名機関として設定するか、サードパーティのプロバイダーの使用を手配する必要があります。

> **Note**
>
> 外部信頼サービスプロバイダーの作成または取得の手順は、このガイドの対象外です。

下位 CA を作成し、外部の署名機関にアクセスできるようになったら、以下のタスクを実行します。

1. から証明書署名リクエスト (CSR) を取得します AWS Private CA。
2. CSR を外部の署名機関に送信し、署名付き CA 証明書とチェーン証明書を取得します。

3. CA 証明書とチェーンを にインポート AWS Private CA して、下位 CA をアクティブ化します。

詳細な手順については、外部署名付きプライベート CA 証明書 を参照してください。

プライベート CA へのアクセスの制御

からプライベート CA に対する必要なアクセス許可を持つユーザーは、その CA を使用して他の証明書に署名 AWS Private CA できます。CA 所有者は、証明書を発行するか、証明書の発行に必要なアクセス許可を、同じ に存在する AWS Identity and Access Management (IAM) ユーザーに委任できます AWS アカウント。別のアカウントに存在するユーザーは AWS 、リソースベースのポリシー を通じて CA 所有者によって承認された場合、証明書を発行することもできます。

認可されたユーザーは、シングルアカウントかクロスアカウントかにかかわらず、証明書の発行時に AWS Private CA または AWS Certificate Manager リソースを使用できます。 AWS Private CA IssueCertificate API または issue-certificate CLI コマンドから発行された証明書は管理されません。このような証明書は、ターゲットデバイスに手動でインストールし、有効期限が切れたら手動で更新する必要があります。ACM コンソール、ACM RequestCertificate API、または request-certificate CLI コマンドから発行された証明書は管理されます。このような証明書は、ACM と統合されたサービスに簡単にインストールできます。CA 管理者が許可し、発行者のアカウントに ACM 用のサービスにリンクされたロールが設定されている場合、マネージド証明書は有効期限が切れると自動的に更新されます。

トピック

- IAM ユーザー用の単一アカウント許可を作成する
- クロスアカウントアクセスのポリシーをアタッチする

IAM ユーザー用の単一アカウント許可を作成する

CA 管理者 (つまり CA の所有者) と証明書発行者が 1 つの AWS アカウントに存在する場合、ベストプラクティスは、アクセス許可が制限された AWS Identity and Access Management (IAM) ユーザーを作成して発行者と管理者ロールを分離することです。での IAM の使用とアクセス許可の例については AWS Private CA、「」を参照してくださいの Identity and Access Management (IAM) AWS Private Certificate Authority。

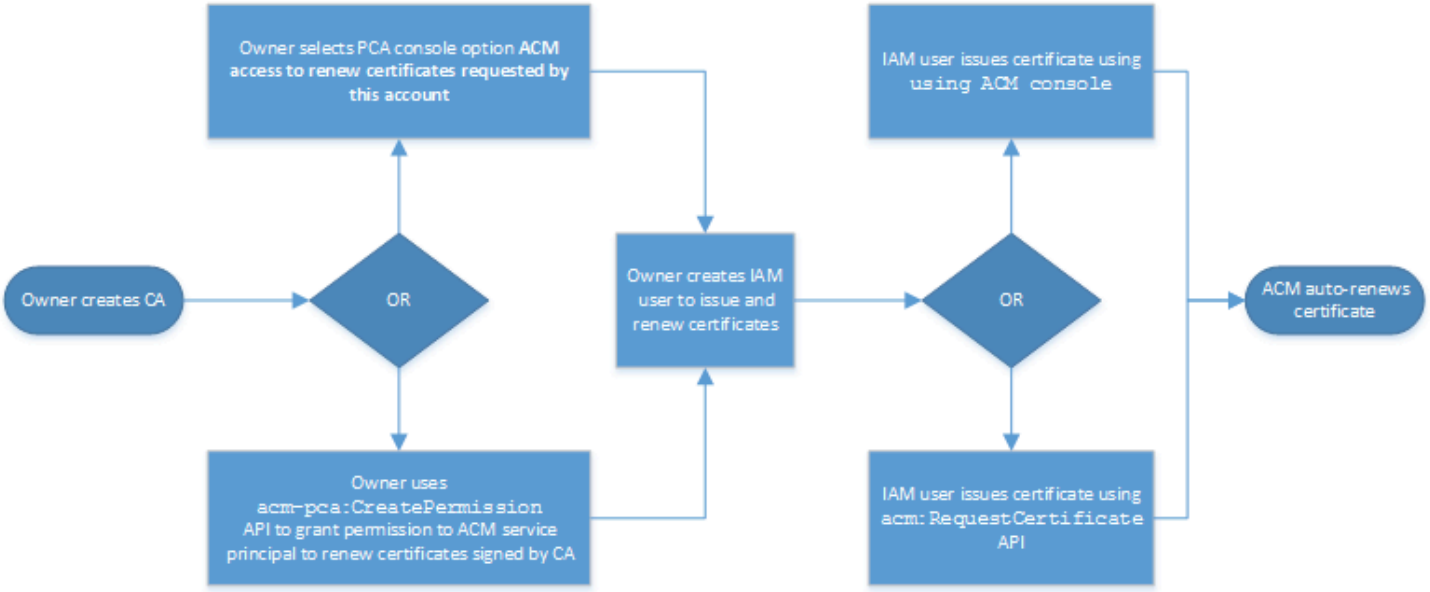
単一アカウントのケース 1: アンマネージド証明書の発行

この場合、アカウント所有者はプライベート CA を作成し、プライベート CA によって署名された証明書を発行する許可を持つ IAM ユーザーを作成します。IAM ユーザーは `IssueCertificate` API を呼び出して証明書を発行します AWS Private CA 。

この方法で発行された証明書は管理されません。つまり、管理者は証明書をエクスポートして、使用する予定のデバイスにインストールする必要があります。また、有効期限が切れたら手動で更新する必要があります。この API を使用して証明書を発行するには、OpenSSL または同様のプログラム AWS Private CA によって の外部で生成される証明書署名リクエスト (CSR) とキーペアが必要です。 OpenSSL 詳細については、「`IssueCertificate` ドキュメント」を参照してください。

単一アカウントのケース 2: ACM によるマネージド証明書の発行

この 2 つ目のケースは、ACM と PCA の両方からの API オペレーションが関わります。アカウント所有者は以前と同様にプライベート CA と IAM ユーザーを作成します。次に、アカウント所有者は ACM サービスプリンシパルに、この CA によって署名されたすべての証明書を自動的に更新する許可を付与します。IAM ユーザーは証明書を再度発行しますが、今回は CSR とキー生成を処理する ACM `RequestCertificate` API を呼び出します。証明書の有効期限が切れると、ACM は更新ワークフローを自動化します。



アカウント所有者は、CA の作成中または作成後に管理コンソールから、または `CreatePermission` PCA API を使用して、更新許可を付与することができます。このワークフローから作成されたマネージド証明書は、ACM と統合された AWS のサービスで使用できます。

以下のセクションには、更新許可を付与する手順があります。

ACM に証明書の更新許可を割り当てる

AWS Certificate Manager (ACM) の マネージド更新 では、パブリック証明書とプライベート証明書の両方の証明書更新プロセスを自動化できます。ACM がプライベート CA によって生成された証明書を自動的に更新するには、ACM サービスプリンシパルに CA 自体から可能なすべての許可を付与する必要があります。これらの更新権限が ACM にない場合は、CA の所有者 (または許可された担当者) は、期限が切れたら各プライベート証明書を手動で再発行する必要があります。

> **Important**
>
> 更新許可を割り当てる手順は、CA 所有者と証明書発行者が同じ AWS アカウントに存在する場合にのみ適用されます。クロスアカウントのシナリオについては、「クロスアカウントアクセスのポリシーをアタッチする」を参照してください。

更新のアクセス許可は、プライベート CA の作成 中に委任することができ、CA が `ACTIVE` 状態である限り、いつでも変更できます。

プライベート CA のアクセス許可は、AWS Private CA コンソール、AWS Command Line Interface (AWS CLI) 、または AWS Private CA API で管理できます。

ACM に プライベート CA の許可を割り当てるには (コンソール)

1. AWS アカウントにサインインし、https://console.aws.amazon.com/acm-pca/home で AWS Private CA コンソールを開きます。
2. [プライベート認証機関] ページで、リストからプライベート CA を選択します。
3. [アクション]、[CA 許可の設定] を選択します。
4. [ACM アクセスを許可してこのアカウントが要求した証明書を更新] を選択します。
5. [保存] を選択します。

AWS Private CA （ AWS CLI) で ACM アクセス許可を管理するには

create-permission コマンドを使用して ACM に許可を割り当てます。ACM が証明書を自動的に更新するようにするには、必要な許可 (IssueCertificate、GetCertificate、ListPermissions) を割り当てる必要があります。

```
$ 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
```

CA によって委任されたアクセス許可 を一覧表示するには、[list-permissions] コマンドを使用します。

```
$ aws acm-pca list-permissions \
     --certificate-authority-arn arn:aws:acm-pca:region:account:certificate-
authority/CA_ID
```

delete-permission コマンドを使用して、CA によって AWS サービスプリンシパルに割り当てられたアクセス許可を取り消します。

```
$ aws acm-pca delete-permission \
     --certificate-authority-arn arn:aws:acm-pca:region:account:certificate-
authority/CA_ID \
     --principal acm.amazonaws.com
```

クロスアカウントアクセスのポリシーをアタッチする

CA 管理者と証明書発行者が異なる AWS アカウントに属している場合、CA 管理者は CA アクセスを共有する必要があります。これは CA にリソースベースのポリシーをアタッチすることでできます。このポリシーは、 AWS アカウント所有者、IAM ユーザー、 AWS Organizations ID、または組織単位 ID である特定のプリンシパルに発行許可を付与します。

CA 管理者は、次の方法でポリシーをアタッチおよび管理できます。

- 管理コンソールでは、 AWS Resource Access Manager (RAM) を使用します。これは、アカウント間で AWS リソースを共有するための標準的な方法です。で CA リソースを別のアカウントの AWS RAM プリンシパルと共有すると、必要なリソースベースのポリシーが CA に自動的にアタッチされます。RAM の詳細については、「AWS RAM ユーザーガイド」を参照してください。

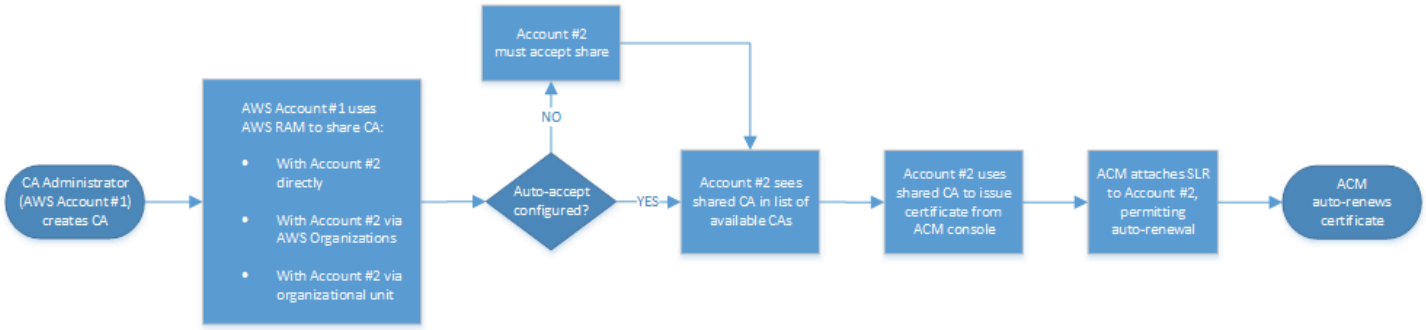
> **Note**
>
> CA を選択し、[アクション]、[リソース共有を管理] の順に選択すると、RAM コンソールを簡単に開くことができます。

- プログラムにより、PCA APIs 、GetPolicy、および PutPolicyを使用しますDeletePolicy。
- AWS CLIで PCA コマンドの put-policy、get-policy、delete-policy を手動で使用する。

RAM アクセスが必要なのはコンソール方式だけです。

クロスアカウントのケース 1: コンソールからマネージド証明書を発行する

この場合、CA 管理者は AWS Resource Access Manager （ AWS RAM) を使用して CA アクセスを別の AWS アカウントと共有します。これにより、そのアカウントはマネージド ACM 証明書を発行できます。この図は、 が CA をアカウントと直接共有することも、アカウントがメンバーである AWS Organizations ID を介して間接的に共有 AWS RAM することもできます。

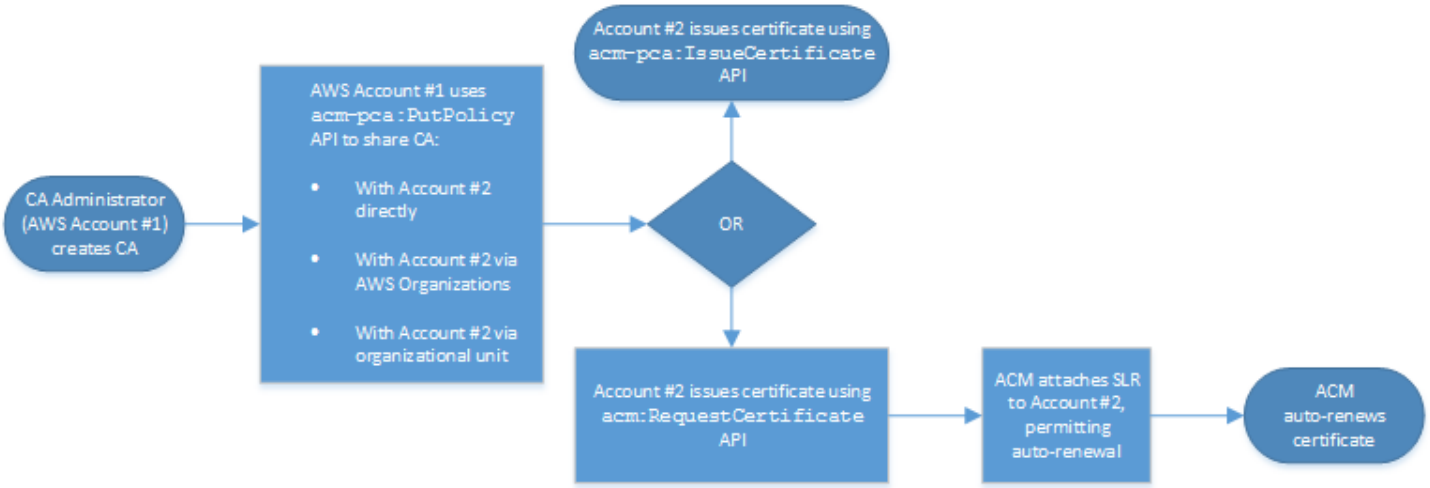


RAM が を介してリソースを共有した後 AWS Organizations、受信者プリンシパルはそのリソースを承諾して有効にする必要があります。受信者は、オファーされた共有を自動的に受け入れ AWS Organizations るように を設定できます。

> **Note**
>
> 受信者アカウントは、ACM での自動更新の設定を行います。通常、共有 CA を初めて使用するときに、ACM は、 AWS Private CAでの自動の証明書呼び出しを許可する、サービスにリンクされたロールをインストールします。これが失敗した場合 (通常は許可がないことが原因)、CA からの証明書は自動的には更新されません。この問題を解決できるのは ACM ユーザーだけで、CA 管理者は解決できません。詳細については、「ACM でのサービスにリンクされたロール (SLR) の使用」を参照してください。

クロスアカウントのケース 2: API または CLI を使用してマネージド証明書およびアンマネージド証明書を発行する

この 2 番目のケースは、 および AWS Private CA API を使用して可能な共有オプション AWS Certificate Manager と発行オプションを示しています。これらのオペレーションはすべて、対応する AWS CLI コマンドを使用して実行することもできます。



この例では API オペレーションを直接使用しているため、証明書発行者は証明書を発行する際に 2 つの API オペレーションから選択できます。PCA API アクション `IssueCertificate` を実行すると、自動的に更新されず、エクスポートして手動でインストールする必要があるアンマネージド証明書が作成されます。ACM API アクションにより、ACM 統合サービスに簡単にインストールでき、自動的に更新できるマネージド証明書RequestCertificateが作成されます。

> **Note**
>
> 受信者アカウントは、ACM での自動更新の設定を行います。通常、共有 CA を初めて使用するときに、ACM は、 AWS Private CAでの自動の証明書呼び出しができるようになる、サービスにリンクされたロールをインストールします。これが失敗した場合 (通常は許可がないことが原因)、CA からの証明書は自動的には更新されず、この問題を解決できるのは ACM ユーザーだけで、CA 管理者は解決できません。詳細については、「ACM でのサービスにリンクされたロール (SLR) の使用」を参照してください。

プライベート CA を一覧表示する

AWS Private CA コンソールまたは を使用して AWS CLI 、所有している、またはアクセスできるプライベート CAsを一覧表示できます。

コンソールを使用して使用可能な CA の一覧を表示するには

1. AWS アカウントにサインインし、https://console.aws.amazon.com/acm-pca/home で AWS Private CA コンソールを開きます。
2. [プライベート認証機関] リストの情報を確認します。右上のページ番号を使用すると、CA の複数のページを移動できます。各 CA は 1 行で、それぞれに次の列の一部またはすべてが表示されます。

- [サブジェクト] — CA の識別名に関する情報。
- [Id] — CA の一意の 32 バイトかつ 16 進数の識別子
- [ステータス] — CA ステータス。取り得る値は、[作成中]、[保留中の証明書]、[アクティブ]、[削除済み]、[無効]、[期限切れ]、[失敗] です。
- [タイプ] — CA のタイプ。取り得る値は、[ルート] と [下位] です。
- [モード] — CA のモード。取り得る値は、[汎用] (任意の有効期限を設定できる証明書を発行) と [有効期間の短い証明書] (最大有効期間が 7 日間の証明書を発行) です。有効期間を短くすることで失効メカニズムの代わりになる場合もあります。デフォルトは [汎用] です。
- 所有者 – CA を所有する AWS アカウント。これは自分のアカウントでも、CA の管理許可を委任されたアカウントでもかまいません。
- [キーアルゴリズム] — CA がサポートするパブリックキーアルゴリズム。取り得る値は [RSA_2048]、[RSA_4096]、[EC_prime256v1]、[EC_secp384r1] です。
- [署名アルゴリズム] — CA が証明書リクエストに署名するのに使用するアルゴリズム。(発行時に証明書に署名するために使用する `SigningAlgorithm` パラメータと混同しないよう注意してください) 取り得る値は、[SHA256WITHECDSA]、[SHA384WITHECDSA]、[SHA512WITHECDSA]、[SHA256WITHRSA]、[SH です。

> **Note**
>
> コンソールの右上隅にある設定アイコンを選択すると、表示する列やその他の設定をカスタマイズできます。

を使用して利用可能な CAsを一覧表示するには AWS CLI

次の例に示すように、 list-certificate-authorities コマンドを使用して使用可能な CAsを一覧表示します。

```
$ aws acm-pca list-certificate-authorities --max-items 10
```

コマンドは以下のような情報を返します。

```
{
   "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
            }
         }
      }
 ...
   ]
}
```

プライベート CA の表示

ACM コンソールまたは を使用して AWS CLI 、プライベート CA に関する詳細なメタデータを表示し、必要に応じていくつかの値を変更できます。CA の更新の詳細については、「プライベート CA の更新」を参照してください。

コンソールで CA の詳細を表示するには

1. AWS アカウントにサインインし、https://console.aws.amazon.com/acm-pca/home で AWS Private CA コンソールを開きます。
2. [プライベート認証機関] リストを確認します。右上のページ番号を使用すると、CA の複数のページを移動できます。
3. 表示された CA についての詳細なメタデータを表示するには、調べたい CA の横のラジオボタンを選択します。すると詳細ペインが開き、以下のタブ付きビューが表示されます。

   - [サブジェクト] タブ — CA の識別名に関する情報。詳細については、「サブジェクトの識別名のオプション」を参照してください。このフィールドに表示されるもの:
     - [サブジェクト] — 入力された名前情報フィールドの概要
     - [組織 (O)] — 会社名など
     - [組織単位 (OU)] — 会社内の部門など
     - [国名 (C)] — 2 文字の国コード
     - [州名/都道府県名] — 州または都道府県の正式名称
     - [地域名] — 都市の名前
     - 共通名 (CN) – CA を識別するための人間が読める文字列。
   - [CA 証明書] タブ — CA 証明書の有効性に関する情報
     - [有効期限] — CA 証明書が有効でなくなる日付と時刻
     - [有効期限] — 有効期限が切れるまでの日数
   - [失効設定] タブ — 現在選択している証明書失効オプション。[編集] を選択して更新します。
     - [証明書失効リスト (CRL) のディストリビューション] — [有効] または [無効] のステータス
     - [オンライン証明書ステータスプロトコル (OCSP)] — [有効] または [無効] のステータス
   - [許可] タブ — 現在選択している、 AWS Certificate Manager (ACM) によるこの CA の証明書更新権限。[編集] を選択して更新します。
   - [更新のための ACM 承認] — 承認済みまたは未承認のステータス
   - [タグ] タブ — この CA に現在割り当てられているカスタマイズ可能なラベル。更新するには [タグの管理] をクリックします。
   - リソース共有タブ – AWS Resource Access Manager (RAM) を通じてこの CA のリソース共有を現在割り当てている。更新するには [リソースシェアを管理] を選択します。
     - [名前] — リソース共有の名前
     - [ステータス] — リソース共有のステータス

4. 検査する CA の [ID] フィールドを選択し、[一般] ペインを開きます。CA の 32 バイトで 16 進数の一意識別子が上部に表示されます。このペインでは他に次の情報が提供されます。

   - [ステータス] — CA ステータス。取り得る値は、[作成中]、[保留中の証明書]、[アクティブ]、[削除済み]、[無効]、[期限切れ]、[失敗] です。
   - [ARN] — CA の Amazon リソースネーム。
   - 所有者 – CA を所有する AWS アカウント。これは自分のアカウント ([セルフ]) でも、CA の管理許可が委任されたアカウントでも構いません。
   - [CA のタイプ] — CA のタイプ。取り得る値は、[ルート] と [下位] です。
   - [作成時刻] - この CA が生成された日時。
   - 有効期限 — CA 証明書の有効期限が切れる日時。
   - [モード] — CA のモード。取り得る値は、[汎用] (任意の有効期限を設定できる証明書) と [有効期間の短い証明書] (最大有効期間が 7 日間の証明書) です。有効期間を短くすることで失効メカニズムの代わりになる場合もあります。デフォルトは [汎用] です。
   - [キーアルゴリズム] — CA がサポートするパブリックキーアルゴリズム。取り得る値は [RSA 2048]、[RSA 4096]、[ECDSA P2567]、[ECDSA P384] です。
   - [署名アルゴリズム] — CA が証明書リクエストに署名するのに使用するアルゴリズム。(発行時に証明書に署名するために使用する `SigningAlgorithm` パラメータと混同しないよう注意してください) 取り得る値は、[SHA256 ECDSA]、[SHA384 ECDSA]、[SHA512 ECDSA]、[SHA256 RSA]、[SHA384 RSA]、[SHA512 RSA] です。
   - [キーストレージセキュリティ標準] — 連邦情報処理基準の適合レベル。取り得る値は [FIPS 140-2 レベル 3 以上] と [FIPS 140-2 レベル 3 以上] です。このパラメータは AWS リージョンによって異なります。

を使用して CA の詳細を表示および変更するには AWS CLI

次のコマンドに示すように、 AWS CLI で describe-certificate-authority コマンドを使用して CA に関する詳細を表示します。

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

コマンドは以下のような情報を返します。

```
{
   "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
         }
      }
   }
}
```

コマンドラインからのプライベート CA の更新については、「CA の更新 (CLI)」を参照してください。

プライベート CA のタグ管理

タグとは、 AWS リソースを識別および整理するためのメタデータとして使用される単語やフレーズのことを指します。各タグは、キーと値から構成されます。 AWS Private CA コンソール、 AWS Command Line Interface （ AWS CLI ）、または PCA API を使用して、プライベート CAs。

プライベート CA のカスタムタグはいつでも追加または削除できます。例えば、Environment=Prod または Environment=Beta のようなキーと値のペアでプライベート CA にタグを付けると、CA の対象となる環境を特定することができます。詳細については、「プライベート CA の作成」を参照してください。

> **Note**
>
> 作成プロシージャ中にプライベート CA にタグをアタッチするには、CA 管理者はまずインライン IAM ポリシーを CreateCertificateAuthority アクションに関連付けて、タグ付

けを明示的に許可する必要があります。詳細については、「T ag-on-create: 作成時に CA にタグをアタッチする」を参照してください。

他の AWS リソースもタグ付けをサポートしています。同じタグを異なるリソースに割り当てて、それらのリソースが関連していることを示すことができます。例えば、CA、Elastic Load Balancing ロードバランサー、その他の関連リソースに `Website=example.com` のようなタグを割り当てることができます。 AWS リソースのタグ付けの詳細については、Amazon EC2 ユーザーガイド」の「Amazon EC2 リソースのタグ付け」を参照してください。 Amazon EC2

AWS Private CA タグには、次の基本的な制限が適用されます。

- プライベート CA あたりのタグの最大数は 50 です。
- タグキーの最大長は 128 文字です。
- タグ値の最大長は 256 文字です。
- タグキーと値には、A-Z、a-z の文字と次の記号を使用できます。+、=、 @、_、%、- (ハイフン)
- タグのキーと値は大文字と小文字が区別されます。
- `aws:` と `rds:` プレフィックスは、 AWS 用に予約されています。キーが `aws:` または `rds:` で始まるタグを追加、編集、削除することはできません。で始まりaws:、tags-per-resource クォータにはカウントrds:されないデフォルトのタグ。
- 複数のサービス間およびリソース間でタグ付けスキーマを使用する場合、他のサービスでは許可される文字の制限が異なる場合があるのでご注意ください。該当するサービスのドキュメントを参照してください。
- AWS Private CA タグは、 の Resource Groups およびタグエディタでは使用できません AWS Management Console。

プライベート CA には、 AWS Private CA コンソール、 AWS Command Line Interface （ AWS CLI ）、または AWS Private CA APIからタグ付けできます。

プライベート CA にタグを付けるには (コンソール)

1. AWS アカウントにサインインし、https://console.aws.amazon.com/acm-pca/home で AWS Private CA コンソールを開きます。
2. [プライベート認証機関] ページで、リストからプライベート CA を選択します。
3. リストの下の詳細領域で、[タグ] タブを選択します。既存のタグの一覧が表示されます。
4. [Manage tags (タグの管理)] を選択します。

5. 新しいタグを追加を選択します。
6. キーと値のペアを入力します。
7. [保存] を選択します。

プライベート CA にタグを付けるには (AWS CLI)

tag-certificate-authority コマンドを使用して、プライベート CA にタグを追加します。

```
$ aws acm-pca tag-certificate-authority \
     --certificate-authority-arn arn:aws:acm-pca:region:account:certificate-
authority/CA_ID \
     --tags Key=Admin,Value=Alice
```

list-tags コマンドを使用して、プライベート CA のタグを一覧表示します。

```
$ aws acm-pca list-tags \
     --certificate-authority-arn arn:aws:acm-pca:region:account:certificate-
authority/CA_ID \
     --max-results 10
```

untag-certificate-authority コマンドを使用して、プライベート CA からタグを削除します。

```
$ aws acm-pca untag-certificate-authority \
     --certificate-authority-arn arn:aws:acm-pca:aregion:account:certificate-
authority/CA_ID \
     --tags Key=Purpose,Value=Website
```

プライベート CA の更新

プライベート CA を作成すると、そのステータスを更新したり、失効設定を変更したりできます。このトピックでは、CA のステータスと CA ライフサイクルの詳細と、CA のコンソールと CLI 更新の例について説明します。

CA ステータスの更新

によって管理される CA のステータスは、ユーザーアクション、または場合によってはサービスアクションから AWS Private CA 取得されます。例えば、CA のステータスは有効期限が切れると変化します。CA 管理者が使用できるステータスオプションは、CA の現在のステータスによって異なります。

AWS Private CA は、次のステータス値をレポートできます。この表には、各状態で使用可能な CA 機能が示されています。

> **Note**
>
> DELETED と FAILED 以外のすべてのステータス値では、CA に対して課金されます。

ステータス	証明書の発行	OCSP で証明書を検証する	CRL の生成	監査の生成	CA 証明書を更新可能	証明書を取り消し可能	CA に料金が発生します
CREATING — CA を作成中です。	なし	いいえ	いいえ	いいえ	いいえ	いいえ	あり
PENDING_CERTIFICATE — CA が作成され、証明書を運用可能にする必要があります。*	なし	いいえ	いいえ	いいえ	いいえ	いいえ	はい
ACTIVE	はい	はい	はい	はい	はい	はい	あり
DISABLED — CA を手動で無効にしました。	なし	はい	はい	はい	いいえ	はい	あり
EXPIRED — CA 証明書の有効期限が切れています。**	なし	いいえ	いいえ	いいえ	はい	いいえ	あり
FAILED	CreateCertificateAuthority アクションが失敗しました。これは、ネットワークの停止、バックエンド障害 AWS 、またはその他のエラーが原因で発生する可能性があり						なし

ステータス	証明書の発行	OCSP で証明書を検証する	CRL の生成	監査の生成	CA 証明書を更新可能	証明書を取り消し可能	CA に料金が発生します
	ます。障害が生じた CA は回復できません。CA を削除し、新しいCA を作成してください。						
DELETED	CA は復元期間内です。復元期間は 7 ~ 30 日間です。この期間が過ぎると、完全に削除されます。 • DELETED ステータスで、証明書の有効期限が切れた CA の RestoreCertificateAuthority API を呼び出すと、CA は EXPIRED に設定されます。 • CA の削除の詳細については、「プライベート CA の削除」を参照してください。						なし

* アクティベーションを完了するには、CSR を生成し、CA から署名付き CA 証明書を取得し、その証明書を AWS Private CAにインポートする必要があります。CSR は、新しい CA (自己署名用) に送信するか、オンプレミスのルート CA または下位 CA に送信することができます。詳細については、「CA 証明書の作成とインストール」を参照してください。

** 有効期限切れ CA のステータスを直接変更することはできません。CA の新しい証明書をインポートすると、証明書の有効期限が切れDISABLEDる前に に設定されACTIVEない限り、 はステータスを に AWS Private CA リセットします。

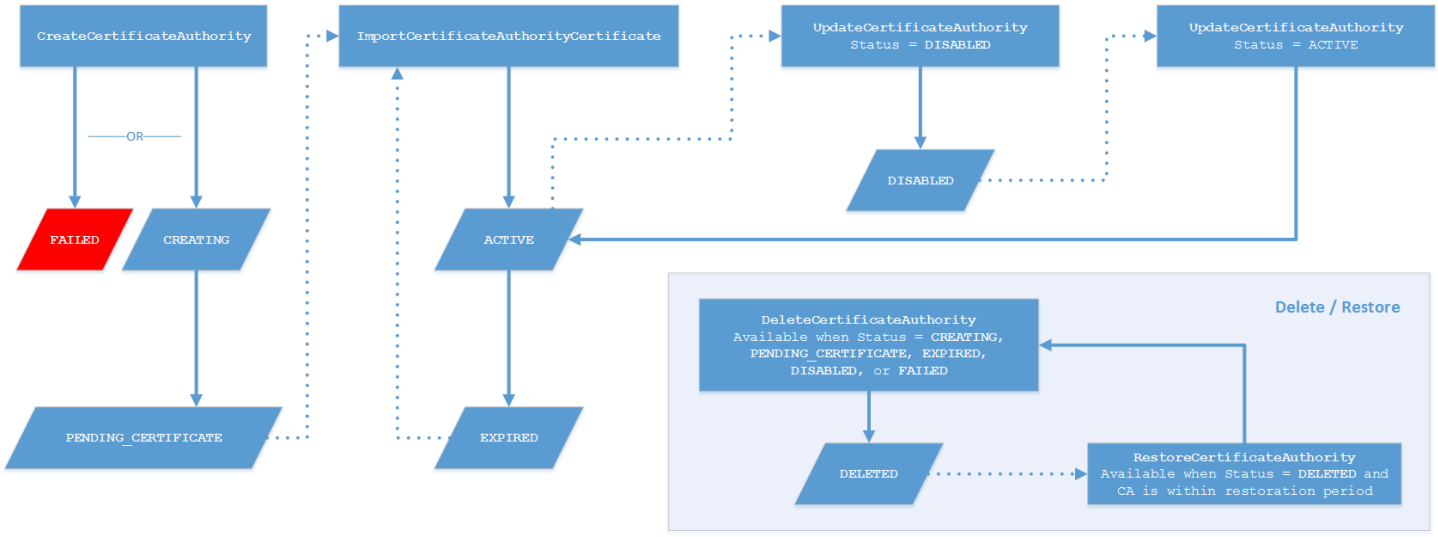
期限切れの CA 証明書に関するその他の考慮事項:

- CA 証明書は自動的には更新されません。による更新の自動化については AWS Certificate Manager、「」を参照してくださいACM に証明書の更新許可を割り当てる。
- 有効期限が切れた CA で新しい証明書を発行しようとすると、 IssueCertificate API は InvalidStateException を返します。有効期限切れのルート CA は、新しい下位証明書を発行する前に、新しいルート CA 証明書に自己署名する必要があります。

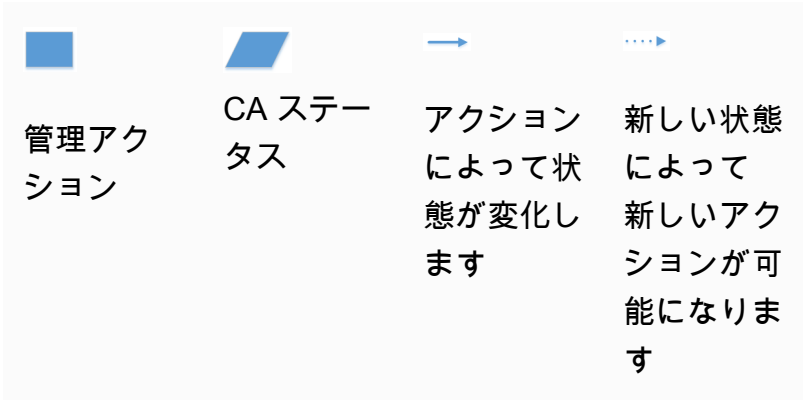
- The `ListCertificateAuthorities` および `DescribeCertificateAuthority` API は、CA のステータスが `ACTIVE` または `DISABLED` に設定されているかどうかにかかわらず、CA 証明書の有効期限が切れているときに `EXPIRED` のステータスを返します。ただし、期限切れ CA が `DELETED` に設定されている場合、`DELETED` のステータスを返します。
- `UpdateCertificateAuthority` API は、有効期限切れの CA のステータスを更新できません。
- `RevokeCertificate` API を使用して、CA 証明書などの有効期限切れの証明書を取り消すことはできません。

CA ステータスと CA ライフサイクル

次の図は、管理アクションと CA ステータスの相互作用としての CA ライフサイクルを示しています。



ダイアグラムキー



図の上部にある管理アクションは、 AWS Private CA コンソール、CLI、または API を介して適用されます。これらのアクションによって、CA の作成、アクティベーション、失効、更新が実行されます。手動操作または自動更新に応じて、CA ステータスが変化します (上記の図では実線で示しています)。ほとんどの場合、新しいステータスによって、CA 管理者は新しいアクション (図では点線で表示) を実行できるようになります。上記の図の右下部分には、削除および復元アクションを許可するステータス値を示しています。

CA の更新 (コンソール)

次の手順は、 AWS Management Consoleを使用して既存の CA 設定を更新する方法を示しています。

CA ステータスの更新 (コンソール)

この例では、有効な CA のステータスが無効に変更されます。

CA のステータスを更新するには

1. AWS アカウントにサインインし、https://console.aws.amazon.com/acm-pca/home で AWS Private CA コンソールを開きます。
2. [プライベート認証機関] ページで、現在アクティブなプライベート CA をリストから選択します。
3. [アクション] メニューで [無効化] を選択してプライベート CA を無効化します。

CA の失効設定の更新 (コンソール)

プライベート CA の失効設定を更新するには、例えば OCSP または CRL のサポートを追加または削除するか、それらの設定を変更します。

> **Note**
>
> CA の失効設定を変更しても、既に発行された証明書には影響しません。マネージド失効を有効にするには、古い証明書を再発行する必要があります。

OCSP では以下の設定を変更します。

- OCSP の有効化または無効化。
- カスタム OCSP 完全修飾ドメイン名 (FQDN) の有効化または無効化。

- FQDN の変更。

CRL では以下のいずれかの設定を変更します。

- プライベート CA が証明書失効リスト (CRL) を生成するかどうか
- CRL の有効期限が切れるまでの日数。は、指定した日数の 1/2 に CRL の再生成を試行 AWS Private CA し始めます。
- CRL の保存先の Amazon S3 バケットの名前。
- Amazon S3 バケット名をパブリックビューから隠すためのエイリアス。

> **Important**
>
> 前述のいずれかのパラメータを変更すると、悪影響が生じることがあります。例としては、CRL 生成の無効化、有効期間の変更、プライベート CA を本番環境に移行した後の S3 バケットの変更などがあります。このような変更を行うと、CRL と現在の CRL 設定に依存する既存の証明書が破損する可能性があります。エイリアスの変更は、古いエイリアスが正しいバケットにリンクされた状態である限り、安全に実行できます。

失効設定を更新するには

1. AWS アカウントにサインインし、https://console.aws.amazon.com/acm-pca/home で AWS Private CA コンソールを開きます。
2. [プライベート認証機関] ページで、リストからプライベート CA を選択します。すると、CA の詳細パネルが開きます。
3. [失効設定] タブを選択し、[編集] を選択します。
4. [証明書失効オプション] には、2 つのオプションが表示されます。

   - [CRL のディストリビューションをアクティブ化]
   - [OCSP をオンにする]

   CA では、これらの失効メカニズムのいずれかを設定することも、いずれも設定しないことも、両方を設定することもできます。任意ではありますが、ベストプラクティスとしてはマネージド失効が推奨されます。このステップを完了する前に、各方法の利点、必要となる事前設定、その他の失効機能に関する情報について、「証明書失効方法の設定」を参照してください。

CRL を設定するには

1. 「CRL のディストリビューションを有効化」を選択します。
2. CRL エントリ用の Amazon S3 バケットを作成するには、「新しい S3 バケットを作成」を選択してください。一意のバケット名を指定します。(バケットへのパスを含める必要はありません)。それ以外の場合は、このオプションを選択しないで、[S3 バケット名] リストから既存のバケットを選択してください。

   新しいバケットを作成すると、 は必要なアクセスポリシー AWS Private CA を作成してアタッチします。既存のバケットを使用する場合は、CRL の生成を開始する前にそのバケットにアクセスポリシーをアタッチする必要があります。Amazon S3 の CRL のアクセスポリシー で説明されているポリシーパターンのいずれかを使用してください。ポリシーのアタッチについては、「Amazon S3 コンソールを使用したバケットポリシーの追加」を参照してください。

   > **Note**
   >
   > AWS Private CA コンソールを使用している場合、次の両方の条件が適用されると、CA の作成は失敗します。
   >
   > - Amazon S3 バケットまたはアカウントでパブリックアクセスブロック設定を実施しています。
   > - Amazon S3 バケットを自動的に作成 AWS Private CA するように に指示しました。
   >
   > この場合、コンソールはデフォルトでパブリックにアクセス可能なバケットを作成しようとしますが、Amazon S3 はこのアクションを拒否します。このような場合は、Amazon S3 設定を確認してください。詳細については、「Amazon S3 ストレージへのパブリックアクセスのブロック」を参照してください。

3. 追加の設定オプションを表示するには、[アドバンスト] を展開します。
   - カスタム CRL 名を追加して、Amazon S3 バケットのエイリアスを作成します。この名前は、RFC 5280 で定義されている「CRL ディストリビューションポイント」拡張で CA が発行した証明書に含まれています。
   - CRL が有効になる日数を入力します。デフォルト値は 7 日です。オンライン CRLs の場合、有効期間は 2～7 日です。 は、指定された期間の中間時点で CRL を再生成 AWS Private CA しようとします。
4. 完了したら、[変更を保存] を選択します。

OCSP を設定するには

1. [証明書失効] ページで [OCSP をオンにする] を選択します。
2. (任意)[カスタム OCSP エンドポイント] フィールドに、OCSP エンドポイントの完全修飾ドメイン名 (FQDN) を入力します。

   このフィールドに FQDN を指定すると、 は AWS OCSP レスポンダーのデフォルト URL の代わりに、発行された各証明書の Authority Information Access 拡張機能に FQDN AWS Private CA を挿入します。エンドポイントは、カスタム FQDN を含む証明書を受け取ると、そのアドレスに OCSP レスポンスを問い合わせます。このメカニズムを機能させるには、さらに 2 つのアクションを実行する必要があります。

   - プロキシサーバーを使用して、カスタム FQDN に到着したトラフィックを AWS OCSP レスポンダーに転送します。
   - 対応する CNAME レコードを DNS データベースに追加します。

   > **Tip**
   >
   > カスタム CNAME を使用して完全な OCSP ソリューションを実装する方法の詳細については、「AWS Private CA OCSP 用のカスタム URL の設定」を参照してください。

   例えば、Amazon Route 53 に表示されるようにカスタマイズされた OCSP の CNAME レコードを次に示します。

レコード名	タイプ	ルーティングポリシー	差別化要因	値/トラフィックのルーティング先
alternative.example.com	CNAME	低	-	proxy.example.com

> **Note**
>
> CNAME の値には、「http://」や「https://」などのプロトコルプレフィックスを含めることはできません。

3. 完了したら、[変更を保存] を選択します。

CA の更新 (CLI)

以下の手順は、 AWS CLIを使用して既存の CA のステータスと失効設定を更新する方法を示しています。

> **Note**
>
> CA の失効設定を変更しても、既に発行された証明書には影響しません。マネージド失効を有効にするには、古い証明書を再発行する必要があります。

プライベート CA のステータスを更新するには (AWS CLI)

update-certificate-authorityコマンドを実行します。

これは、ステータスが DISABLED の既存の CA を ACTIVE に設定する場合に便利です。まず、以下のコマンドで CA の初期ステータスを確認します。

```
$ 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
```

この結果、次のような出力が得られます。

```
{
    "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": "DOC-EXAMPLE-BUCKET1"
   },
            "OcspConfiguration": {
                "Enabled": false
            }
        }
    }
}
```

以下のコマンドは、プライベート CA のステータスを ACTIVE に設定します。これは CA に有効な証明書がインストールされている場合にのみ可能です。

```
$ 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"
```

CA の新しいステータスを検査します。

```
$ 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
```

ステータスは ACTIVE と表示されています。

```
{
    "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": "DOC-EXAMPLE-BUCKET1"
            },
            "OcspConfiguration": {
                "Enabled": false
            }
        }
    }
}
```

場合によっては、アクティブな CA に失効メカニズムが設定されていないことがあります。証明書失効リスト (CRL) の使用を開始する場合は、次の手順を使用します。

CRL を既存の CA に追加するには (AWS CLI)

1. 次のコマンドを使用して、CA の現在のステータスを調べます。

```
$ 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
```

出力から、CA のステータスは ACTIVE であっても CRL を使用するようには設定されていないことが確認されます。

```
{
    "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. CRL 設定パラメータを定義するファイルを、revoke_config.txt のような名前で作成して保存します。

```
{
   "CrlConfiguration":{
      "Enabled": true,
      "ExpirationInDays": 7,
      "S3BucketName": "bucket-name"
   }
}
```

> **Note**
>
> Matter デバイス認証 CA を更新して CRLsを有効にする場合は、現在の Matter 標準に準拠できるように、発行された証明書から CDP 拡張機能を省略するように設定する必要があります。これを行うには、次に示すように CRL 設定パラメータを定義します。
>
> ```
> {
>    "CrlConfiguration":{
>       "Enabled": true,
>       "ExpirationInDays": 7,
>       "S3BucketName": "bucket-name"
>       "CrlDistributionPointExtensionConfiguration":{
>          "OmitExtension": true
>       }
>    }
> }
> ```

3. update-certificate-authority コマンドと失効設定ファイルを使用して CA を更新します。

```
$ 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. CA のステータスをもう一度調べます。

```
$ 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
```

出力から、今は CA が CRL を使用するように設定されていることが確認されます。

```
{
    "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_numbner",
        "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": "DOC-EXAMPLE-BUCKET1",
            },
            "OcspConfiguration": {
                "Enabled": false
            }
        }
    }
```

```
}
```

場合によっては、前の手順のように CRL を有効にする代わりに、OCSP 失効サポートを追加する必要があります。その場合は、次の手順に従ってください。

既存の CA に OCSP サポートを追加するには (AWS CLI)

1. OCSP パラメータを定義するファイルを、`revoke_config.txt` のような名前で作成して保存します。

```
{
   "OcspConfiguration":{
      "Enabled":true
   }
}
```

2. update-certificate-authority コマンドと失効設定ファイルを使用して CA を更新します。

```
$ 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. CA のステータスをもう一度調べます。

```
$ 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
```

出力から、今は CA が OCSP を使用するように設定されていることが確認されます。

```
{
    "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**
>
> CA では CRL と OCSP の両方のサポートを設定することもできます。

プライベート CA の削除

プライベート CA は、 AWS Management Console または から AWS CLI 完全に削除できます。たとえば、ある CA を 1 つ削除して、新しいプライベートキー持つ新しい CA に置き換えることができます。CA を安全に削除するには、次のステップを実行します。

1. 代替 CA を作成します。
2. 新しいプライベート CA の稼働後、古いプライベート CA を無効にします。ただし、すぐには削除しないでください。

3. 古い CA によって発行されたすべての証明書が期限切れになるまで、その古い CA は無効のままにします。
4. 古い CA を削除します。

AWS Private CA は、削除リクエストを処理する前に、発行されたすべての証明書の有効期限が切れているかどうかをチェックしません。監査報告書を生成し、期限切れになっている証明書を確認できます。CA が無効になっている間、証明書を取り消すことができますが、新しい証明書を発行することはできません。

発行されたすべての証明書が期限切れになる前にプライベート CA を削除する必要がある場合は、CA 証明書を取り消すこともお勧めします。CA 証明書は親 CA の CRL に一覧表示され、プライベート CA はクライアントによって信頼されなくなります。

> **Important**
>
> プライベート CA は、PENDING_CERTIFICATE、CREATING、EXPIRED、 DISABLED、または FAILED 状態である場合に削除できます。ACTIVE 状態の CA を削除するためには、まずその CA を削除する必要があります。さもないと、削除リクエストで例外が発生します。PENDING_CERTIFICATE または DISABLED の状態にあるプライベート CA を削除する場合、復元期間を 7～30 日 (デフォルトは 30 日) に設定することができます。この期間中、ステータスは DELETED に設定され、CA は復元可能です。CREATING または FAILED の状態で削除されたプライベート CA は、復元期間が割り当てられていないため、復元できません。詳細については、「プライベート CA の復元」を参照してください。
> プライベート CA が削除された後は、それに対して課金されません。ただし、削除された CA が復元されると、削除と復元の間の料金が課金されます。詳細については、「料金」を参照してください。

プライベート CA を削除するには (コンソール)

1. AWS アカウントにサインインし、https://console.aws.amazon.com/acm-pca/home で AWS Private CA コンソールを開きます。
2. [プライベート認証機関] ページで、リストからプライベート CA を選択します。
3. CA が ACTIVE の状態にある場合、先にその CA を無効化する必要があります。[アクション] メニューで、[無効化] を選択します。プロンプトが表示されたら、[私はリスクを理解しており、続行します] を選択します。
4. ACTIVE の状態にない CA の場合は、[アクション]、[削除] を選択します。

5. CA が DISABLED、EXPIRED、または PENDING_CERTIFICATE の状態の場合、[CA を削除] ページでは 7～30 日の復元期間を指定できます。プライベート CA がこれらの状態のいずれでもない場合、後で復元することはできず、削除は完全なものになります。
6. [削除] を選択します。
7. プライベート CA を完全に削除する場合、確認のプロンプトが表示されたら、[完全に削除] を選択します。プライベート CA のステータスが DELETED に変わります。ただし、復元期間の終了前には、プライベート CA を復元することができます。DELETED 状態のプライベート CA の復元期間を確認するには、 DescribeCerticateAuthorityまたは ListCertificateAuthorities API オペレーションを呼び出します。

プライベート CA を削除するには (AWS CLI)

プライベート CA を削除するには、 delete-certificate-authority コマンドを使用します。

```
$ 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
```

プライベート CA の復元

CA を削除したときに指定した復元期間の間は、削除されたプライベート CA を復元できます。復元期間は 7～30 日です。プライベート CA は、復元期間の最後に完全に削除されます。詳細については、「プライベート CA の削除」を参照してください。完全に削除されたプライベート CA を復元することはできません。

> **Note**
>
> プライベート CA が削除された後は、それに対して課金されません。ただし、削除された CA が復元されると、削除と復元の間の料金が課金されます。詳細については、「料金」を参照してください。

プライベート CA の復元 (コンソール)

を使用してプライベート CA を AWS Management Console 復元できます。

プライベート CA を復元するには (コンソール)

1. AWS アカウントにサインインし、https://console.aws.amazon.com/acm-pca/home で AWS Private CA コンソールを開きます。
2. [プライベート認証機関] ページで、削除したプライベート CA をリストから選択します。
3. [アクション] メニューで、[復元] を選択します。
4. [CA を復元] ページで、再度 [復元] を選択します。
5. 成功すると、プライベート CA のステータスは、削除前の状態に戻ります。[アクション]、[有効化]、もう一度[有効化] を選択して、ステータスを ACTIVE に変更します。削除時にプライベート CA がPENDING_CERTIFICATE 状態だった場合は、CA 証明書をアクティブ化する前に、プライベート CA に CA 証明書をインポートする必要があります。

プライベート CA の復元 (AWS CLI)

restore-certificate-authority コマンドを使用して、 DELETED状態にある削除されたプライベート CA を復元します。次の手順では、プライベート CA の削除、復元、および再アクティブ化に必要なプロセス全体について説明します。

プライベート CA を削除、復元、および再アクティブ化するには (AWS CLI)

1. プライベート CA を削除します。

   delete-certificate-authority コマンドを実行してプライベート CA を削除します。プライベート CA のステータスが DISABLED または PENDING_CERTIFICATE である場合、--permanent-deletion-time-in-days パラメータを設定して、プライベート CA の復元期間を 7～30 日で指定することができます。復元期間を指定しない場合、デフォルトは 30 日です。成功すると、このコマンドによりプライベート CA の状態は DELETED に設定されます。

   > **Note**
   >
   > 復元可能であるためには、削除の時点でのプライベート CA のステータスは DISABLED または PENDING_CERTIFICATE である必要があります。

```
$ 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. プライベート CA を復元します。

   restore-certificate-authority コマンドを実行して、プライベート CA を復元します。delete-certificate-authority コマンドで設定した復元期間が終了する前に、このコマンドを実行する必要があります。成功すると、このコマンドにより、プライベート CA のステータスは、削除前のステータスに設定されます。

```
$ aws acm-pca restore-certificate-authority \
    --certificate-authority-arn arn:aws:acm-pca:region:account:certificate-authority/CA_ID
```

3. プライベート CA を ACTIVE に設定します。

   update-certificate-authority コマンドを実行して、プライベート CA のステータスを に変更しますACTIVE。

```
$ aws acm-pca update-certificate-authority \
    --certificate-authority-arn arn:aws:acm-pca:region:account:certificate-authority/CA_ID \
    --status ACTIVE
```

証明書管理

プライベート認証局 (CA) を作成して有効化し、アクセスを設定すると、管理者または権限のあるユーザーがこのセクションで説明するタスクを実行できます。CA 用の AWS Identity and Access Management (IAM) ポリシーをまだ設定していない場合は、このガイドの「Identity and Access Management」セクションでポリシーの設定について詳しく知ることができます。シングルアカウントシナリオとクロスアカウントシナリオでの CA アクセスの設定については、「プライベート CA へのアクセスの制御」を参照してください。

トピック

プライベートエンドエンティティ証明書の発行

プライベート CA を設定すると、AWS Certificate Manager (ACM) または AWS Private CA からプライベートエンドエンティティ証明書をリクエストできます。次の表では、この 2 つのサービスの機能を比較します。

機能	ACM	AWS Private CA
エンドエンティティ証明書を発行する	✓ (RequestCertificate または コンソールを使用)	✓ (を使用IssueCertificate)
ロードバランサーおよびインターネット向け AWS サービスとの関係	✓	非サポート
マネージド証明書の更新	✓	ACM により、間接的にサポートされています

機能	ACM	AWS Private CA
コンソールのサポート	✓	非サポート
API サポート	✓	✓
CLI サポート	✓	✓

AWS Private CA が証明書を作成するとき、証明書の種類とパスの長さを指定するテンプレートに従います。証明書を作成する API または CLI ステートメントにテンプレート ARN が指定されていない場合、EndEntityCertificate/V1 テンプレートはデフォルトで適用されます。使用可能な証明書テンプレートの詳細については、「証明書テンプレートについて」を参照してください。

ACM 証明書は公共の信頼に基づいて設計されていますが、AWS Private CA はユーザーのプライベート PKI のニーズに応えます。そのため、ACM では許可されていない方法で、AWS Private CA API と CLI を使用して証明書を設定できます。これには以下が含まれます。

- 任意の件名を持つ証明書の作成。
- サポートされている任意のプライベートキーアルゴリズムとキー長 を使用する。
- サポートされている署名アルゴリズム のいずれかを使用する。
- プライベート CA とプライベート証明書の有効期間の指定。

AWS Private CA を使用してプライベート TLS 証明書を作成すると、それを ACM にインポートして、サポートされている AWS サービスで使用できます。

> **Note**
>
> 以下の手順、 issue-certificate コマンド、または IssueCertificate API アクションを使用して作成された証明書は、 の外部で使用するために直接エクスポートすることはできません AWS。ただし、プライベート CA を使用して ACM を通じて発行された証明書に署名することができ、それらの証明書はシークレットキーとともにエクスポートできます。詳細については、「ACM ユーザーガイド」の「プライベート証明書のリクエスト」と「プライベート証明書のエクスポート」を参照してください。

標準証明書の発行 (AWS CLI)

CLI コマンド issue-certificate AWS Private CA または API アクションIssueCertificateを使用して、エンドエンティティ証明書をリクエストできます。このコマンドでは、証明書の発行に使用するプライベート CA の Amazon リソースネーム (ARN) が必要です。また、OpenSSL などのプログラムを使用して、証明書署名リクエスト (CSR) を生成する必要があります。

AWS Private CA API または AWS CLI を使用してプライベート証明書を発行する場合、証明書は管理されません。つまり、ACM コンソール、ACM CLI、または ACM API を使用して証明書を表示またはエクスポートすることはできず、証明書は自動的に更新されません。ただし、PCA の get-certificate コマンドを使用して証明書の詳細を取得することができます。CA を所有している場合は監査レポート を作成できます。

証明書を作成する際の考慮事項

- RFC 5280 に準拠している場合、ドメイン名 (技術的には Common Name) の長さは、ピリオドを含む 64 オクテット (文字) を超えることはできません。より長いドメイン名を追加するには、[サブジェクト代替名] フィールドにドメイン名を指定します。このフィールドでは、最大 253 オクテットの名前がサポートされます。
- AWS CLI バージョン 1.6.3 以降を使用している場合は、CSRs などの base64 でエンコードされた入力ファイルを指定するfileb://ときに プレフィックスを使用します。こうすることで、AWS Private CA はデータを正しく解析します。

次の OpenSSL コマンドは、証明書の CSR とプライベートキーを生成します。

```
$ openssl req -out csr.pem -new -newkey rsa:2048 -nodes -keyout private-key.pem
```

CSR の内容は次のように検査できます。

```
$ openssl req -in csr.pem -text -noout
```

結果の出力は、次の省略された例のようになります。

```
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
```

次のコマンドは、証明書を作成します。テンプレートが指定されていないため、ベースエンドエンティティ証明書がデフォルトで発行されます。

```
$ 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"
```

発行された証明書の ARN が返されます。

```
{
   "CertificateArn":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID"
}
```

> **Note**
>
> AWS Private CA は issue-certificate コマンドを受信すると、シリアル番号の付いた ARN を即時に返します。ただし、証明書の処理は非同期で行われるため、失敗する可能性があります。この場合、新しい ARN を使用する get-certificate コマンドも失敗します。

APIPassthrough テンプレートを使用して、カスタムサブジェクト名で証明書を発行します。

この例では、カスタマイズされたサブジェクト名要素を含む証明書が発行されます。標準証明書の発行 (AWS CLI) にあるような CSR の指定に加えて、issue-certificate コマンドには、APIPassthrough テンプレートの ARN と、カスタム属性とそのオブジェクト識別子 (OID) を指定する JSON 設定ファイルの 2 つの引数を追加で渡します。`StandardAttributes` と `CustomAttributes` を組み合わせて使用することはできませんが、標準 OID を `CustomAttributes` の一部として渡すことはできます。デフォルトのサブジェクト名 OID を以下の表に示します (RFC 4519 および グローバル OID リファレンスデータベース からの情報)。

サブジェクト名	略語	オブジェクト ID
countryName	c	2.5.4.6
commonName	cn	2.5.4.3
dnQualifier [識別名修飾子]		2.5.4.46
generationQualifier		2.5.4.44
givenName		2.5.4.42
initials		2.5.4.43
locality	l	2.5.4.7
organizationName	o	2.5.4.10
organizationalUnitName	ou	2.5.4.11
pseudonym		2.5.4.65

サブジェクト名	略語	オブジェクト ID
serialNumber		2.5.4.5
st [状態]		2.5.4.8
surname	sn	2.5.4.4
title		2.5.4.12
domainComponent	dc	0.9.2342.19200300.100.1.25
userid		0.9.2342.19200300.100.1.1

サンプル設定ファイル `api_passthrough_config.txt` には以下のコードが含まれています。

```
{
  "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"
      }
    ]
  }
}
```

次のコマンドを使用して、証明書を発行します。

```
$ 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
```

発行された証明書の ARN が返されます。

```
{
   "CertificateArn":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID"
}
```

以下のように証明書をローカルで取得します。

```
$ 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
```

OpenSSL を使用して、証明書の内容を検査できます。

```
$ openssl x509 -in cert.pem -text -noout
```

> **Note**
>
> 発行する各証明書にカスタム属性を渡すプライベート CA を作成することもできます。

APIPassthrough テンプレートを使用して、カスタム拡張付きの証明書を発行します。

この例では、カスタマイズされた拡張を含む証明書が発行されます。そのためには、APIPassthrough テンプレートの ARN、カスタム拡張を指定する JSON 設定ファイル、および標準証明書の発行 (AWS CLI) に示されたような CSR の 3 つの引数を issue-certificate コマンドに渡す必要があります。

サンプル設定ファイル api_passthrough_config.txt には以下のコードが含まれています。

```
{
  "Extensions": {
    "CustomExtensions": [
      {
        "ObjectIdentifier": "2.5.29.30",
        "Value": "MBWgEzARgg8ucGVybWl0dGVkLnRlc3Q=",
        "Critical": true
      }
    ]
  }
}
```

カスタマイズされた証明書は次のように発行されます。

```
$ 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
```

発行された証明書の ARN が返されます。

```
{
   "CertificateArn":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID"
}
```

以下のように証明書をローカルで取得します。

```
$ 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
```

OpenSSL を使用して、証明書の内容を検査できます。

```
$ openssl x509 -in cert.pem -text -noout
```

プライベート 証明書を取得する

AWS Private CA API と AWS CLI を使用して、プライベート証明書を発行できます。その場合は、AWS CLI または AWS Private CA API を使用して証明書を取得できます。ACM を使用してプライベート CA を作成し、証明書をリクエストした場合、ACM を使用して証明書および暗号化されたプライベートキーをエクスポートできます。詳細については「プライベート証明書のエクスポート」を参照してください。

エンドエンティティ証明書を取得するには

AWS CLI コマンドの get-certificate を使用してプライベートエンドエンティティ証明書を取得します。GetCertificate API オペレーションを使用することもできます。出力は sed に似たパーサーである jq でフォーマットすることをお勧めします。

> **Note**
>
> 証明書を取り消す場合は、get-certificate コマンドを使用して 16 進数形式のシリアル番号を取得します。また、監査レポートを作成して 16 進数シリアル番号を取得することもできます。詳細については、「プライベート CA での監査レポートの使用」を参照してください。

```
$ 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'
```

このコマンドは、以下の標準形式の証明書と証明書チェーンを出力します。

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

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

CA 証明書を取得するには

AWS Private CA API および AWS CLI を使用してプライベート認証機関 (CA) の CA 証明書を取得できます。get-certificate-authority-certificate コマンドを実行します。また、GetCertificateAuthorityCertificate オペレーションを呼び出すこともできます。出力は sed に似たパーサーである jq でフォーマットすることをお勧めします。

```
$ 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'
```

このコマンドは CA 証明書を以下の標準形式で出力します。

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

プライベート証明書の一覧表示

プライベート証明書を一覧表示するには、監査レポートを生成し、S3 バケットから取得して、必要に応じてレポートの内容を解析します。AWS Private CA 認証情報レポート作成の詳細については、「プライベート CA での監査レポートの使用」を参照してください。S3 バケットからオブジェクトを取得する方法については、「Amazon Simple Storage Service ユーザーガイド」の「オブジェクトのダウンロード」を参照してください。

以下の例は、監査レポートを作成し、それを解析して有用なデータを見つける方法を示しています。結果は JSON 形式で、データは sed に似たパーサーである jq を使用してフィルタリングされます。

1. 監査報告書を作成する。

次のコマンドは、指定した CA の監査レポートを生成します。

```
$ 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
```

成功すると、コマンドは新しい監査レポートの ID と場所を返します。

```
{
   "AuditReportId":"audit_report_ID",
   "S3Key":"audit-report/CA_ID/audit_report_ID.json"
}
```

2. 監査レポートを取得してフォーマットします。

このコマンドは、監査レポートを取得し、その内容を標準出力で表示し、結果をフィルタリングして 2020-12-01 以降に発行された証明書のみを表示します。

```
$ 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")'
```

返された項目は以下のようになります。

```
{
   "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. 監査レポートをローカルに保存します。

複数のクエリを実行する場合は、監査レポートをローカルファイルに保存すると便利です。

```
$ aws s3api get-object \
     --region region \
     --bucket bucket_name \
     --key audit-report/CA_ID/audit_report_ID.json > my_local_audit_report.json
```

以前と同じフィルターで同じ出力が得られます。

```
$ 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. 日付範囲内のクエリ

次のように日付範囲内に発行された証明書をクエリできます。

```
$ cat my_local_audit_report.json | jq '.[] | select(.issuedAt >= "2020-11-01"
 and .issuedAt <= "2020-11-10")'
```

フィルタリングされた内容は標準出力に表示されます。

```
{
   "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. 指定したテンプレートに従って証明書を検索します。

次のコマンドは、テンプレート ARN を使用してレポートコンテンツをフィルタリングします。

```
$ cat my_local_audit_report.json | jq '.[] | select(.templateArn == "arn:aws:acm-
pca:::template/RootCACertificate/V1")'
```

出力には、一致する証明書レコードが表示されます。

```
{
   "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. 失効した証明書をフィルタリングします。

失効した証明書をすべて検索するには、次のコマンドを使用します。

```
$ cat my_local_audit_report.json | jq '.[] | select(.revokedAt != null)'
```

失効した証明書は次のように表示されます。

```
{
   "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. 正規表現を使用してフィルタリングします。

次のコマンドは、「leaf」という文字列を含むサブジェクト名を検索します。

```
$ cat my_local_audit_report.json | jq '.[] | select(.subject|test("leaf"))'
```

一致する証明書レコードは次のように返されます。

```
{
   "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"
}
```

プライベート証明書とそのシークレットキーのエクスポート

AWS Private CA は署名して発行したプライベート証明書を直接エクスポートすることはできません。ただし、AWS Certificate Manager でそのような証明書を暗号化されたシークレットキーとともにエクスポートすることができます。そうすれば、証明書は完全に移植可能になり、プライベート PKI のどこにでもデプロイできます。詳細については、「AWS Certificate Manager ユーザーガイド」の「プライベート証明書のエクスポート」を参照してください。

さらに AWS Certificate Manager では、ACM コンソール、 ACM API の `RequestCertificate` アクション、または AWS CLI の ACM セクション内の request-certificate コマンドを使用して発行されたプライベート証明書の管理更新も可能です。更新についての詳細は、「プライベート PKI の証明書の更新」を参照してください。

プライベート証明書の取り消し

AWS Private CA 証明書は、revoke-certificate コマンドまたは API アクションを使用して取り消すことができます。 AWS CLI RevokeCertificateたとえば、シークレットキーが漏洩したり、関連するドメインが無効になったりした場合、予定されている有効期限が切れる前に証明書を取り消す必要があ

る場合があります。失効を有効にするには、証明書を使用するクライアントが、安全なネットワーク接続を構築しようとするたびに失効状態を確認する方法が必要です。

AWS Private CA には、失効ステータスの確認をサポートする 2 つの完全管理メカニズムが用意されています。オンライン証明書ステータスプロトコル (OCSP) と証明書失効リスト (CRLs) です OCSP では、クライアントは権限のある失効データベースをクエリして、リアルタイムでステータスを返します。CRL を使用すると、クライアントは証明書を、定期的にダウンロードおよび保存される失効した証明書のリストと照合します。クライアントは、失効した証明書の受け入れを拒否します。

OCSP と CRL はどちらも、証明書に埋め込まれた検証情報に依存します。このため、発行前に、発行元の CA がこれらのメカニズムのいずれかまたは両方をサポートするように設定する必要があります。AWS Private CA を通してのマネージド型失効の選択と実装については、「証明書失効方法の設定」を参照してください。

取り消された証明書は常に AWS Private CA 監査レポートに記録されます。

> **Note**
>
> クロスアカウント証明書の発行者には、発行する証明書を取り消すための追加の権限が必要です。それ以外の場合は、CA 所有者が失効を行う必要があります。クロスアカウント発行者による失効を有効にするには、CA 管理者は同じ CA を指す 2 つの RAM 共有を作成する必要があります。
>
> 1. `AWSRAMRevokeCertificateCertificateAuthority` 権限による共有。
> 2. `AWSRAMDefaultPermissionCertificateAuthority` 権限による共有。

証明書を取り消すには

RevokeCertificate API アクションまたは revoke-certificate コマンドを使用して、プライベート PKI 証明書を取り消します。シリアル番号は 16 進形式である必要があります。get-certificate コマンドを呼び出して、シリアル番号を取得できます。`revoke-certificate` コマンドはレスポンスを返しません。

```
$ 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"
```

失効した証明書と OCSP

証明書を取り消すと、OCSP レスポンスに新しいステータスが反映されるまでに最大 60 分かかることがあります。一般に、OCSP は失効情報の配信が速い傾向があります。これは、クライアントが数日間キャッシュすることがある CRL とは異なり、OCSP レスポンスは通常クライアントによってキャッシュされないためです。

CRL で取り消された証明書

CRL は通常、証明書が取り消された約 30 分後に更新されます。何らかの理由で CRL の更新に失敗した場合、AWS Private CA は 15 分ごとに試行を行います。

Amazon では CloudWatch、メトリクス `CRLGenerated`および のアラームを作成できます`MisconfiguredCRLBucket`。詳細については、「サポートされているメトリクス CloudWatch」を参照してください。CRL の作成と設定の詳細については、「証明書失効リスト (CRL) の計画」を参照してください。

以下の例には、証明書失効リスト (CRL) の取り消された証明書を示しています。

```
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
```

監査レポートの取り消された証明書

取り消した証明書を含むすべての証明書には、プライベート CA の監査レポートが含まれます。次の例では、1 つの発行証明書および 1 つの取り消した証明書の監査レポートを示しています。詳細については、「プライベート CA での監査レポートの使用」を参照してください。

```
[
   {
      "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"
   }
]
```

更新された証明書のエクスポートの自動化

AWS Private CA を使用して CA を作成すると、その CA を AWS Certificate Manager にインポートして、ACM に証明書の発行と更新を管理させることができます。更新中の証明書が統合サービスに関連付けられている場合、サービスは新しい証明書をシームレスに適用します。ただし、証明書が元々 PKI 環境の他の場所 (オンプレミスサーバーやアプライアンスなど) で使用するためにエクスポートされたものである場合は、更新後に再度エクスポートする必要があります。

Amazon EventBridge と AWS Lambda を使用して ACM エクスポートプロセスを自動化するサンプルソリューションについては、「更新された証明書のエクスポートの自動化」を参照してください。

証明書テンプレートについて

AWS Private CA は設定テンプレートを使用して CA 証明書とエンドエンティティ証明書の両方を発行します。PCA コンソールから CA 証明書を発行すると、適切なルート CA 証明書テンプレートまたは下位 CA 証明書テンプレートが自動的に適用されます。

CLI または API を使用して証明書を発行すると、`IssueCertificate` アクションのパラメーターとしてテンプレート ARN を指定できます。(ARN を指定しない場合、`EndEntityCertificate/V1` テンプレートがデフォルトで適用されます。) 詳細については、 IssueCertificate API および issue-certificate コマンドのドキュメントを参照してください。

> **Note**
>
> プライベート CA へのクロスアカウント共有アクセス権を持つ AWS Certificate Manager (ACM) ユーザーは、CA が署名したマネージド証明書を発行できます。クロスアカウント発行者はリソースベースのポリシーによって制約され、以下のエンドエンティティ証明書テンプレートにのみアクセスできます。
>
> - EndEntityCertificate/V1
> - EndEntityClientAuthCertificate/V1
> - EndEntityServerAuthCertificate/V1
> - BlankEndEntityCertificate_APIPassthrough/V1

- BlankEndEntityCertificate_APICSRPassthrough/V1
- SubordinateCACertificate _PathLen0/V1

詳細については、「リソースベースのポリシー」を参照してください。

トピック

テンプレートの種類

AWS Private CA は 4 種類のテンプレートをサポートします。

- ベーステンプレート

  パススルーパラメータを使用できない定義済みテンプレート。

- CSRPassthrough テンプレート

  CSR パススルーを許可することで、対応するベーステンプレートバージョンを拡張するテンプレート。証明書の発行に使用される CSR 内の拡張は、発行された証明書にコピーされます。CSR にテンプレート定義と矛盾する拡張値が含まれている場合は、常にテンプレート定義が優先されます。優先度の詳細については、「テンプレートの操作順序」を参照してください。

- APIPassthrough テンプレート

  CSR パススルーを許可することで、対応するベーステンプレートバージョンを拡張するテンプレート。管理者や他の中間システムに知られている動的値は、証明書を要求するエンティティには知られていない場合、テンプレートで定義できない場合、また CSR で使用できない場合があります。ただし、CA 管理者は Active Directory などの別のデータソースから追加情報を取得して、リクエストを完了することができます。たとえば、マシンがどの組織単位に属しているかわからない場合、管理者は Active Directory で情報を検索し、その情報を JSON 構造に含めることで証明書要求に追加できます。

  `IssueCertificate` アクションの の `ApiPassthrough` パラメータの値は、発行された証明書にコピーされます。`ApiPassthrough` パラメータにテンプレート定義と矛盾する情報が含まれて

いる場合は、常にテンプレート定義が優先されます。優先度の詳細については、「テンプレートの操作順序」を参照してください。

- APICSRPassthrough テンプレート

  API と CSR の両方のパススルーを許可することで、対応するベーステンプレートバージョンを拡張するテンプレート。証明書の発行に使用された CSR の拡張は発行された証明書にコピーされ、`IssueCertificate` アクションの `ApiPassthrough` パラメータの値もコピーされます。テンプレート定義、API パススルー値、CSR パススルー拡張に矛盾がある場合は、テンプレート定義が最も優先され、続いて API パススルー値、CSR パススルー拡張の順に優先されます。優先度の詳細については、「テンプレートの操作順序」を参照してください。

次の表に、AWS Private CA によってサポートされるすべてのテンプレートタイプとその定義へのリンクを示します。

> **Note**
>
> GovCloud リージョンのテンプレート ARNsAWS GovCloud (US)「 ユーザーガイドAWS Private Certificate Authority」の「」を参照してください。

ベーステンプレート

テンプレート名	[テンプレート ARN]	証明書タイプ
CodeSigningCertificate/V1	`arn:aws:acm-pca:::template/CodeSigningCertificate/V1`	コード署名
EndEntityCertificate/V1	`arn:aws:acm-pca:::template/EndEntityCertificate/V1`	エンドエンティティ
EndEntityClientAuthCertificate/V1	`arn:aws:acm-pca:::template/EndEntityClientAuthCertificate/V1`	エンドエンティティ

テンプレート名	[テンプレート ARN]	証明書タイプ
EndEntityServerAuthCertificate/V1	arn:aws:acm-pca:::template/EndEntityServerAuthCertificate/V1	エンドエンティティ
OCSPSigningCertificate/V1	arn:aws:acm-pca:::template/OCSPSigningCertificate/V1	OCSP 署名
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
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 テンプレート

テンプレート名	[テンプレート ARN]	証明書タイプ
BlankEndEntityCertificate_CSRPassthrough/V1	arn:aws:acm-pca:::template/BlankEndEntityCertificate_CSRPassthrough/V1	エンドエンティティ
BlankEndEntityCertificateCriticalBasicConstraints__CSRPassthrough/V1	arn:aws:acm-pca:::template/BlankEndEntityCertificate_CriticalBasicConstraints_CSRPassthrough/V1	エンドエンティティ
BlankSubordinateCACertificate_PathLen0_CSRPassthrough /V1	arn:aws:acm-pca:::template/BlankSubordinateCACertificate_PathLen0_CSRPassthrough/V1	CA
BlankSubordinateCACertificate_PathLen1_CSRPassthrough /V1	arn:aws:acm-pca:::template/BlankSubordinateCACertificate_PathLen1_CSRPassthrough/V1	CA
BlankSubordinateCACertificate_PathLen2_CSRPassthrough /V1	arn:aws:acm-pca:::template/BlankSubordinateCACertificate_PathLen2_CSRPassthrough/V1	CA
BlankSubordinateCACertificate_PathLen3_CSRPassthrough /V1	arn:aws:acm-pca:::template/BlankSubordinateCACertifica	CA

テンプレート名	[テンプレート ARN]	証明書タイプ
	te_PathLen3_CSRPas sthrough/V1	
CodeSigningCertificate_CSRP assthrough/V1	arn:aws:acm-pca::: template/CodeSigni ngCertificate_CSRP assthrough/V1	コード署名
EndEntityCertificate_CSRPas sthrough/V1	arn:aws:acm-pca::: template/EndEntity Certificate_CSRPas sthrough/V1	エンドエンティティ
EndEntityClientAuthCertific ate_CSRPassthrough/V1	arn:aws:acm-pca::: template/EndEntity ClientAuthCertific ate_CSRPassthrough/V1	エンドエンティティ
EndEntityServerAuthCertific ate_CSRPassthrough/V1	arn:aws:acm-pca::: template/EndEntity ServerAuthCertific ate_CSRPassthrough/V1	エンドエンティティ
OCSP SigningCertificate _CSRPassthrough /V1	arn:aws:acm-pca::: template/OCSPSigni ngCertificate_CSRP assthrough/V1	OCSP 署名
SubordinateCACertificate _PathLen0_CSRPassthrough /V1	arn:aws:acm-pca::: template/Subordina teCACertificate_Pa thLen0_CSRPassthrough/ V1	CA

テンプレート名	[テンプレート ARN]	証明書タイプ
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 テンプレート

テンプレート名	[テンプレート ARN]	証明書タイプ
BlankEndEntityCertificate_APIPassthrough/V1	arn:aws:acm-pca:::template/BlankEndEntityCertificate_APIPassthrough/V1	エンドエンティティ
BlankEndEntityCertificateCriticalBasicConstraints__APIPassthrough/V1	arn:aws:acm-pca:::template/BlankEndEntityCertificate_CriticalBasicConstraints_APIPassthrough/V1	エンドエンティティ

テンプレート名	[テンプレート ARN]	証明書タイプ
CodeSigningCertificate_APIPassthrough/V1	arn:aws:acm-pca:::template/CodeSigningCertificate_APIPassthrough/V1	コード署名
EndEntityCertificate_APIPassthrough/V1	arn:aws:acm-pca:::template/EndEntityCertificate_APIPassthrough/V1	エンドエンティティ
EndEntityClientAuthCertificate_APIPassthrough/V1	arn:aws:acm-pca:::template/EndEntityClientAuthCertificate_APIPassthrough/V1	エンドエンティティ
EndEntityServerAuthCertificate_APIPassthrough/V1	arn:aws:acm-pca:::template/EndEntityServerAuthCertificate_APIPassthrough/V1	エンドエンティティ
OCSP SigningCertificate _APIPassthrough /V1	arn:aws:acm-pca:::template/OCSPSigningCertificate_APIPassthrough/V1	OCSP 署名
RootCACertificate_APIPassthrough/V1	arn:aws:acm-pca:::template/RootCACertificate_APIPassthrough/V1	CA
BlankRootCACertificate _APIPassthrough /V1	arn:aws:acm-pca:::template/BlankRootCACertificate_APIPassthrough/V1	CA

テンプレート名	[テンプレート ARN]	証明書タイプ
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
BlankSubordinateCACertificate_PathLen0_APIPassthrough /V1	arn:aws:acm-pca:::template/BlankSubordinateCACertificate_PathLen0_APIPassthrough/V1	CA

テンプレート名	[テンプレート ARN]	証明書タイプ
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
BlankSubordinateCACertificate_PathLen3_APIPassthrough /V1	arn:aws:acm-pca:::template/BlankSubordinateCACertificate_PathLen3_APIPassthrough/V1	CA

APICSRPassthrough テンプレート

テンプレート名	[テンプレート ARN]	証明書タイプ
BlankEndEntityCertificate_APICSRPassthrough/V1	arn:aws:acm-pca:::template/BlankEndEntityCertificate_APICSRPassthrough/V1	エンドエンティティ
BlankEndEntityCertificate_CriticalBasicConstraints_APICSRPassthrough /V1	arn:aws:acm-pca:::template/BlankEndEntityCertificate_CriticalBasicConstraints_APICSRPassthrough/V1	エンドエンティティ
CodeSigningCertificate_APICSRPassthrough/V1	arn:aws:acm-pca:::template/CodeSigningCertificate_APICSRPassthrough/V1	コード署名
EndEntityCertificate_APICSRPassthrough/V1	arn:aws:acm-pca:::template/EndEntityCertificate_APICSRPassthrough/V1	エンドエンティティ
EndEntityClientAuthCertificate_APICSRPassthrough/V1	arn:aws:acm-pca:::template/EndEntityClientAuthCertificate_APICSRPassthrough/V1	エンドエンティティ
EndEntityServerAuthCertificate_APICSRPassthrough/V1	arn:aws:acm-pca:::template/EndEntityServerAuthCertific	エンドエンティティ

テンプレート名	[テンプレート ARN]	証明書タイプ
	ate_APICSRPassthrough/V1	
OCSP SigningCertificate_APICSRPassthrough /V1	arn:aws:acm-pca:::template/OCSPSigningCertificate_APICSRPassthrough/V1	OCSP 署名
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
BlankSubordinateCACertificate_PathLen1_APICSRPassthrough / V1	arn:aws:acm-pca:::template/BlankSubordinateCACertificate_PathLen1_APICSRPassthrough/V1	CA

テンプレート名	[テンプレート ARN]	証明書タイプ
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

テンプレートの操作順序

発行された証明書に含まれる情報は、テンプレート定義、API パススルー、CSR パススルー、CA 設定の 4 つのソースから取得できます。

API パススルー値は API パススルーまたは APICSR パススルーテンプレートを使用する場合にのみ考慮されます。CSR パススルーは、CSRPassthrough または APICSR パススルーテンプレートを使用する場合にのみ優先されます。これらの情報源に矛盾がある場合は、通常、拡張値ごとにテンプレート定義が最も優先され、次に API パススルー値、CSR パススルー拡張が続くという一般的なルールが適用されます。

例

1. EndEntityClientAuthCertificate_APIPassthrough のテンプレート定義は、「TLS ウェブサーバー認証、TLS ウェブクライアント認証」という値で ExtendedKeyUsage 拡張機能を定義します。 ExtendedKeyUsage が CSR または `IssueCertificateApiPassthrough`パラメータで定義されている場合、テンプレート定義が優先されるため、 `ApiPassthrough`の値は ExtendedKeyUsage 無視され、テンプレートが CSR パススルーのバリエーションではないため、ExtendedKeyUsage 値の CSR 値は無視されます。

   > **Note**
   >
   > それでも、テンプレート定義は CSR の他の値 (件名 や サブジェクトの代替名 など) を上書きします。テンプレートが CSR パススルー型ではない場合でも、テンプレート定義が常に最優先されるため、これらの値は CSR から取得されます。

2. EndEntityClientAuthCertificate_APICSRPassthrough のテンプレート定義では、サブジェクト代替名 (SAN) 拡張子が API または CSR からコピーされるものとして定義されます。SAN 拡張が CSR で定義され、`IssueCertificate  ApiPassthrough` パラメータで指定されている場合、API パススルー値は CSR パススルー値よりも優先されるため、API パススルー値が優先されます。

テンプレートの定義

以下のセクションでは、サポートされている AWS Private CA 証明書テンプレートに関する設定の詳細について説明します。

BlankEndEntityCertificate_APIPassthrough /V1 定義

エンドエンティティ証明書テンプレートが空白の場合、X.509 基本制約のみが存在するエンドエンティティ証明書を発行できます。これは AWS Private CA が発行可能な最も単純なエンドエンティティ証明書ですが、API 構造を使用してカスタマイズできます。基本制約拡張は、証明書が CA 証明書かどうかを定義します。エンドエンティティ証明書テンプレートを空白にすると、CA 証明書ではなくエンドエンティティ証明書が確実に発行されるように、基本制約の値が FALSE になります。

空白のパススルーテンプレートを使用して、キー使用法 (KU) と拡張キー使用法 (EKU) に特定の値を必要とするスマートカード証明書を発行できます。たとえば、拡張キーの使用にはクライアント認証とスマートカードログオンが必要な場合があります。また、キーの使用にはデジタル署名、否認防止、および鍵の暗号化が必要な場合があります。他のパススルーテンプレートとは異なり、空白のエンドエンティティ証明書テンプレートでは KU および EKU 拡張機能の設定が可能です。ここで、KU はサポートされている 9 つの値

(digitalSignature 、nonRepudiation 、keyEncipherment 、dataEncipherment 、keyAgreement keyCertSign、cRLSign 、encipherOnly decipherOnly) のいずれかであり、EKU はサポートされている値 (serverAuth 、clientAuth 、codesigning、emailProtection、timetamping、OCSPSigning) とカスタム拡張のいずれかです。

BlankEndEntityCertificate_APIPassthrough/V1

X509v3 パラメータ	値
サブジェクトの代替名	[API または CSR からのパススルー]
件名	[API または CSR からのパススルー]
基本制約	CA:FALSE
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]
CRL ディストリビューションポイント *	[CA 設定からのパススルー]

* CRL ディストリビューションポイントは、CA で CRL 生成が有効に設定されている場合にのみテンプレートに含まれます。

BlankEndEntityCertificate_APICSRPassthrough /V1 定義

空白テンプレートの全般的な情報については、「BlankEndEntityCertificate_APIPassthrough /V1 定義」を参照してください。

BlankEndEntityCertificate_APICSRPassthrough/V1

X509v3 パラメータ	値
サブジェクトの代替名	[API または CSR からのパススルー]
件名	[API または CSR からのパススルー]
基本制約	CA:FALSE
権限キー識別子	[CA 証明書による SKI]

X509v3 パラメータ	値
サブジェクトキー識別子	[CSR から取得]
CRL ディストリビューションポイント *	[CA 設定または CSR からのパススルー]

* CRL ディストリビューションポイントは、CA で CRL 生成が有効に設定されている場合にのみテンプレートに含まれます。

BlankEndEntityCertificate_CriticalBasicConstraints_APICSRPassthrough /V1 定義

空白テンプレートの全般的な情報については、「BlankEndEntityCertificate_APIPassthrough /V1 定義」を参照してください。

BlankEndEntityCertificate_CriticalBasicConstraints_APICSRPassthrough /V1

X509v3 パラメータ	値
サブジェクトの代替名	[API または CSR からのパススルー]
件名	[API または CSR からのパススルー]
基本制約	重要、CA:FALSE
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]
CRL ディストリビューションポイント *	[CA 設定、API、または CSR からのパススルー]

* CRL ディストリビューションポイントは、CA で CRL 生成が有効に設定されている場合にのみテンプレートに含まれます。

BlankEndEntityCertificateCriticalBasicConstraints_APIPassthrough /V1 定義

空白テンプレートの全般的な情報については、「BlankEndEntityCertificate_APIPassthrough /V1 定義」を参照してください。

BlankEndEntityCertificateCriticalBasicConstraints__APIPassthrough/V1

X509v3 パラメータ	値
サブジェクトの代替名	[API または CSR からのパススルー]
件名	[API または CSR からのパススルー]
基本制約	重要、CA:FALSE
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]
CRL ディストリビューションポイント *	[CA 設定または API からのパススルー]

* CRL ディストリビューションポイントは、CA で CRL 生成が有効に設定されている場合にのみテンプレートに含まれます。

BlankEndEntityCertificateCriticalBasicConstraints_CSRPassthrough /V1 定義

空白テンプレートの全般的な情報については、「BlankEndEntityCertificate_APIPassthrough /V1 定義」を参照してください。

BlankEndEntityCertificateCriticalBasicConstraints__CSRPassthrough/V1

X509v3 パラメータ	値
サブジェクトの代替名	[API または CSR からのパススルー]
件名	[API または CSR からのパススルー]
基本制約	重要、CA:FALSE
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]
CRL ディストリビューションポイント *	[CA 設定または CSR からのパススルー]

* CRL ディストリビューションポイントは、CA で CRL 生成が有効に設定されている場合にのみテンプレートに含まれます。

BlankEndEntityCertificate_CSRPassthrough /V1 定義

空白テンプレートの全般的な情報については、「BlankEndEntityCertificate_APIPassthrough /V1 定義」を参照してください。

BlankEndEntityCertificate_CSRPassthrough/V1

X509v3 パラメータ	値
サブジェクトの代替名	[CSR からのパススルー]
件名	[CSR からのパススルー]
基本制約	CA:FALSE
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]
CRL ディストリビューションポイント *	[CA 設定または CSR からのパススルー]

* CRL ディストリビューションポイントは、CA で CRL 生成が有効に設定されている場合にのみテンプレートに含まれます。

BlankSubordinateCACertificate _PathLen0_CSRPassthrough /V1 定義

空白テンプレートの全般的な情報については、「BlankEndEntityCertificate_APIPassthrough /V1 定義」を参照してください。

BlankSubordinateCACertificate _PathLen0_CSRPassthrough /V1

X509v3 パラメータ	値
サブジェクトの代替名	[CSR からのパススルー]
件名	[CSR からのパススルー]
基本制約	重要、CA:TRUE、pathlen: 0

X509v3 パラメータ	値
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]
CRL ディストリビューションポイント *	[CA 設定または CSR からのパススルー]

* CRL ディストリビューションポイントは、CA で CRL 生成が有効に設定されている場合にのみテンプレートに含まれます。

BlankSubordinateCACertificate _PathLen0_APICSRPassthrough /V1 定義

空白テンプレートの全般的な情報については、「BlankEndEntityCertificate_APIPassthrough /V1 定義」を参照してください。

BlankSubordinateCACertificate _PathLen0_APICSRPassthrough /V1

X509v3 パラメータ	値
サブジェクトの代替名	[API または CSR からのパススルー]
件名	[API または CSR からのパススルー]
基本制約	重要、CA:TRUE、pathlen: 0
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]
CRL ディストリビューションポイント *	[CA 設定または CSR からのパススルー]

* CRL ディストリビューションポイントは、CA で CRL 生成が有効に設定されている場合にのみテンプレートに含まれます。

BlankSubordinateCACertificate _PathLen0_APIPassthrough /V1 定義

空白テンプレートの全般的な情報については、「BlankEndEntityCertificate_APIPassthrough /V1 定義」を参照してください。

BlankSubordinateCACertificate _PathLen0_APIPassthrough /V1

X509v3 パラメータ	値
サブジェクトの代替名	[API または CSR からのパススルー]
件名	[API または CSR からのパススルー]
基本制約	重要、CA:TRUE、pathlen: 0
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]
CRL ディストリビューションポイント *	[CA 設定からのパススルー]

BlankSubordinateCACertificate _PathLen1_APIPassthrough /V1 定義

空白テンプレートの全般的な情報については、「BlankEndEntityCertificate_APIPassthrough /V1 定義」を参照してください。

BlankSubordinateCACertificate _PathLen1_APIPassthrough /V1

X509v3 パラメータ	値
サブジェクトの代替名	[API または CSR からのパススルー]
件名	[API または CSR からのパススルー]
基本制約	重要、CA:TRUE、pathlen: 1
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]
CRL ディストリビューションポイント *	[CA 設定からのパススルー]

* CRL ディストリビューションポイントは、CA で CRL 生成が有効に設定されている場合にのみテンプレートに含まれます。

BlankSubordinateCACertificate _PathLen1_CSRPassthrough /V1 定義

空白テンプレートの全般的な情報については、「BlankEndEntityCertificate_APIPassthrough /V1 定義」を参照してください。

BlankSubordinateCACertificate _PathLen1_CSRPassthrough /V1

X509v3 パラメータ	値
サブジェクトの代替名	[CSR からのパススルー]
件名	[CSR からのパススルー]
基本制約	重要、CA:TRUE、pathlen: 1
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]
CRL ディストリビューションポイント *	[CA 設定または CSR からのパススルー]

* CRL ディストリビューションポイントは、CA で CRL 生成が有効に設定されている場合にのみテンプレートに含まれます。

BlankSubordinateCACertificate _PathLen1_APICSRPassthrough /V1 定義

空白テンプレートの全般的な情報については、「BlankEndEntityCertificate_APIPassthrough /V1 定義」を参照してください。

BlankSubordinateCACertificate _PathLen1_APICSRPassthrough /V1

X509v3 パラメータ	値
サブジェクトの代替名	[API または CSR からのパススルー]
件名	[API または CSR からのパススルー]
基本制約	重要、CA:TRUE、pathlen: 1
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]

X509v3 パラメータ	値
CRL ディストリビューションポイント *	[CA 設定または CSR からのパススルー]

* CRL ディストリビューションポイントは、CA で CRL 生成が有効に設定されている場合にのみテンプレートに含まれます。

BlankSubordinateCACertificate _PathLen2_APIPassthrough /V1 定義

空白テンプレートの全般的な情報については、「BlankEndEntityCertificate_APIPassthrough /V1 定義」を参照してください。

BlankSubordinateCACertificate _PathLen2_APIPassthrough /V1

X509v3 パラメータ	値
サブジェクトの代替名	[API または CSR からのパススルー]
件名	[API または CSR からのパススルー]
基本制約	重要、CA:TRUE、pathlen: 2
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]
CRL ディストリビューションポイント *	[CA 設定からのパススルー]

* CRL ディストリビューションポイントは、CA で CRL 生成が有効に設定されている場合にのみテンプレートに含まれます。

BlankSubordinateCACertificate _PathLen2_CSRPassthrough /V1 定義

空白テンプレートの全般的な情報については、「BlankEndEntityCertificate_APIPassthrough /V1 定義」を参照してください。

BlankSubordinateCACertificate _PathLen2_CSRPassthrough /V1

X509v3 パラメータ	値
サブジェクトの代替名	[CSR からのパススルー]
件名	[CSR からのパススルー]
基本制約	重要、CA:TRUE、pathlen: 2
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]
CRL ディストリビューションポイント *	[CA 設定または CSR からのパススルー]

* CRL ディストリビューションポイントは、CA で CRL 生成が有効に設定されている場合にのみテンプレートに含まれます。

BlankSubordinateCACertificate _PathLen2_APICSRPassthrough /V1 定義

空白テンプレートの全般的な情報については、「BlankEndEntityCertificate_APIPassthrough /V1 定義」を参照してください。

BlankSubordinateCACertificate _PathLen2_APICSRPassthrough /V1

X509v3 パラメータ	値
サブジェクトの代替名	[API または CSR からのパススルー]
件名	[API または CSR からのパススルー]
基本制約	重要、CA:TRUE、pathlen: 2
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]
CRL ディストリビューションポイント *	[CA 設定または CSR からのパススルー]

* CRL ディストリビューションポイントは、CA で CRL 生成が有効に設定されている場合にのみテンプレートに含まれます。

BlankSubordinateCACertificate _PathLen3_APIPassthrough /V1 定義

空白テンプレートの全般的な情報については、「BlankEndEntityCertificate_APIPassthrough /V1 定義」を参照してください。

BlankSubordinateCACertificate _PathLen3_APIPassthrough /V1

X509v3 パラメータ	値
サブジェクトの代替名	[API または CSR からのパススルー]
件名	[API または CSR からのパススルー]
基本制約	重要、CA:TRUE、pathlen: 3
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]
CRL ディストリビューションポイント *	[CA 設定からのパススルー]

* CRL ディストリビューションポイントは、CA で CRL 生成が有効に設定されている場合にのみテンプレートに含まれます。

BlankSubordinateCACertificate _PathLen3_CSRPassthrough /V1 定義

空白テンプレートの全般的な情報については、「BlankEndEntityCertificate_APIPassthrough /V1 定義」を参照してください。

BlankSubordinateCACertificate _PathLen3_CSRPassthrough /V1

X509v3 パラメータ	値
サブジェクトの代替名	[CSR からのパススルー]
件名	[CSR からのパススルー]
基本制約	重要、CA:TRUE、pathlen: 3

X509v3 パラメータ	値
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]
CRL ディストリビューションポイント *	[CA 設定または CSR からのパススルー]

* CRL ディストリビューションポイントは、CA で CRL 生成が有効に設定されている場合にのみテンプレートに含まれます。

BlankSubordinateCACertificate _PathLen3_APICSRPassthrough /V1 定義

空白テンプレートの全般的な情報については、「BlankEndEntityCertificate_APIPassthrough /V1 定義」を参照してください。

BlankSubordinateCACertificate _PathLen3_APICSRPassthrough

X509v3 パラメータ	値
サブジェクトの代替名	[API または CSR からのパススルー]
件名	[API または CSR からのパススルー]
基本制約	重要、CA:TRUE、pathlen: 3
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]
CRL ディストリビューションポイント *	[CA 設定または CSR からのパススルー]

* CRL ディストリビューションポイントは、CA で CRL 生成が有効に設定されている場合にのみテンプレートに含まれます。

CodeSigningCertificate/V1 定義

このテンプレートは、コード署名用の証明書を作成するために使用されます。AWS Private CA のコード署名証明書は、プライベート CA インフラストラクチャに基づくコード署名ソリューションで使用できます。たとえば、AWS Private CA のコード署名を使用している顧客は、AWS IoT でコード

署名証明書を生成し、AWS Certificate Manager にインポートできます。詳細については、「 のコード署名AWS IoTとは」および「コード署名証明書を取得してインポートする」を参照してください。

CodeSigningCertificate/V1

X509v3 パラメータ	値
サブジェクトの代替名	[CSR からのパススルー]
件名	[CSR からのパススルー]
基本制約	CA:FALSE
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]
キーの用途	重要、デジタル署名
拡張キーの用途	重要、コード署名
CRL ディストリビューションポイント *	[CA 設定からのパススルー]

*CRL ディストリビューションポイントは、CA で CRL 生成が有効に設定されている場合にのみテンプレートに含まれます。

CodeSigningCertificate_APICSRPassthrough /V1 定義

このテンプレートは CodeSigningCertificate/V1 を拡張して、API と CSR のパススルー値をサポートします。

CodeSigningCertificate_APICSRPassthrough/V1

X509v3 パラメータ	値
サブジェクトの代替名	[API または CSR からのパススルー]
件名	[API または CSR からのパススルー]
基本制約	CA:FALSE
権限キー識別子	[CA 証明書による SKI]

X509v3 パラメータ	値
サブジェクトキー識別子	[CSR から取得]
キーの用途	重要、デジタル署名
拡張キーの用途	重要、コード署名
CRL ディストリビューションポイント *	[CA 設定または CSR からのパススルー]

* CRL ディストリビューションポイントは、CA で CRL 生成が有効に設定されている場合にのみテンプレートに含まれます。

CodeSigningCertificate_APIPassthrough /V1 定義

このテンプレートは `CodeSigningCertificate` テンプレートとほぼ同じですが、違いが 1 つあります。このテンプレートでは、拡張がテンプレートで指定されていない場合、AWS Private CA は API (CSR) を通して証明書に追加の拡張を渡します。テンプレートで指定された拡張は、常に API 内の拡張を上書きします。

CodeSigningCertificate_APIPassthrough/V1

X509v3 パラメータ	値
サブジェクトの代替名	[API または CSR からのパススルー]
件名	[API または CSR からのパススルー]
基本制約	`CA:FALSE`
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]
キーの用途	重要、デジタル署名
拡張キーの用途	重要、コード署名
CRL ディストリビューションポイント *	[CA 設定からのパススルー]

* CRL ディストリビューションポイントは、CA で CRL 生成が有効に設定されている場合にのみテンプレートに含まれます。

CodeSigningCertificate_CSRPassthrough /V1 定義

このテンプレートは `CodeSigningCertificate` テンプレートと同じですが、違いが 1 つあります。このテンプレートでは、拡張機能がテンプレートで指定されていない場合、AWS Private CA は証明書署名リクエスト (CSR) から証明書に追加の拡張機能を渡します。テンプレートで指定された拡張機能は、常に CSR 内の拡張機能を上書きします。

CodeSigningCertificate_CSRPassthrough/V1

X509v3 パラメータ	値
サブジェクトの代替名	[CSR からのパススルー]
件名	[CSR からのパススルー]
基本制約	`CA:FALSE`
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]
キーの用途	重要、デジタル署名
拡張キーの用途	重要、コード署名
CRL ディストリビューションポイント *	[CA 設定または CSR からのパススルー]

*CRL ディストリビューションポイントは、CA で CRL 生成が有効に設定されている場合にのみテンプレートに含まれます。

EndEntityCertificate/V1 定義

このテンプレートは、オペレーティングシステムやウェブサーバーなどのエンドエンティティの証明書を作成するために使用されます。

EndEntityCertificate/V1

X509v3 パラメータ	値
サブジェクトの代替名	[CSR からのパススルー]
件名	[CSR からのパススルー]
基本制約	CA:FALSE
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]
キーの用途	重要、デジタル署名、キー暗号化
拡張キーの用途	TLS ウェブサーバー認証、TLS ウェブクライアント認証
CRL ディストリビューションポイント *	[CA 設定からのパススルー]

*CRL ディストリビューションポイントは、CA で CRL 生成が有効に設定されている場合にのみテンプレートに含まれます。

EndEntityCertificate_APICSRPassthrough /V1 定義

このテンプレートは EndEntityCertificate/V1 を拡張して、API と CSR のパススルー値をサポートします。

EndEntityCertificate_APICSRPassthrough/V1

X509v3 パラメータ	値
サブジェクトの代替名	[API または CSR からのパススルー]
件名	[API または CSR からのパススルー]
基本制約	CA:FALSE
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]

X509v3 パラメータ	値
キーの用途	重要、デジタル署名、キー暗号化
拡張キーの用途	TLS ウェブサーバー認証、TLS ウェブクライアント認証
CRL ディストリビューションポイント *	[CA 設定または CSR からのパススルー]

* CRL ディストリビューションポイントは、CA で CRL 生成が有効に設定されている場合にのみテンプレートに含まれます。

EndEntityCertificate_APIPassthrough /V1 定義

このテンプレートは `EndEntityCertificate` テンプレートとほぼ同じですが、違いが 1 つあります。このテンプレートでは、拡張がテンプレートで指定されていない場合、AWS Private CA は API (CSR) を通して証明書に追加の拡張を渡します。テンプレートで指定された拡張は、常に API 内の拡張を上書きします。

EndEntityCertificate_APIPassthrough/V1

X509v3 パラメータ	値
サブジェクトの代替名	[API または CSR からのパススルー]
件名	[API または CSR からのパススルー]
基本制約	CA:FALSE
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]
キーの用途	重要、デジタル署名、キー暗号化
拡張キーの用途	TLS ウェブサーバー認証、TLS ウェブクライアント認証
CRL ディストリビューションポイント *	[CA 設定からのパススルー]

* CRL ディストリビューションポイントは、CA で CRL 生成が有効に設定されている場合にのみテンプレートに含まれます。

EndEntityCertificate_CSRPassthrough /V1 定義

このテンプレートは EndEntityCertificate テンプレートと同じですが、違いが 1 つあります。このテンプレートでは、拡張機能がテンプレートで指定されていない場合、AWS Private CA は証明書署名リクエスト (CSR) から証明書に追加の拡張機能を渡します。テンプレートで指定された拡張機能は、常に CSR 内の拡張機能を上書きします。

EndEntityCertificate_CSRPassthrough/V1

X509v3 パラメータ	値
サブジェクトの代替名	[CSR からのパススルー]
件名	[CSR からのパススルー]
基本制約	CA:FALSE
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]
キーの用途	重要、デジタル署名、キー暗号化
拡張キーの用途	TLS ウェブサーバー認証、TLS ウェブクライアント認証
CRL ディストリビューションポイント *	[CA 設定または CSR からのパススルー]

*CRL ディストリビューションポイントは、CA で CRL 生成が有効に設定されている場合にのみテンプレートに含まれます。

EndEntityClientAuthCertificate/V1 定義

このテンプレートは EndEntityCertificate とほぼ同じですが、唯一違う点は拡張キー使用値で、TLS ウェブクライアント認証に制限されます。

EndEntityClientAuthCertificate/V1

X509v3 パラメータ	値
サブジェクトの代替名	[CSR からのパススルー]
件名	[CSR からのパススルー]
基本制約	CA:FALSE
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]
キーの用途	重要、デジタル署名、キー暗号化
拡張キーの用途	TLS ウェブクライアント認証
CRL ディストリビューションポイント *	[CA 設定または CSR からのパススルー]

*CRL ディストリビューションポイントは、CA で CRL 生成が有効に設定されている場合にのみテンプレートに含まれます。

EndEntityClientAuthCertificate_APICSRPassthrough /V1 定義

このテンプレートは EndEntityClientAuthCertificate/V1 を拡張して、API と CSR のパススルー値をサポートします。

EndEntityClientAuthCertificate_APICSRPassthrough/V1

X509v3 パラメータ	値
サブジェクトの代替名	[API または CSR からのパススルー]
件名	[API または CSR からのパススルー]
基本制約	CA:FALSE
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]

X509v3 パラメータ	値
キーの用途	重要、デジタル署名、キー暗号化
拡張キーの用途	TLS ウェブクライアント認証
CRL ディストリビューションポイント *	[CA 設定または CSR からのパススルー]

* CRL ディストリビューションポイントは、CA で CRL 生成が有効に設定されている場合にのみテンプレートに含まれます。

EndEntityClientAuthCertificate_APIPassthrough /V1 定義

このテンプレートは `EndEntityClientAuthCertificate` テンプレートと同じですが、 違いが 1 つあります。このテンプレートでは、拡張がテンプレートで指定されていない場合、AWS Private CA はAPI を通して証明書に追加の拡張を渡します。テンプレートで指定された拡張は、常に API 内の拡張を上書きします。

EndEntityClientAuthCertificate_APIPassthrough/V1

X509v3 パラメータ	値
サブジェクトの代替名	[API または CSR からのパススルー]
件名	[API または CSR からのパススルー]
基本制約	CA:FALSE
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]
キーの用途	重要、デジタル署名、キー暗号化
拡張キーの用途	TLS ウェブクライアント認証
CRL ディストリビューションポイント *	[CA 設定からのパススルー]

* CRL ディストリビューションポイントは、CA で CRL 生成が有効に設定されている場合にのみテンプレートに含まれます。

EndEntityClientAuthCertificate_CSRPassthrough /V1 定義

このテンプレートは EndEntityClientAuthCertificate テンプレートと同じですが、 違いが 1 つあります。このテンプレートでは、拡張機能がテンプレートで指定されていない場合、AWS Private CA は証明書署名リクエスト (CSR) から証明書に追加の拡張機能を渡します。テンプレートで指定された拡張機能は、常に CSR 内の拡張機能を上書きします。

EndEntityClientAuthCertificate_CSRPassthrough/V1

X509v3 パラメータ	値
サブジェクトの代替名	[CSR からのパススルー]
件名	[CSR からのパススルー]
基本制約	CA:FALSE
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]
キーの用途	重要、デジタル署名、キー暗号化
拡張キーの用途	TLS ウェブクライアント認証
CRL ディストリビューションポイント *	[CA 設定または CSR からのパススルー]

*CRL ディストリビューションポイントは、CA で CRL 生成が有効に設定されている場合にのみテンプレートに含まれます。

EndEntityServerAuthCertificate/V1 定義

このテンプレートは EndEntityCertificate とほぼ同じですが、唯一違う点は拡張キー使用法の値で、TLS ウェブサーバー認証に制限されます。

EndEntityServerAuthCertificate/V1

X509v3 パラメータ	値
サブジェクトの代替名	[CSR からのパススルー]
件名	[CSR からのパススルー]

X509v3 パラメータ	値
基本制約	CA:FALSE
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]
キーの用途	重要、デジタル署名、キー暗号化
拡張キーの用途	TLS ウェブサーバー認証
CRL ディストリビューションポイント *	[CA 設定からのパススルー]

*CRL ディストリビューションポイントは、CA で CRL 生成が有効に設定されている場合にのみテンプレートに含まれます。

EndEntityServerAuthCertificate_APICSRPassthrough /V1 定義

このテンプレートは EndEntityServerAuthCertificate/V1 を拡張して、API と CSR のパススルー値をサポートします。

EndEntityServerAuthCertificate_APICSRPassthrough/V1

X509v3 パラメータ	値
サブジェクトの代替名	[API または CSR からのパススルー]
件名	[API または CSR からのパススルー]
基本制約	CA:FALSE
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]
キーの用途	重要、デジタル署名、キー暗号化
拡張キーの用途	TLS ウェブサーバー認証
CRL ディストリビューションポイント *	[CA 設定または CSR からのパススルー]

* CRL ディストリビューションポイントは、CA で CRL 生成が有効に設定されている場合にのみテンプレートに含まれます。

EndEntityServerAuthCertificate_APIPassthrough /V1 定義

このテンプレートは EndEntityServerAuthCertificate テンプレートと同じですが、 違いが 1 つあります。このテンプレートでは、拡張がテンプレートで指定されていない場合、AWS Private CA はAPI を通して証明書に追加の拡張を渡します。テンプレートで指定された拡張は、常に API 内の拡張を上書きします。

EndEntityServerAuthCertificate_APIPassthrough/V1

X509v3 パラメータ	値
サブジェクトの代替名	[API または CSR からのパススルー]
件名	[API または CSR からのパススルー]
基本制約	CA:FALSE
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]
キーの用途	重要、デジタル署名、キー暗号化
拡張キーの用途	TLS ウェブサーバー認証
CRL ディストリビューションポイント *	[CA 設定からのパススルー]

* CRL ディストリビューションポイントは、CA で CRL 生成が有効に設定されている場合にのみテンプレートに含まれます。

EndEntityServerAuthCertificate_CSRPassthrough /V1 定義

このテンプレートは EndEntityServerAuthCertificate テンプレートと同じですが、 違いが 1 つあります。このテンプレートでは、拡張機能がテンプレートで指定されていない場合、AWS Private CA は証明書署名リクエスト (CSR) から証明書に追加の拡張機能を渡します。テンプレートで指定された拡張機能は、常に CSR 内の拡張機能を上書きします。

EndEntityServerAuthCertificate_CSRPassthroughV1

X509v3 パラメータ	値
サブジェクトの代替名	[CSR からのパススルー]
件名	[CSR からのパススルー]
基本制約	CA:FALSE
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]
キーの用途	重要、デジタル署名、キー暗号化
拡張キーの用途	TLS ウェブサーバー認証
CRL ディストリビューションポイント *	[CA 設定または CSR からのパススルー]

*CRL ディストリビューションポイントは、CA で CRL 生成が有効に設定されている場合にのみテンプレートに含まれます。

OCSPSigningCertificate/V1 定義

このテンプレートは、OCSP 応答に署名するための証明書を作成するために使用されます。テンプレートは `CodeSigningCertificate` テンプレートと同じですが、拡張キー使用法の値がコード署名ではなく OCSP 署名を指定している点が異なります。

OCSPSigningCertificate/V1

X509v3 パラメータ	値
サブジェクトの代替名	[CSR からのパススルー]
件名	[CSR からのパススルー]
基本制約	CA:FALSE
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]

X509v3 パラメータ	値
キーの用途	重要、デジタル署名
拡張キーの用途	重要、OCSP 署名
CRL ディストリビューションポイント *	[CA 設定からのパススルー]

*CRL ディストリビューションポイントは、CA で CRL 生成が有効に設定されている場合にのみテンプレートに含まれます。

OCSP SigningCertificate_APICSRPassthrough /V1 定義

このテンプレートは、OCSP SigningCertificate/V1 を拡張して API および CSR パススルー値をサポートします。

OCSP SigningCertificate_APICSRPassthrough /V1

X509v3 パラメータ	値
サブジェクトの代替名	[API または CSR からのパススルー]
件名	[API または CSR からのパススルー]
基本制約	`CA:FALSE`
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]
キーの用途	重要、デジタル署名
拡張キーの用途	重要、OCSP 署名
CRL ディストリビューションポイント *	[CA 設定または CSR からのパススルー]

* CRL ディストリビューションポイントは、CA で CRL 生成が有効に設定されている場合にのみテンプレートに含まれます。

OCSP SigningCertificate_APIPassthrough /V1 定義

このテンプレートは OCSPSigningCertificate テンプレートと同じですが、 違いが 1 つあります。このテンプレートでは、拡張がテンプレートで指定されていない場合、AWS Private CA はAPI を通して証明書に追加の拡張を渡します。テンプレートで指定された拡張は、常に API 内の拡張を上書きします。

OCSP SigningCertificate_APIPassthrough /V1

X509v3 パラメータ	値
サブジェクトの代替名	[API または CSR からのパススルー]
件名	[API または CSR からのパススルー]
基本制約	CA:FALSE
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]
キーの用途	重要、デジタル署名
拡張キーの用途	重要、OCSP 署名
CRL ディストリビューションポイント *	[CA 設定からのパススルー]

* CRL ディストリビューションポイントは、CA で CRL 生成が有効に設定されている場合にのみテンプレートに含まれます。

OCSP SigningCertificate_CSRPassthrough /V1 定義

このテンプレートは OCSPSigningCertificate テンプレートと同じですが、 違いが 1 つあります。このテンプレートでは、拡張機能がテンプレートで指定されていない場合、AWS Private CA は証明書署名リクエスト (CSR) から証明書に追加の拡張機能を渡します。テンプレートで指定された拡張機能は、常に CSR 内の拡張機能を上書きします。

OCSP SigningCertificate_CSRPassthrough /V1

X509v3 パラメータ	値
サブジェクトの代替名	[CSR からのパススルー]
件名	[CSR からのパススルー]
基本制約	CA:FALSE
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]
キーの用途	重要、デジタル署名
拡張キーの用途	重要、OCSP 署名
CRL ディストリビューションポイント *	[CA 設定または CSR からのパススルー]

*CRL ディストリビューションポイントは、CA で CRL 生成が有効に設定されている場合にのみテンプレートに含まれます。

RootCACertificate/V1 定義

このテンプレートは、自己署名ルート CA 認定を交付するために使用されます。CA 認定には、証明書が CA 認定の交付に使用できることを指定するために、[CA] フィールドが TRUE に設定された、重要な基本制約拡張機能が含まれています。テンプレートではパスの長さ (pathLenConstraint) を指定しません。これは、階層の今後の拡張が妨げられる可能性があるためです。CA 認定を TLS クライアントまたはサーバー証明書として使用できないようにするため、拡張キーの使用は除外されます。自己署名証明書を取り消すことができないため、CRL 情報は指定されません。

RootCACertificate/V1

X509v3 パラメータ	値
サブジェクトの代替名	[CSR からのパススルー]
件名	[CSR からのパススルー]
基本制約	重要、CA:TRUE

X509v3 パラメータ	値
サブジェクトキー識別子	[CSR から取得]
キーの用途	重要、デジタル署名 keyCertSign、CRL 記号
CRL ディストリビューションポイント	該当なし

RootCACertificate_APIPassthrough/V1 定義

このテンプレートは RootCACertificate/V1 を拡張して API パススルー値をサポートします。

RootCACertificate_APIPassthrough/V1

X509v3 パラメータ	値
サブジェクトの代替名	[API または CSR からのパススルー]
件名	[API または CSR からのパススルー]
基本制約	重要、CA:TRUE
権限キー識別子	[API からのパススルー]
サブジェクトキー識別子	[CSR から取得]
キーの用途	重要、デジタル署名 keyCertSign、CRL 記号
CRL ディストリビューションポイント *	該当なし

BlankRootCACertificate _APIPassthrough /V1 定義

空のルート証明書テンプレートを使用すると、X.509 の基本制約のみを含むルート証明書を発行できます。これは が発行AWS Private CAできる最も簡単なルート証明書ですが、 API 構造を使用してカスタマイズできます。基本的な制約拡張は、証明書が CA 証明書であるかどうかを定義します。ルート証明書テンプレートを空白にすると、ルート CA 証明書が確実に発行されるように、基本的な制約TRUEとして の値が強制されます。

空のパススルールートテンプレートを使用して、キー使用法 (KU) に特定の値を
必要とするルート証明書を発行できます。例えば、キーの使用には、 ではなく

cRLSignkeyCertSignと が必要になる場合がありますdigitalSignature。空白以外のルートパススルー証明書テンプレートとは異なり、空白のルート証明書テンプレートでは KU 拡張機能を設定できます。ここで、KU はサポートされている 9 つの値 (digitalSignature、、nonRepudiation、keyEnciphermentdataEncipherment、、cRLSignenc keyAgreement keyCertSignのいずれかになりますdecipherOnly。

BlankRootCACertificate _APIPassthrough /V1

X509v3 パラメータ	値
サブジェクトの代替名	[API または CSR からのパススルー]
件名	[API または CSR からのパススルー]
基本制約	重要、CA:TRUE
サブジェクトキー識別子	[CSR から取得]

BlankRootCACertificate _PathLen0_APIPassthrough /V1 定義

空白のルート CA テンプレートの一般的な情報については、「」を参照してください???。

BlankRootCACertificate _PathLen0_APIPassthrough /V1

X509v3 パラメータ	値
サブジェクトの代替名	[API または CSR からのパススルー]
件名	[API または CSR からのパススルー]
基本制約	重要、CA:TRUE、pathlen: 0
サブジェクトキー識別子	[CSR から取得]

BlankRootCACertificate _PathLen1_APIPassthrough /V1 定義

空白のルート CA テンプレートの一般的な情報については、「」を参照してください???。

BlankRootCACertificate _PathLen1_APIPassthrough /V1

X509v3 パラメータ	値
サブジェクトの代替名	[API または CSR からのパススルー]
件名	[API または CSR からのパススルー]
基本制約	重要、CA:TRUE、pathlen: 1
サブジェクトキー識別子	[CSR から取得]

BlankRootCACertificate _PathLen2_APIPassthrough /V1 定義

空白のルート CA テンプレートの一般的な情報については、「」を参照してください???。

BlankRootCACertificate _PathLen2_APIPassthrough /V1

X509v3 パラメータ	値
サブジェクトの代替名	[API または CSR からのパススルー]
件名	[API または CSR からのパススルー]
基本制約	重要、CA:TRUE、pathlen: 2
サブジェクトキー識別子	[CSR から取得]

BlankRootCACertificate _PathLen3_APIPassthrough /V1 定義

空白のルート CA テンプレートの一般的な情報については、「」を参照してください???。

BlankRootCACertificate _PathLen3_APIPassthrough /V1

X509v3 パラメータ	値
サブジェクトの代替名	[API または CSR からのパススルー]
件名	[API または CSR からのパススルー]
基本制約	重要、CA:TRUE、pathlen: 3

X509v3 パラメータ	値
サブジェクトキー識別子	[CSR から取得]

SubordinateCACertificate _PathLen0/V1 定義

このテンプレートは、パス長が の下位 CA 証明書を発行するために使用されます0。CA 認定には、証明書が CA 認定の交付に使用できることを指定するために、[CA] フィールドが TRUE に設定された、重要な基本制約拡張機能が含まれています。拡張キーの使用法が含まれていないため、CA 認定が TLS クライアントまたはサーバー証明書として使用されなくなります。

証明パスの詳細については、「証明パスに長さの制約を設定する」を参照してください。

SubordinateCACertificate _PathLen0/V1

X509v3 パラメータ	値
サブジェクトの代替名	[CSR からのパススルー]
件名	[CSR からのパススルー]
基本制約	重要、CA:TRUE、pathlen: 0
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]
キーの用途	重要、デジタル署名、keyCertSign 、CRL 署名
CRL ディストリビューションポイント *	[CA 設定からのパススルー]

*CRL ディストリビューションポイントは、CA が CRL 生成を有効にして設定されている場合にのみ、このテンプレートで発行される証明書に含まれます。

SubordinateCACertificate _PathLen0_APICSRPassthrough /V1 定義

このテンプレートは SubordinateCACertificate _PathLen0/V1 を拡張して、API および CSR パススルー値をサポートします。

SubordinateCACertificate _PathLen0_APICSRPassthrough /V1

X509v3 パラメータ	値
サブジェクトの代替名	[API または CSR からのパススルー]
件名	[API または CSR からのパススルー]
基本制約	重要、CA:TRUE、pathlen: 0
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]
キーの用途	重要、デジタル署名、keyCertSign 、CRL 署名
CRL ディストリビューションポイント *	[CA 設定または CSR からのパススルー]

* CRL ディストリビューションポイントは、CA で CRL 生成が有効に設定されている場合にのみテンプレートに含まれます。

SubordinateCACertificate _PathLen0_APIPassthrough /V1 定義

このテンプレートは SubordinateCACertificate _PathLen0/V1 を拡張して API パススルー値をサポートします。

SubordinateCACertificate _PathLen0_APIPassthrough /V1

X509v3 パラメータ	値
サブジェクトの代替名	[API または CSR からのパススルー]
件名	[API または CSR からのパススルー]
基本制約	重要、CA:TRUE、pathlen: 0
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]

X509v3 パラメータ	値
キーの用途	重要、デジタル署名、keyCertSign 、CRL 署名
CRL ディストリビューションポイント *	[CA 設定からのパススルー]

* CRL ディストリビューションポイントは、CA で CRL 生成が有効に設定されている場合にのみテンプレートに含まれます。

SubordinateCACertificate _PathLen0_CSRPassthrough /V1 定義

このテンプレートは SubordinateCACertificate_PathLen0 テンプレートと同じですが、違いが 1 つあります。このテンプレートでは、拡張機能がテンプレートで指定されていない場合、AWS Private CA は証明書署名リクエスト (CSR) から証明書に追加の拡張機能を渡します。テンプレートで指定された拡張機能は、常に CSR 内の拡張機能を上書きします。

> **Note**
>
> カスタムの追加拡張を含む CSR は、AWS Private CA の外部で作成する必要があります。

SubordinateCACertificate _PathLen0_CSRPassthrough /V1

X509v3 パラメータ	値
サブジェクトの代替名	[CSR からのパススルー]
件名	[CSR からのパススルー]
基本制約	重要、CA:TRUE、pathlen: 0
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]
キーの用途	重要、デジタル署名、keyCertSign 、CRL 署名
CRL ディストリビューションポイント *	[CA 設定または CSR からのパススルー]

*CRL ディストリビューションポイントは、CA が CRL 生成を有効にして構成されている場合にのみ、このテンプレートで発行された証明書に含まれます。

SubordinateCACertificate _PathLen1/V1 定義

このテンプレートは、パス長が の下位 CA 証明書を発行するために使用されます1。CA 認定には、証明書が CA 認定の交付に使用できることを指定するために、[CA] フィールドが TRUE に設定された、重要な基本制約拡張機能が含まれています。拡張キーの使用法が含まれていないため、CA 認定が TLS クライアントまたはサーバー証明書として使用されなくなります。

証明パスの詳細については、「証明パスに長さの制約を設定する」を参照してください。

SubordinateCACertificate _PathLen1/V1

X509v3 パラメータ	値
サブジェクトの代替名	[CSR からのパススルー]
件名	[CSR からのパススルー]
基本制約	重要、CA:TRUE、pathlen: 1
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]
キーの用途	重要、デジタル署名、keyCertSign 、CRL 署名
CRL ディストリビューションポイント *	[CA 設定からのパススルー]

*CRL ディストリビューションポイントは、CA が CRL 生成を有効にして構成されている場合にのみ、このテンプレートで発行された証明書に含まれます。

SubordinateCACertificate _PathLen1_APICSRPassthrough /V1 定義

このテンプレートは SubordinateCACertificate _PathLen1/V1 を拡張して、API および CSR パススルー値をサポートします。

SubordinateCACertificate _PathLen1_APICSRPassthrough /V1

X509v3 パラメータ	値
サブジェクトの代替名	[API または CSR からのパススルー]
件名	[API または CSR からのパススルー]
基本制約	重要、CA:TRUE、pathlen: 1
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]
キーの用途	重要、デジタル署名、keyCertSign 、CRL 署名
CRL ディストリビューションポイント *	[CA 設定または CSR からのパススルー]

* CRL ディストリビューションポイントは、CA で CRL 生成が有効に設定されている場合にのみテンプレートに含まれます。

SubordinateCACertificate _PathLen1_APIPassthrough /V1 定義

このテンプレートは SubordinateCACertificate _PathLen0/V1 を拡張して API パススルー値をサポートします。

SubordinateCACertificate _PathLen1_APIPassthrough /V1

X509v3 パラメータ	値
サブジェクトの代替名	[API または CSR からのパススルー]
件名	[API または CSR からのパススルー]
基本制約	重要、CA:TRUE、pathlen: 1
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]

X509v3 パラメータ	値
キーの用途	重要、デジタル署名、keyCertSign 、CRL 署名
CRL ディストリビューションポイント *	[CA 設定からのパススルー]

* CRL ディストリビューションポイントは、CA で CRL 生成が有効に設定されている場合にのみテンプレートに含まれます。

SubordinateCACertificate _PathLen1_CSRPassthrough /V1 定義

このテンプレートは SubordinateCACertificate_PathLen1 テンプレートと同じですが、違いが 1 つあります。このテンプレートでは、拡張機能がテンプレートで指定されていない場合、AWS Private CA は証明書署名リクエスト (CSR) から証明書に追加の拡張機能を渡します。テンプレートで指定された拡張機能は、常に CSR 内の拡張機能を上書きします。

> **Note**
>
> カスタムの追加拡張を含む CSR は、AWS Private CA の外部で作成する必要があります。

SubordinateCACertificate _PathLen1_CSRPassthrough /V1

X509v3 パラメータ	値
サブジェクトの代替名	[CSR からのパススルー]
件名	[CSR からのパススルー]
基本制約	重要、CA:TRUE、pathlen: 1
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]
キーの用途	重要、デジタル署名、keyCertSign 、CRL 署名
CRL ディストリビューションポイント *	[CA 設定または CSR からのパススルー]

*CRL ディストリビューションポイントは、CA が CRL 生成を有効にして構成されている場合にのみ、このテンプレートで発行された証明書に含まれます。

SubordinateCACertificate _PathLen2/V1 定義

このテンプレートは、パスの長さが 2 の下位 CA 認定を交付するために使用されます。CA 認定には、証明書が CA 認定の交付に使用できることを指定するために、[CA] フィールドが TRUE に設定された、重要な基本制約拡張機能が含まれています。拡張キーの使用法が含まれていないため、CA 認定が TLS クライアントまたはサーバー証明書として使用されなくなります。

証明パスの詳細については、「証明パスに長さの制約を設定する」を参照してください。

SubordinateCACertificate _PathLen2/V1

X509v3 パラメータ	値
サブジェクトの代替名	[CSR からのパススルー]
件名	[CSR からのパススルー]
基本制約	重要、CA:TRUE、pathlen: 2
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]
キーの用途	重要、デジタル署名、keyCertSign 、CRL 署名
CRL ディストリビューションポイント *	[CA 設定からのパススルー]

*CRL ディストリビューションポイントは、CA が CRL 生成を有効にして構成されている場合にのみ、このテンプレートで発行された証明書に含まれます。

SubordinateCACertificate _PathLen2_APICSRPassthrough /V1 定義

このテンプレートは SubordinateCACertificate _PathLen2/V1 を拡張して、API と CSR のパススルー値をサポートします。

SubordinateCACertificate _PathLen2_APICSRPassthrough /V1

X509v3 パラメータ	値
サブジェクトの代替名	[API または CSR からのパススルー]
件名	[API または CSR からのパススルー]
基本制約	重要、CA:TRUE、pathlen: 2
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]
キーの用途	重要、デジタル署名、keyCertSign 、CRL 署名
CRL ディストリビューションポイント *	[CA 設定または CSR からのパススルー]

* CRL ディストリビューションポイントは、CA で CRL 生成が有効に設定されている場合にのみテンプレートに含まれます。

SubordinateCACertificate _PathLen2_APIPassthrough /V1 定義

このテンプレートは SubordinateCACertificate _PathLen2/V1 を拡張して API パススルー値をサポートします。

SubordinateCACertificate _PathLen2_APIPassthrough /V1

X509v3 パラメータ	値
サブジェクトの代替名	[API または CSR からのパススルー]
件名	[API または CSR からのパススルー]
基本制約	重要、CA:TRUE、pathlen: 2
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]

X509v3 パラメータ	値
キーの用途	重要、デジタル署名、keyCertSign 、CRL 署名
CRL ディストリビューションポイント *	[CA 設定からのパススルー]

* CRL ディストリビューションポイントは、CA で CRL 生成が有効に設定されている場合にのみテンプレートに含まれます。

SubordinateCACertificate _PathLen2_CSRPassthrough /V1 定義

このテンプレートは SubordinateCACertificate_PathLen2 テンプレートと同じですが、違いが 1 つあります。このテンプレートでは、拡張機能がテンプレートで指定されていない場合、AWS Private CA は証明書署名リクエスト (CSR) から証明書に追加の拡張機能を渡します。テンプレートで指定された拡張機能は、常に CSR 内の拡張機能を上書きします。

> **Note**
>
> カスタムの追加拡張を含む CSR は、AWS Private CA の外部で作成する必要があります。

SubordinateCACertificate _PathLen2_CSRPassthrough /V1

X509v3 パラメータ	値
サブジェクトの代替名	[CSR からのパススルー]
件名	[CSR からのパススルー]
基本制約	重要、CA:TRUE、pathlen: 2
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]
キーの用途	重要、デジタル署名、keyCertSign 、CRL 署名
CRL ディストリビューションポイント *	[CA 設定または CSR からのパススルー]

*CRL ディストリビューションポイントは、CA が CRL 生成を有効にして構成されている場合にのみ、このテンプレートで発行された証明書に含まれます。

SubordinateCACertificate _PathLen3/V1 定義

このテンプレートは、パスの長さが 3 の下位 CA 認定を交付するために使用されます。CA 認定には、証明書が CA 認定の交付に使用できることを指定するために、[CA] フィールドが TRUE に設定された、重要な基本制約拡張機能が含まれています。拡張キーの使用法が含まれていないため、CA 認定が TLS クライアントまたはサーバー証明書として使用されなくなります。

証明パスの詳細については、「証明パスに長さの制約を設定する」を参照してください。

SubordinateCACertificate _PathLen3/V1

X509v3 パラメータ	値
サブジェクトの代替名	[CSR からのパススルー]
件名	[CSR からのパススルー]
基本制約	重要、CA:TRUE、pathlen: 3
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]
キーの用途	重要、デジタル署名、keyCertSign 、CRL 署名
CRL ディストリビューションポイント *	[CA 設定からのパススルー]

*CRL ディストリビューションポイントは、CA が CRL 生成を有効にして構成されている場合にのみ、このテンプレートで発行された証明書に含まれます。

SubordinateCACertificate _PathLen3_APICSRPassthrough /V1 定義

このテンプレートは SubordinateCACertificate _PathLen3/V1 を拡張して、API および CSR パススルー値をサポートします。

SubordinateCACertificate _PathLen3_APICSRPassthrough /V1

X509v3 パラメータ	値
サブジェクトの代替名	[API または CSR からのパススルー]
件名	[API または CSR からのパススルー]
基本制約	重要、CA:TRUE、pathlen: 3
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]
キーの用途	重要、デジタル署名、keyCertSign 、CRL 署名
CRL ディストリビューションポイント *	[CA 設定または CSR からのパススルー]

* CRL ディストリビューションポイントは、CA で CRL 生成が有効に設定されている場合にのみテンプレートに含まれます。

SubordinateCACertificate _PathLen3_APIPassthrough /V1 定義

このテンプレートは SubordinateCACertificate _PathLen3/V1 を拡張して API パススルー値をサポートします。

SubordinateCACertificate _PathLen3_APIPassthrough /V1

X509v3 パラメータ	値
サブジェクトの代替名	[API または CSR からのパススルー]
件名	[API または CSR からのパススルー]
基本制約	重要、CA:TRUE、pathlen: 3
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]

X509v3 パラメータ	値
キーの用途	重要、デジタル署名、keyCertSign 、CRL 署名
CRL ディストリビューションポイント *	[CA 設定からのパススルー]

* CRL ディストリビューションポイントは、CA で CRL 生成が有効に設定されている場合にのみテンプレートに含まれます。

SubordinateCACertificate _PathLen3_CSRPassthrough /V1 定義

このテンプレートは SubordinateCACertificate_PathLen3 テンプレートと同じですが、違いが 1 つあります。このテンプレートでは、拡張機能がテンプレートで指定されていない場合、AWS Private CA は証明書署名リクエスト (CSR) から証明書に追加の拡張機能を渡します。テンプレートで指定された拡張機能は、常に CSR 内の拡張機能を上書きします。

> **Note**
>
> カスタムの追加拡張を含む CSR は、AWS Private CA の外部で作成する必要があります。

SubordinateCACertificate _PathLen3_CSRPassthrough /V1

X509v3 パラメータ	値
サブジェクトの代替名	[CSR からのパススルー]
件名	[CSR からのパススルー]
基本制約	重要、CA:TRUE、pathlen: 3
権限キー識別子	[CA 証明書による SKI]
サブジェクトキー識別子	[CSR から取得]
キーの用途	重要、デジタル署名、keyCertSign 、CRL 署名
CRL ディストリビューションポイント *	[CA 設定または CSR からのパススルー]

*CRL ディストリビューションポイントは、CA が CRL 生成を有効にして構成されている場合にのみ、このテンプレートで発行された証明書に含まれます。

AWS Private CA API の使用 (Java の例)

HTTP リクエストを送信してサービスをプログラムでインタラクティブに操作するために AWS Private Certificate Authority API を使用できます。サービスは HTTP レスポンスを返します。詳細については、「AWS Private Certificate Authority API Reference」を参照してください。

HTTP API に加えて、AWS SDK とコマンドラインツールを使用して AWS Private CA を操作できます。これは HTTP API で推奨されています。詳細については、「Amazon ウェブ サービスのツール」を参照してください。次のトピックでは、AWS SDK for Java を使用して AWS Private CA API をプログラミングする方法を説明します。

GetCertificateAuthorityCsr、GetCertificate、および DescribeCertificateAuthorityAuditReportオペレーションはウェーターをサポートします。ウェイターを使用して、特定のリソースの存在または状態に基づいてコードの進行を制御できます。詳細については、 AWSデベロッパーブログの以下のトピックと のウェーターAWS SDK for Javaを参照してください。

トピック

- ルート CA をプログラムで作成して有効にする
- 下位 CA をプログラム的に作成して有効にする
- CreateCertificateAuthority
- を使用して CreateCertificateAuthority Active Directory をサポートする
- CreateCertificateAuthorityAuditReport
- CreatePermission
- DeleteCertificateAuthority
- DeletePermission
- DeletePolicy
- DescribeCertificateAuthority
- DescribeCertificateAuthorityAuditReport
- GetCertificate
- GetCertificateAuthorityCertificate
- GetCertificateAuthorityCsr
- GetPolicy
- ImportCertificateAuthorityCertificate
- IssueCertificate

- ListCertificateAuthorities
- ListPermissions
- ListTags
- PutPolicy
- RestoreCertificateAuthority
- RevokeCertificate
- TagCertificateAuthorities
- UntagCertificateAuthority
- UpdateCertificateAuthority
- カスタムサブジェクト名で CA と証明書を作成する
- カスタム拡張を持つ証明書を作成する

ルート CA をプログラムで作成して有効にする

この Java サンプルは、次の AWS Private CA API アクションを使用してルート CA をアクティブ化する方法を示しています。

- 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);
    }
}
```

下位 CA をプログラム的に作成して有効にする

この Java サンプルは、次の AWS Private CA API アクションを使用して下位 CA をアクティブ化する方法を示しています。

- 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

次の Java サンプルは、 CreateCerticateAuthorityオペレーションの使用方法を示しています。

このオペレーションはプライベートの下位証明書認証機関 (CA) を作成します。CA 設定、失効設定を指定する必要があり、CA タイプ、およびオプションの冪等性トークン。

CA 設定で、次のように指定します。

- CA プライベートキーの作成に使用されるアルゴリズム名とキーサイズ
- CA が署名に使用する署名アルゴリズムのタイプ
- X.500 件名情報

CRL 設定で、次のように指定します。

- CRL の有効期間の日数 (CRL の有効期間)
- CRL を含む Amazon S3 バケット
- CA によって発行された証明書に含まれている S3 バケットの CNAME エイリアス

成功すると、この関数は CA の Amazon リソースネーム (ARN) を返します。

```
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);
    }
}
```

出力は次のようになります。

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

を使用して CreateCertificateAuthority Active Directory をサポートする

次の Java サンプルは、 CreateCerticateAuthorityオペレーションを使用して、Microsoft Active Directory (AD) の Enterprise NTAuth ストアにインストールできる CA を作成する方法を示しています。

このオペレーションでは、カスタムオブジェクト識別子 (OID) を使用してプライベートルート認証機関 (CA) を作成します。詳細および AWS CLI の同等オペレーションの例については、「Active Directory ログイン用に CA を作成する」を参照してください。

成功すると、この関数は CA の Amazon リソースネーム (ARN) を返します。

```
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);
    }
}
```

出力は次のようになります。

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

CreateCertificateAuthorityAuditReport

次の Java サンプルは、 CreateCertificateAuthorityAuditReportオペレーションの使用方法を示しています。

この関数は、証明書が発行または取り消されるたびにそれを一覧表示する監査報告書を作成します。この報告書は、入力時に指定する Amazon S3 バケットに保存されます。30 分に 1 回新しいレポートを生成できます。

```
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);

    }
}
```

出力は次のようになります。

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

CreatePermission

次の Java サンプルは、 CreatePermissionオペレーションの使用方法を示しています。

このオペレーションは、プライベート CA から指定された AWS サービスプリンシパルにアクセス許可を割り当てます。サービスには、プライベート CA から証明書を作成および取得するアクセス許可を与えることができます。また、プライベート CA が付与したアクティブなアクセス許可を一覧表示することもできます。ACM を介して証明書を自動的に更新するには、CA から ACM サービスプリンシパル (ListPermissions) にすべての可能なアクセス許可 (IssueCertificate、GetCertificate、および) を割り当てる必要がありますacm.amazonaws.com。CA の ARN は、 ListCertificateAuthorities関数を呼び出すことで見つけることができます。

アクセス許可を作成したら、 ListPermissions関数で検査するか、 DeletePermission関数で削除できます。

```
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

次の Java サンプルは、 DeleteCertificateAuthorityオペレーションの使用方法を示しています。

このオペレーションは、 CreateCertificateAuthorityオペレーションを使用して作成したプライベート認証機関 (CA) を削除します。DeleteCertificateAuthority オペレーションでは、削除する CA の ARN を指定する必要があります。ARN は、 ListCertificateAuthoritiesオペレーションを呼び出すことで確認できます。プライベート CA のステータスが CREATING または PENDING_CERTIFICATE である場合は、即座に削除できます。ただし、証明書を既にインポートしている場合は、すぐに削除することはできません。まず、 UpdateCertificateAuthorityオペレーションを呼び出して CA を無効にし、 Statusパラメータを に設定する必要がありますDISABLED。その後、DeleteCertificateAuthority オペレーションで PermanentDeletionTimeInDays パラメータを使用して、7～30 の日数を指定できます。その期間中、プライベート CA は disabled ステータスに復元できます。デフォルトでは、PermanentDeletionTimeInDays パラメータを設定しない場合、復元期間は 30 日です。この期間が経過すると、プライベート CA は完全に削除され、復元できなくなります。詳細については、「CA の復元」を参照してください。

RestoreCertificateAuthority オペレーションの使用方法を示す Java の例については、「」を参照してください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

次の Java サンプルは、 DeletePermissionオペレーションの使用方法を示しています。

オペレーションは、プライベート CA が CreatePermissionsオペレーションを使用して AWSサービスプリンシパルに委任したアクセス許可を削除します。CA の ARN は、 ListCertificateAuthorities関数を呼び出すことで見つけることができます。ListPermissions 関数を呼び出すことで、CA が付与したアクセス許可を検査できます。

```
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

次の Java サンプルは、 DeletePolicyオペレーションの使用方法を示しています。

このオペレーションは、プライベート CA にアタッチされているリソースベースのポリシーを削除します。リソースベースのポリシーは、クロスアカウント CA 共有を有効にするのに使用します。プライベート CA の ARN は、 ListCertificateAuthoritiesアクションを呼び出すことで確認できます。

関連する API アクションには、 PutPolicyおよび が含まれます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.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 AWS 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;
        }
     }
}
```

DescribeCertificateAuthority

次の Java サンプルは、 DescribeCertificateAuthorityオペレーションの使用方法を示しています。

このオペレーションは、プライベート認証機関 (CA) に関する情報をリストします。プライベート CA の ARN (Amazon リソースネーム) を指定する必要があります。出力には、CA のステータスが含まれます。これは、次のいずれかとなります。

- CREATING — AWS Private CA がプライベート認証機関を作成中です。
- PENDING_CERTIFICATE — 証明書が保留中です。オンプレミスルート CA または下位 CA を使用してプライベート CA CSR に署名し、PCA にインポートする必要があります。
- ACTIVE — プライベート CA がアクティブです。
- DISABLED — プライベート CA が無効になっています。
- EXPIRED — プライベート CA 証明書の有効期限が切れています。
- FAILED — プライベート CA を作成することはできません。
- DELETED — プライベート CA は復元期間内にあります。その後は完全に削除されます。

```
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

次の Java サンプルは、 DescribeCertificateAuthorityAuditReportオペレーションの使用方法を示しています。

オペレーションは、 CreateCertificateAuthorityAuditReportオペレーションを呼び出して作成した特定の監査レポートに関する情報を一覧表示します。監査情報は、認証機関 (CA) プライベートキーが使用されるたびに作成されます。プライベートキーは、証明書を発行、CRL に署名、または証明書を取り消すときに使用されます。

```
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);
    }
}
```

出力は次のようになります。

```
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

次の Java サンプルは、 GetCertificateオペレーションの使用方法を示しています。

このオペレーションは、プライベート CA から証明書を取得します。IssueCertificate オペレーションを呼び出すと、証明書の ARN が返されます。GetCertificate オペレーションを呼び出すときは、プライベート CA の ARN と、発行済み証明書の ARN の両方を指定する必要があります。ISSUED 状態の場合は、証明書を取得できます。CreateCertificateAuthorityAuditReport オペレーションを呼び出して、プライベート CA によって発行および取り消されたすべての証明書に関する情報を含むレポートを作成できます。

```
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);
    }
}
```

出力は、指定した認証機関 (CA) および証明書について、以下に類似した証明書チェーンになります。

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

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

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

GetCertificateAuthorityCertificate

次の Java サンプルは、 GetCertificateAuthorityCertificateオペレーションの使用方法を示しています。

このオペレーションは、プライベート認証機関 (CA) の証明書と証明書チェーンを取得します。証明書とチェーンのどちらも、PEM 形式の base64 エンコード文字列です。このチェーンには CA 証明書は含まれません。チェーンの各証明書は、前の証明書に署名します。

```
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);
    }
}
```

出力は、指定した次の証明書チェーン (CA) に類似した証明書およびチェーンになります。

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

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

GetCertificateAuthorityCsr

次の Java サンプルは、 GetCertificateAuthorityCsrオペレーションの使用方法を示しています。

このオペレーションは、プライベート認証機関 (CA) の証明書署名リクエスト (CSR) を取得します。CSR は、 CreateCertificateAuthorityオペレーションを呼び出すと作成されます。CSR をオンプレミスの X.509 インフラストラクチャに持ち込み、ルート CA または下位 CA を使用して署名します。次に、 ImportCertificateAuthorityCertificateオペレーションを呼び出して、署名された証明書を ACM PCA にインポートします。CSR は PEM 形式の base64 エンコード文字列として返されます。

```
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);
   }
}
```

指定する認証機関 (CA) に対して、出力は次のようになります。証明書署名リクエスト (CSR) は base64 でエンコードされた PEM 形式です。これをローカルファイルに保存し、オンプレミスの X.509 インフラストラクチャに持ち込み、ルート CA または下位 CA を使用して署名します。

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

GetPolicy

次の Java サンプルは、 GetPolicyオペレーションの使用方法を示しています。

このオペレーションは、プライベート CA にアタッチされているリソースベースのポリシーを取得します。リソースベースのポリシーは、クロスアカウント CA 共有を有効にするのに使用します。プライベート CA の ARN は、 ListCertificateAuthoritiesアクションを呼び出すことで確認できます。

関連する API アクションには、 PutPolicyおよび が含まれます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

次の Java サンプルは、 ImportCertificateAuthorityCertificateオペレーションの使用方法を示しています。

このオペレーションは、署名されたプライベート CA 認定を AWS Private CA にインポートします。このオペレーションを呼び出す前に、 CreateCertificateAuthorityオペレーションを呼び出してプライベート認証機関を作成する必要があります。次に、 GetCertificateAuthorityCsrオペレーションを呼び出して、証明書署名リクエスト (CSR) を生成する必要があります。CSR をオンプレミスの CA に持ち込み、ルート証明書または下位の証明書を使用して署名します。証明書チェーンを作成し、署名証明書および証明書チェーンを作業ディレクトリにコピーします。

```
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

次の Java サンプルは、 IssueCertificateオペレーションの使用方法を示しています。

このオペレーションでは、プライベート認証機関 (CA) を使用してエンドエンティティ証明書を発行します。このオペレーションは、証明書の Amazon リソースネーム (ARN) を返します。を呼び出しGetCertificateて ARN を指定することで、証明書を取得できます。

> **Note**
>
> IssueCertificate オペレーションでは、証明書テンプレートを指定する必要があります。この例では EndEntityCertificate/V1 テンプレートを使用しています。使用可能なすべてのテンプレートについては、「証明書テンプレートについて」を参照してください。

```
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);
    }
}
```

出力は次のようになります。

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

ListCertificateAuthorities

次の Java サンプル例は、ListCertificateAuthorities オペレーションを使用する方法を示しています。

このオペレーションは、CreateCertificateAuthority オペレーションを使用して作成したプライベート証明書機関 (CA) を一覧表示します。

```
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());
    }
}
```

一覧表示する証明書機関がある場合、出力は次のようになります。

```
[{
 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

次の Java サンプルは、 ListPermissionsオペレーションの使用方法を示しています。

このオペレーションでは、プライベート CA が割り当てた権限があれば、その権限が一覧表示されます。IssueCertificate、、 GetCertificateなどのアクセス許可はListPermissions、CreatePermissionオペレーションで AWSサービスプリンシパルに割り当て、 DeletePermissionsオペレーションで取り消すことができます。

```
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);
    }
}
```

指定されたプライベート CA がサービスプリンシパルにアクセス許可を割り当てている場合、出力は次のようになります。

```
[{
       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

次の Java サンプルは、 ListTagsオペレーションの使用方法を示しています。

このオペレーションは、プライベート CA に関連付けられたタグ (ある場合) を一覧表示します。タグは、CA を特定し、整理するのに使用できるラベルです。各タグは、キー、および値 (オプション) で構成されます。TagCertificateAuthority オペレーションを呼び出して、CA に 1 つ以上のタグを追加します。UntagCertificateAuthority オペレーションを呼び出してタグを削除します。

```
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);
    }
}
```

リストするタグがある場合、出力は次のようになります。

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

PutPolicy

次の Java サンプルは、 PutPolicyオペレーションの使用方法を示しています。

このオペレーションにより、リソースベースのポリシーがプライベート CA にアタッチされ、クロスアカウント共有が可能になります。ポリシーによって承認されると、別の AWS アカウントに属するプリンシパルは、所有していないプライベート CA を使用してプライベートエンドエンティティ証明書を発行および更新できます。プライベート CA の ARN は、 ListCertificateAuthoritiesアクションを呼び出すことで確認できます。ポリシーの例については、「Resource-Based Policies」の「AWS Private CA ガイダンス」を参照してください。

ポリシーが CA にアタッチされたら、 GetPolicyアクションでポリシーを検査するか、 DeletePolicyアクションでポリシーを削除できます。

```
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

次の Java サンプルは、 RestoreCertificateAuthorityオペレーションの使用方法を示しています。プライベート CA は、復元期間の間は、いつでも復元できます。現在、この期間は削除の日から 7 ~ 30 日間で設定でき、CA を削除するときに定義できます。詳細については、「CA の復元」を参照してください。DeleteCertificateAuthority の Java の例も参照してください。

```
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

次の Java サンプルは、 RevokeCertificateオペレーションの使用方法を示しています。

このオペレーションは、 IssueCertificateオペレーションを呼び出して発行した証明書を取り消します。プライベート CA の作成時あるいは更新時に証明書失効リスト (CRL) を有効にすると、取り消

し済み証明書に関する情報が CRL に入ります。AWS Private CA は、指定した Amazon S3 バケットに CRL を書き込みます。詳細については、「CrlConfiguration構造」を参照してください。

```
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

次の Java サンプルは、 TagCertificateAuthorityオペレーションの使用方法を示しています。

このオペレーションは、1 つまたは複数のタグをプライベート CA に追加します。タグは、AWS リソースを特定し、整理するのに使用できるラベルです。各タグは、キー、および値 (オプション) で構成されます。このオペレーションを呼び出すときは、Amazon リソースネーム (ARN) でプライベート CA を指定します。キーと値のペアを使用してタグを指定します。その CA の特定の特性を識別するために、1 つのプライベート CA だけにタグを適用できます。または、これらの CA 間で共通のリレーションシップをフィルタリングするには、同じタグを複数のプライベート CA に適用できます。1 つ以上のタグを削除するには、 UntagCertificateAuthorityオペレーションを使用します。ListTags オペレーションを呼び出して、CA に関連付けられているタグを確認します。

```
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

次の Java サンプルは、 UntagCertificateAuthorityオペレーションの使用方法を示しています。

このオペレーションは、1 つまたは複数のタグをプライベート CA から削除します。タグは、キーと値のペアから構成されます。このオペレーションを呼び出すときにタグの値部分を指定しない場合、このタグは値にかかわらず削除されます。値を指定する場合、タグは指定された値に関連付けられている場合にのみ削除されます。プライベート CA にタグを追加するには、 TagCertificateAuthorityオペレーションを使用します。ListTags オペレーションを呼び出して、CA に関連付けられているタグを確認します。

```
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

次の Java サンプルは、 UpdateCertificateAuthorityオペレーションの使用方法を示しています。

このオペレーションでは、プライベート認証機関 (CA) のステータスまたは設定を更新します。プライベート CA を更新するには、そのステータスが ACTIVE または DISABLED であることが必要です。ACTIVE 状態のプライベート CA は無効化できます。DISABLED 状態の CA は再有効化できます。

```
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;
        }
    }
}
```

カスタムサブジェクト名で CA と証明書を作成する

CustomAttribute オブジェクトを使用すると、管理者はカスタムオブジェクト ID (OID) をプライベート CA や証明書に渡すことができます。カスタム OID を使用すると、組織の構造とニーズを反映した特殊なサブジェクト名階層を作成できます。カスタマイズした証明書は、ApiPassthrough テンプレートのいずれかを使用して作成する必要があります。テンプレートの詳細については、「テンプレートの種類」を参照してください。カスタム属性の使用の詳細については、「プライベートエンドエンティティ証明書の発行」と「CA を作成するための手順 (CLI) 」を参照してください。

StandardAttributes は CustomAttributes と組み合わせて使用することはできません。ただし、標準 OID を CustomAttributes の一部として渡すことはできます。デフォルトのサブジェクト名 OID を次の表に示します。

サブジェクト名	オブジェクト ID
国	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
状態	2.5.4.8
姓	2.5.4.4
タイトル	2.5.4.12

トピック

- CustomAttribute を持つ CA を作成する
- CustomAttribute を持つ CA を発行する

CustomAttribute を持つ CA を作成する

```
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);
    }
}
```

CustomAttribute を持つ CA を発行する

```
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);
   }
}
```

カスタム拡張を持つ証明書を作成する

CustomExtension オブジェクトを使用すると、管理者はプライベート証明書にカスタム X.509 拡張を設定できます。カスタマイズされた証明書は、ApiPassthrough テンプレートのいずれかを使用して作成する必要があります。テンプレートの詳細については、「テンプレートの種類」を参照してください。カスタム拡張の使用の詳細については、「プライベートエンドエンティティ証明書の発行」を参照してください。

トピック

- NameConstraints 拡張を持つ下位 CA を有効にする
- QC ステートメント拡張を持つ証明書を発行する

NameConstraints 拡張を持つ下位 CA を有効にする

```
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);
    }
}
```

QC ステートメント拡張を持つ証明書を発行する

```
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);
   }
}
```

AWS Private CA API を使用した Matter 規格の実装 (Java の例)

AWS Private Certificate Authority API を使用して Matter 接続規格に準拠する証明書を作成できます。Matter は、複数のエンジニアリングプラットフォームにわたるモノのインターネット (IoT) デバイスのセキュリティと一貫性を向上させる証明書設定を規定しています。Matter の詳細については、「buildwithmatter.com」を参照してください。

このセクションでの Java の例では、HTTP リクエストを送信してサービスとやりとりします。サービスは HTTP レスポンスを返します。詳細については、「AWS Private Certificate Authority API Reference」を参照してください。

HTTP API に加えて、AWS SDK とコマンドラインツールを使用して AWS Private CA を操作できます。これは HTTP API で推奨されています。詳細については、「Amazon ウェブ サービスのツール」を参照してください。次のトピックでは、AWS SDK for Java を使用して AWS Private CA API をプログラミングする方法を説明します。

GetCertificateAuthorityCsr、GetCertificate、および DescribeCertificateAuthorityAuditReportオペレーションはウェーターをサポートします。ウェイターを使用して、特定のリソースの存在または状態に基づいてコードの進行を制御できます。詳細については、 AWSデベロッパーブログの以下のトピックと のウェーターAWS SDK for Javaを参照してください。

2023 年 10 月にリリースされたMatter 1.2 は、証明書失効リスト (CRLsを使用した DAC 失効をサポートしています。現在のMatt 標準に準拠できるように、Matter証明書を発行するCAsのCRL失効を有効にすると、 `CrlConfiguration` オブジェクトの `CrlDistributionPointExtensionConfiguration`構造で、 `OmitExtension`を に設定します`true`。

通常、CAs発行する証明書に CRL ディストリビューションポイント (CDP) を埋め込み、証明書チェーンの検証を実行する証明書利用者が CRL を取得し、証明書のステータスをチェックできるようにします。Matter では、CDP URI は証明書に書き込まれません。代わりに、ユーザーは信頼できるMatterデータストアであるMatter分散コンプライアンス台帳 (DCL) からCDPsを取得します。CDP URI をMatter DCLにアップロードして、DACsの検証時に検出できるようにする必要があります。CDP URI の決定の詳細については、「」を参照してくださいCRL ディストリビューションポイント (CDP) URI の確認 。Matter の詳細については、Matter DCL ドキュメントを参照してください。

トピック

- 製品認証機関 (PAA) をアクティブ化する
- 製品認証中間 (PAI) をアクティブ化する
- Device Attestation Certificate (DAC) を作成する
- ノード運用証明書 (NOC) のルート CA を有効にします。
- ノード運用証明書 (NOC) の下位 CA をアクティブ化する
- ノード運用証明書 (NOC) を作成する

製品認証機関 (PAA) をアクティブ化する

この Java サンプルでは、 RootCACertificate_APIPassthrough/V1 定義 テンプレートを使用して、製品認証用のMatter ルート CA (PAA)証明書を作成およびインストールする方法を示します。AuthorityKeyIdentifier (AKI) 拡張機能は PAAsではオプションです。AKI を設定するには、Base64-encoded AKI 値を生成し、それを に渡す必要があります CustomExtension。

この例では次の AWS Private CA API アクションを呼び出します。

- CreateCertificateAuthority
- GetCertificateAuthorityCsr
- IssueCertificate
- GetCertificate
- ImportCertificateAuthorityCertificate

問題が発生した場合は、「トラブルシューティング」セクションの「Matter 規格の使用」を参照してください。

```
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);
    }
}
```

製品認証中間 (PAI) をアクティブ化する

この Java サンプルでは、 BlankSubordinateCACertificate _PathLen0_APIPassthrough /V1 定義テンプレートを使用して、製品認証用のMatter Subordinate CA (PAI)証明書を作成およびインストールする方法を示します。Base64-encoded KeyUsage 値を生成し、 に渡す必要がありますCustomExtension。

この例では次の AWS Private CA API アクションを呼び出します。

- CreateCertificateAuthority
- GetCertificateAuthorityCsr
- IssueCertificate
- GetCertificate
- ImportCertificateAuthorityCertificate

- GetCertificateAuthorityCertificate

問題が発生した場合は、「トラブルシューティング」セクションの「Matter 規格の使用」を参照してください。

```
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);
    }
}
```

Device Attestation Certificate (DAC) を作成する

この Java サンプルは、BlankEndEntityCertificate_CriticalBasicConstraints_APIPassthrough /V1 テンプレートを使用してMatterデバイス認証証明書を作成する方法を示しています。Base64-encoded KeyUsage 値を生成し、 に渡す必要があります CustomExtension。

この例では次の AWS Private CA API アクションを呼び出します。

- IssueCertificate

問題が発生した場合は、「トラブルシューティング」セクションの「Matter 規格の使用」を参照してください。

```
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);
    }
}
```

ノード運用証明書 (NOC) のルート CA を有効にします。

この Java サンプルでは、 RootCACertificate_APIPassthrough/V1 定義 テンプレートを使用してMatterルート CA証明書を作成してインストールし、NOCsを発行する方法を示します。AuthorityKeyIdentifier (AKI) 拡張機能は、NOC ルート CA 証明書ではオプションです。AKI を設定するには、Base64-encoded AKI 値を生成し、それを に渡す必要があります CustomExtension。

この例では次の AWS Private CA API アクションを呼び出します。

- CreateCertificateAuthority
- GetCertificateAuthorityCsr
- IssueCertificate
- GetCertificate
- ImportCertificateAuthorityCertificate

問題が発生した場合は、「トラブルシューティング」セクションの「Matter 規格の使用」を参照してください。

```
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);
    }
}
```

ノード運用証明書 (NOC) の下位 CA をアクティブ化する

この Java サンプルでは、 BlankSubordinateCACertificate _PathLen0_APIPassthrough /V1 定義 テンプレートを使用してMatter Subordinate CA証明書を発行およびインストールしてNOCs を発行する方法を示します。Base64-encoded KeyUsage 値を生成し、 に渡す必要があります CustomExtension。

この例では次の AWS Private CA API アクションを呼び出します。

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

問題が発生した場合は、トラブルシューティングセクションMatter 規格の使用の「」を参照してください。

```
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);
    }
}
```

ノード運用証明書 (NOC) を作成する

この Java サンプルは、BlankEndEntityCertificate_CriticalBasicConstraints_APIPassthrough /V1 テンプレートを使用してMatterノード運用証明書を作成する方法を示しています。Base64-encoded KeyUsage 値を生成し、 に渡す必要があります CustomExtension。

この例では次の AWS Private CA API アクションを呼び出します。

- IssueCertificate

問題が発生した場合は、「トラブルシューティング」セクションの「Matter 規格の使用」を参照してください。

```
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);
   }
}
```

AWS Private CA API を使用してモバイル運転ライセンス (mDL) 標準を実装する (Java の例)

AWS Private Certificate Authority API を使用して、モバイル運転ライセンス (mDL) の ISO/IEC 標準に準拠する証明書を作成できます。この標準は、証明書設定を含む、モバイルデバイスに関連する運転免許証の実装に関するインターフェイス仕様を確立します。

このセクションでの Java の例では、HTTP リクエストを送信してサービスとやりとりします。サービスは HTTP レスポンスを返します。詳細については、「AWS Private Certificate Authority APIリファレンス」を参照してください。

HTTP API に加えて、 AWS SDKs とAWS CLIツールを使用して を管理することもできますAWS Private CA。SDK または を HTTP API AWS CLI経由で使用することをお勧めします。詳細については、「Amazon ウェブ サービスのツール」を参照してください。次のトピックでは、AWS SDK for Java を使用して AWS Private CA API をプログラミングする方法を説明します。

`GetCertificateAuthorityCsr`、`GetCertificate`、および `DescribeCertificateAuthorityAuditReport`オペレーションはウェーターをサポートします。ウェイターを使用して、特定のリソースの存在または状態に基づいてコードの進行を制御できます。詳細については、 AWSデベロッパーブログの以下のトピックと AWS SDK for Java のウェーターを参照してください。

トピック

- 発行機関認証機関 (IACA) 証明書をアクティブ化する
- ドキュメント署名者証明書を作成する

発行機関認証機関 (IACA) 証明書をアクティブ化する

この Java サンプルでは、 BlankRootCACertificate _PathLen0_APIPassthrough /V1 定義 テンプレートを使用して ISO/IEC mDL 標準 - 準拠の発行機関認証局 (IACA) 証明書を作成およびインストールする方法を示します。`KeyUsage`、、`IssuerAlternativeName`および の base64 でエンコードされた値を生成し`CRLDistributionPoint`、 に渡す必要があります`CustomExtensions`。

> **Note**
>
> IACA リンク証明書は、古い IACA ルート証明書から新しい IACA ルート証明書への信頼パスを確立します。発行機関は、IACA 再キープロセス中に IACA リンク証明書を生成して配布で

きます。IACA リンク証明書は、 pathLen=0が設定された IACA ルート証明書を使用して発行することはできません。

この例では次の AWS Private CA API アクションを呼び出します。

- 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);
    }
}
```

ドキュメント署名者証明書を作成する

この Java サンプルは、BlankEndEntityCertificate_APIPassthrough /V1 テンプレートを使用して ISO/IEC mDL 標準 - 準拠のドキュメント署名者証明書を作成する方法を示しています。KeyUsage、、および の base64 でエンコードされた値を生成CRLDistributionPointしIssuerAlternativeName、 に渡す必要がありますCustomExtensions。

この例では次の AWS Private CA API アクションを呼び出します。

- 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);
    }
}
```

外部署名付きプライベート CA 証明書

プライベート CA 階層の信頼ルートが AWS Private CA の外部の CA でなければならない場合は、独自のルート CA を作成して自己署名できます。または、組織が運営する外部のプライベート CA によって署名されたプライベート CA 証明書を取得することもできます。ソースに関わらず、この外部から取得した CA を使用して、AWS Private CA が管理するプライベートの下位 CA 証明書に署名できます。

> **Note**
>
> 外部信頼サービスプロバイダーの作成または取得の手順は、このガイドの対象外です。

AWS Private CA で外部の親 CA を使用すると、RFC 5280 の「名前制約」セクションで定義されている CA 名制約を適用できます。名前制約により、CA 管理者は証明書のサブジェクト名を制限できます。

外部 CA を持つプライベートの下位 CA 証明書に署名する予定の場合、有効な CA を AWS Private CA に導入する前に完了しておくべきタスクが 3 つあります。

1. 証明書署名リクエスト (CSR) を生成します。
2. CSR を外部の署名機関に送信し、署名付き CA 証明書とチェーン証明書を取得します。
3. 署名付き証明書を AWS Private CA にインストールします。

次の手順では、AWS Management Console または AWS CLI を使用してこれらのタスクを実行する方法について説明します。

外部署名付き CA 証明書の取得とインストールを行うには (コンソール)

1. (オプション) CA の詳細ページをまだ開いていない場合は、https://console.aws.amazon.com/acm-pca/home の AWS Private CA コンソールを開きます。[プライベート認証局] ページで、ステータスが [証明書保留中]、[有効]、[無効]、または [期限切れ] の下位 CA を選択します。
2. [アクション]、[CA 証明書をインストール] を選択し、[下位 CA 証明書をインストール] ページを開きます。
3. [下位 CA 証明書のインストール]ページの [CA タイプの選択] で、[外部プライベート CA] を選択します。

4. [この CA の CSR] の下に、CSR の Base64 でエンコードされた ASCII テキストがコンソールに表示されます。[コピー] ボタンを使用してテキストをコピーするか、[CSR をファイルにエクスポート] を選択してローカルに保存できます。

   > **Note**
   >
   > CSR テキストの正確な形式を保持してコピー & ペーストする必要があります。

5. 外部の署名機関から署名付き証明書を取得するためのオフライン手順をすぐに実行できない場合は、ページを一度閉じて、署名付き証明書と証明書チェーンを入手してからもう一度開いてみてください。

   それ以外の場合は、準備ができたら、次のいずれかを行います。

   - 証明書本文と証明書チェーンの Base64 でエンコードされた ASCII テキストをそれぞれのテキストボックスに貼り付けます。
   - [アップロード] を選択して、証明書本文と証明書チェーンをローカルファイルからそれぞれのテキストボックスにロードします。

6. [確認してインストール] を選択します。

外部署名付き CA 証明書を取得およびインストールするには (CLI)

1. get-certificate-authority-csr コマンドを使用して、プライベート CA の証明書署名リクエスト (CSR) を取得します。CSR を画面に送信するには、`--output text` オプションを使用して各行の末尾から CR/LF 文字を削除します。CSR をファイルに送信するには、リダイレクトオプション (>) を使用して、ファイル名を指定します。

   ```
   $ 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
   ```

   CSR をローカルファイルとして保存した後は、次の OpenSSL コマンドを使用して確認できます。

   ```
   openssl req -in path_to_CSR_file -text -noout
   ```

このコマンドを処理すると、次のような出力が生成されます。CA 拡張機能は TRUE であり、CSR が CA 証明書用であることを示しています。

```
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. CSR を外部の署名機関に送信し、Base64 PEM エンコードされた署名付き証明書および証明書チェーンを含むファイルを取得します。
3. import-certificate-authority-certificate コマンドを使用して、プライベート CA 証明書ファイルとチェーンファイルを にインポートしますAWS Private CA。

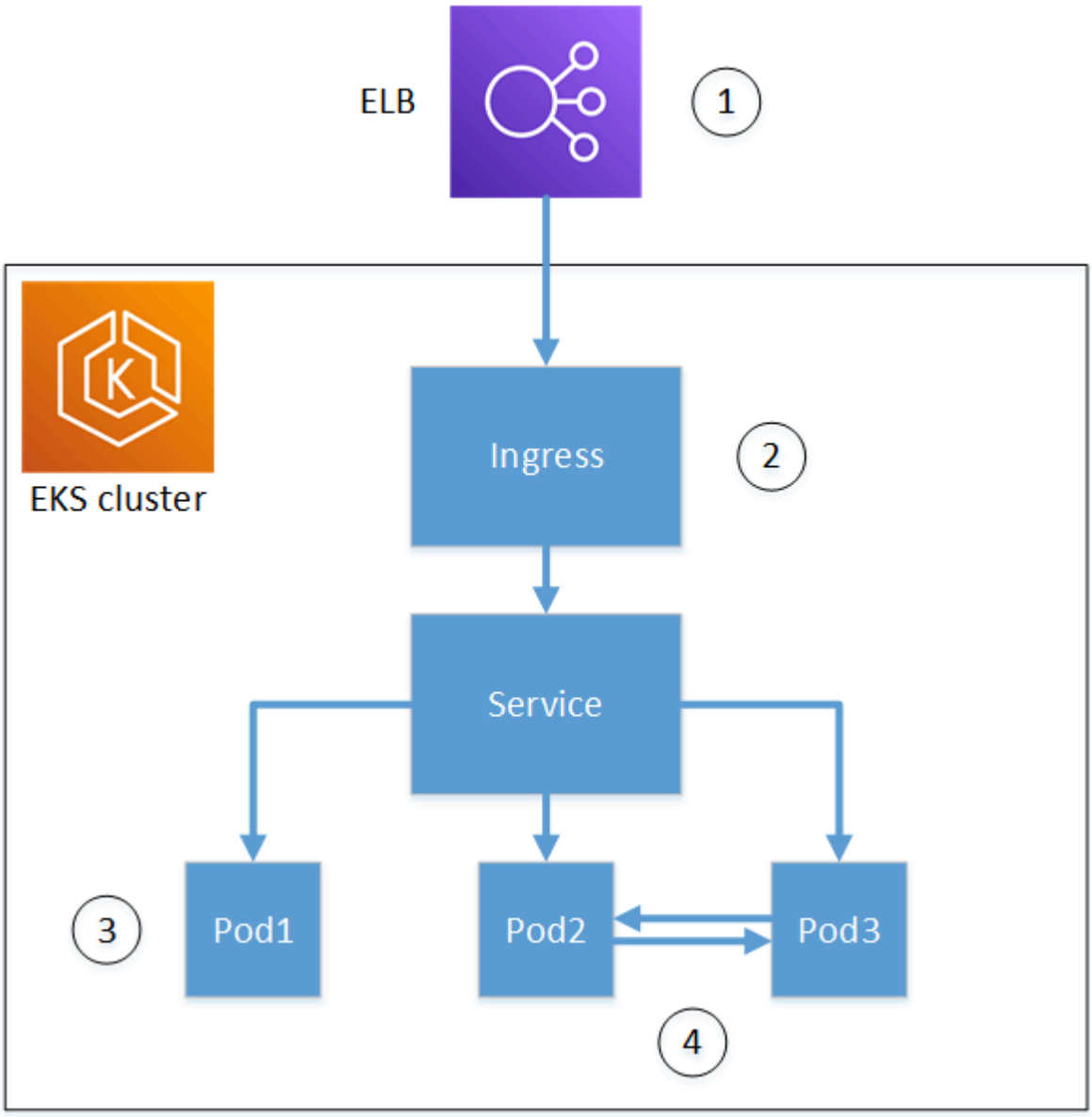
```
$ aws acm-pca import-certificate-authority-certificate \
--certificate-authority-arn arn:aws:acm-pca:region:account:\
certificate-authority/12345678-1234-1234-1234-123456789012 \
--certificate file://C:\example_ca_cert.pem \
--certificate-chain file://C:\example_ca_cert_chain.pem
```

AWS Private CA での Kubernetes のセキュリティ保護

Kubernetes コンテナとアプリケーションはデジタル証明書を使用して、TLS を介した安全な認証と暗号化を行います。Kubernetes の TLS 証明書のライフサイクル管理で広く採用されているソリューションは cert-manager です。これは、証明書のリクエストおよび Kubernetes コンテナへの配布と、証明書の更新の自動化を行う Kubernetes のアドオンです。

AWS Private CA は、クラスターにプライベートキーを保存せずに CA をセットアップしたい cert-manager ユーザーaws-privateca-issuer向けに、cert-manager、 へのオープンソースプラグインを提供します。CA の運用に対するアクセスの制御と監査に関する規制要件があるユーザーは、このソリューションを使用して監査可能性を高め、コンプライアンスをサポートできます。AWS プライベート CA 発行者プラグインは、Amazon Elastic Kubernetes Service (Amazon EKS)、AWS 上のセルフマネージド型の Kubernetes またはオンプレミスの Kubernetes で使用できます。

下の図は、Amazon EKS クラスターで TLS を使用する際で使用できるオプションの一部です。さまざまなリソースを含むこのサンプルクラスターは、ロードバランサーの背後にあります。数字は、外部ロードバランサー、イングレスコントローラー、サービス内の個々のポッド、相互に安全に通信するポッドのペアなど、TLS で保護された通信の可能なエンドポイントを示しています。



1. ロードバランサーでの終了。

   Elastic Load Balancing (ELB) は AWS Certificate Manager 統合サービスです。つまり、プライベート CA で ACM をプロビジョニングし、証明書に署名し、ELB コンソールを使用してインストールすることができます。このソリューションは、リモートクライアントとロードバランサーの間にある暗号化通信を提供します。データは暗号化されずに EKS クラスターに渡されます。あるいは、非 AWS ロードバランサーにプライベート証明書を提供して TLS を終了させることもできます。

2. Kubernetes イングレスコントローラーでの終了

Ingress コントローラーは、ネイティブのロードバランサーおよびルーターとして EKS クラスター内にあります。cert-manager と の両方をインストールしaws-privateca-issuer、プライベート CA を使用してクラスターをプロビジョニングした場合、Kubernetes はコントローラーに署名付き TLS 証明書をインストールして、外部通信のクラスターのエンドポイントとして機能することができます。ロードバランサーとイングレスコントローラー間の通信は暗号化され、受信後、データは暗号化されずにクラスターのリソースに渡されます。

3. ポッドでの終了。

   各ポッドは、ストレージとネットワークリソースを共有する 1 つ以上のコンテナのグループです。cert-manager と の両方をインストールしaws-privateca-issuer、プライベート CA を使用してクラスターをプロビジョニングした場合、Kubernetes は必要に応じて署名付き TLS 証明書をポッドにインストールできます。ポッドで終了する TLS 接続は、デフォルトではクラスター内の他のポッドには使用できません。

4. ポッド間の安全な通信。

   また、複数のポッドを相互に通信できるようにする証明書をプロビジョニングすることもできます。以下のシナリオが考えられます。

   - Kubernetes が生成した自己署名証明書によるプロビジョニング これによりポッド間の通信は保護されますが、自己署名証明書は HIPAA や FIPS の要件を満たしません。
   - プライベート CA によって署名された証明書によるプロビジョニング 上記の番号 2 と 3 と同様に、これには cert-manager と の両方をインストールしaws-privateca-issuer、プライベート CA を使用してクラスターをプロビジョニングする必要があります。その後、Kubernetes は必要に応じて署名付き TLS 証明書をポッドにインストールできます。

証明書マネージャーのクロスアカウント使用

CA へのクロスアカウントアクセス権を持つ管理者は、cert-manager を使用して Kubernetes クラスターをプロビジョニングできます。詳細については、「プライベート CA へのクロスアカウントアクセスに関するセキュリティのベストプラクティス」を参照してください。

> **Note**
>
> クロスアカウントシナリオでは、特定の AWS Private CA 証明書テンプレートのみが使用できます。使用可能なテンプレートのリストについては、「the section called “サポートされている証明書テンプレート ”」を参照してください。

サポートされている証明書テンプレート

以下の表は、cert-manager で Kubernetes クラスターをプロビジョニングするために使用できる AWS Private CA テンプレートの一覧です。

Kubernetes でサポートされているテンプレート	クロスアカウント使用のサポート
BlankEndEntityCertificate_CSRPassthrough / V1 定義	
CodeSigningCertificate/V1 定義	
EndEntityCertificate/V1 定義	✓
EndEntityClientAuthCertificate/V1 定義	✓
EndEntityServerAuthCertificate/V1 定義	✓
OCSPSigningCertificate/V1 定義	

ソリューション例

以下の統合ソリューションは、Amazon EKS クラスターの AWS Private CA にアクセスを設定する方法を示しています。

- AWS Private CA と Amazon EKS を使用した TLS 対応の Kubernetes クラスター
- 新しい AWS Load Balancer Controller を使用した Amazon EKS での end-to-end TLS 暗号化の設定

AWS Private CA Connector for Active Directory

AWS Private CA Connector for Active Directory とは

AWS Private CA は AWS Managed Microsoft AD に必要な証明書を発行および管理できます。AWS Private CA Connector for Active Directory (Connector for AD) を使用すると、オンプレミスのエンタープライズ CA やその他のサードパーティ CA を、所有する管理されたプライベート CA に置き換えることができ、AD が管理するユーザー、グループ、およびマシンに証明書を登録できるようになります。

Connector for AD と AWS Managed Microsoft AD を併用すると、AD とパブリックキーインフラストラクチャをクラウドに移行することで、オンプレミスインフラストラクチャをなくせます。オンプレミス AD と AWS Private CA の併用を検討している場合、この機能を AWS Managed Microsoft AD Connector と統合することもできます。

トピック

- Connector for AD を初めて使用する場合
- Connector for AD へのアクセス
- Connector for AD の料金

Connector for AD を初めて使用する場合

Connector for AD を初めて使用する場合は、以下のセクションを初めに読むことをお勧めします。

- とは AWS Private CA
- AWS Directory Service とは

Connector for AD へのアクセス

Connector for AD にはAWS CLI、コンソール、、および APIs からアクセスできます。コネクタには、コンソール、AWS Private CAコンソール、AWS Directory ServiceまたはAWS Management Console検索バーで Connector for AD を検索することでアクセスできます。

Connector for AD の料金

Connector for AD は、追加費用なしで AWS Private CA の機能として提供されます。お支払いいただくのは、プライベート認証機関と、プライベート認証機関を通じて発行する証明書の料金のみです。

AWS Private CA の最新の料金情報については、「AWS Private Certificate Authority 料金表」を参照してください。AWS 料金計算ツール でコストを見積もることもできます。

AWS Private CA Connector for Active Directory 入門

AWS Private CA Connector for Active Directory を使用すると、認証と暗号化のためにプライベート CA から Active Directory オブジェクトに証明書を発行できます。コネクタを作成すると、AWS Private Certificate Authority はディレクトリオブジェクトが証明書をリクエストするためのエンドポイントを VPC に作成します。

証明書を発行するには、コネクタと、そのコネクタの AD 互換テンプレートを作成します。テンプレートを作成すると、AD グループの登録許可を設定できます。

トピック

Connector for AD の前提条件

Connector for AD には以下が必要です。

Connector for AD を作成するには、AWS Private Certificate Authority (CA) とディレクトリを設定する必要があります。次に、プライベート CA とディレクトリを Connector for AD サービスと共有する必要があります。コネクタを作成するには、セキュリティグループと IAM ポリシーを正しく設定する必要もあります。

AWS Private CA

ディレクトリオブジェクトに証明書を発行するために AWS Private CA を設定します。詳細については、「プライベート CA 管理」を参照してください。

Connector for AD を作成するには、AWS Private CA が `Active` 状態になっている必要があります。プライベート CA のサブジェクト名には共通名が含まれている必要があります。共通名のないプライベート CA を使用してコネクタを作成しようとすると、コネクタの作成に失敗します。

アクティブディレクトリ

AWS Private CA に加えて、仮想プライベートクラウド (VPC) 内に Active Directory が必要です。Connector for AD は、AWS Directory Service が提供する以下のディレクトリタイプをサポートしています。

- AWS Managed Microsoft Active Directory : AWS Directory Serviceでは、Microsoft Active Directory (AD) をマネージドサービスとして実行できます。 は AWS Directory Service for Microsoft Active Directoryとも呼ばれAWS Managed Microsoft AD、Windows Server 2019 を使用しています。AWS Managed Microsoft AD では、Microsoft SharePoint、カスタム .NET、および SQL Server ベースのアプリケーションなど、ディレクトリ対応ワークロードを AWS クラウド で実行できます。
- Active Directory Connector: AD Connector は、クラウドに情報をキャッシュせずに、ディレクトリリクエストをオンプレミスの Microsoft Active Directory にリダイレクトできるディレクトリゲートウェイです。AD Connector は、Amazon EC2 でホストされているドメインへの接続をサポートしています。

> **Note**
>
> Connector for AD と AWS Managed Microsoft AD を併用する場合、ドメインコントローラーの登録はサポートされません。

サービスアカウント

Directory Service AD Connector を使用する際には、サービスアカウントに追加の権限を委任する必要があります。サービスアカウントにアクセスコントロールリスト (ACL) を設定して、次の機能を許可します。

- サービスプリンシパル名 (SPN) をそれ自体に対して追加および削除する。
- 以下のコンテナで証明機関を作成および更新します。

```
#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
```

- NT AuthCertificates 認証機関 (CA) オブジェクトを作成および更新します。注: NT AuthCertificates CA オブジェクトが存在する場合は、そのオブジェクトに対するアクセス許可を委任する必要があります。オブジェクトが存在しない場合は、Public Key Services コンテナに子オブジェクトを作成する権限を委任する必要があります。

```
#objects
CN=NTAuthCertificates,CN=Public Key Services,CN=Services,CN=Configuration
```

> **Note**
>
> AWS Managed Microsoft AD を使用している場合は、ディレクトリで Connector for AD サービスを認証すると追加の権限が自動的に委任されます。この前提条件となる手順はスキップできます。

この PowerShell スクリプトを使用して、追加のアクセス許可を委任できます。NT AuthCertifiates 認証機関オブジェクトが作成されます。「myconnectoraccount」をサービスアカウント名で置き換えてください。

```
$AccountName = 'myconnectoraccount'
# DO NOT modify anything below this comment.
# Getting Active Directory information.
Import-Module -Name 'ActiveDirectory'
$RootDSE = Get-ADRootDSE

# Getting AD Connector service account Information
$AccountProperties = Get-ADUser -Identity $AccountName
$AccountSid = New-Object -TypeName 'System.Security.Principal.SecurityIdentifier'
 $AccountProperties.SID.Value
[System.GUID]$ServicePrincipalNameGuid = (Get-ADObject -SearchBase
 $RootDse.SchemaNamingContext -Filter { lDAPDisplayName -eq 'servicePrincipalName' } -
Properties 'schemaIDGUID').schemaIDGUID
$AccountAclPath = $AccountProperties.DistinguishedName

# Getting ACL settings for AD Connector service account.
```

```
$AccountAcl = Get-ACL -Path "AD:\$AccountAclPath"

# Setting ACL allowing the AD Connector service account the ability to add and remove a
 Service Principal Name (SPN) to itself
$AccountAccessRule = New-Object -TypeName
 'System.DirectoryServices.ActiveDirectoryAccessRule' $AccountSid, 'WriteProperty',
 'Allow', $ServicePrincipalNameGuid, 'None'
$AccountAcl.AddAccessRule($AccountAccessRule)
Set-ACL -AclObject $AccountAcl -Path "AD:\$AccountAclPath"

# Add ACLs allowing AD Connector service account the ability to create certification
 authorities
[System.GUID]$CertificationAuthorityGuid = (Get-ADObject -SearchBase
 $RootDse.SchemaNamingContext -Filter { lDAPDisplayName -eq 'certificationAuthority' }
 -Properties 'schemaIDGUID').schemaIDGUID
$CAAccessRule = New-Object -TypeName
 'System.DirectoryServices.ActiveDirectoryAccessRule' $AccountSid,
 'ReadProperty,WriteProperty,CreateChild,DeleteChild', 'Allow',
 $CertificationAuthorityGuid, 'None'
$PKSDN = "CN=Public Key Services,CN=Services,CN=Configuration,
$($RootDSE.rootDomainNamingContext)"
$PKSACL = Get-ACL -Path "AD:\$PKSDN"
$PKSACL.AddAccessRule($CAAccessRule)
Set-ACL -AclObject $PKSACL -Path "AD:\$PKSDN"

$AIADN = "CN=AIA,CN=Public Key Services,CN=Services,CN=Configuration,
$($RootDSE.rootDomainNamingContext)"
$AIAACL = Get-ACL -Path "AD:\$AIADN"
$AIAACL.AddAccessRule($CAAccessRule)
Set-ACL -AclObject $AIAACL -Path "AD:\$AIADN"

$CertificationAuthoritiesDN = "CN=Certification Authorities,CN=Public Key
 Services,CN=Services,CN=Configuration,$($RootDSE.rootDomainNamingContext)"
$CertificationAuthoritiesACL = Get-ACL -Path "AD:\$CertificationAuthoritiesDN"
$CertificationAuthoritiesACL.AddAccessRule($CAAccessRule)
Set-ACL -AclObject $CertificationAuthoritiesACL -Path "AD:\$CertificationAuthoritiesDN"

$NTAuthCertificatesDN = "CN=NTAuthCertificates,CN=Public Key
 Services,CN=Services,CN=Configuration,$($RootDSE.rootDomainNamingContext)"
If (-Not (Test-Path -Path "AD:\$NTAuthCertificatesDN")) {
New-ADObject -Name 'NTAuthCertificates' -Type 'certificationAuthority' -OtherAttributes
 @{certificateRevocationList=[byte[]]'00';authorityRevocationList=[byte[]]'00';cACertificate=[b
 -Path "CN=Public Key Services,CN=Services,CN=Configuration,
$($RootDSE.rootDomainNamingContext)" }
```

```
$NTAuthCertificatesACL = Get-ACL -Path "AD:\$NTAuthCertificatesDN"
$NullGuid = [System.GUID]'00000000-0000-0000-0000-000000000000'
$NTAuthAccessRule = New-Object -TypeName
 'System.DirectoryServices.ActiveDirectoryAccessRule' $AccountSid,
 'ReadProperty,WriteProperty', 'Allow', $NullGuid, 'None'
$NTAuthCertificatesACL.AddAccessRule($NTAuthAccessRule)
Set-ACL -AclObject $NTAuthCertificatesACL -Path "AD:\$NTAuthCertificatesDN"
```

IAM ポリシー

AD のコネクタを作成するには、コネクタリソースの作成、Connector for AD サービスとのプライベート CA の共有、およびディレクトリによる Connector for AD サービスの承認を許可する IAM ポリシーが必要です。

これはユーザーマネージドポリシーの例です。

```
{
    "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": {
```

```
                "StringLike": {
                    "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 には、コンソールとコマンドラインの両方で使用するための追加の AWS RAM 権限が必要です。

```
{
    "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": "*"
        }
    ]
}
```

AWS Private CA を Connector for AD と共有する

AWS Resource Access Manager サービスプリンシパル共有を使用して、AWS Private CA をコネクタサービスと共有する必要があります。

AWS コンソールでコネクタを作成すると、リソース共有が自動的に作成されます。

AWS CLI を使用してリソース共有を作成する場合は、AWS RAM create-resource-share コマンドを使用します。

次のコマンドでリソース共有が作成されます。

```
$  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
```

が呼び出すサービスプリンシパル CreateConnector には、PCA に対する証明書発行のアクセス許可があります。Connector for AD を使用するサービスプリンシパルによる AWS Private CA リソースへの一般アクセスを阻むには、`CalledVia` を使用して権限を制限します。

ディレクトリを使用して Connector for AD を認証します。

Connector for AD サービスをディレクトリで認証して、コネクタがディレクトリと通信できるようにします。Connector for AD サービスを認証するには、ディレクトリ登録を作成します。ディレクトリ登録の作成の詳細については、「ディレクトリ登録の管理」を参照してください。

セキュリティグループ

VPC と Connector for AD コネクタ間の通信は AWS PrivateLink 経由で行われるため、VPC で TCP と UDP のポート 443 を開くインバウンドルールを持つセキュリティグループが必要です。コネクタの作成時に、このセキュリティグループを要求されます。ソースをカスタムとして指定し、VPC の CIDR ブロックを選択できます。これをさらに制限することもできます (IP、CIDR、セキュリティグループ ID など)。

コネクタを作成する

詳細については、「手順 コネクタの作成」を参照してください。

AD ポリシーを設定する

Connector for AD は、お客様のグループポリシーオブジェクト (GPO) 設定を表示または管理できません。GPO は、お客様の AWS Private CA や他の認証サーバー、証明書交付サーバーへの AD リクエストのルーティングを制御します。GPO 設定が無効な場合、リクエストが正しくルーティングされない可能性があります。Connector for AD 設定を構成してテストするのはお客様次第です。

グループポリシーは 1 つのコネクタに関連付けられており、1 つの AD に対して複数のコネクタを作成することができます。グループポリシー設定が異なる場合、各コネクタへのアクセス制御を管理するのはユーザー次第です。

データプレーン呼び出しのセキュリティは、Kerberos および VPC 設定によって異なります。VPC にアクセスできる人なら誰でも、対応する AD に対して認証されている限り、データプレーン呼び出しを行うことができます。これは、 の境界外に存在し、承認と認証 AWSAuth の管理はお客様次第です。

Active Directory では、以下の手順に従って、コネクタを作成したときに生成された URI を指す GPO を作成します。このステップは、Connector for AD をコンソールまたはコマンドラインから使用するのに必要です。

GPO を設定する

1. DC でサーバーマネージャーを開きます。
2. [ツール] に移動し、コンソールの右上隅にある [グループポリシー管理] を選択します。
3. [フォレスト] > [ドメイン] に移動します。ドメイン名を選択し、ドメインを右クリックします。「このドメインに GPO を作成してここにリンクする」を選択し、名前に PCA GPO を入力します。
4. これで、新しく作成した GPO がドメイン名の下に表示されます。
5. 「PCA GPO」を選択し、「編集」を選択します。ダイアログボックスが開き、「This is a link and that changes will be globally propagated (これはリンクであり、変更は全体に適用されます)」という警告メッセージが表示されたら、メッセージを了承して続行します。グループポリシー管理エディタが開きます。
6. 「グループポリシー管理エディタ」で、[コンピュータの設定] > [ポリシー] > [Windows 設定] > [セキュリティ設定] > [パブリックキーポリシー] (フォルダを選択) に移動します。
7. [オブジェクトタイプ] に移動し、[証明書サービスクライアント - 証明書登録ポリシー] を選択します。
8. オプションで、[設定モデル] を [有効] に変更します。
9. [Active Directory 登録ポリシー] が オン で、[有効] になっていることを確認します。[追加] を選択します。
10. 証明書登録ポリシーサーバー ウィンドウが開くはずです。
11. コネクタを作成したときに生成された証明書登録ポリシーサーバーエンドポイントを [登録サーバーポリシー URI を入力] フィールドに入力します。
12. [認証タイプ] は [Windows 統合] のままにします。

13. [検証] を選択します。検証が成功したら、追加を選択します。ダイアログボックスが閉じます。
14. [証明書サービスクライアント - 証明書登録ポリシー] に戻り、新しく作成したコネクタの横にあるチェックボックスをオンにして、コネクタをデフォルトの登録ポリシーにします。
15. 「Active Directory 登録ポリシー」を選択し、「削除」を選択します。
16. 確認ダイアログボックスで [はい] を選択して LDAP ベースの認証を削除します。
17. [証明書サービスクライアント] > [証明書登録ポリシー] ウィンドウで [適用] と [OK] を選択し、閉じます。
18. [パブリックキーポリシー] フォルダに移動し、[証明書サービスクライアント - 自動登録] を選択します。
19. [設定モデル] オプションを [有効] に変更します。
20. [期限切れの証明書を更新する] と [証明書を更新する] の両方がオンになっていることを確認します。他の設定はそのままにします。
21. 適用を選択し、OK を選択し、ダイアログボックスを閉じます。

次に、ユーザー設定用のパブリックキーポリシーを設定します。[ユーザー設定] > [ポリシー] > [Windows 設定] > [セキュリティ設定] > [パブリックキーポリシー] に移動します。ステップ 6 からステップ 21 までの手順に従って、ユーザー設定用のパブリックキーポリシーを設定します。

GPO とパブリックキーポリシーの設定が完了すると、ドメイン内のオブジェクトは AWS Private CA Connector for AD に証明書を要求し、AWS Private CA によって発行された証明書を取得します。

テンプレートを作成する

詳細については、「手順 コネクタテンプレートの作成」を参照してください。

AD グループ許可を管理する

詳細については、「手順 テンプレートの AD グループと許可を管理する」を参照してください。

AWS Private CA Connector for Active Directory の手順

このセクションの手順では、コネクタの作成、テンプレートの設定、AWS Private CA と Active Directory の統合の方法について説明します。これらの操作は、AWS Private CA Connector for AD コンソールから、AWS CLI の Connector for AD セクションを使用するか、または AWS Private CA Connector for AD API を使用して実行できます。

> **Note**
>
> AWS Private CA Connector for AD は AWS Private CA と緊密に統合されていますが、この 2 つのサービスには別々の API があります。詳細については、 AWS Private Certificate Authority API リファレンスおよび AWS Private CA Connector for Active Directory API リファレンスを参照してください。

手順

- コネクタの作成
- コネクタテンプレートの作成
- Active Directory のコネクタの一覧表示
- コネクタテンプレートの一覧表示
- コネクタの詳細の表示
- コネクタテンプレートの詳細を表示する
- ディレクトリ登録の管理
- テンプレートの AD グループと許可を管理する
- サービスプリンシパル名の設定
- Connector for AD リソースのタグ付け

コネクタの作成

以下の手順に従って、AWS Private CA Connector for Active Directory のコンソール、コマンドライン、または API を使用してコネクタを作成します。

コネクタの作成 (コンソール)

AWS コンソールを使用してコネクタの作成と設定を行うには、次の手順を実行します。

タスク

- コンソールを開く
- [コネクタを作成] を開く
- ディレクトリを選択または作成する
- プライベート CA の選択
- タグ付けの設定

- 確認と作成

コンソールを開く

AWS アカウントにサインインし、**https://console.aws.amazon.com/pca-connector-ad/home** の AWS Private CA Connector for Active Directory コンソールを開きます。

[コネクタを作成] を開く

初回サービスのランディングページまたは [Connectors for Active Directory] ページで、[コネクタを作成] を選択します。

ディレクトリを選択または作成する

[プライベート CA Connector for Active Directory を作成] ページで、[Active Directory] セクションに情報を入力します。

- [Active Directory タイプを選択] で、次の 2 つの使用可能なタイプのいずれかを選択します。
  - AWS Directory Service for Microsoft Active Directory – によって管理される Active Directory を指定しますAWS Directory Service。
  - AWS AD Connector を備えたオンプレミス Active Directory — AD Connector を使用して、オンプレミスでホストしている Active Directory にアクセスします。
- [ディレクトリを選択] で、リストからディレクトリを選択します。

  または、[ディレクトリを作成] を選択して、新しいウィンドウに AWS Directory Service コンソールを開くこともできます。新しいディレクトリの作成が完了したら、AWS Private CA Connector for Active Directory コンソールに戻り、ディレクトリのリストを更新します。新しいディレクトリが選択可能になっているはずです。

  > **Note**
  >
  > ディレクトリを作成する場合、Connector for AD は AWS Directory Service コンソールで提供される次のディレクトリタイプのみをサポートしていることに注意してください。
  >
  > - AWS Managed Microsoft AD
  > - AD Connector

- [VPC エンドポイントのセキュリティグループを選択] で、リストからセキュリティグループを選択します。

または、[セキュリティグループを作成] を選択して、Amazon EC2 コンソールから [セキュリティグループを作成] ページを新しいウィンドウで開くこともできます。セキュリティグループの作成が完了したら、AWS Private CA Connector for Active Directory コンソールに戻り、セキュリティグループのリストを更新します。新しいセキュリティグループが選択可能になっているはずです。

プライベート CA の選択

[プライベート認証機関] セクションで、リストからプライベート CA を選択します。

または、[プライベート CA を作成] を選択して、AWS Private CA コンソールから [プライベート認証機関] ページを新しいウィンドウで開くこともできます。CA の作成が完了したら、AWS Private CA Connector for Active Directory コンソールに戻り、CA のリストを更新します。新しい CA が選択可能になっているはずです。

タグ付けの設定

[タグ — オプション] ペインでは、AD リソースのメタデータを適用したり削除したりできます。タグはキーと値の文字列ペアで、キーはリソース固有のものでなければならず、値は任意です。このペインでは、リソースの既存のタグがテーブルに表示されます。以下のアクションがサポートされています。

- [タグの管理] を選択して [タグの管理] ページを開きます。
- [新しいタグを追加] を選択してタグを作成します。[キー] フィールドと、任意で [値] フィールドに情報を入力します。[変更を保存] を選択してタグを適用します。
- タグの横にある [削除] ボタンを選択して削除対象としてマークし、[変更を保存] を選択して確定します。

確認と作成

必要な情報を入力して選択内容を確認したら、[コネクタを作成] を選択します。[Connectors for Active Directory] の詳細ページが開き、コネクタの作成中に進行状況を確認できます。

コネクタの作成プロセスが完了したら、そのコネクタにサービスプリンシパル名を割り当てます。

Active Directory のコネクタを作成する (AWS CLI)

CLI を使用して Active Directory のコネクタを作成するには、AWS CLI の AWS Private CA Connector for Active Directory セクションで create-connector コマンドを使用します。

Active Directory のコネクタを作成する (API)

API を使用して Active Directory 用のコネクタを作成するには、 AWS Private CA Connector for Active Directory API の CreateConnectorアクションを使用します。

コネクタテンプレートの作成

コネクタテンプレートの作成 (コンソール)

AWS コンソールを使用してコネクタテンプレートの作成と設定を行うには、次の手順を実行します。

タスク

- コンソールを開く
- コネクタの選択
- テンプレートセクションを見つける
- テンプレート作成方法
- テンプレートの設定
- 証明書設定の構成
- リクエスト処理と登録の設定を構成する
- キー使用拡張の設定
- アプリケーションポリシーの割り当て
- カスタムアプリケーションポリシーの設定
- 暗号化設定の構成
- グループと許可の設定
- 置き換えるテンプレートの設定
- タグ付けの設定
- 確認と作成

コンソールを開く

AWS アカウントにサインインし、**https://console.aws.amazon.com/pca-connector-ad/home** の AWS Private CA Connector for Active Directory コンソールを開きます。

コネクタの選択

[Connectors for Active Directory] リストからコネクタを選択し、[詳細を表示] を選択します。

テンプレートセクションを見つける

コネクタの詳細ページで [テンプレート] セクションを見つけ、[テンプレートを作成] を選択します。

テンプレート作成方法

[テンプレートを作成] ページの [テンプレート作成方法] セクションで、いずれかの方法オプションを選択します。

- [事前定義されたテンプレートから開始] (デフォルト) — AD アプリケーション用の定義済みテンプレートのリストから選択します。
  - [コード署名]
  - [コンピュータ]
  - [ドメインコントローラーの認証]
  - [EFS 回復エージェント]
  - [登録エージェント]
  - [登録エージェント (コンピュータ)]
  - [IPSec]
  - [Kerberos 認証]
  - [RAS および IAS サーバー]
  - [スマートカードログオン]
  - [信頼リストの署名]
  - [ユーザー署名]
  - [ワークステーション認証]
- [作成した既存のテンプレートから開始] — 以前に作成したカスタムテンプレートのリストから選択します。
- [空白のテンプレートから開始] — まったく新しいテンプレートの作成を開始するには、このオプションを選択します。

テンプレートの設定

[テンプレートの設定] セクションで、以下の情報を指定します。

- [テンプレート名] — テンプレートの名前。
- [テンプレートスキーマのバージョン] — テンプレートのスキーマバージョン。スキーマのバージョンは、テンプレートオプションの可用性に次のように影響します。

  スキーマバージョン 2

  - Windows XP / Windows Server 2003 以降のクライアント互換性をサポートします。
  - 従来の暗号化サービスプロバイダーのみをサポートします。

  スキーマバージョン 3

  - Windows Vista / Windows Server 2008 以降のクライアント互換性をサポートします。
  - リクエスト元が既存のキーで更新できるようにすることをサポートします。
  - キーストレージプロバイダーのみをサポートします。

  スキーマバージョン 4

  - Windows 8 / Windows Server 2012 以降のクライアント互換性をサポートします。
  - リクエスト元が既存のキーで更新できるようにすることをサポートします。
  - 従来の暗号化サービスプロバイダーとキーストレージプロバイダーをサポートします。
- [クライアントの互換性] — テンプレートと互換性のある最低オペレーティングシステムレベル。リストにあるオプションのいずれかを選択します。
  - [Windows XP / Windows Server 2003]
  - [Windows Vista / Windows Server 2008]
  - [Windows 7 / Windows Server 2008 R2]
  - [Windows 8 以降 / Windows Server 2012]
  - [Windows 8 以降 / Windows Server 2012 R2]
  - [Windows 8 以降 / Windows Server 2016 以降]

証明書設定の構成

[証明書の設定] セクションで、このテンプレートに基づく証明書について以下の設定を定義します。

- [証明書タイプ] — [ユーザー] 証明書と [コンピュータ] 証明書のどちらを作成するかを指定します。
- [自動登録] — このテンプレートに基づいて証明書の自動登録を有効にするかどうかを選択します。

- [有効期間] — 証明書の有効期間を、時間、日、週、月、または年の整数値で指定します。最小値は 2 時間です。
- [更新期間] — 証明書の更新期間を、時間、日、週、月、または年の整数値で指定します。更新期間は有効期間の 75% を超えないようにする必要があります。
- [サブジェクト名] — Active Directory に含まれる情報に基づいて、サブジェクト名に含めるオプションを 1 つ以上選択します。

> **Note**
>
> 少なくとも 1 つのサブジェクト名またはサブジェクト代替名オプションを指定する必要があります。

  - [共通名]
  - [共通名としての DNS]
  - [ディレクトリパス]
  - Email(メール)
- [サブジェクトの代替名] — Active Directory に含まれる情報に基づいて、サブジェクトの代替名に含めるオプションを 1 つ以上選択します。

> **Note**
>
> 少なくとも 1 つのサブジェクト名またはサブジェクト代替名オプションを指定する必要があります。

  - [ディレクトリ GUID]
  - [DNS 名]
  - [ドメイン DNS]
  - Email(メール)
  - [サービスプリンシパル名 (SPN)]
  - [ユーザープリンシパル名 (UPN)]

リクエスト処理と登録の設定を構成する

[証明書のリクエスト処理と登録のオプション] セクションで、以下のオプションのいずれかを選択して、テンプレートに基づいて証明書の目的を指定します。

- 署名
- 暗号化
- [署名と暗号化]
- [署名とスマートカードのログオン]

次に、次の機能のうち有効にするものを選択します。オプションは、証明書の目的によって異なります。

- [無効な証明書を削除 (アーカイブしない)]
- [対称アルゴリズムを含める]
- [エクスポート可能なプライベートキー]

最後に、証明書登録オプションを選択します。オプションは、証明書の目的によって異なります。

- [ユーザー入力は不要です]
- [登録中にユーザーにプロンプトを表示]
- [登録中にユーザーにプロンプトを表示し、ユーザー入力を要求する]

キー使用拡張の設定

[キー使用拡張の設定] セクションで、署名と暗号化キーの使用に関するオプションを選択します。

[[署名キーの使用]]

- [デジタル署名]
- [署名は発行元の証明である (非否認)]

[暗号化キーの使用]

- [キー暗号化しないキー交換を許可する (鍵の承諾)]
- [キー暗号化したキー交換のみを許可する (キー暗号化)]

- [ユーザーデータの暗号化を許可 (データ暗号化)]

また、どちらの種類のキーでも、[キー使用の拡張をクリティカルにする] を選択することができます。

アプリケーションポリシーの割り当て

[アプリケーションポリシー] セクションで、適用するアプリケーションポリシーをすべて選択します。使用可能なポリシーは、複数のページにまたがって表示されます。一部のポリシーは、以前の設定により事前に選択されている場合があります。

カスタムアプリケーションポリシーの設定

[カスタムアプリケーションポリシー] セクションでは、テンプレートにカスタム OID を追加し、アプリケーションポリシーの拡張が重要かどうかを指定できます。

暗号化設定の構成

[暗号化設定] セクションで、このテンプレートに基づく証明書の暗号化設定を以下のカテゴリから選択します。

1. セクション上部の内容は、以前に選択した テンプレート作成方法 と テンプレートの設定 によって決まります。

   - テンプレートの設定 でデフォルトの [テンプレートバージョン 2] をそのまま使用すると、次のステータスメッセージがここに表示されます。
     - [暗号化プロバイダーのカテゴリ]
     - [レガシー暗号化サービスプロバイダー]

     この場合、構成する設定はなく、次のステップに進むことができます。
   - テンプレートの設定 で [テンプレートバージョン 3] を指定した場合、次のステータスメッセージがここに表示されます。
     - [暗号化プロバイダーのカテゴリ]
     - [キーストレージプロバイダー]

     また、表示されているオプション [ECDH_P256]、[ECDH_P384]、[ECDH_P521]、[RSA] から [キーアルゴリズム] を選択する必要があります。
   - テンプレートの設定 で [テンプレートバージョン 4] を指定した場合は、[キーストレージプロバイダー] と [レガシー暗号化サービスプロバイダー] のどちらかを選択する必要がありま

す。[キーストレージプロバイダー] を選択した場合は、[キーアルゴリズム] も表示されているオプション [ECDH_P256]、[ECDH_P384]、[ECDH_P521]、[RSA] から選択する必要があります。

2. [最小キーサイズ (ビット)] — 最小キーサイズを指定します。この設定は、どの暗号化プロバイダーを利用できるかに影響します。
3. [リクエストに使用できる暗号プロバイダーを選択] — 次の 2 つの使用可能なオプションから 1 つを選択します。

   - [リクエストは対象のコンピュータで使用可能な任意のプロバイダーを使用できます]
   - [リクエストは次の選択されたプロバイダーのいずれかを使用する必要があります]

     このオプションを選択すると、[暗号化プロバイダー]リストが開きます。[順序] 列のボタンを使用してプロバイダーを選択し、優先順位を付けることができます。次のプロバイダーがサポートされています。

     - [Microsoft Base 暗号化プロバイダー v1.0]
     - [Microsoft Base DSS およびディフィーヘルマン暗号化プロバイダー]
     - [Microsoft Base スマートカード暗号化プロバイダー]
     - [Microsoft DH SChannel 暗号化プロバイダー]
     - [Microsoft 拡張暗号化プロバイダー v1.0]
     - [Microsoft 拡張 DSS およびディフィーヘルマン暗号化プロバイダー]
     - [Microsoft 拡張 RSA および AES 暗号化プロバイダー]
     - [Microsoft RSA SChannel 暗号化プロバイダー]

グループと許可の設定

[グループと許可] セクションでは、登録に関するテンプレート、既存のグループ、許可を確認できます。また、[新しいグループと許可を追加] ボタンを選択して新しいグループと許可を追加することもできます。このボタンで次の情報を要求するフォームが開きます。

- Display name (表示名)
- [セキュリティ識別子] (SID)
- [登録] (ALLOW | DENY | NOT SET のオプション)
- [自動登録] (ALLOW | DENY | NOT SET のオプション)

置き換えるテンプレートの設定

[テンプレートを置き換える] セクションでは、AD で作成された 1 つ以上のテンプレートが現在のテンプレートで置き換わることを Active Directory に通知できます。[Active Directory の置き換えるテンプレートを追加] を選択し、置き換えるテンプレートの共通名を指定することで、置き換えるテンプレートを適用します。

タグ付けの設定

[タグ — オプション] ペインでは、AD リソースのメタデータを適用したり削除したりできます。タグはキーと値の文字列ペアで、キーはリソース固有のものでなければならず、値は任意です。このペインでは、リソースの既存のタグがテーブルに表示されます。以下のアクションがサポートされています。

- [タグの管理] を選択して [タグの管理] ページを開きます。
- [新しいタグを追加] を選択してタグを作成します。[キー] フィールドと、任意で [値] フィールドに情報を入力します。[変更を保存] を選択してタグを適用します。
- タグの横にある [削除] ボタンを選択して削除対象としてマークし、[変更を保存] を選択して確定します。

確認と作成

必要な情報を入力して選択内容を確認したら、[テンプレートを作成] を選択します。すると [テンプレートの詳細] が開き、新しいテンプレートの設定の確認、テンプレートの編集または削除、グループと許可の管理、優先されるテンプレートの管理、タグの管理、証明書保有者の自動再登録の設定を行うことができます。

コネクタテンプレートの作成 (CLI)

AWS CLI の AWS Private CA Connector for Active Directory セクションで create-template コマンドを使用します。

コネクタテンプレートの作成 (API)

AWS Private CA Connector for Active Directory API で CreateTemplate アクションを使用します。

Active Directory のコネクタの一覧表示

AWS Private CA Connector for Active Directory コンソールまたは AWS CLI を使用して、所有しているコネクタの一覧を表示することもできます。

コンソールを使用してコネクタの一覧を表示するには

1. AWS アカウントにサインインし、**https://console.aws.amazon.com/pca-connector-ad/home** の AWS Private CA Connector for Active Directory コンソールを開きます。
2. [Connectors for Active Directory] リストの情報を確認します。右上のページ番号を使用すると、コネクタの複数のページを移動できます。各コネクタは 1 行で、デフォルトでは以下の情報列が表示されます。

- [コネクタ ID] — コネクタの一意の ID。
- [ディレクトリ名] — コネクタに関連付けられた Active Directory リソース。
- [コネクタの状態] — コネクタの状態。取り得る値は、[作成中] | [アクティブ] | [削除中] | [失敗] です。
- [サービスプリンシパル名ステータス] — コネクタに関連付けられているサービスプリンシパル名 (SPN) のステータス。取り得る値は、[作成中] | [アクティブ] | [削除中] | [失敗] です。
- [ディレクトリ登録ステータス] — アソシエイトディレクターの登録ステータス。取り得る値は、[作成中] | [アクティブ] | [削除中] | [失敗] です。
- [作成時刻] — コネクタ作成時のタイムスタンプ。

コンソールの右上隅の歯車アイコンを選択すると、ページに表示されるコネクタの数を、[ページサイズ] 設定を使用して表示するコネクタの数をカスタマイズできます。

AWS CLI を使用してコネクタの一覧を表示するには

list-connectors コマンドを使用してコネクタの一覧を表示します。

API を使用してコネクタの一覧を表示するには

AWS Private CA Connector for Active Directory API で ListConnectors アクションを使用します。

コネクタテンプレートの一覧表示

AWS Private CA Connector for Active Directory コンソールまたは AWS CLI を使用して、所有しているコネクタのテンプレートの一覧を表示することもできます。コネクタテンプレートは AWS Private CA BlankEndEntityCertificate_APIPassthrough/V1 テンプレートに基づいています。

コンソールを使用してテンプレートの一覧を表示するには

1. AWS アカウントにサインインし、**https://console.aws.amazon.com/pca-connector-ad/home** の AWS Private CA Connector for Active Directory コンソールを開きます。
2. [Connectors for Active Directory] リストからコネクタを選択し、[詳細を表示] を選択します。
3. コネクタの詳細ページで、[テンプレート] セクションの情報を確認します。右上のページ番号を使用すると、テンプレートの複数のページを移動できます。各テンプレートは 1 行で、以下の情報列が表示されます。

- [テンプレート名] — 人が読めるテンプレート名。
- [テンプレートのステータス] — テンプレートのステータス。取り得る値は、[アクティブ] | [削除中] です。
- [テンプレート ID] — テンプレートの一意識別子。

AWS CLI を使用してテンプレートの一覧を表示するには

list-templates コマンドを使用して、指定したコネクタのテンプレートの一覧を表示します。

API を使用してテンプレートの一覧を表示するには

AWS Private CA Connector for Active Directory API で ListTemplates アクションを使用して、指定したコネクタのテンプレートの一覧を表示します。

コネクタの詳細の表示

以下の手順に従って、AWS Private CA Connector for Active Directory のコンソール、コマンドライン、または API でコネクタの設定の詳細を表示します。

コネクタの表示 (コンソール)

コネクタの詳細を表示するには (コンソール)

1. AWS アカウントにサインインし、**https://console.aws.amazon.com/pca-connector-ad/home** の AWS Private CA Connector for Active Directory コンソールを開きます。
2. [Connectors for Active Directory] リストからコネクタを選択し、[詳細を表示] を選択します。

3. コネクタの詳細ページで、[コネクタの詳細] ペインの情報を確認します。内容は次のとおりです。

   - [コネクタ ID]
   - [コネクタステータス]
   - [その他のステータス詳細]
   - [コネクタ ARN]
   - [証明書登録ポリシーサーバーエンドポイント]
   - [ディレクトリ名]
   - [ディレクトリ ID]
   - AWS Private CA サブジェクト
   - AWS Private CA ステータス
   - [VPC エンドポイントとセキュリティグループ]
4. [テンプレート] ペインでは、コネクタに関連付けられているテンプレートを作成または管理できます。
5. [サービスプリンシパル名 (SPN)] ペインでは、コネクタに関連付けられているサービスプリンシパル名を表示できます。
6. [ディレクトリ登録] ペインでは、コネクタに関連付けられているディレクトリ登録を表示または変更できます。
7. [タグ — オプション]ペインでは、コネクタに関連付けられたタグを作成または管理できます。

コネクタの表示 (CLI)

AWS CLI の AWS Private CA Connector for Active Directory セクションで get-connector マンドを使用します。

コネクタの表示 (API)

AWS Private CA Connector for Active Directory API で GetConnector アクションを使用します。

コネクタテンプレートの詳細を表示する

AWS Private CA Connector for Active Directory のコンソール、コマンドライン、または API を使用して、コネクタテンプレートの設定の詳細を表示するには、以下の手順に従います。

テンプレートの表示 (コンソール)

コネクタテンプレートの詳細を表示するには (コンソール)

1. AWS アカウントにサインインし、**https://console.aws.amazon.com/pca-connector-ad/home** の AWS Private CA Connector for Active Directory コンソールを開きます。
2. [Connectors for Active Directory] リストからコネクタを選択し、[詳細を表示] を選択します。
3. コネクタの詳細ページで、[テンプレート] セクションの情報を確認し、検査するテンプレートを選択します。[詳細を表示] を選択します。
4. 詳細ページの [テンプレートの詳細] ペインには、テンプレートに関する次の情報が表示されます。

   - [テンプレート名]
   - [テンプレート ID]
   - [テンプレートのステータス]
   - [テンプレートスキーマのバージョン]
   - [テンプレートのバージョン]
   - [テンプレート ARN]
   - [証明書タイプ]
   - [自動登録が有効になっています]
   - [有効期間]
   - [更新期間]
   - [サブジェクト名の要件]
   - [サブジェクトの代替名の要件]
   - [証明書のリクエストと登録の設定]
   - [暗号化プロバイダーのカテゴリ]
   - [キーアルゴリズム]
   - [最小キーサイズ (ビット)]
   - ハッシュアルゴリズム
   - [暗号化プロバイダー]
   - [キー使用拡張の設定]

このペインでは、[編集]、[削除]、および [アクション] ボタンを使用して次のアクションを実行することもできます。

- [Edit] (編集)
- [Delete] (削除)
- [グループと許可を管理] — 詳細については、「グループとアクセス許可の設定」を参照してください。
- [優先されるテンプレートを管理] — 詳細については、「レビューと作成」を参照してください。
- [タグの管理] — 詳細については、「Connector for AD リソースのタグ付け」を参照してください。
- [すべての証明書所有者を再登録] — この設定により、テンプレートのメジャーバージョンが自動的に増えるようになります。テンプレートによる登録が許可されている Active Directory グループのすべてのメンバーは、そのテンプレートを使用して発行された新しい証明書を受け取ります。詳細については、「UpdateTemplate API」を参照してください。

5. 下のペインには、テンプレートの設定を変更できるタブが一列に表示されます。

- [グループと許可] — このテンプレートを使用して証明書を登録する Active Directory グループの許可を表示および管理します。詳細については、「グループとアクセス許可を設定する」を参照してください。
- [アプリケーションポリシー] — テンプレートアプリケーションポリシーを表示および管理します。詳細については、「アプリケーションポリシーの割り当て」を参照してください。
- [優先されるテンプレート] — 優先されるテンプレートを表示および管理します。詳細については、「レビューと作成」を参照してください。
- [タグオプション] — このテンプレートに対するタグ付けを表示および管理します。詳細については、「Connector for AD リソースのタグ付け」を参照してください。

コネクタテンプレートの詳細を表示するには (AWS CLI)

テンプレートの表示 (CLI)

AWS CLI の AWS Private CA Connector for Active Directory セクションで get-template コマンドを使用します。

テンプレートの表示 (API)

コネクタテンプレートの詳細を表示するには (API)

AWS Private CA Connector for Active Directory API で GetTemplate アクションを使用します。

ディレクトリ登録の管理

ディレクトリ登録を管理するには (コンソール)

コネクタのディレクトリ登録は、AWS Private CA Connector for Active Directory コンソールのトップレベルから管理できます。このトピックでは、使用可能な管理オプションについて説明します。

1. AWS アカウントにサインインし、**https://console.aws.amazon.com/pca-connector-ad/home** の AWS Private CA Connector for Active Directory コンソールを開きます。
2. 左側のナビゲーションエリアで、[ディレクトリ登録] を選択します。
3. [ディレクトリ登録] ページには、以下のフィールドを含む登録済みディレクトリの表が表示されます。

   - [ディレクトリ ID] — ディレクトリの一意の ID
   - [ディレクトリ名] — ディレクトリドメインサイト名
   - [ディレクトリタイプ]
   - [登録済み] — 登録のステータス。サポートされている値は、CREATING | ACTIVE | DELETING | FAILED です。
   - [ディレクトリのステータス] — ディレクトリのステータス

   [ディレクトリを登録] を使用して新しい登録を作成できます。
4. 管理のために、表示されている登録の中から 1 つを選択できます。すると、[登録の詳細を表示] ボタンと [ディレクトリの登録解除] ボタンが有効になります。[登録の詳細を表示] ボタンをクリックすると、登録の詳細ページが開きます。
5. [ディレクトリ登録の詳細] ペインには次の情報が表示されます。

   - [ディレクトリドメインサイト名]
   - [ディレクトリ ID] — ディレクトリの一意の ID。リンクを選択すると、AWS Directory Service コンソールに移動します。
   - [ディレクトリタイプ]

- [ステータス] — ディレクトリのステータス
- [ディレクトリ登録 ARN] — ディレクトリ登録の Amazon リソース名
- [その他のステータス情報]

6. [コネクタおよびサービスプリンシパル名 (SPN)] ペインでは、コネクタの SPN を管理できます。詳細については、「コネクタの詳細表示」を参照してください。
7. [タグ — オプション] ペインでは、AD リソースのメタデータを適用したり削除したりできます。タグはキーと値の文字列ペアで、キーはリソース固有のものでなければならず、値は任意です。このペインでは、リソースの既存のタグがテーブルに表示されます。以下のアクションがサポートされています。

   - [タグの管理] を選択して [タグの管理] ページを開きます。
   - [新しいタグを追加] を選択してタグを作成します。[キー] フィールドと、任意で [値] フィールドに情報を入力します。[変更を保存] を選択してタグを適用します。
   - タグの横にある [削除] ボタンを選択して削除対象としてマークし、[変更を保存] を選択して確定します。

ディレクトリ登録を管理するには (CLI)

作成: AWS CLI のAWS Private CA Connector for Active Directory セクションで create-directory-registration コマンドを使用します。

取得: AWS CLI の AWS Private CA Connector for Active Directory セクションの get-directory-registration コマンド。

一覧表示: AWS CLI の AWS Private CA Connector for Active Directory セクションの list-directory-registrations コマンド。

削除: AWS CLI の AWS Private CA Connector for Active Directory セクションの delete-directory-registration コマンド。

ディレクトリ登録を管理するには (API)

作成: AWS Private CA Connector for Active Directory API の CreateDirectoryRegistration アクション。

取得: AWS Private CA Connector for Active Directory API の GetDirectoryRegistration アクション。

一覧表示: AWS Private CA Connector for Active Directory API の ListDirectoryRegistrations アクション。

削除: AWS Private CA Connector for Active Directory API の DeleteDirectoryRegistration アクション。

テンプレートの AD グループと許可を管理する

テンプレートのグループと許可を管理するには (コンソール)

既存のテンプレートのグループと許可は、テンプレートの詳細ページから管理できます。詳細については、「コネクタテンプレートの詳細を表示する」を参照してください。

特定のテンプレートの証明書を登録できるまたは登録できないグループに対する許可を設定します。グループのセキュリティ識別子 (SID) を指定します。次に、グループの登録許可と自動登録許可を設定します。自動登録を行うには、登録と自動登録の両方を「許可」に設定する必要があります。

Active Directory でグループセキュリティ識別子を検索する

以下のスクリプトを使用して Active Directory 内のグループセキュリティ識別子を検索できます。

```
$ Get-ADGroup -Identity "my_active_directory_group_name"
```

テンプレートのグループと許可を管理するには (CLI)

作成: AWS CLI の AWS Private CA Connector for Active Directory セクションの create-template-group-access-control-entry コマンド。

更新: AWS CLI の AWS Private CA Connector for Active Directory セクションの update-template-group-access-control-entry コマンド。

取得: AWS CLI の AWS Private CA Connector for Active Directory セクションの get-template-group-access-control-entry コマンド。

一覧表示: AWS CLI の AWS Private CA Connector for Active Directory セクションの list-template-group-access-control-entries コマンド。

削除: AWS CLI の AWS Private CA Connector for Active Directory セクションの delete-template-group-access-control-entries コマンド。

テンプレートのグループと許可を管理するには (API)

作成: AWS Private CA Connector for Active Directory API の CreateTemplateGroupAccessControlEntry アクション。

更新: AWS Private CA Connector for Active Directory API の UpdateTemplateGroupAccessControlEntry アクション。

取得: AWS Private CA Connector for Active Directory API の GetTemplateGroupAccessControlEntry アクション。

一覧表示: AWS Private CA Connector for Active Directory API の ListTemplateGroupAccessControlEntries アクション。

削除: AWS Private CA Connector for Active Directory API の DeleteTemplateGroupAccessControlEntry アクション。

サービスプリンシパル名の設定

サービスプリンシパル名を管理するには (コンソール)

既存の AD コネクタのサービスプリンシパル名 (SPN) は、コネクタの詳細ページから管理できます。詳細については、「ディレクトリ登録の管理」の「View connector details」を参照してください。

サービスプリンシパル名を管理するには (CLI)

作成: AWS CLI の AWS Private CA Connector for Active Directory セクションの create-service-principal-name コマンド。

取得: AWS CLI の AWS Private CA Connector for Active Directory セクションの get-service-principal-name コマンド。

リスト: AWS CLI の AWS Private CA Connector for Active Directory セクションの list-service-principal-names コマンド。

削除: AWS CLI の AWS Private CA Connector for Active Directory セクションの delete-service-principal-name コマンド。

サービスプリンシパル名 (API) を管理するには

作成: AWS Private CA Connector for Active Directory API の CreateServicePrincipalName アクション。

取得: AWS Private CA Connector for Active Directory API の GetServicePrincipalName アクション。

リスト: AWS Private CA Connector for Active Directory API の ListServicePrincipalNames アクション。

削除: AWS Private CA Connector for Active Directory API の DeleteServicePrincipalName アクション。

Connector for AD リソースのタグ付け

コネクタ、テンプレート、およびディレクトリの登録にタグを適用できます。タグ付けを行うとリソースにメタデータが追加され、整理と管理に役立ちます。

リソースのタグ付けを管理するには (コンソール)

既存リソースのタグ付けは、リソースの詳細ページで管理します。詳細については、次の手順を参照してください。

- コネクタテンプレートの詳細を表示する
- ディレクトリ登録の管理

リソースタグ付けを管理するには (CLI)

タグ付け: AWS CLI の AWS Private CA Connector for Active Directory セクションの tag-resource コマンド。

タグの一覧表示: AWS CLI の AWS Private CA Connector for Active Directory セクションの list-tags-for-resource コマンド。

タグ付け解除: AWS CLI の AWS Private CA Connector for Active Directory セクションの untag-resource コマンド。

リソースタグ付けを管理するには (API)

タグ付け: AWS Private CA Connector for Active Directory API の TagResource アクション。

タグの一覧表示: AWS Private CA Connector for Active Directory API の ListTagsForResource アクション。

タグ付け解除: AWS Private CA Connector for Active Directory API の UntagResource アクション。

重要 - 機密データを含むオブジェクトのラベルとしてタグを使用できます。ただし、タグそのものに個人を特定できる情報 (PII)、機微情報、機密情報を入れないでください。

AWS Private CA Connector for Simple Certificate Enrollment Protocol (SCEP) (プレビュー)

Connector for SCEP は のプレビューリリースであり、変更 AWS Private CA される可能性があります。

Connector for SCEP とは

Connector for Simple Certificate Enrollment Protocol (SCEP) AWS Private Certificate Authority は、SCEP 対応のモバイルデバイスとネットワーク機器にリンクします。Connector for SCEP を使用すると、 AWS Private CA を使用して証明書を発行し、SCEP デバイスを登録できます。Connector for SCEP は、一般的なモバイルデバイス管理 (MDM) システムで使用でき、SCEP をサポートするクライアントまたはエンドポイントで動作するように設計されています。

トピック

Connector for SCEP の機能

SCEP プロトコルのサポート - SCEP は、認証局 (CA) からデジタル ID 証明書を取得し、モバイルデバイスやネットワーク機器に配布するための広く採用されているプロトコルです。Connector for SCEP を使用すると、SCEP を使用してエンドポイントを登録できます。

モバイルデバイスの登録 - Connector for SCEP は、Microsoft Intune や Jamf Pro などの一般的な MDM システムで使用できます。

大規模な証明書の発行 - コネクタの SCEP エンドポイントを介して証明書をリクエストするように SCEP 対応デバイスを設定すると、クライアントは から証明書を自動的にリクエストできます AWS Private CA。

Connector for SCEP の使用を開始する方法

開始するには、 Connector for SCEP マネジメントコンソールからガイド付きウィザードを起動します。これにより、コネクタを作成し、コネクタで使用するプライベート CA を指定できます。これらのステップを完了すると、Connector for SCEP は、MDM システムまたはネットワーク機器に入力できるエンドポイントおよびその他の設定パラメータを提供します。MDM システムまたはネットワーク機器を設定すると、クライアントは自動的に に証明書をリクエストします AWS Private CA。Connector for SCEP の使用を開始する方法の詳細については、「」を参照してくださいAWS Private Certificate Authority Connector for SCEP の開始方法。

関連サービス

Connector for SCEP は、次の AWS サービスに関連しています。

- AWS Private Certificate Authority - 独自のプライベート CA を運用するための先行投資や継続的なメンテナンスコストなしで、可用性の高いプライベート CA サービス AWS Private CA を提供します。
- AWS Private CA Connector for Active Directory - Connector for AD は、Active Directory (AD) をにリンクします AWS Private CA。コネクタは、 から AD によって管理されるユーザーとマシン AWS Private CA への証明書の交換をブローカーします。

Connector for SCEP へのアクセス

Connector for SCEP コネクタは、次のいずれかのインターフェイスを使用して作成、アクセス、管理できます。

- AWS Management Console - Connector for SCEP へのアクセスに使用できるウェブインターフェイスを提供します。「 Connector for SCEP 管理コンソール」を参照してください。
- AWS Command Line Interface - Connector for SCEP など、さまざまな AWS のサービス用のコマンドを提供します。 AWS CLI は、Windows、macOSでサポートされています。詳細については、「AWS Command Line Interface」を参照してください。
- AWS SDKs - 言語固有の APIs を提供し、署名の計算、リクエストの再試行処理、エラー処理など、接続の詳細の多くを処理します。詳細については、「AWS Command Line Interface」を参照してください。
- Connector for SCEP API - HTTPS リクエストを使用して呼び出す低レベルの API アクションを提供します。Connector for SCEP API の使用は、サービスにアクセスする最も直接的な方法です。

ただし、 Connector for SCEP API では、アプリケーションがリクエストに署名するハッシュの生成やエラー処理など、低レベルの詳細を処理する必要があります。詳細については、「 Connector for SCEP API reference」を参照してください。

Connector for SCEP の料金

Connector for SCEP は、 の機能として追加料金 AWS Private CA なしで提供されます。コネクタの作成と更新に使用した AWS Private Certificate Authority オペレーションと証明書に対してのみ料金が発生します。

最新の AWS Private CA 料金情報については、「 の料金AWS Private Certificate Authority」を参照してください。AWS 料金計算ツール でコストを見積もることもできます。

Connector for SCEP の概念

> Connector for SCEP は のプレビューリリースであり、変更 AWS Private CA される可能性があります。

Connector for SCEP は、 のアドオン機能です AWS Private Certificate Authority。

Connector for SCEP の主な概念は次のとおりです。

証明書署名リクエスト (CSR)

: デジタル証明書を発行するために CA に提供される必要な情報。この情報には、パブリックキーと ID が含まれます。

チャレンジパスワード

: SCEP プロトコルは、CA から証明書を発行する前に、チャレンジパスワードを使用してリクエストを認証します。Connector for SCEP は、コネクタタイプに基づいて SCEP チャレンジパスワードを処理します。詳細については、「Connector for SCEP の仕組み」を参照してください。

証明書の失効

: 証明書の失効は、発行した証明書の有効期限が切れる前に失効するプロセスです。API、 AWS SDK、 AWS Command Line Interfaceまたは RevokeCertificateで を呼び出すことで、コネクタに関連付けられたプライベート CA 証明書を取り消すことができます AWS CloudFormation。

SCEP 用コネクタ

SCEP 対応デバイス AWS Private CA への SCEP リンク用のコネクタ。

モバイルデバイス管理

モバイルデバイス管理 (MDM) を使用すると、IT 管理者はスマートフォン、タブレット、その他のエンドポイントやデバイスでポリシーを制御、保護、適用できます。多くの MDM システムは、SCEP ベースの証明書登録用の組み込み統合を提供します。

SCEP

SCEP は、証明書を自動的に配布するための標準化プロトコル (RFC 8894) です。プロトコルは、デバイスが CA に証明書をリクエストするためのエンドポイントを提供します。SCEP はチャレンジパスワードを使用して、デバイスへの証明書の発行を承認します。SCEP は、一般的にモバイルデバイス管理 (MDM) システムおよびネットワーク機器に適用されます。MDM ソリューションを使用すると、IT 管理者はスマートフォン、タブレット、Apple ワークステーションなどの他のエンティティでポリシーを制御、保護、適用できます。Microsoft Intune、Apple MDM、Jamf Pro など、ほとんどの MDM ソリューションは SCEP をサポートしています。ルーター、ロードバランサー、Wi-Fi ハブ、VPN デバイス、ファイアウォールなどのほとんどのネットワーク機器は、証明書の自動登録に SCEP を使用します。

SCEP プロファイル

SCEP プロファイルには、証明書プロファイルの定義に使用される設定パラメータが含まれています。これには、証明書の有効期間、キーサイズ、SCEP 設定名、チャレンジパスワード、失敗した再試行回数と再試行間隔、および証明書の発行に関連するその他の情報が含まれます。MDM システムと証明書管理プラットフォームは通常、認証用の証明書をリクエストする SCEP プロファイルをクライアントに送信します。

Connector for SCEP の仕組み

Connector for SCEP は のプレビューリリースであり、変更 AWS Private CA される可能性があります。

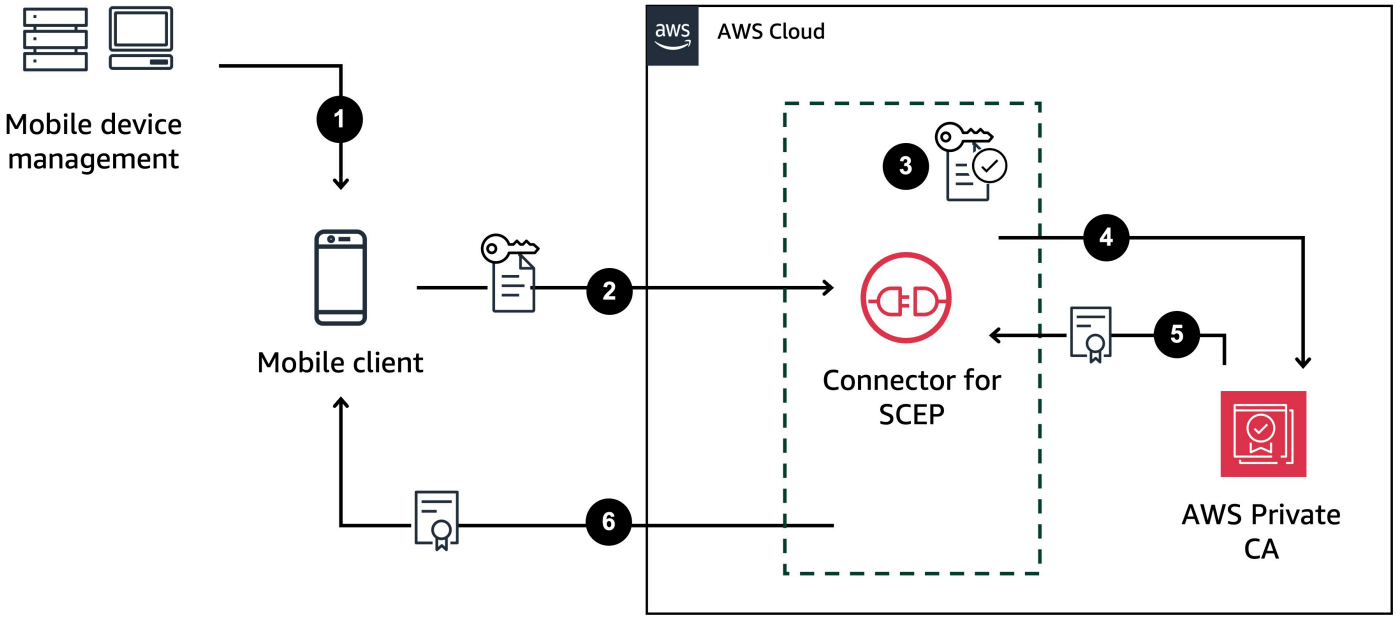
簡易証明書登録プロトコル (SCEP) は、証明書の登録と更新に使用される標準プロトコルです。Connector for SCEP は RFC 8894 ベースの SCEP サーバーで、 から SCEP クライアント AWS Private Certificate Authority に証明書を自動的に発行します。コネクタを作成すると、Connector for SCEP は SCEP クライアントが証明書をリクエストするための HTTPS エンドポイントを提供しま

す。クライアントは、サービスへの証明書署名リクエスト (CSR) の一部として含まれるチャレンジパスワードを使用して認証します。Connector for SCEP は、Microsoft Intune や Jamf Pro などの一般的なモバイルデバイス管理 (MDM) ソリューションでモバイルデバイスを登録するために使用できます。SCEP をサポートする任意のクライアントまたはエンドポイントで動作します。

Connector for SCEP には、汎用コネクタと Connector for SCEP for Microsoft Intune の 2 種類のコネクタが用意されています。以下のセクションでは、それらの仕組みについて説明します。

汎用

汎用コネクタは、特別なコネクタがある Microsoft Intune を除き、SCEP をサポートするエンドポイントで動作するように設計されています。汎用コネクタでは、SCEP チャレンジパスワードを管理します。次の図では、例としてモバイルデバイス管理 (MDM) システムを使用していますが、同様の SCEP 対応システムまたはデバイスを使用している場合も同じ機能が適用されます。



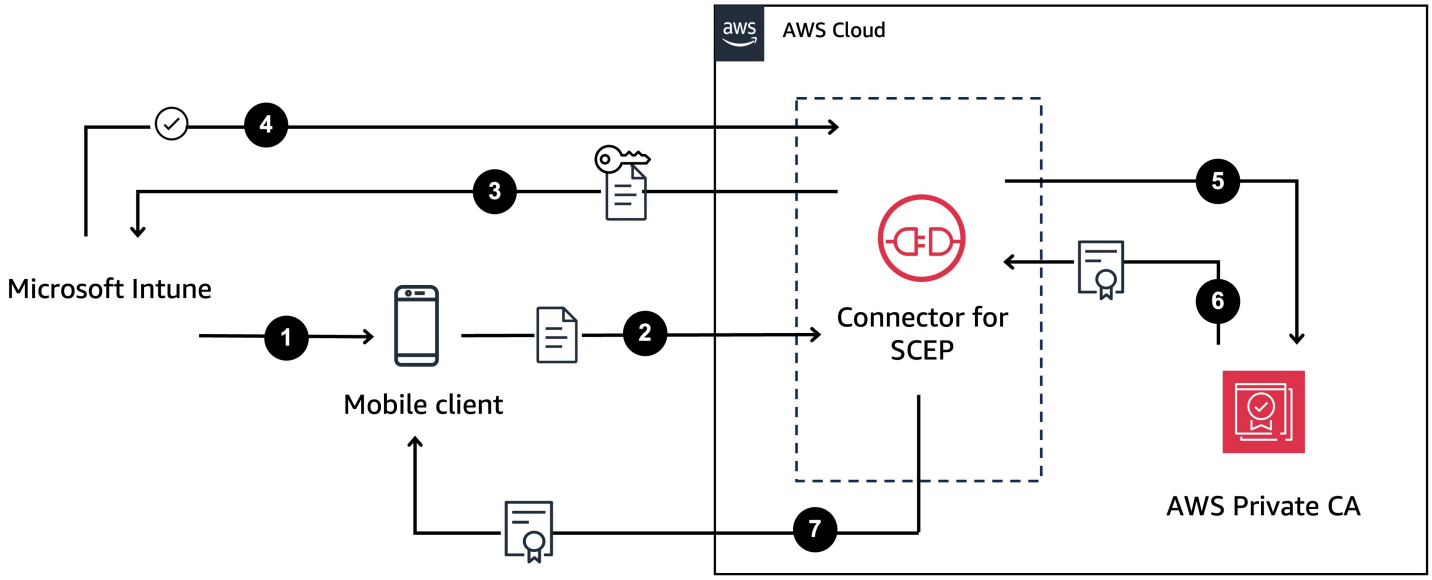
1. MDM システム (または類似のデバイスまたはシステム) は、SCEP プロファイルをモバイルクライアントに送信します。SCEP プロファイルには、証明書の有効期間、キーサイズ、SCEP 設定名、チャレンジパスワード、失敗した試行回数と再試行間隔、証明書の発行に関連するその他の情報など、証明書プロファイルの定義に使用される設定パラメータが含まれています。
2. モバイルクライアントは証明書をリクエストし、チャレンジパスワードを含む証明書署名リクエスト (CSR) も送信します。
3. Connector for SCEP はチャレンジパスワードを検証します。有効な場合、サービスはモバイルクライアント AWS Private CA に代わって から証明書をリクエストします。
4. AWS Private CA は証明書を発行し、Connector for SCEP に送信します。

5. Connector for SCEP は、発行された証明書をモバイルクライアントに送信します。

AWS Private Certificate Authority Microsoft Intune 用 SCEP 用コネクタ

AWS Private CA Connector for SCEP for Microsoft Intune は、Microsoft Intune で使用するように設計されています。Connector for SCEP for Microsoft Intune コネクタタイプでは、Microsoft Intune を使用して SCEP チャレンジパスワードを管理します。Microsoft Intune で Connector for SCEP を使用する方法の詳細については、「」を参照してくださいMicrosoft Intune の SCEP に Connector を使用する。

Connector for SCEP を Microsoft Intune で使用すると、Microsoft API から Microsoft Intune にアクセスすることで、特定の機能が有効になります。Connector for SCEP および付随する AWS サービスを使用しても、Microsoft Intune サービスを使用するための有効なライセンスは必要ありません。Microsoft Intune® アプリ保護ポリシー も確認してください。



1. Microsoft Intune は SCEP プロファイルをモバイルクライアントに送信します。プロファイルには、モバイルクライアントが CSR に配置する暗号化されたチャレンジパスワードが含まれています。
2. モバイルクライアントは証明書をリクエストし、CSR を Connector for SCEP に送信します。
3. Connector for SCEP は、認可のために CSR を Microsoft Intune に送信します。
4. Microsoft Intune は CSR のチャレンジパスワードを復号します。有効な場合、Microsoft Intune は Connector for SCEP に承認を送信し、モバイルクライアントに証明書を発行します。
5. Connector for SCEP は、モバイルクライアント AWS Private CA に代わって から証明書をリクエストします。

6. AWS Private CA は証明書を発行し、Connector for SCEP に送信します。
7. Connector for SCEP は、発行された証明書をモバイルクライアントに送信します。

Connector for SCEP を使用する際の考慮事項と制限事項

考慮事項

CA オペレーションモード

Connector for SCEP は、汎用オペレーションモードを使用するプライベート CAs でのみ使用できます。Connector for SCEP は、デフォルトで有効期間が 1 年の証明書を発行します。有効期間が短い証明書モードを使用するプライベート CA は、有効期間が 7 日を超える証明書の発行をサポートしていません。オペレーションモードの詳細については、「」を参照してください認証局モード。

チャレンジパスワード

- チャレンジパスワードは慎重に配布し、信頼度の高い個人やクライアントとのみ共有してください。1 つのチャレンジパスワードを使用して、任意のサブジェクト と SANsを持つ任意の証明書を発行できます。これにより、セキュリティ上のリスクが生じます。
- 汎用コネクタを使用する場合は、チャレンジパスワードを手動で頻繁にローテーションすることをお勧めします。

RFC 8894 への準拠

Connector for SCEP は、HTTP エンドポイントではなく HTTPS エンドポイントを提供することで、RFC 8894 プロトコルから逸脱します。

CSRs

- Connector for SCEP に送信される証明書署名リクエスト (CSR) に拡張キー使用 (EKU) 拡張機能が含まれていない場合は、EKU 値を に設定しますclientAuthentication。詳細については、「4.2.1.12」を参照してください。RFC 5280 での拡張キーの使用。
- CSR では、 ValidityPeriodおよびValidityPeriodUnitsカスタム属性がサポートされています。 CSRs CSR に が含まれていない場合ValidityPeriod、1 年間の有効期間を持つ証明書が発行されます。MDM システムでこれらの属性を設定できない場合があります。ただし、設定できる場合は、サポートされます。これらの属性の詳細については、szENROLLMENT_NAME_VALUE_PAIR」を参照してください。

エンドポイントの共有

コネクタのエンドポイントは、信頼された当事者にのみ配布します。エンドポイントはシークレットとして扱います。一意の完全修飾ドメイン名とパスを見つけられるすべてのユーザーが CA 証明書を取得できるためです。

制限事項

Connector for SCEP には以下の制限が適用されます。

動的チャレンジパスワード

静的チャレンジパスワードは、汎用コネクタでのみ作成できます。汎用コネクタで動的パスワードを使用するには、コネクタの静的パスワードを使用する独自のローテーションメカニズムを構築する必要があります。Connector for SCEP for Microsoft Intune コネクタタイプは、Microsoft Intune を使用して管理する動的パスワードをサポートします。

HTTP

Connector for SCEP は HTTPS のみをサポートし、HTTP コールのリダイレクトを作成します。システムが HTTP に依存している場合は、Connector for SCEP が提供する HTTP リダイレクトに対応できることを確認してください。

共有プライベート CAs

Connector for SCEP は、自分が所有者であるプライベート CAsでのみ使用できます。

Connector for SCEP のセットアップ

> Connector for SCEP は のプレビューリリースであり、変更 AWS Private CA される可能性があります。

このセクションの手順は、 Connector for SCEP の使用を開始するのに役立ちます。 AWS アカウントが既に作成されていることを前提としています。このページのステップを完了したら、SCEP 用のコネクタの作成に進むことができます。

トピック

- ステップ 1: AWS Identity and Access Management ポリシーを作成する

- ステップ 2: プライベート CA を作成する
- ステップ 3: を使用してリソース共有を作成する AWS Resource Access Manager

ステップ 1: AWS Identity and Access Management ポリシーを作成する

SCEP 用のコネクタを作成するには、コネクタに必要なリソースを作成および管理し、ユーザーに代わって証明書を発行する機能を Connector for SCEP に付与する IAM ポリシーを作成する必要があります。IAM の詳細については、「IAM ユーザーガイド」の「IAM とは」を参照してください。

次の例は、 Connector for SCEP に使用できるカスタマー管理ポリシーです。

```
{
    "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": {
                "StringLike": {
                    "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": "*"
        }
    ]
}
```

ステップ 2: プライベート CA を作成する

Connector for SCEP を使用するには、 からコネクタにプライベート CA AWS Private Certificate Authority を関連付ける必要があります。SCEP プロトコルに固有のセキュリティ脆弱性があるため、コネクタ専用のプライベート CA を使用することをお勧めします。

プライベート CA は、次の要件を満たしている必要があります。

- アクティブ状態で、汎用動作モードを使用する必要があります。
- プライベート CA を所有している必要があります。クロスアカウント共有を通じて共有されたプライベート CA を使用することはできません。

Connector for SCEP で使用するようにプライベート CA を設定するときは、次の考慮事項に注意してください。

- DNS 名の制約 — SCEP デバイス用に発行された証明書でどのドメインが許可または禁止されるかを制御する方法として、DNS 名の制約を使用することを検討してください。詳細については、「 で DNS 名制約を適用する方法 AWS Private Certificate Authority」を参照してください。
- 失効 — プライベート CA で OCSP CRLs を有効にして、失効を許可します。詳細については、「証明書失効方法の設定」を参照してください。

- PII – CA 証明書には、個人を特定できる情報 (PII) やその他の機密情報や機密情報を追加しないことをお勧めします。セキュリティが悪用された場合、これは機密情報の漏洩を制限するのに役立ちます。
- ルート証明書を信頼ストアに保存 – ルート CA 証明書をデバイス信頼ストアに保存して、証明書との戻り値を検証できるようにしますGetCertificateAuthorityCertificate。に関連する信頼ストアについては AWS Private CA、「」を参照してくださいルート CA 。

プライベート CA を作成する方法については、「」を参照してくださいプライベート CA の作成。

ステップ 3: を使用してリソース共有を作成する AWS Resource Access Manager

、 AWS SDK AWS Command Line Interface、または Connector for SCEP API をプログラムで使用して Connector for SCEP を使用している場合は、 AWS Resource Access Manager サービスプリンシパル共有を使用して、プライベート CA を Connector for SCEP と共有する必要があります。これにより、 Connector for SCEP がプライベート CA に共有アクセスできるようになります。 AWS コンソールでコネクタを作成すると、自動的にリソース共有が作成されます。リソース共有の詳細については、「 ユーザーガイド」の「リソース共有の作成AWS RAM 」を参照してください。

を使用してリソース共有を作成するには AWS CLI、 コマンドを使用できます AWS RAM create-resource-share。次のコマンドは、リソース共有を作成します。resource*-arns* の値として、共有するプライベート CA の ARN を指定します。

```
$  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
```

を呼び出すサービスプリンシパルCreateConnectorには、プライベート CA に対する証明書発行のアクセス許可があります。Connector for SCEP を使用するサービスプリンシパルが AWS Private CA リソースに一般アクセスできないようにするには、 を使用してアクセス許可を制限しますCalledVia。

AWS Private Certificate Authority Connector for SCEP の開始方法

Connector for SCEP は のプレビューリリースであり、変更 AWS Private CA される可能性があります。

AWS Private Certificate Authority Connector for SCEP を使用すると、プライベート CA から SCEP 対応デバイスおよびモバイルデバイス管理 (MDM) システムに証明書を発行できます。コネクタを作成すると、 は証明書をリクエストするためのパブリック SCEP URL AWS Private Certificate Authority を作成し、MDM システムに統合するために使用できる情報も提供します。

証明書を発行するには、 AWS Private Certificate Authority プライベート CA を作成し、コネクタを作成し、SCEP 対応の MDM システムとデバイスをコネクタに証明書をリクエストするように設定する必要があります。

トピック

開始する前に

次のチュートリアルでは、SCEP 用のコネクタを作成するプロセスについて説明します。

このチュートリアルに従うには、プライベート CA と SCEP 対応デバイスが必要です。また、まず Connector for SCEP のセットアップセクションに記載されている前提条件を満たす必要があります。

次の手順では、コンソールを使用してコネクタを作成する方法について説明します AWS 。

タスク

- ステップ 1: コネクタを作成する
- ステップ 2: コネクタの詳細を MDM システムにコピーする

ステップ 1: コネクタを作成する

汎用用のコネクタまたは Microsoft Intune 用 SCEP 用のコネクタを作成します。汎用コネクタは SCEP 対応エンドポイントで使用するように設計されており、SCEP チャレンジパスワードを管理します。Connector for SCEP for Microsoft Intune は Microsoft Intune で使用するためのもので、ユーザーは Microsoft Intune を使用してチャレンジパスワードを管理します。

General-purpose

汎用コネクタを作成するには

AWS アカウントにサインインし、 で Connector for SCEP コンソールを開きますhttps://console.aws.amazon.com/pca-connector-scep/home 。

1. [コネクターを作成] をクリックします。
2. コネクタの作成ページで、オプションで名前タグフィールドにわかりやすい名前をコネクタに付けます。コネクタのリストに名前が表示されます。必要に応じて、Add more tags を選択してコネクタにタグを追加できます。タグは、 AWS リソースに割り当てるラベルです。各タグは、キー、および値 (オプション) で構成されます。タグを使用して、リソースを検索およびフィルタリングしたり、 AWS コストを追跡したりできます。
3. コネクタタイプ で、汎用 を選択します。
4. プライベート CA で、このコネクタで使用するプライベート CA を選択します。または、プライベート CA の作成 を選択して新しい CA を作成します。SCEP プロトコルには固有の脆弱性があるため、このコネクタ専用のプライベート CA を使用することをお勧めします。新しい CA を作成した場合は、 で作成が完了したら AWS Private CA、 Connector for SCEP コンソールに戻り、プライベート CAsのリストを更新します。新しいプライベート CA を選択できるはずです。
5. チャレンジパスワード で、チャレンジパスワード を自動的に生成 を選択します。このコネクタを作成すると、静的チャレンジパスワードが生成されます。
6. コネクタの作成 を選択します。

Microsoft Intune

Microsoft Intune 用の SCEP 用コネクタを作成するには

AWS アカウントにサインインし、 で Connector for SCEP コンソールを開きますhttps://console.aws.amazon.com/pca-connector-scep/home 。

1. [コネクターを作成] をクリックします。
2. コネクタの作成ページで、オプションで名前タグフィールドにわかりやすい名前をコネクタに付けます。コネクタのリストに名前が表示されます。必要に応じて、Add more tags を選択してコネクタにタグを追加できます。タグは、 AWS リソースに割り当てるラベルです。各タグは、キー、および値 (オプション) で構成されます。タグを使用して、リソースを検索およびフィルタリングしたり、 AWS コストを追跡したりできます。
3. コネクタタイプ で、Microsoft Intune を選択します。

   a. アプリケーション (クライアント) ID に、Microsoft Entra ID アプリ登録からアプリケーション (クライアント) ID を入力します。Connector for SCEP で Microsoft Intune を使用する方法については、「」を参照してくださいMDM システムで SCEP に Connector を使用する。

   b. ディレクトリ (テナント) ID またはプライマリドメイン には、Microsoft Entra ID アプリ登録からディレクトリ (テナント) ID またはプライマリドメインを入力します。
4. プライベート CA で、このコネクタで使用するプライベート CA を選択します。または、プライベート CA の作成 を選択して新しい CA を作成します。SCEP プロトコルには固有の脆弱性があるため、このコネクタ専用のプライベート CA を使用することをお勧めします。新しい CA を作成した場合は、 で作成が完了したら AWS Private CA、 Connector for SCEP コンソールに戻り、プライベート CAsのリストを更新します。新しいプライベート CA を選択できるはずです。
5. コネクタの作成 を選択します。

ステップ 2: コネクタの詳細を MDM システムにコピーする

コネクタを作成したら、コネクタから MDM システムに次の詳細をコピーする必要があります。コンソールを使用してコネクタの詳細を表示するには、コネクタ for SCEP コンソールページのリストからコネクタを選択します。

- パブリック SCEP URL - これは、SCEP クライアントが証明書をリクエストするコネクタのエンドポイントです。このエンドポイントは信頼できるエンティティにのみ提供するよう注意してください。
- (汎用) チャレンジパスワード - チャレンジパスワード で、前の手順で自動生成したパスワードを選択し、パスワードを表示 を選択してパスワードを表示します。追加のパスワードを作成するには、「パスワードの作成」を選択します。パスワードは慎重に配布し、信頼できる個人とクライアントのみに配布してください。1 つのチャレンジパスワードを使用して、任意のサブジェクトと SANsで任意の証明書を発行できるため、慎重に処理する必要があります。

- (Microsoft Intune) Open ID 値 - Microsoft Intune と統合する場合は、Open ID 発行者 、Open ID 件名 、および Open ID 対象者を Microsoft Entra アプリ登録の OpenID Connect (OIDC) 認証情報にコピーする必要があります。詳細については、「MDM システムで SCEP に Connector を使用する」を参照してください。

MDM システムで SCEP に Connector を使用する

Connector for SCEP は のプレビューリリースであり AWS Private Certificate Authority 、変更される可能性があります。

以下のセクションでは、特定のモバイルデバイス管理 (MDM) システムで Connector for SCEP を使用する方法について説明します。

トピック

- Jamf Pro での SCEP 用コネクタの使用
- Microsoft Intune の SCEP に Connector を使用する

Jamf Pro での SCEP 用コネクタの使用

Jamf Pro モバイルデバイス管理 (MDM) ソリューションでは、 を外部認証機関 (CA) AWS Private CA として使用できます。このガイドでは、SCEP 用 AWS Private Certificate Authority コネクタを作成した後に Jamf Pro を SCEP プロキシとして設定する方法について説明します。

Jamf Pro の要件

Jamf Pro の実装は、次の要件を満たしている必要があります。

- Jamf Pro 10.0.0 以降を使用する必要があります。
- Jamf Pro で証明書ベースの認証を有効にする設定を有効にする必要があります。この設定の詳細については、Jamf Pro ドキュメントの Jamf Pro セキュリティ設定ページを参照してください。

前提条件

Jamf Pro で Connector for SCEP を使用するには、まずプライベート CA と SCEP 用の汎用コネクタを作成する必要があります。手順については、「Connector for SCEP のセットアップ」を参照してください。

Jamf Pro で外部 CA AWS Private CA として を設定する

SCEP 用のコネクタを作成したら、Jamf Pro で を外部 CA AWS Private CA として設定する必要があります。をグローバルの外部 CA AWS Private CA として設定できます。または、Jamf Pro 設定プロファイルを使用して、組織内のデバイスのサブセットに証明書を発行するなど、ユースケースごとに から AWS Private CA 異なる証明書を発行することもできます。Jamf Pro 設定プロファイルの実装に関するガイダンスは、このドキュメントの範囲外です。

Jamf Pro で を外部 CA AWS Private CA として設定するには

1. Jamf Pro コンソールで、設定 > グローバル > PKI 証明書 に移動して PKI 証明書の設定ページに移動します。
2. 管理証明書テンプレート タブを選択します。
3. 外部 CA を選択します。
4. [Edit] (編集) を選択します。
5. (オプション) 設定プロファイル の SCEP Proxy として Jamf Pro を有効にする を選択します。Jamf Pro 設定プロファイルを使用して、特定のユースケースに合わせたさまざまな証明書を発行できます。Jamf Pro で設定プロファイルを使用する方法のガイダンスについては、Jamf Pro ドキュメントの「Jamf Pro を設定プロファイルの SCEP プロキシとして有効にする」を参照してください。
6. コンピュータとモバイルデバイスの登録に SCEP 対応の外部 CA を使用する を選択します。
7. (オプション) コンピュータとモバイルデバイスの登録に SCEP プロキシとして Jamf Pro を使用する を選択します。プロファイルのインストールに障害が発生した場合は、「」を参照してくださいプロファイルのインストール失敗のトラブルシューティング。
8. Connector for SCEP パブリック SCEP URL をコネクタの詳細から Jamf Pro の URL フィールドにコピーして貼り付けます。コネクタの詳細を表示するには、コネクタ for SCEP リストからコネクタを選択します。または、 を呼び出して URL を取得しGetConnector、レスポンスからEndpoint値をコピーすることもできます。
9. (オプション) 名前フィールドにインスタンスの名前を入力します。例えば、 という名前を付けることができますAWS Private CA。

10. チャレンジタイプの静的 を選択します。
11. コネクタのチャレンジパスワードをコピーし、チャレンジフィールドに貼り付けます。コネクタのチャレンジパスワードを表示するには、 AWS コンソールでコネクタの詳細ページに移動し、パスワードの表示ボタンを選択します。または、Password を呼び出してレスポンスからPassword値をコピーすることで、コネクタのチャレンジGetChallengeパスワードを取得することもできます。
12. チャレンジパスワードをチャレンジの検証フィールドに貼り付けます。
13. キーサイズ を選択します。キーサイズは 2048 以上をお勧めします。
14. (オプション) デジタル署名 として使用する を選択します。Wi-Fi や VPN などのリソースへの安全なアクセスをデバイスに付与するには、認証目的でこれを選択します。
15. (オプション) キー暗号化 に使用する を選択します。
16. (オプション) フィンガープリントフィールドに 16 進数の文字列を入力します。プライベート CA のフィンガープリントを作成する方法については、「」を参照してください (オプション) CA フィンガープリントを追加する。
17. [Save] を選択します。

プロファイル署名証明書を作成してアップロードする

Jamf Pro で Connector for SCEP を使用するには、コネクタに関連付けられているプライベート CA の署名証明書と CA 証明書を提供する必要があります。これを行うには、両方の証明書を含むプロファイル署名証明書キーストアを Jamf Pro にアップロードします。内部プロセスを使用して証明書署名リクエスト (CSR) を生成し、 によって署名してもらう必要があります AWS Private Certificate Authority。次の手順では、証明書キーストアを作成して Jamf Pro にアップロードする方法について説明します。次の例では OpenSSL を使用していますが、任意の方法を使用して証明書署名リクエストを生成できます。

1. OpenSSL を使用して、次のコマンドを実行してプライベートキーを生成します。

```
openssl genrsa -out local.key 2048
```

2. 証明書署名リクエスト (CSR) を生成します。

```
openssl req -new -key local.key -sha512 -out local.csr -
subj "/CN=MySigningCertificate/O=MyOrganization" -addext
 keyUsage=critical,digitalSignature,nonRepudiation
```

3. を使用して AWS CLI、ステップ 2 で生成した CSR を使用して署名証明書を発行します。次のコマンドを実行し、レスポンスに証明書 ARN を書き留めます。

```
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. ステップ 3 の証明書 ARN を使用して次のコマンドを実行して、署名証明書を取得します。

```
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. 次のコマンドを実行して CA 証明書を取得します。

```
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. OpenSSL を使用して、署名証明書キーストアを p12 形式で出力します。ステップ 4 と 5 で生成したcrtファイルを使用します。次のコマンドを実行します。

```
openssl pkcs12 -export -in local.crt -inkey local.key -certfile ca.crt -name "CA
 Chain" -out local.p12
```

7. プロンプトが表示されたら、エクスポートパスワードを入力します。このパスワードはキーストアパスワードであり、後で使用する必要があります。
8. Jamf Pro で、管理証明書テンプレートに移動し、外部 CA ペインに移動します。
9. 外部 CA ペインの下部で、署名の変更と CA 証明書 を選択します。
10. 画面の指示に従って、外部 CA の署名証明書と CA 証明書をアップロードします。

(オプション) CA フィンガープリントを追加する

CA フィンガープリントを追加すると、マネージドデバイスは CA を検証し、CA からの証明書のみをリクエストできます。

1. AWS Private CA コンソールまたは を使用して、プライベート CA 証明書を取得しますGetCertificateAuthorityCertificate。ca.pem ファイルとして保存します。
2. OpenSSL で、次のコマンドを実行します。

```
openssl x509 -in ca.pem -sha256 -fingerprint
```

3. 前の手順で参照したフィンガープリントフィールドに出力をコピーして貼り付けます。

(オプション) ユーザーが開始した登録時に証明書をインストールする

ユーザーが開始した登録中にコネクタのプライベート CA 証明書をクライアントまたはデバイスにインストールするには、Jamf Pro ユーザーが開始した登録設定を構成します。これにより、Jamf Pro は AWS Private CA 証明書をリクエストするときに、クライアントまたはデバイスに証明書をインストールできます。Connector for SCEP の実装と互換性があることを確認するために、設定をテストするのはユーザーの責任です。Jamf Pro のユーザー主導登録設定の詳細については、Jamf Pro ドキュメントの「ユーザー主導登録設定」を参照してください。

プロファイルのインストール失敗のトラブルシューティング

コンピュータおよびモバイルデバイスの登録で SCEP Proxy として Jamf Pro を使用する を有効にした後にプロファイルのインストールに失敗する場合は、以下を試してください。

エラーメッセージ	緩和策
`Profile installation failed. Unable to obtain certificate from SCEP server at "<your-jamf-endpoint>.jamfcloud.com". <MDM-SCEP:15001>`	登録中にこのエラーメッセージが表示された場合は、登録を再試行してください。登録が成功するまでに数回の試行が必要になる場合があります。
`Profile installation failed. Unable to obtain certificate from SCEP server at "<your-jamf-endpoint>.jamfcloud.com". <MDM-SCEP:14006>`	チャレンジパスワードの設定が間違っている可能性があります。Jamf Pro のチャレンジパスワードがコネクタのチャレンジパスワードと一致していることを確認します。

Microsoft Intune の SCEP に Connector を使用する

Microsoft Intune モバイルデバイス管理 (MDM) システムでは、 を外部認証機関 (CA) AWS Private CA として使用できます。このガイドでは、Microsoft Intune 用 コネクタを作成した後に Microsoft Intune を設定する方法について説明します。

前提条件

Connector for SCEP for Microsoft Intune を作成する前に、次の前提条件を満たす必要があります。

- Entra ID を作成します。
- Microsoft Intune テナントを作成します。
- Microsoft Entra ID でアプリ登録を作成します。アプリケーション登録のアプリケーションレベルのアクセス許可を管理する方法については、Microsoft Entra ドキュメントの「Microsoft Entra ID でアプリケーションがリクエストしたアクセス許可を更新する」を参照してください。アプリ登録には、次のアクセス許可が必要です。
  - Intune で scep_challenge_provider を設定します。
  - Microsoft Graph の場合、Application.Read.All と User.Read を設定します。
- アプリケーション登録管理者の同意でアプリケーションを許可する必要があります。詳細については、Microsoft Entra ドキュメントの「テナント全体の管理者にアプリケーションへの同意を付与する」を参照してください。

> **Tip**
>
> アプリケーション登録を作成するときは、アプリケーション (クライアント) ID とディレクトリ (テナント) ID またはプライマリドメイン を書き留めます。Connector for SCEP for Microsoft Intune を作成するときは、これらの値を入力します。これらの値を取得する方法については、Microsoft Entra ドキュメントの「リソースにアクセスできる Microsoft Entra アプリケーションおよびサービスプリンシパルを作成する」を参照してください。

Microsoft Entra ID アプリケーションを使用するアクセス AWS Private CA 許可を付与する

Connector for SCEP for Microsoft Intune を作成したら、Microsoft アプリ登録でフェデレーション認証情報を作成して、Connector for SCEP が Microsoft Intune と通信できるようにする必要があります。

Microsoft Intune で を外部 CA AWS Private CA として設定するには

1. Microsoft Entra ID コンソールで、アプリ登録 に移動します。

2. Connector for SCEP で使用するように作成したアプリケーションを選択します。クリックするアプリケーションのアプリケーション (クライアント) ID は、コネクタの作成時に指定した ID と一致する必要があります。
3. マネージドドロップダウンメニューから証明書とシークレットを選択します。
4. フェデレーティッド認証情報タブを選択します。
5. 認証情報の追加 を選択します。
6. フェデレーティッド認証情報シナリオのドロップダウンメニューから、その他の発行者 を選択します。
7. Connector for SCEP for Microsoft Intune の詳細から OpenID 発行者の値をコピーして発行者フィールドに貼り付けます。コネクタの詳細を表示するには、 AWS コンソールの Connectors for SCEP リストからコネクタを選択します。または、 を呼び出して URL を取得しGetConnector、レスポンスからIssuer値をコピーすることもできます。
8. Connector for SCEP for Microsoft Intune の詳細から OpenID オーディエンス値をコピーして、オーディエンスフィールドに貼り付けます。コネクタの詳細を表示するには、 AWS コンソールの Connectors for SCEP リストからコネクタを選択します。または、 を呼び出して URL を取得しGetConnector、レスポンスからSubject値をコピーすることもできます。
9. (オプション) 名前フィールドにインスタンスの名前を入力します。例えば、 という名前を付けることができますAWS Private CA。
10. (オプション) 説明フィールドに説明を入力します。
11. 対象者フィールドで編集 (オプション) を選択します。OpenID サブジェクト値をコネクタからサブジェクトフィールドにコピーして貼り付けます。OpenID 発行者の値は、 AWS コンソールのコネクタの詳細ページで表示できます。または、 を呼び出して URL を取得しGetConnector、レスポンスからAudience値をコピーすることもできます。
12. [追加] を選択します。

Microsoft Intune 設定プロファイルを設定する

Microsoft Intune を呼び出す AWS Private CA アクセス許可を付与したら、Microsoft Intune を使用して、証明書の発行のために Connector for SCEP に連絡するようにデバイスに指示する Microsoft Intune 設定プロファイルを作成する必要があります。

1. 信頼できる証明書設定プロファイルを作成します。Connector for SCEP で使用しているチェーンのルート CA 証明書を Microsoft Intune にアップロードして、信頼を確立する必要があります。信頼できる証明書設定プロファイルを作成する方法については、Microsoft Intune ドキュメントの「Microsoft Intune の信頼できるルート証明書プロファイル」を参照してください。

2. 新しい証明書が必要な場合に、デバイスがコネクタを指す SCEP 証明書設定プロファイルを作成します。設定プロファイルのプロファイルタイプは SCEP 証明書 である必要があります。設定プロファイルのルート証明書には、前のステップで作成した信頼された証明書を使用していることを確認してください。

   SCEP サーバー URLs の場合、コネクタの詳細からパブリック SCEP URL をコピーして SCEP サーバー URLsフィールドに貼り付けます。コネクタの詳細を表示するには、コネクタ for SCEP リストからコネクタを選択します。または、 を呼び出して URL を取得しGetConnector、レスポンスからEndpoint値をコピーすることもできます。Microsoft Intune で設定プロファイルを作成する方法については、Microsoft Intune ドキュメントの「Microsoft Intune で SCEP 証明書プロファイルを作成して割り当てる」を参照してください。

   > **Note**
   >
   > Mac 以外の OS および iOS デバイスの場合、設定プロファイルに有効期間を設定しないと、Connector for SCEP は有効期間が 1 年の証明書を発行します。設定プロファイルで拡張キー使用量 (EKU) 値を設定しない場合、Connector for SCEP は で設定された EKU で証明書を発行しますClient Authentication (Object Identifier: 1.3.6.1.5.5.7.3.2)。macOS または iOS デバイスの場合、Microsoft Intune は設定プロファイルの ExtendedKeyUsageまたは Validityパラメータを尊重しません。これらのデバイスの場合、Connector for SCEP はクライアント認証を通じてこれらのデバイスに 1 年間の有効期間を持つ証明書を発行します。

Connector for SCEP への接続を確認する

Connector for SCEP エンドポイントを指す Microsoft Intune 設定プロファイルを作成したら、登録されたデバイスが証明書をリクエストできることを確認します。確認するには、ポリシーの割り当てに失敗していないことを確認します。確認するには、Intune ポータルでデバイス > デバイスの管理 > 設定に移動し、設定ポリシーの割り当て失敗 に何もリストされていないことを確認します。存在する場合は、前の手順の情報を使用してセットアップを確認します。セットアップが正しく、それでも失敗する場合は、「モバイルデバイスから利用可能なデータを収集する」を参照してください。

デバイス登録の詳細については、Microsoft Intune ドキュメントの「デバイス登録とは」を参照してください。

トラブルシューティング

AWS Private CAを使用する際に問題がある場合は、以下のトピックを参照してください。

トピック

プライベート CA 証明書の作成と署名

プライベート CA を作成したら、CSR を取得し、X.509 インフラストラクチャで中間 CA または ルート CA に送信する必要があります。お客様の CA は、CSR を使用してプライベート CA 証明書を作成し、その証明書に署名してからお客様に返します。

残念ながら、プライベート CA 証明書の作成と署名に関連する問題について具体的なアドバイスをすることはできません。X.509 インフラストラクチャおよびその中の CA 階層の詳細は、この AWS Private CA ドキュメントでは扱いません。

OCSP 応答のレイテンシー

発信者が地域のエッジキャッシュや発行元の CA の地域から地理的に離れていると、OCSP の応答が遅くなる可能性があります。リージョナルエッジキャッシュの可用性についての詳細は、「グローバルエッジネットワーク」を参照してください。証明書は、使用する地域に近い地域で発行することをお勧めします。

CRL バケットの作成を許可するように Amazon S3 を設定する

Amazon S3 の [パブリックアクセスブロック (バケット設定)] がアカウントで強制されている場合、プライベート CA が CRL バケットを作成できないことがあります。このような場合は、Amazon S3

設定を確認してください。詳細については、「Amazon S3 パブリックアクセスのブロックの使用」を参照してください。

自己署名の CA 証明書の取り消し

自己署名 CA 認定を取り消すことはできません。代わりに、CA を削除する必要があります。

例外処理

AWS Private CA コマンドは、いくつかの理由で失敗することがあります。各例外とその解決に関する推奨事項については、以下の表を参照してください。

AWS Private CA 例外

によって返された例外 AWS Private CA	説明	修正
`AccessDeniedException`	指定されたコマンドを使用するために必要なアクセス許可が、プライベート CA によって呼び出し側アカウントに委任されていません。	でのアクセス許可の委任については AWS Private CA、「」を参照してくださいACM に証明書の更新許可を割り当てる。
`InvalidArgsException`	証明書の作成または更新リクエストが、無効なパラメータを使用して行われました。	コマンドの各マニュアルを参照して、入力パラメータが有効であることを確認します。新しい証明書を作成する場合は、リクエストされた署名アルゴリズムが CA のキータイプで使用できることを確認してください。
`InvalidStateException`	関連付けられたプライベート CA が `ACTIVE` 状態ではないため、証明書を更新できません。	[プライベート CA を復元します] を試みます。プライベート CA が復元期間外にある場合、CA を復元できないため、証明書を更新できません。

によって返された例外 AWS Private CA	説明	修正
`LimitExceededException`	各認証機関 (CA) には、発行できる証明書のクォータがあります。指定された証明書に関連付けられているプライベート CA がクォータに達しました。詳細については、「 AWS 全般のリファレンス ガイド」の「Service Quotas」を参照してください。	クォータの引き上げをリクエストするには、 AWS Support センターにお問い合わせください。
`MalformedCSRException`	AWS Private CA に送信された証明書署名要求 (CSR) を、検証または検証できません。	CSR が正しく生成され、設定されていることを確認します。
`OtherException`	内部エラーにより、リクエストに失敗しました。	コマンドを再度実行してみます。問題が解決しない場合は、 AWS Support センターにお問い合わせください。
`RequestFailedException`	AWS 環境のネットワーク問題により、リクエストが失敗しました。	リクエストを再試行します。問題が解決しない場合は、Amazon VPC (VPC) 設定を確認してください。
`ResourceNotFoundException`	証明書を発行したプライベート CA は削除済みであり、存在しません。	別のアクティブ CA に新しい証明書をリクエストします。

によって返された例外 AWS Private CA	説明	修正
`ThrottlingException`	リクエストされた API アクションは、クォータを超過したため、失敗しました。	発行する呼び出しが、 AWS Private CAで許可された数を超えていないことを確認してください。 `ThrottlingException` エラーは、クォータを超過した状態ではなく、一時的な状態が発生したためにも発生することがあります。このエラーが発生し、発行したコールがクォータを超過していない場合、リクエストを再試行してください。 クォータに達した場合、増加をリクエストすることができます。詳細については、「 AWS 全般のリファレンスガイド」のService Quotas」を参照してください。
`ValidationException`	リクエストの入力パラメータが正しくフォーマットされていないか、ルート証明書の有効期間が、リクエストされた証明書の有効期間より前に終了します。	コマンドの入力パラメータの構文要件と、CA のルート証明書の有効期間を確認します。有効期間の変更については、「プライベート CA の更新」を参照してください 。

Matter 規格の使用

Matter 接続規格は、モノのインターネット (IoT) デバイスのセキュリティと一貫性を向上させる証明書構成を規定しています。Matter 準拠のルート CA、中間 CA、およびエンドエンティティ証明書を

作成するための Java サンプルは AWS Private CA API を使用した Matter 規格の実装 (Java の例) にあります。

トラブルシューティングに役立つように、Matter 開発者は chip-cert という証明書検証ツールを提供しています。次の表に、ツールが報告するエラーとその対処方法を示します。

エラーコード	意味	修正
0x0000035	BasicConstraints 、KeyUsage、Exte KeyUsage 拡張は「重要」とマークする必要があります。	ユースケースに適したテンプレートを選択していることを確認さい。
0x0000000	権限キー識別子拡張が存在している必要があります。	AWS Private CA は、ルート証明書に権限キー識別子拡張を設定ん。CSR を使用して Base64-encodedされたAuthorityKeyIdent生成し、それを に渡す必要がありますCustomExtension。詳細は、「ノード運用証明書 (NOC) のルート CA を有効にします。び「製品認証機関 (PAA) をアクティブ化する」を参照してくだ
0x000000E	証明書の有効期限が切れています。	使用する証明書の有効期限が切れていないことを確認してくだ
0x0000004	証明書チェーンの検証に失敗しました。	このエラーは、提供された Java の例 を使用せずに Matter 準拠ドエンティティ証明書を作成しようとすると発生する可能性がす。この例では、 AWS Private CA API を使用して適切に設定渡します KeyUsage。 デフォルトでは、 は 9 ビットの KeyUsage 拡張値 AWS Private生成し、9 番目のビットは余分なバイトになります。Matter はマット変換中に余分なバイトを無視するため、チェーン検証がす。ただし、APIPassthrough テンプレートCustomExtensio使用して、 KeyUsage値に正確なバイト数を設定できます。例は、ノード運用証明書 (NOC) を作成するを参照してください。 サンプルコードを変更したり、OpenSSL などの代替の X.509 リティを使用する場合は、チェーン検証エラーを回避するため検証を行う必要があります。

エラーコード	意味	修正

		変換が口スレスであることを確認するには
		
		1. openssl を使用して、ノード (エンドエンティティ) 証明書書に有効なチェーンが含まれていることを確認します。こは、rcac.pem はルート CA 証明書、icac.pem は中間 C書、noc.pem はノード証明書です。
		
		`openssl verify -verbose -CAfile <(cat rcac.pem icac noc.pem`
		
		2. chip-cert を使用して PEM 形式のノード証明書を TLV (タさ、値) 形式に変換し、元に戻します。
		
		`./chip-cert convert-cert noc.pem noc.chip -c`
		`./chip-cert convert-cert noc.chip noc_converted.pem`
		
		noc.pem および noc_converted.pem ファイルは、文ツールで確認されたものとまったく同じである必要があり

AD エラーと障害用のコネクタ

Connector for AD を使用する際のエラーや作成失敗の診断と修正には、こちらの情報を参考にしてください。

トピック

- エラー
- コネクタ作成の失敗
- SPN 作成の失敗

エラー

Connector for AD は、いくつかの理由でエラーメッセージを送信します。各エラーと解決に関する推奨事項については、次の表を参照してください。これらのエラーは、Amazon EventBridge Scheduler イベント (イベントソース: `aws.pca-connector-ad`) にサブスクライブするか、Windows で手動登録を使用して受信できます。

エラーコード	意味	修正
0x8FFFA000	Kerberos 認証 に失敗しました。	ディレクトリにアクセスできること、およびクライアントがユーザーまたはコンピュータであることを確認します。自動登録を使用している場合は、リソースサービスプリンシパルを修正します AWS 。Active Directory UI を使用して証明書を取得している場合は、`gpupdate /force` を実行してください。
0x8FFFA001	SOAP メッセージにはアクションヘッダーが含まれている必要があります。	アクションヘッダーの追加
0x8FFFA002	コネクタは接続先のプライベート CA にアクセスできません。	プライベート CA と Connector for AD サービスの間で共有する AWS Resource Access Manager (RAM) を作成して、プライベート CA をコネクタと共有します。
0x8FFFA003	このコネクタのプライベート CA はアクティブではありません。	プライベート CA をアクティブ状態に移行します。プライベート CA が証明書が保留中の状態の場合は、CA 証明書をインストールしてください。

エラーコード	意味	修正
0x8FFFA004	このコネクタのプライベート CA が存在しません。	認証局が削除済み状態の場合は、アクティブ状態に移行します。プライベート CA が完全に削除された場合は、別の CA で新しいコネクタを作成してください。
0x8FFFA005	テンプレートは証明書サブジェクトの `directoryGuid` 属性またはサブジェクトの代替名を指定しましたが、その属性がリクエスト元の AD オブジェクトに見つかりませんでした。	Active Directory はユーザーのディレクトリ用に `directoryGuid` を生成しませんでした。Active Directory のトラブルシューティング。
0x8FFFA006	テンプレートは証明書サブジェクトの `dnsHostName` 属性またはサブジェクトの代替名を指定しましたが、その属性がリクエスト元の AD オブジェクトに見つかりませんでした。	AD オブジェクトに `dnsHostName` 属性を追加します。
0x8FFFA007	テンプレートは証明書サブジェクトまたはサブジェクトの代替名に含めるメール属性を指定しましたが、その属性がリクエスト元の AD オブジェクトに見つかりませんでした。	AD オブジェクトにメール属性を追加します。

エラーコード	意味	修正
0x8FFFA008	SOAP メッセージには、http://schemas.microsoft.com/windows/pki/2009/01/enrollmentpolicy/IPolicy/GetPolicies または http://schemas.microsoft.com/windows/pki/2009/01/enrollment/RST/wstep のアクションヘッダーが必要です。	指定された値のいずれかを使用するようにアクションヘッダーを更新してください。
0x8FFFA009	は でエンコード BinarySecurityToken する必要がありますhttp://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd#base64binary 。	バイナリセキュリティトークンのタイプを更新してください。
0x8FFFA00A	BinarySecurityToken が無効です。	CSR が正しく生成されていることを確認してください。
0x8FFFA00B	には、 http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd#PKCS7 または の値タイプ BinarySecurityToken が必要ですhttp://schemas.microsoft.com/windows/pki/2009/01/enrollment#PKCS10 。	バイナリセキュリティトークンの値タイプを有効な値に更新してください。

エラーコード	意味	修正
0x8FFFA00C	BinarySecurityToken 含まれている無効な CMS。	Base64 は有効ですが、暗号メッセージ構文 (CMS) が無効です。CMS の構文を確認してください。
0x8FFFA00D	に無効な CSR BinarySecurityToken が含まれていました。	CSR が正しく生成されていることを確認してください。
0x8FFFA00E	プライベート CA は特定のテンプレートを使用して証明書を発行できませんでした。	から検証例外を確認します AWS Private CA。検証例外は、Amazon EventBridge または で表示できます AWS CloudTrail。
0x8FFFA00F	SOAP メッセージは `http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue` のリクエストタイプである必要があります。	リクエストタイプを `http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue` に設定します。
0x8FFFA010	SOAP メッセージには、コネクタの `CertificateEnrollmentPolicyServerEndpoint` フィールドまたは XCEP 応答の URI フィールドのいずれかの to ヘッダーが必要です。	リクエストセキュリティトークンのヘッダーを XCEP レスポンスの `CertificateEnrollmentPolicyServerEndpoint` フィールドまたは URI フィールドに設定します。
0x8FFFA011	SOAP メッセージにはアクションヘッダーを 1 つだけ含める必要があります。	リクエストセキュリティトークンの SOAP メッセージヘッダーを確認し、ヘッダーを正しく設定してください。

エラーコード	意味	修正
0x8FFFA012	SOAP メッセージには messageId ヘッダーを 1 つだけ含める必要があります。	リクエストセキュリティトークンの SOAP メッセージヘッダーを確認し、ヘッダーを正しく設定してください。
0x8FFFA013	SOAP メッセージには to ヘッダーを 1 つだけ含める必要があります。	リクエストセキュリティトークンの SOAP メッセージヘッダーを確認し、ヘッダーを正しく設定してください。
0x8FFFA014	リクエスト元にはリクエストされたテンプレートへのアクセス権がありません。	アクセス制御エントリを作成して、リクエスト元のグループが要求されたテンプレートを使用して登録できるようにします。
0x8FFFA015	CertificateTemplateInformation または CertificateTemplateName 拡張機能のいずれかが に存在する必要があります BinarySecurityToken。	セキュリティ拡張を CSR に追加してください。
0x8FFFA016	要求されたテンプレートは、指定されたコネクタでは見つかりませんでした。	テンプレートは各コネクタの子リソースです。createTemplate を使用してコネクタ用のテンプレートを作成します。
0x8FFFA017	リクエストのスロットリングにより、リクエストが拒否されました。	リクエストの速度を遅くしてください。
0x8FFFA018	SOAP メッセージには to ヘッダーが含まれている必要があります。	SOAP メッセージのヘッダーを確認してください。

エラーコード	意味	修正
0x8FFFA019	ヘッダーが認識されないため、SOAP メッセージを処理できませんでした。	SOAP メッセージのヘッダーを確認してください。
0x8FFFA01A	テンプレートでは、証明書のサブジェクトまたはサブジェクトの代替名に UPN 属性を含めるように指定しましたが、その属性がリクエスト元の AD オブジェクトに見つかりませんでした。	UPN を Active Directory オブジェクトに追加します。

コネクタ作成の失敗

コネクタの作成は、さまざまな理由で失敗することがあります。コネクタの作成に失敗すると、API レスポンスに失敗の理由が表示されます。コンソールを使用している場合は、コネクタの詳細コンテナ内の「ステータスの詳細の追加」フィールドのコネクタの詳細ページに失敗の理由が表示されます。次の表は、障害の理由と解決の推奨手順を示しています。

失敗ステータス	説明	修正
`CA_CERTIFICATE_REGISTRATION_FAILED`	Connector for AD は、CA 証明書をディレクトリにインポートできません。	「前提条件」ページを確認し、サービスアカウントに適切なアクセス許可があることを確認します。サービスアカウントに正しいアクセス許可を委任したら、障害が発生したコネクタを削除し、新しいコネクタを作成します。アクセス許可の委任の詳細については、「 AWS Directory Service 管理ガイド」の「サービスアカウントに権限を委

失敗ステータス	説明	修正
		任する」を参照してください。
DIRECTORY_ACCESS_DENIED	Connector for AD がディレクトリにアクセスできません。	Connector for AD に ディレクトリへのアクセスを許可する必要があります。IAM ポリシー セクションを確認して、AWS アカウントに関連付けられた IAM ポリシーでディレクトリへのアクセスと説明が許可されていることを確認します。 AWS ロールに正しいアクセス許可を付与したら、障害が発生したコネクタを削除し、新しいコネクタを作成します。
INTERNAL_FAILURE	Connector for AD で内部障害が発生しました。	後ほどもう一度試してください。」 障害が発生したコネクタを削除し、新しいコネクタを作成します。

失敗ステータス	説明	修正
PRIVATECA_ACCESS_DENIED	Connector for AD がプライベート CA にアクセスできません。	「前提条件」ページを確認し、コネクタを作成するアクセス許可があることを確認します。詳細については、「IAM ポリシー」を参照してください。 AWS CLI または API を使用してコネクタを作成する場合は、前提条件ページを確認し、 を使用して Connector for AD とプライベート CA を共有していることを確認します AWS Resource Access Manager。 IAM アクセス許可と AWS RAM リソース共有を確認して修正したら、障害が発生したコネクタを削除し、新しいコネクタを作成します。
PRIVATECA_RESOURCE_NOT_FOUND	Connector for AD は、指定されたプライベート CA を見つけることができません。	正しいプライベート CA Amazon リソースネーム (ARN) を指定し、障害が発生したコネクタを削除して、意図したプライベート CA ARN を使用して新しいコネクタを作成します。

失敗ステータス	説明	修正
SECURITY_GROUP_NOT_IN_VPC	セキュリティグループは、ディレクトリをホストする VPC にありません。	ディレクトリをホストする VPC にあるセキュリティグループを使用します。詳細については、「セキュリティグループ」を参照してください。障害が発生したコネクタを削除し、VPC 内のセキュリティグループを使用して新しいコネクタを作成します。
VPC_ACCESS_DENIED	Connector for AD は、ディレクトリをホストする Amazon VPC にアクセスできません。	IAM 許可をチェックします。障害が発生したコネクタを削除し、新しいコネクタを作成します。アクセス許可を含む IAM ポリシーの例については、「」を参照してください。 IAM ポリシー
VPC_ENDPOINT_LIMIT_EXCEEDED	Connector for AD は Amazon VPC にエンドポイントを作成できません。アカウント用に作成できる VPC エンドポイントの制限に達しました。	Amazon VPC エンドポイントを削除するか、制限の引き上げをリクエストします。2 つのステップのいずれかを実行したら、障害が発生したコネクタを削除し、新しいコネクタを作成します。クォータの詳細については、Amazon Virtual Private Cloud Service クォータ」を参照してください。

失敗ステータス	説明	修正
VPC_RESOURCE_NOT_FOUND	Connector for AD は、指定された VPC を見つけることができません。	正しい VPC を指定し、VPC が存在することを確認します。次に、障害が発生したコネクタを削除し、正しい VPC ID を使用して新しいコネクタを作成します。

SPN 作成の失敗

サービスプリンシパル名 (SPN) の作成は、さまざまな理由で失敗することがあります。SPN の作成に失敗すると、API レスポンスに失敗の理由が表示されます。コンソールを使用している場合は、サービスプリンシパル名 (SPN) コンテナ内の「追加ステータスの詳細」フィールドのコネクタの詳細ページに失敗の理由が表示されます。次の表は、障害の理由と解決の推奨手順を示しています。

失敗ステータス	説明	修正
DIRECTORY_ACCESS_DENIED	Connector for AD はディレクトリにアクセスできません。	Connector for AD に ディレクトリへのアクセスを許可します。ディレクトリアクセスを許可するアクセス許可を含む IAM ポリシーの例については、「」を参照してくださいIAM ポリシー。
DIRECTORY_NOT_REACHABLE	Connector for AD はディレクトリにアクセスできません。	AWS と ディレクトリ間のネットワークを確認し、SPN をもう一度作成してみてください。
DIRECTORY_RESOURCE_NOT_FOUND	Connector for AD は、指定されたディレクトリを見つけることができません。	正しいディレクトリ ID を指定してから、障害が発生したコ

失敗ステータス	説明	修正
		ネクタを削除し、目的のディレクトリ ID を使用して新しいコネクタを作成します。
INTERNAL_FAILURE	Connector for AD で内部障害が発生しました。	後ほどもう一度試してください。」
SPN_EXISTS_ON_DIFFERENT_AD_OBJECT	サービスプリンシパル名 (SPN) は、別の Active Directory オブジェクトに存在します。	Active Directory オブジェクトから SPN を削除し、もう一度 SPN を作成してみてください。
SPN_LIMIT_EXCEEDED	ディレクトリあたりの SPN の上限に達したため、Connector for AD は SPNsを作成できません。ディレクトリあたりの SPNs の最大数は 10 です。	アカウントから 1 つ以上の SPNsを削除し、もう一度 SPN を作成してみてください。

Connector for AD コネクタの作成失敗エラー

Connector for AD コネクタの作成は、さまざまな理由で失敗することがあります。コネクタの作成に失敗すると、API レスポンスに失敗の理由が表示されます。コンソールを使用している場合は、コネクタの詳細コンテナ内の「ステータスの詳細の追加」フィールドのコネクタの詳細ページに失敗の理由が表示されます。次の表は、障害の理由と解決の推奨手順を示しています。

用語と概念

次の用語と概念は、AWS Private Certificate Authority (AWS Private CA) で作業する際に役立ちます。

トピック

信頼

ウェブブラウザがウェブサイトのアイデンティティを信頼するために、ウェブサイトの証明書がブラウザによって検証できる必要があります。しかし、ブラウザは CA ルート証明書と呼ばれるほんのわずかな証明書のみを信頼します。認証機関 (CA) と呼ばれる信頼されるサードパーティは、ウェブサイトのアイデンティティを検証し、ウェブサイト運営者に署名されたデジタル証明書を発行します。こうしてブラウザはデジタル署名を確認して、ウェブサイトのアイデンティティを検証します。検証に成功した場合は、ブラウザのアドレスバーにロックアイコンが表示されます。

TLS サーバー証明書

HTTPS トランザクションは、サーバーを認証するためにサーバー証明書を要求します。サーバー証明書とは、証明書のパブリックキーを証明書の件名にバインドする X.509 v3 データ構造です。TLS 証明書は、認証機関 (CA) によって署名されています。これには、サーバーの名前、有効期間、パブリックキー、署名アルゴリズムなどが含まれます。

証明書の署名

デジタル署名は、証明書上の暗号化されたハッシュです。署名は、証明書データの整合性を確認するために使用します。プライベート CA は、可変サイズの証明書コンテンツに SHA256 などの暗号化ハッシュ関数を使用して署名を作成します。このハッシュ関数は、効果的に偽造不可能な固定サイズのデータ文字列を生成します。この文字列はハッシュと呼ばれます。次に CA は、プライベートキーを使用してハッシュ値を暗号化し、暗号化したハッシュを証明書に連結します。

認証局

認証機関 (CA) は、デジタル証明書を発行し、必要に応じて、取り消します。最も一般的なタイプの証明書は ISO X.509 規格に基づくものです。X.509 証明書は、証明書のサブジェクトのアイデンティティを確認し、そのアイデンティティをパブリックキーにバインドします。サブジェクトは、ユーザー、アプリケーション、コンピュータ、またはその他のデバイスです。CA は証明書に署名するために、コンテンツをハッシュし、そのハッシュを証明書のパブリックキーに関連するプライベートキーで暗号化します。サブジェクトのアイデンティティを確認する必要があるクライアントアプリケーション (ウェブブラウザなど) は、パブリックキーを使用して証明書の署名を復号します。次に証明書のコンテンツをハッシュし、ハッシュした値と復号した署名を比較して一致するかどうかを確認します。証明書の署名の詳細については、「証明書の署名」を参照してください。

AWS Private CA を使用してプライベート CA を作成し、このプライベート CA を使用して証明書を発行できます。プライベート CA は、組織内で使用するプライベート SSL/TLS 証明書のみを発行します。詳細については、「プライベート証明書」を参照してください。プライベート CA を使用するには、事前に証明書が必要です。詳細については、「CA 証明書」を参照してください。

ルート CA

ルート CA とは暗号構築ブロックで、証明書を発行できる信頼のルートのことです。証明書に署名 (発行) するためのプライベートキーと、ルート CA を識別し、プライベートキーを CA 名にバインドするルート証明書で構成されています。ルート証明書は、環境内の各エンティティの信頼ストアに配布されます。管理者は、信頼する CA のみを含めるように信頼ストアを構築します。管理者は、信頼ストアをオペレーティングシステム、インスタンス、および環境内のエンティティのホストマシンイメージに更新または構築します。リソースが互いに接続しようとした場合、各エンティティが提示する証明書を確認します。クライアントは、証明書の有効性、および証明書から信頼ストアにインストールされているルート証明書へのチェーンが存在するかどうかを確認します。これらの条件が満たされると、リソース間で「ハンドシェイク」が実行されます。このハンドシェイクは、各エンティ

ティのアイデンティティを暗号的に証明し、それらの間に暗号化された通信チャネル (TLS/SSL) を作成します。

CA 証明書

認証機関 (CA) 証明書では、CA のアイデンティティを確認し、これを証明書内のパブリックキーにバインドします。

AWS Private CA を使用して、プライベートルート CA またはプライベート下位 CA を作成し、それぞれが CA 証明書によって裏付けられます。下位 CA 証明書は、信頼のチェーンの上位の別の CA 証明書によって署名されます。ただし、ルート CA の場合、証明書は自己署名です。また、外部ルート権限 (オンプレミスでホストされているなど) を確立することもできます。その後、ルート権限を使用して、AWS Private CA によってホストされる下位ルート CA 証明書に署名できます。

次の例では、AWS Private CA X.509 CA 証明書内の一般的なフィールドを示します。CA 証明書の場合、CA: フィールドの Basic Constraints 値は TRUE に設定します。

```
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
```

ルート CA 証明書

認証機関 (CA) は通常、相互の親子関係を明確に定義した他の複数の CA を含む段階構造内に存在します。子または下位の CA はその親 CA に証明され、証明書チェーンを造り上げます。階層の最上部にある CA はルート CA とされ、その証明書はルート証明書と呼ばれます。この証明書は通常、自己署名されています。

エンドエンティティ証明書

エンドエンティティ証明書は、サーバー、インスタンス、コンテナ、デバイスなどのリソースを識別します。CA 証明書とは異なり、エンドエンティティ証明書は証明書の発行に使用できません。エンドエンティティ証明書のその他の一般的な用語は、「クライアント」証明書または「リーフ」証明書です。

自己署名証明書

上位の CA ではなく、発行者によって署名された証明書。CA が維持する安全なルートから発行された証明書とは異なり、自己署名証明書は自身のルートで動作するため、大きな制限があります。自己署名証明書はワイヤー暗号化での提供には使用できるものの、アイデンティティの確認はできず、取り消しもできません。セキュリティの観点からは受け入れられません。しかし、生成が容易で、専門知識やインフラストラクチャを必要とせず、多くのアプリケーションがそれらを受け入れるために、組織により使用されます。自己署名証明書の発行には、何の制御もありません。有効期限の日付を確認する方法が無いため、これを使う組織は、証明書の有効期限切れによって使用不能に陥るリスクが大きくなります。

プライベート証明書

AWS Private CA 証明書は、組織内で使用できるプライベート SSL/TLS 証明書ですが、パブリックインターネットでは信頼されません。これらの証明書を使用して、クライアント、サーバー、アプリケーション、サービス、デバイス、ユーザーなどのリソースを識別します。安全な暗号化された通信

チャネルを確立する場合、各リソースは次のような証明書と暗号化手法を使用してそのアイデンティティを別のリソースに対して証明します。内部の API エンドポイント、ウェブサーバー、VPN ユーザー、IoT デバイス、およびその他多くのアプリケーションでは、プライベート証明書を使用して、安全なオペレーションに必要な暗号化された通信チャネルを確立します。デフォルトでは、プライベート証明書はパブリックに信頼されません。内部管理者は、プライベート証明書を信頼するように明示的にアプリケーションを設定し、証明書を配布する必要があります。

```
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
```

証明書のパス

証明書に依存するクライアントは、可能であれば中間証明書のチェーンを通して、エンドエンティティ証明書から信頼されたルートへのパスが存在することを検証します。クライアントは、パス上の各証明書が有効である (失効していない) ことを確認します。また、エンドエンティティ証明書の有効期限が切れていない、整合性 (改ざんまたは変更されていない)、証明書の制約が適用されているかどうかも確認します。

パスの長さの制約

CA 証明書pathLenConstraintの基本的な制約は、その下のチェーンに存在する可能性のある下位 CA 証明書の数を設定します。たとえば、パスの長さの制約が 0 の CA 証明書は、下位 CA を持つことはできません パス長の制約が 1 の CA は、その下に最大 1 つのレベルの下位 CA を持つことができます。RFC 5280 では、これを「有効な証明書パスでこの non-self-issued 証明書に続く中間証明書の最大数」と定義しています。パスの長さの値にはエンドエンティティ証明書は含まれていませんが、検証チェーンの「長さ」や「深さ」に関する非公式な表現には含まれている場合があり、混乱を招きます。

ドキュメント履歴

以下の表は、このドキュメントの 2018 年 1 月以降の大きな変更点をまとめたものです。ここに表示されている主要な変更に加えて、その内容の説明と例を向上し、ユーザーから寄せられるフィードバックにも応える目的で、このドキュメントは頻繁に更新されます。重要な変更についての通知を受け取るには、右上隅にあるリンクを使用して、RSS フィードをサブスクライブします。

変更	説明	日付
Connector for AD のトラブルシューティングガイダンスを更新	Connector for AD のインスタンスで`Kerberos authentication failed`エラーが発生した場合は、ディレクトリにアクセスできること、およびクライアントがユーザーまたはコンピュータであることを確認してください。	2024 年 7 月 3 日
監査レポートの制約を追加	AWS Private CA は、監査レポートに使用されるバケットでの Amazon S3 オブジェクトロックの使用をサポートしていません。	2024 年 7 月 3 日
中国リージョンで SM2 をサポートするようになりました	AWS Private CA は、中国リージョンでのみ SM2 署名アルゴリズムをサポートするようになりました。	2024 年 6 月 27 日
AWS Private CA が Connector for SCEP をサポート (プレビュー)	Connector for SCEP を使用して、SCEP 対応のクライアントとデバイス AWS Private CA にリンクします。	2024 年 6 月 11 日
新しいコネクタのトラブルシューティングガイダンス	コネクタのトラブルシューティングと SPN 作成の失敗に	2024 年 4 月 4 日

	関する 2 つの新しいセクションを追加しました。	
Matter 用の CDP 拡張機能の追加	Matter の Certificate Revocation List Distribution Point (CDP) 拡張機能のサポートを追加しました。	2024 年 1 月 25 日
AWS Private CA mDL の API サポート	モバイル運転免許証 (mDL) の ISO/IEC 規格に準拠した証明書を作成するための API サポートを追加しました。	2024 年 1 月 16 日
AWS Private CA Connector for Active Directory	Connector for AD のユーザーガイド、API、および CLI サポート 詳細については、「Connector for AD のドキュメント」を参照してください。	2023 年 8 月 24 日
セキュリティポリシー名を新しいサービス名と一致するように変更する	で標準のアクセス許可を指定する AWS マネージド IAM ポリシーの新しい名前を採用しました AWS Private CA。詳細については、「AWS マネージドポリシー」を参照してください。	2023 年 2 月 13 日
AWS 管理ポリシーの変更トラッカーの追加	で標準アクセス許可を指定する AWS マネージド IAM ポリシーの変更を追跡するために追加されたドキュメント AWS Private CA。詳細については、「AWS Private CA の AWS マネージドポリシーに関する更新」を参照してください。	2022 年 11 月 11 日

有効期間の短い証明書を発行する CA の API と CLI のサポート	CA 使用モードの導入により、CA が汎用証明書または短期証明書のみのどちらを発行するかを設定できます。詳細については「認証局モード」を参照してください。	2022 年 10 月 24 日
サービスのブランド変更とコンソールの更新	サービスの名前が AWS Private Certificate Authority () に変更されましたAWS Private CA。 AWS Private CA コンソールでは、完全なドキュメントにリンクする統合ヘルプパネルなど、使いやすさが向上しています。	2022 年 9 月 27 日
Matter 準拠の証明書サポート	Matter 準拠の CA 証明書とエンドエンティティ証明書のサポートに、3 つの新しい証明書テンプレートが追加されました。詳細については、「証明書テンプレートの説明」を参照してください。	2022 年 7 月 20 日
新しいリージョンのサポート	アジアパシフィック (ジャカルタ) にエンドポイントが追加されました。 AWS Private CA エンドポイントの完全なリストについては、「ACM Private Certificate Authority Endpoints and Quotas」を参照してください。	2022 年 5 月 4 日

カスタム属性と拡張のサポート	CustomAttribute オブジェクトを使用してカスタマイズされた CAs と証明書を設定し、CustomExtension オブジェクトを使用してカスタマイズされた証明書を設定します。	2022 年 3 月 16 日
マネージド OCSP のサポート	OCSP を含む失効オプションについては、「証明書失効方法の設定」を参照してください。	2021 年 8 月 18 日
CRL の S3 のブロックパブリックアクセス機能のサポート	「S3 のブロックパブリックアクセス機能の有効化」を参照してください。	2021 年 5 月 27 日
新規および更新された Java 実装例	「ACM プライベート CA API の使用 (Java サンプル)」 を参照してください。	2020 年 9 月 9 日
新しいリージョンのサポート	アフリカ (ケープタウン) および欧州 (ミラノ) にエンドポイントが追加されました。 AWS Private CA エンドポイントの詳しいリストについては、「AWS Certificate Manager プライベート認証機関のエンドポイントとクォータ」を参照してください。	2020 年 8 月 27 日

クロスアカウントプライベート CA アクセスがサポートされています	AWS Certificate Manager ユーザーは、所有していないプライベート CAsを使用して証明書を発行する権限を付与されます。詳細については、「プライベート CA へのクロスアカウントアクセス」を参照してください。	2020 年 8 月 17 日
VPC エンドポイント (PrivateLink) のサポート	ネットワークセキュリティを強化するための VPC エンドポイント (AWS PrivateLink) の使用のサポートが追加されました。詳細については、「ACM プライベート CA VPC エンドポイント (AWS PrivateLink) 」を参照してください。	2020 年 3 月 26 日
専用セキュリティセクションが追加されました	のセキュリティドキュメント AWS が専用のセキュリティセクションに統合されました。セキュリティの詳細については、AWS Certificate Manager「 プライベート認証機関 のセキュリティ」を参照してください。	2020 年 3 月 26 日
テンプレート ARN が監査報告書に追加されました。	詳細については、「プライベート CA の監査報告書の作成」を参照してください。	2020 年 3 月 6 日

CloudFormation サポート	のサポートが追加されました AWS CloudFormation。詳細については、「 AWS CloudFormation ユーザーガイド」の「ACMPCA リソースタイプのリファレンス」を参照してください。	2020 年 1 月 22 日
CloudWatch イベント統合	CA の作成、証明書の発行、CRL の作成など、非同期イベントの CloudWatch イベントとの統合。詳細については、「イベントの使用 CloudWatch 」を参照してください。	2019 年 12 月 23 日
FIPS エンドポイント	(米国東部) および AWS GovCloud （ AWS GovCloud 米国西部) に追加された FIPS エンドポイント。 AWS Private CA エンドポイントの完全なリストについては、AWS Certificate Manager 「 Private Certificate Authority Endpoints and Quotas」を参照してください。	2019 年 12 月 13 日

タグベースのアクセス許可	新しい API TagResource 、UntagResource 、および ListTagsForResource を使用してサポートされるタグベースのアクセス許可。タグベースのコントロールに関する一般的な情報については、「IAM リソースタグを使用した IAM ユーザーおよびロールへのアクセス制御」を参照してください。	2019 年 11 月 5 日
名前制約の強制	インポートされた CA 認定にサブジェクト名の制約を適用するためのサポートが追加されました。詳細については、「プライベート CA に名前制約を適用する」を参照してください。	2019 年 10 月 28 日
新しい証明書テンプレート	でコード署名用のテンプレートを含む、新しい証明書テンプレートが追加されましたAWS Signer。詳細については、「テンプレートの使用」を参照してください。	2019 年 10 月 1 日
CA の計画	AWS Private CAを使用したPKI の計画に関する新しいセクションが追加されました。詳細については、「ACM プライベート CA デプロイの計画」を参照してください。	2019 年 9 月 30 日

リージョンサポートの追加	AWS アジアパシフィック (香港) リージョンのリージョンサポートを追加しました。サポートされているリージョンの完全なリストについては、「AWS Certificate Manager プライベート認証機関のエンドポイントとクォータ」を参照してください。	2019 年 7 月 24 日
完全プライベート CA 階層のサポートを追加	ルート CA の作成とホストのサポートにより、外部の親が不要になります。	2019年6月20日
リージョンサポートの追加	AWS GovCloud （米国西部および米国東部) リージョンのリージョンサポートを追加しました。サポートされているリージョンの完全なリストについては、「AWS Certificate Manager プライベート認証機関のエンドポイントとクォータ」を参照してください。	2019 年 5 月 8 日
リージョンサポートの追加	AWS アジアパシフィック (ムンバイおよびソウル）、米国西部 (北カリフォルニア）、欧州 (パリおよびストックホルム) の各リージョンに対するリージョンのサポートを追加しました。サポートされているリージョンの完全なリストについては、「AWS Certificate Manager プライベート認証機関のエンドポイントとクォータ」を参照してください。	2019 年 4 月 4 日

証明書の更新ワークフローのテスト	ACM マネージド更新ワークフローの設定を手動でテストできるようになりました。詳細については、「ACM のマネージド更新設定のテスト」を参照してください。	2019 年 3 月 14 日
リージョンサポートの追加	AWS EU (ロンドン) リージョンのリージョンサポートを追加しました。サポートされているリージョンの完全なリストについては、「AWS Certificate Manager プライベート認証機関のエンドポイントとクォータ」を参照してください。	2018 年 8 月 1 日
削除された CA を復元する	プライベート CA の復元を使用すると、認証機関 (CA) が削除されてから最大 30 日間、その CA を復元できます。詳細については、「プライベート CA の復元」を参照してください。	2018 年 6 月 20 日

以前の更新

次の表に、2018 年 6 月 AWS Private Certificate Authority 以前の のドキュメントリリース履歴を示します。

変更	説明	日付
新規ガイド	このリリースでは AWS Private Certificate Authorityを導入しています。	2018 年 04 月 4 日

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。