Add a permission의

AWS Private Certificate Authority

버전 latest

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

AWS Private Certificate Authority: Add a permission의

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

Amazon의 상표 및 트레이드 드레스는 Amazon 외 제품 또는 서비스와 함께, Amazon 브랜드 이미지를 떨어뜨리거나 고객에게 혼동을 일으킬 수 있는 방식으로 사용할 수 없습니다. Amazon이 소유하지 않은 기타 모든 상표는 Amazon과 제휴 관계이거나 관련이 있거나 후원 관계와 관계없이 해당 소유자의 자산입니다.

Table of Contents

AWS Private CA란 무엇인가요? 1
리전별 가용성 1
통합 서비스 2
지원되는 알고리즘 2
RFC 5280 규정 준수 3
가격 책정 4
에 대한 용어 및 개념 AWS Private CA 5
신뢰 5
TLS 서버 인증서 6
인증서 서명 6
인증 기관 6
루트 CA 6
CA 인증서 7
루트 CA 인증서 8
최종 엔터티 인증서 8
자체 서명된 인증서 8
프라이빗 인증서 8
인증서 경로 9
경로 길이 제약 10
내 요구에 가장 적합한 인증서 서비스는 무엇입니까? 11
모범 사례 12
CA 구조 및 정책 문서화 12
가능한 경우 루트 CA 사용 최소화 12
루트 CA에 고유한 권한 부여 AWS 계정 13
별도의 관리자 및 발급자 역할 13
인증서의 관리형 해지 구현 13
켜기 AWS CloudTrail 14
CA 프라이빗 키 업데이트 14
미사용 CA 삭제 14
CRL에 대한 퍼블릭 액세스 차단 14
Amazon EKS 애플리케이션 모범 사례 15
와 AWS Private CA 함께 사용 AWS SDK for Java 16
API 예제 16
프로그래밍 방식으로 루트 CA 생성 및 활성화 17

프로그래밍 방식으로 하위 CA 생성 및 활성화 26
CreateCertificateAuthority 35
Active Directory를 지원하기 위해 인증서 기관 생성 사용 39
CreateCertificateAuthorityAuditReport 48
CreatePermission 50
DeleteCertificateAuthority 52
DeletePermission 55
DeletePolicy 57
DescribeCertificateAuthority 59
DescribeCertificateAuthorityAuditReport 61
GetCertificate 64
GetCertificateAuthorityCertificate 66
GetCertificateAuthorityCsr 68
GetPolicy 71
ImportCertificateAuthorityCertificate 73
IssueCertificate 76
ListCertificateAuthorities 79
ListPermissions 83
ListTags 85
PutPolicy 87
RestoreCertificateAuthority 90
RevokeCertificate 91
TagCertificateAuthorities 94
UntagCertificateAuthority 96
UpdateCertificateAuthority 98
사용자 지정 주제 이름을 사용하여 CA 및 인증서 생성 100
사용자 지정 확장으로 인증서 생성 108
Matter 예제 123
제품 인증 기관(PAA) 활성화 124
제품 증명 중간(PAI) 활성화 134
디바이스 인증 인증서(DAC) 생성 145
노드 운영 인증서(NOC)에 대한 루트 CA를 활성화합니다. 150
노드 운영 인증서(NOC)에 대한 하위 CA 활성화 159
노드 운영 인증서(NOC) 생성 169
mDL 예제 174
발급 기관 인증 기관(IACA) 인증서 활성화 174

문서 서명자 인증서 생성 ... 183
용 솔루션 설계 AWS Private CA ... 189
CA 계층 구조 설계 ... 189
최종 엔터티 인증서 검증 ... 191
CA 계층 구조의 계획 ... 192
인증 경로에서 길이 제약 조건 설정 ... 195
CA 수명 주기 관리 ... 197
유효 기간 선택 ... 197
CA 승계 관리 ... 199
CA 취소 ... 200
계획 인증서 해지 ... 200
요구 사항 ... 202
CRL 설정 ... 202
OCSP URL 사용자 지정 ... 209
CA 모드 ... 211
범용(기본값) ... 212
수명이 짧은 인증서 ... 212
복원력 계획 ... 213
이중화 및 재해 복구 ... 213
인증 기관 ... 214
설정 ... 215
에 가입 AWS 계정 ... 215
관리자 액세스 권한이 있는 사용자 생성 ... 215
설치 AWS Command Line Interface ... 217
프라이빗 CA 생성 ... 217
CLI 예 ... 224
CA 인증서 설치 ... 236
호환되는 서명 알고리즘 ... 236
루트 CA 인증서 설치 ... 239
에서 호스팅하는 하위 CA 인증서 설치 AWS Private CA ... 246
외부 상위 CA에서 서명한 하위 CA 인증서 설치 ... 247
액세스 제어 ... 248
IAM 사용자를 위한 단일 계정 권한 생성 ... 248
교차 계정 액세스를 위한 정책 연결 ... 251
프라이빗 CAs 나열 ... 253
프라이빗 CA 보기 ... 255

태그 추가 258
CA 상태 260
CA 상태와 CA 수명 주기 간의 관계 262
CA 업데이트 263
CA 업데이트(콘솔) 263
CA(CLI) 업데이트 267
CA 삭제 275
CA 복원 276
프라이빗 CA 복원(콘솔) 277
프라이빗 CA 복원(AWS CLI) 277
외부에서 서명된 CA 인증서 278
인증서 발급 및 관리 283
프라이빗 최종 엔터티 인증서 발급 283
표준 인증서 발급(AWS CLI) 284
APIPassThrough 템플릿을 사용하여 사용자 지정 보안 주체 이름이 포함된 인증서를 발급합니다. 287
APIPassThrough 템플릿을 사용하여 사용자 지정 확장이 포함된 인증서를 발급합니다. 289
프라이빗 인증서 검색 290
프라이빗 인증서 나열 292
인증서 내보내기 297
프라이빗 인증서 취소 297
해지된 인증서 및 OCSP 298
CRL의 해지된 인증서 298
감사 보고서에서 해지된 인증서 299
내보내기 자동화 300
인증서 템플릿 301
템플릿 종류 301
템플릿 작업 순서 312
템플릿 정의 313
보안 353
IAM 354
API 권한 355
AWS 관리형 정책 360
고객 관리형 정책 362
인라인 정책 363
크로스 계정 액세스 368

리소스 기반 정책 ... 369
데이터 보호 ... 373
AWS Private CA 프라이빗 키의 스토리지 및 보안 규정 준수 ... 374
Active Directory용 AWS Private CA 커넥터의 데이터 암호화 ... 374
규정 준수 확인 ... 374
감사 보고서 생성 ... 375
인프라 보안 ... 382
VPC 엔드포인트(AWS PrivateLink) ... 382
듀얼 스택 엔드포인트 지원 ... 386
IAM 및에서 IPv6 주소 사용 AWS Private CA ... 387
CP/CPS ... 388
CP/CPS 요구 사항 및 책임 ... 389
리소스 모니터링 ... 398
AWS Private CA CloudWatch 지표 ... 398
CloudWatch Events AWS Private CA 로 모니터링 ... 399
프라이빗 CA를 생성할 때 성공 또는 실패 ... 400
인증서 발급 시 성공 또는 실패 ... 401
인증서 해지 시 성공 ... 402
CRL 생성 시 성공 또는 실패 ... 403
CA 감사 보고서 생성 시 성공 또는 실패 ... 405
CloudTrail 로그 ... 406
AWS Private CA CloudTrail의 정보 ... 406
AWS Private CA 관리 이벤트 ... 407
예제 AWS Private CA 이벤트 ... 408
문제 해결 ... 411
인증서 해지 문제 ... 411
OCSP 응답 지연 시간 ... 411
자체 서명된 인증서의 취소 ... 411
예외 메시지 ... 411
문제 준수 인증서 오류 ... 414
를 사용하여 Kubernetes 보호 AWS Private CA ... 416
개념 ... 416
고려 사항 ... 418
cert-manager의 교차 계정 사용 ... 418
시작하기 ... 419
cert-manager 설치 ... 422

IAM 권한 구성 423
AWS Private CA 클러스터 발급자 설치 및 구성 425
cert-manager를 사용하여 AWS Private CA 클라이언트 인증서 관리 429
첫 번째 TLS 인증서 발급 430
예시 431
모니터링 432
문제 해결 432
Active Directory용 커넥터 374
AD 사용자를 위한 커넥터를 처음 사용하시나요? 434
AD용 액세스 커넥터 434
가격 책정 434
설정 435
1단계:를 사용하여 프라이빗 CA 생성 AWS Private CA 435
2단계: Active Directory 설정 435
(Active Directory 커넥터만 해당) 3단계: 서비스 계정에 권한 위임 435
4단계: IAM 정책 생성 436
5단계: AD용 커넥터와 프라이빗 CA 공유 439
6단계: 디렉터리 등록 생성 440
7단계: 보안 그룹 구성 440
8단계: 디렉터리 객체에 대한 네트워크 액세스 구성 440
시작하기 441
시작하기 전 준비 사항 441
1단계: 커넥터 생성 441
2단계: Microsoft Active Directory 정책 구성 442
3단계: 템플릿 생성 443
4단계: Microsoft 그룹 권한 구성 443
Active Directory용 커넥터 444
커넥터 생성 444
템플릿 생성 446
템플릿 업데이트 450
커넥터 목록 451
작업 템플릿 나열 452
뷰 커넥터 453
템플릿 보기 454
디렉터리 등록 456
템플릿 액세스 제어 항목 458

서비스 보안 주체 이름 ... 459
Tags ... 460
EventBridge와 통합 ... 461
EventBridge가 AD 이벤트용 커넥터를 라우팅하는 방법 ... 462
AD 이벤트용 커넥터 ... 462
이벤트 패턴 생성 ... 463
이벤트 수신 ... 463
Active Directory용 커넥터 문제 해결 ... 464
AD용 커넥터 오류 코드 ... 464
커넥터 생성 실패 ... 469
SPN 생성 실패 ... 472
템플릿 업데이트 문제 ... 473
SCEP용 커넥터 ... 475
특성 ... 475
SCEP용 커넥터 시작하기 ... 476
관련 서비스 ... 476
SCEP용 액세스 커넥터 ... 476
가격 책정 ... 477
개념 ... 477
고려 사항 및 제한 사항 ... 478
고려 사항 ... 478
제한 사항 ... 479
설정 ... 480
1단계: AWS Identity and Access Management 정책 생성 ... 480
2단계: 프라이빗 CA 생성 ... 481
3단계: 리소스 공유 생성 ... 482
시작하기 ... 483
시작하기 전 준비 사항 ... 483
1단계: 커넥터 생성 ... 483
2단계: MDM 시스템에 커넥터 세부 정보 복사 ... 485
MDM 시스템 구성 ... 486
범용 커넥터 ... 486
AWS Private CA Microsoft Intune용 SCEP용 커넥터 ... 487
Jamf Pro 구성 ... 488
Microsoft Intune 구성 ... 494
Omnissa Workspace ONE 구성 ... 497

보안 502
VPC 엔드포인트(AWS PrivateLink) 503
모니터링 507
EventBridge를 사용하여 자동화 507
CloudTrail 로그 513
문제 해결 521
HTTP 오류 522
클라이언트 오류 537
Service quotas 539
문서 기록 540
이전 업데이트 548
..... dxlix

AWS Private CA란 무엇인가요?

AWS Private CA 를 사용하면 온프레미스 CA 운영에 대한 투자 및 유지 관리 비용 없이 루트 및 하위 CAs를 포함한 프라이빗 인증 기관(CA) 계층 구조를 생성할 수 있습니다. 사설 CA는 다음과 같은 시나리오에서 유용한 최종 엔터티 X.509 인증서를 발급할 수 있습니다.

- 암호화된 TLS 통신 채널 생성
- 사용자, 컴퓨터, API 엔드포인트 및 IoT 디바이스 인증
- 암호화 서명 코드
- 인증서 해지 상태를 얻기 위한 온라인 인증서 상태 프로토콜(OCSP) 구현

AWS Private CA 작업은에서 AWS Management Console, AWS Private CA API를 사용하여 또는를 사용하여 액세스할 수 있습니다 AWS CLI.

주제

- 의 리전별 가용성 AWS Private Certificate Authority
- 와 통합된 서비스 AWS Private Certificate Authority
- 에서 지원되는 암호화 알고리즘 AWS Private Certificate Authority
- 의 RFC 5280 규정 준수 AWS Private Certificate Authority
- 에 대한 요금 AWS Private Certificate Authority
- 에 대한 용어 및 개념 AWS Private CA

의 리전별 가용성 AWS Private Certificate Authority

대부분의 AWS 리소스와 마찬가지로 사설 인증 기관(CAs)은 리전 리소스입니다. 하나 이상의 리전에서 사설 CA를 사용하려면 이러한 리전에서 CA를 생성해야 합니다. 리전 간에 사설 CA를 복사할 수 없습니다. AWS Private CA에 대한 리전별 가용성을 확인하려면 AWS 일반 참조 또는 AWS 리전 표에서 AWS 리전 및 엔드포인트를 방문하세요.

> **Note**
>
> ACM은 현재 AWS Private CA 가 아닌 일부 리전에서 사용할 수 있습니다.

와 통합된 서비스 AWS Private Certificate Authority

AWS Certificate Manager 를 사용하여 프라이빗 인증서를 요청하는 경우 해당 인증서를 ACM과 통합된 모든 서비스와 연결할 수 있습니다. 이는 AWS Private CA 루트에 연결된 인증서와 외부 루트에 연결된 인증서 모두에 적용됩니다. 자세한 내용은 AWS Certificate Manager 사용 설명서의 통합 서비스를 참조하세요.

또한 프라이빗 CA를 Amazon Elastic Kubernetes Service에 통합하여 Kubernetes 클러스터 내에서 인증서 발급을 제공할 수 있습니다. 자세한 내용은 를 사용하여 Kubernetes 보호 AWS Private Certificate Authority 단원을 참조하십시오.

> **Note**
>
> Amazon Elastic Kubernetes Service는 ACM 통합 서비스가 아닙니다.

AWS Private CA API 또는를 사용하여 인증서를 AWS CLI 발급하거나 ACM에서 프라이빗 인증서를 내보내는 경우 원하는 위치에 인증서를 설치할 수 있습니다.

에서 지원되는 암호화 알고리즘 AWS Private Certificate Authority

AWS Private CA 는 프라이빗 키 생성 및 인증서 서명을 위해 다음과 같은 암호화 알고리즘을 지원합니다.

지원되는 알고리즘

프라이빗 키 알고리즘	서명 알고리즘
ML_DSA_44	ML_DSA_44
ML_DSA_65	ML_DSA_65
ML_DSA_87	ML_DSA_87
RSA_2048	SHA256WITHRSA
RSA_3072	SHA384WITHRSA
RSA_4096	SHA512WITHRSA
EC_prime256v1	SHA256WITHECDSA

프라이빗 키 알고리즘	서명 알고리즘
EC_secp384r1	SHA384WITHECDSA
EC_secp521r1	SHA512WITHECDSA
SM2(중국 리전 전용)	SM3WITHSM2

이 목록은 콘솔, API 또는 명령줄을 AWS Private CA 통해에서 직접 발급한 인증서에만 적용됩니다. 가에서 CA를 사용하여 인증서를 AWS Certificate Manager 발급 AWS Private CA하면 이러한 알고리즘을 일부만 지원합니다. 자세한 내용은 AWS Certificate Manager 사용 설명서의 프라이빗 인증서 요청을 참조하세요.

> **Note**
>
> RSA 또는 ECDSA의 경우 지정된 서명 알고리즘 패밀리는 CA 프라이빗 키의 키 알고리즘 패밀리와 일치해야 합니다.
> ML-DSA의 경우 해시 함수는 알고리즘 자체의 일부로 정의됩니다. ML-DSA에서 다른 해시 함수를 선택하는 옵션은 없습니다. APIs와의 이전 버전과의 호환성을 유지하기 위해 키 알고리즘 및 서명 알고리즘에 동일한 값이 사용됩니다.

의 RFC 5280 규정 준수 AWS Private Certificate Authority

AWS Private CA 는 RFC 5280에 정의된 특정 제약 조건을 적용하지 않습니다. 반대의 상황도 마찬가지입니다. 사설 CA에 해당되는 특정한 추가 제약 조건이 적용됩니다.

적용

- 날짜 이후에 적용되지 않음 RFC 5280에 따라 AWS Private CA 는 CA 인증서 발급 날짜 `Not After`보다 `Not After`일 늦은 날짜의 인증서 발급을 금지합니다.
- Basic constraints. AWS Private CA enforces 가져온 CA 인증서의 기본 제약 조건 및 경로 길이를 적용합니다.

  기본 제약 조건은 인증서에 의해 식별된 리소스가 CA인지 여부와 인증서를 발급할 수 있는지 여부를 나타냅니다. AWS Private CA 로 가져온 CA 인증서에는 기본 제약 조건의 확장이 포함되어야 하며 확장은 `critical`로 표시되어야 합니다 . `critical` 플래그 외에도 다음과 같은 이유로 검증 예외와 함께 실패하여 기본 제약 조건을 set. AWS Private CA enforces로 설정해야 `CA=true` 합니다.

- 확장은 CA 인증서에 포함되지 않습니다.
- 확장은 `critical`로 표시되어 있지 않습니다.

경로 길이(pathLenConstraint)는 가져온 CAs 인증서에서 다운스트림으로 존재할 수 있는 하위 CA 수를 결정합니다.는 다음과 같은 이유로 검증 예외와 함께 실패하여 경로 길이를 AWS Private CA 적용합니다.

- CA 인증서를 가져오면 CA 인증서 또는 체인의 CA 인증서에서 경로 길이 제약 조건을 위반할 수 있습니다.
- 인증서를 발급하면 경로 길이 제약 조건을 위반할 수 있습니다.

- 이름 제약 조건은 인증 경로의 후속 인증서에 있는 모든 제목 이름이 위치해야 하는 이름 공간을 나타냅니다. 주체 고유 이름 및 주체 대체 이름에 제한이 적용됩니다.

적용되지 않음

- 인증서 정책. 인증서 정책은 CA가 인증서를 발급하는 조건을 규제합니다.
- anyPolicy를 금지합니다. CAs에 발급된 인증서에 사용됩니다.
- 발급자 대체 이름입니다. CA 인증서 발급자와 추가 자격 증명을 연결할 수 있습니다.
- 정책 제약 조건. 이러한 제약 조건은 CA의 하위 CA 인증서 발급 능력을 제한합니다.
- 정책 매핑. CA 인증서에 사용됩니다. 하나 이상의 OIDs 각 페어에는 issuerDomainPolicy와 subjectDomainPolicy가 포함됩니다.
- Subject Directory 속성입니다. 주제의 식별 속성을 전달하는 데 사용됩니다.
- 주체 정보 액세스. 확장이 표시되는 인증서의 주체에 대한 정보 및 서비스에 액세스하는 방법입니다.
- 보안 주체 키 식별자(SKI) 및 인증 기관 키 식별자(AKI). RFC는 CA 인증서에 SKI 확장이 포함되도록 요구합니다. CA에서 발급한 인증서에는 CA 인증서의 SKI. AWS does와 일치하는 AKI 확장이 포함되어야 합니다. CA 인증서에 SKI가 포함되어 있지 않으면 발급된 최종 엔터티 또는 하위 CA 인증서 AKI가 대신하여 발급자 퍼블릭 키의 SHA-1 해시가 됩니다.
- SubjectPublicKeyInfo 및 보안 주체 대체 이름(SAN). AWS Private CA 은 인증서를 발급할 때 유효성 검사를 수행하지 않고 제공된 CSR에서 SucjectPublicKeyInfo 및 SAN 확장을 복사합니다.

에 대한 요금 AWS Private Certificate Authority

계정에는 사설 CA를 생성한 시점부터 각 사설 CA에 대한 월별 요금이 청구됩니다. 발급한 각 인증서에 대해서도 요금이 청구됩니다. 이 요금에는 ACM에서 내보내는 인증서와 AWS Private CA API 또

는 AWS Private CA CLI에서 생성하는 인증서가 포함됩니다. 삭제한 프라이빗 CA에 대해서는 요금이 부과되지 않습니다. 그러나 사설 CA를 복원하면 삭제부터 복원까지의 기간에 대한 요금이 청구됩니다. 액세스할 수 없는 프라이빗 키를 가진 프라이빗 인증서는 무료입니다. 여기에는 Elastic Load Balancing, CloudFront, API Gateway와 같은 통합 서비스와 함께 사용되는 인증서가 포함됩니다.

최신 AWS Private CA 요금 정보는 AWS Private Certificate Authority 요금을 참조하세요. AWS 요금 계산기를 사용하여 비용을 추정할 수도 있습니다.

에 대한 용어 및 개념 AWS Private CA

다음 용어와 개념은 작업 시 도움이 될 수 있습니다 AWS Private Certificate Authority.

주제

- 신뢰
- TLS 서버 인증서
- 인증서 서명
- 인증 기관
- 루트 CA
- CA 인증서
- 루트 CA 인증서
- 최종 엔터티 인증서
- 자체 서명된 인증서
- 프라이빗 인증서
- 인증서 경로
- 경로 길이 제약

신뢰

웹 브라우저가 웹 사이트의 자격 증명을 신뢰하려면 브라우저가 웹 사이트의 인증서를 확인할 수 있어야 합니다. 하지만 브라우저는 CA 루트 인증서로 알려진 소수의 인증서만 신뢰합니다. 인증 기관(CA)으로 알려진 신뢰할 수 있는 타사는 웹 사이트의 자격 증명을 확인하고 서명된 디지털 인증서를 웹 사이트 운영자에게 발행합니다. 그러면 브라우저는 디지털 서명을 점검하여 웹 사이트의 자격 증명을 확인할 수 있습니다. 확인에 성공하면 브라우저는 주소 표시줄에 자물쇠 아이콘을 표시합니다.

TLS 서버 인증서

HTTPS 트랜잭션에서는 서버를 인증하기 위해 서버 인증서가 필요합니다. 서버 인증서는 인증서의 퍼블릭 키를 인증서의 보안 주체와 바인딩하는 X.509 v3 데이터 구조입니다. TLS 인증서는 인증 기관(CA)이 서명을 합니다. 여기에는 서버 이름, 유효 기간, 퍼블릭 키, 서명 알고리즘 등이 포함됩니다.

인증서 서명

디지털 서명은 인증서에 대한 암호화된 해시입니다. 서명은 인증서 데이터의 무결성을 확인하는 데 사용됩니다. 사설 CA는 다양한 크기의 인증서 내용에 대해 SHA256과 같은 암호화 해시 함수를 사용하여 서명을 생성합니다. 이 해시 함수는 효과적으로 위조 불가한 고정 크기 데이터 문자열을 생성합니다. 이 문자열을 해시라고 합니다. 그런 다음 CA는 개인 키로 해시 값을 암호화하고 암호화된 해시를 인증서와 연결합니다.

인증 기관

인증 기관(CA) 는 디지털 인증서를 발급하고 필요한 경우 해지합니다. 가장 일반적인 유형의 인증서는 ISO X.509 표준을 기반으로 합니다. X.509 인증서는 인증서 보안 주체의 ID를 확인하고 해당 ID를 공개 키에 바인딩합니다. 보안 주체는 사용자, 애플리케이션, 컴퓨터 또는 기타 장치일 수 있습니다. CA는 인증서 내용을 해시한 다음, 인증서의 퍼블릭 키와 관련된 프라이빗 키로 해당 해시를 암호화하여 인증서를 서명합니다. 보안 주체의 자격 증명을 확인해야 하는 웹 브라우저와 같은 클라이언트 애플리케이션은 공개 키를 사용하여 인증서 서명을 해독합니다. 그런 다음 인증서 내용을 해시하고 해시된 값을 해독된 서명과 비교하여 일치하는지 확인합니다. 인증서에 대한 자세한 내용은 인증서 서명 섹션을 참조하세요.

AWS Private CA 를 사용하여 프라이빗 CA를 생성하고 프라이빗 CA를 사용하여 인증서를 발급할 수 있습니다. 프라이빗 CA는 조직 내에서 사용할 수 있는 프라이빗 SSL/TLS 인증서만 발급합니다. 자세한 내용은 프라이빗 인증서 단원을 참조하십시오. 또한 프라이빗 CA를 사용하려면 인증서가 필요합니다. 자세한 내용은 CA 인증서 단원을 참조하십시오.

루트 CA

인증서 발급의 기반이 되는 암호화 구성 요소 및 신뢰 루트입니다. 루트 CA는 인증서 서명(발급)을 위한 프라이빗 키와 루트 CA를 식별하고 프라이빗 키를 CA 이름에 바인딩하는 루트 인증서로 구성됩니다. 루트 인증서는 환경 내에 있는 각 엔터티의 신뢰 저장소에 배포됩니다. 관리자는 신뢰하는 CA만 포함하도록 신뢰 저장소를 구성합니다. 관리자는 해당 환경에 있는 엔터티의 운영 체제, 인스턴스 및 호스트 컴퓨터 이미지로 신뢰 저장소를 업데이트하거나 빌드합니다. 리소스는 상호 연결을 시도할 때 각 엔터티가 제시하는 인증서를 확인합니다. 클라이언트는 인증서의 유효 기간을 검사하고 인증서에서 신뢰 저장소에 설치된 루트 인증서까지의 체인이 존재하는지 여부를 확인합니다. 이러한 조건이 충족

되면 리소스 간에 “핸드셰이크”가 수행됩니다. 이러한 핸드셰이크는 각 엔터티의 신원을 암호로 상대에게 증명하고, 이들 간에 암호화된 통신 채널(TLS/SSL)을 생성합니다.

CA 인증서

인증 기관(CA) 인증서는 CA의 자격 증명 확인 및 인증서에 포함된 퍼블릭 키와 바인딩합니다.

AWS Private CA 를 사용하여 프라이빗 루트 CA 또는 프라이빗 하위 CA를 생성할 수 있으며, 각 CA는 CA 인증서로 지원됩니다. 하위 CA 인증서는 신뢰 체인에서 보다 상위에 있는 또 다른 CA 인증서에 의해 서명됩니다. 그러나 루트 CA의 경우 인증서는 자체 서명됩니다. 외부 루트 기관(예: 온프레미스에서 호스팅됨)을 설정할 수도 있습니다. 그런 다음 루트 기관을 사용하여 AWS Private CA에서 호스팅하는 하위 루트 CA 인증서에 서명할 수 있습니다.

다음 예제에서는 AWS Private CA X.509 CA 인증서에 포함된 일반적인 필드를 보여줍니다. CA 인증서의 경우 `Basic Constraints` 필드의 `CA:` 값은 `TRUE`로 설정된다는 점에 유의하십시오.

```
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
```

루트 CA 인증서

인증 기관(CA)는 일반적으로 명확하게 정의된 상위-하위 관계에 있는 여러 다른 CA가 포함된 계층 구조 내에 존재합니다. 하위 또는 종속 CA는 상위 CA에서 인증되어 인증서 체인을 생성합니다. 계층 구조의 최상위에 있는 CA를 루트 CA라고 하며 해당 인증서를 루트 인증서라고 합니다. 이 인증서는 일반적으로 자체 서명됩니다.

최종 엔터티 인증서

최종 엔터티 인증서는 서버, 인스턴스, 컨테이너 또는 디바이스 같은 리소스를 식별합니다. CA 인증서와 달리 최종 엔터티 인증서를 사용해 인증서를 발급할 수 없습니다. 최종 엔터티 인증서의 다른 일반적인 용어는 "클라이언트" 또는 "잎" 인증서입니다.

자체 서명된 인증서

상위 CA 대신 발급자가 서명한 인증서입니다. CA에서 유지 관리하는 안전한 루트에서 발급된 인증서와 달리, 자체 서명된 인증서는 자체 루트의 역할을 하므로 유선 암호화를 제공하는 데는 사용할 수 있지만, ID를 확인하는 데는 사용할 수 없기 때문에 해지가 불가능하다는 심각한 한계가 있습니다. 보안 관점에서는 용납할 수 없습니다. 그러나 조직은 쉽게 생성할 수 있고, 전문 지식이나 인프라가 필요하지 않으며, 많은 애플리케이션에서 수용한다는 점에서 이를 사용합니다. 자체 서명된 인증서를 발급하기 위한 컨트롤은 없습니다. 이러한 인증서는 만료 날짜를 추적할 수 있는 방법이 없으므로 인증서를 사용하는 조직은 인증서 만료로 인한 가동 중단이라는 커다란 위험 부담을 안게 됩니다.

프라이빗 인증서

AWS Private CA 인증서는 조직 내에서 사용할 수 있지만 퍼블릭 인터넷에서는 신뢰할 수 없는 프라이빗 SSL/TLS 인증서입니다. 이를 사용하여 클라이언트, 서버, 애플리케이션, 서비스, 장치 및 사용자와 같은 리소스를 식별할 수 있습니다. 암호화된 보안 통신 채널을 구축할 때 각 리소스는 다음과 같은 인증서와 암호화 기술을 사용하여 다른 리소스에 대한 자격 증명을 증명합니다. 내부 API 엔드포인트, 웹 서버, VPN 사용자, IoT 디바이스 및 기타 여러 애플리케이션은 프라이빗 인증서를 사용하여 보안 운영에 필요한 암호화된 통신 채널을 설정합니다. 기본적으로 프라이빗 인증서는 공개적으로 신뢰되지 않습니다. 내부 관리자는 프라이빗 인증서를 신뢰하고 인증서를 배포하도록 애플리케이션을 명시적으로 구성해야 합니다.

```
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
```

인증서 경로

인증서에 의존하는 클라이언트는 최종 엔터티 인증서에서 중간 인증서 체인을 통해 신뢰할 수 있는 루트까지의 경로가 존재하는지 확인합니다. 클라이언트는 경로에 있는 각 인증서가 유효한지(해지되지

않았는지) 확인합니다. 또한 최종 엔터티 인증서가 만료되지 않았는지, 무결성을 갖추고 있는지(변조 또는 수정되지 않았는지), 인증서의 제약 조건이 적용되는지 확인합니다.

경로 길이 제약

CA 인증서에 대한 기본 제약 조건 pathLenConstraint는 아래 체인에 존재할 수 있는 하위 CA 인증서 수를 설정합니다. 예를 들어 경로 길이 제한 조건이 0인 CA 인증서에는 하위 CA가 있을 수 없습니다. 경로 길이 제한 조건이 있는 CA는 그 아래에 최대 1개의 하위 CA 수준을 가질 수 있습니다. RFC 5280에서는 이를 “유효한 인증 경로에서 이 인증서를 따를 수 있는 자체 발급되지 않은 중간 인증서의 최대 수”로 정의합니다. 경로 길이 값에는 최종 엔터티 인증서가 제외되지만 검증 체인의 “길이” 또는 “깊이”에 대한 비공식 언어에는 최종 엔터티 인증서가 포함될 수 있어 혼란을 야기할 수 있습니다.

내 요구에 가장 적합한 인증서 서비스는 무엇입니까?

X.509 인증서를 발급하고 배포하기 위한 두 가지 AWS 서비스가 있습니다. 필요에 맞는 서비스를 선택하십시오. 고려 사항에는 퍼블릭 또는 프라이빗 인증서, 사용자 지정 인증서, 다른 AWS 서비스에 배포하려는 인증서 또는 자동 인증서 관리 및 갱신이 필요한지 여부가 포함됩니다.

1. AWS Private CA - 이 서비스는 AWS 클라우드 내부에 PKI(퍼블릭 키 인프라)를 구축하는 기업 고객을 대상으로 하며 조직 내에서 비공개로 사용할 수 있도록 고안되었습니다. 를 사용하면 자체 CA 계층 구조를 생성하고 내부 사용자 AWS Private CA, 컴퓨터, 애플리케이션, 서비스, 서버 및 기타 디바이스를 인증하고 컴퓨터 코드에 서명하기 위한 인증서를 발급할 수 있습니다. 사설 CA에서 발급한 인증서는 인터넷이 아닌 조직 내에서만 신뢰할 수 있습니다.

   사설 CA를 생성하면 외부 CA에서 유효성 검사를 받지 않고 직접 인증서를 발급하고 조직의 내부 요구 사항에 맞게 사용자 지정할 수 있습니다. 예를 들어 다음을 수행할 수 있습니다.

   - 보안 주체 이름으로 인증서를 생성합니다.
   - 만료 날짜가 있는 인증서를 생성합니다.
   - 지원되는 모든 개인 키 알고리즘과 키 길이를 사용하세요.
   - 지원되는 서명 알고리즘을 모두 사용하세요.
   - 템플릿을 사용하여 인증서 발급을 제어합니다.

   이 서비스에 적합한 위치에 있습니다. 시작하려면 https://console.aws.amazon.com/acm-pca/ 콘솔에 로그인하세요.
2. AWS Certificate Manager (ACM) -이 서비스는 공개적으로 신뢰할 수 있는 보안 웹이 필요한 엔터프라이즈 고객을 위해 TLS를 사용하여 인증서를 관리합니다. ACM 인증서를 AWS Elastic Load Balancing, Amazon CloudFront, Amazon API Gateway 및 기타 통합 서비스에 배포할 수 있습니다. 이러한 종류의 가장 일반적인 애플리케이션은 중요한 트래픽 요구 사항을 가진 안전한 공개 웹사이트입니다.

   이 서비스에서는 ACM에서 제공하는 공인 인증서(ACM 인증서) 또는 ACM으로 가져온 인증서를 사용할 수 있습니다. AWS Private CA 를 사용하여 CA를 생성하는 경우 ACM은 해당 프라이빗 CA에서 인증서 발급을 관리하고 인증서 갱신을 자동화할 수 있습니다.

   자세한 내용은 AWS Certificate Manager 사용 설명서를 참조하십시오.

AWS Private CA 모범 사례

모범 사례는를 AWS Private CA 효과적으로 사용하는 데 도움이 되는 권장 사항입니다. 다음 모범 사례는 현재 AWS Certificate Manager 및 AWS Private CA 고객의 실제 경험을 기반으로 합니다.

CA 구조 및 정책 문서화

AWS 에서는 CA 운영에 대한 모든 정책 및 관행을 문서화할 것을 권장합니다. 여기에는 다음이 포함될 수 있습니다.

- CA 구조에 대한 의사 결정을 위한 추론
- CA 및 CA 관계를 보여주는 다이어그램
- CA 유효 기간에 대한 정책
- CA 승계 계획
- 경로 길이에 대한 정책
- 권한 카탈로그
- 관리 제어 구조에 대한 설명
- 보안

이 정보는 CP(인증 정책)와 CPS(인증 사례 명세서)라는 두 문서에서 캡처할 수 있습니다. CA 작업에 대한 중요한 정보를 캡처하기 위한 프레임워크는 RFC 3647을 참조하십시오.

가능한 경우 루트 CA 사용 최소화

루트 CA는 일반적으로 중간 CA에 대한 인증서를 발급하기 위한 용도로만 사용해야 합니다. 이렇게 하면 중간 CA가 최종 엔터티 인증서를 발급하는 일상적인 작업을 수행하는 동안 루트 CA를 손상 없이 저장할 수 있습니다.

그러나 조직의 현재 관행이 루트 CA에서 직접 최종 엔터티 인증서를 발급하는 경우는 보안 및 운영 제어를 개선하면서이 워크플로를 지원할 AWS Private CA 수 있습니다. 이 시나리오에서 최종 엔터티 인증서를 발급하려면 루트 CA가 최종 엔터티 인증서 템플릿을 사용하도록 허용하는 IAM 권한 정책이 필요합니다. IAM 정책에 대한 자세한 내용은 용 Identity and Access Management(IAM) AWS Private Certificate Authority 단원을 참조하십시오.

> **Note**
>
> 이 구성의 경우, 제한 조건으로 인해 운영 문제가 야기될 수 있습니다. 예를 들어 루트 CA가 손상되거나 손실된 경우 새 루트 CA를 생성해서 환경의 모든 클라이언트에 배포해야 합니다. 이러한 복구 프로세스가 완료될 때까지 새 인증서를 발급할 수 없습니다. 루트 CA에서 직접 인증서를 발급하면 액세스를 제한하고 루트에서 발급된 인증서 수를 한정하지 못하게 되므로 둘 다 루트 CA 관리에 대한 모범 사례로 간주됩니다.

루트 CA에 고유한 권한 부여 AWS 계정

서로 다른 두 AWS 계정에 루트 CA와 하위 CA를 생성하는 것이 모범 사례입니다. 이렇게 하면 루트 CA에 대한 추가 보호 및 액세스 제어를 제공할 수 있습니다. 이를 위해 한 계정의 하위 CA에서 CSR을 내보내고 다른 계정의 루트 CA로 CSR을 서명하면 됩니다. 이 접근 방식의 이점은 계정별로 CA에 대한 제어를 분리할 수 있다는 것입니다. 단점은 AWS Management Console 마법사를 사용하여 루트 CA에서 하위 CA의 CA 인증서에 서명하는 프로세스를 간소화할 수 없다는 것입니다.

> **Important**
>
> 에 액세스할 때마다 다중 인증(MFA)을 사용하는 것이 좋습니다 AWS Private CA.

별도의 관리자 및 발급자 역할

CA 관리자 역할은 최종 엔터티 인증서만 발급하면 되는 사용자와는 분리되어야 합니다. CA 관리자와 인증서 발급자가 동일한에 있는 경우 해당 용도에 맞는 IAM 사용자를 생성하여 발급자 권한을 제한 AWS 계정할 수 있습니다.

인증서의 관리형 해지 구현

관리형 해지는 인증서가 해지되면 인증서 클라이언트에게 자동으로 알림을 제공합니다. 암호화 정보가 손상되었거나 잘못 발급된 경우 인증서를 해지해야 할 수 있습니다. 클라이언트는 일반적으로 해지된 인증서 수락을 거부합니다.는 관리형 해지를 위한 두 가지 표준 옵션인 온라인 인증서 상태 프로토콜(OCSP)과 인증서 해지 목록(CRLs AWS Private CA 제공합니다. 자세한 내용은 AWS Private CA 인증서 해지 방법 계획 단원을 참조하십시오.

켜기 AWS CloudTrail

프라이빗 CA를 생성하고 운영을 시작하기 전에 CloudTrail 로깅을 활성화합니다. CloudTrail을 사용하면 계정에 대한 AWS API 호출 기록을 검색하여 배포를 모니터링할 수 있습니다 AWS . 이 기록에는 AWS Management Console, AWS SDKs, AWS Command Line Interface및 상위 수준 AWS 서비스에서 수행된 API 호출이 포함됩니다. 또한 어떤 사용자 및 계정이 PCA API 작업을 호출했는지, 어떤 소스 IP 주소에 호출이 수행되었는지, 언제 호출이 발생했는지도 확인할 수 있습니다. API를 사용해 CloudTrail을 애플리케이션에 통합시킴으로써 조직에 대한 추적 생성을 자동화하고 추적 상태를 확인하며 관리자가 CloudTrail 로깅을 활성화하고 비활성화하는 방식을 제어할 수 있습니다. 자세한 내용은 추적 생성을 참조하세요. AWS Private CA 작업에 대한 예제 추적을 를 사용하여 AWS Private Certificate Authority API 호출 로깅 AWS CloudTrail 보려면 로 이동합니다.

CA 프라이빗 키 업데이트

프라이빗 CA의 프라이빗 키를 정기적으로 업데이트하는 것이 가장 좋습니다. 새 CA 인증서를 가져와서 키를 업데이트하거나 사설 CA를 새 CA로 바꿀 수 있습니다.

> **Note**
>
> CA 자체를 교체하는 경우 CA의 ARN이 변경된다는 점에 유의하세요. 이로 인해 하드 코딩된 ARN에 의존하는 자동화가 실패할 수 있습니다.

미사용 CA 삭제

프라이빗 CA를 영구적으로 삭제할 수 있습니다. CA가 더 이상 필요하지 않거나 새 프라이빗 키가 있는 CA로 교체하려는 경우 그렇게 하는 것이 좋습니다. CA를 안전하게 삭제하려면 프라이빗 CA 삭제에 설명된 절차를 따르는 것이 좋습니다.

> **Note**
>
> AWS 는 CA가 삭제될 때까지 CA에 대한 요금을 청구합니다.

CRL에 대한 퍼블릭 액세스 차단

AWS Private CA 에서는 CRLs이 포함된 버킷에서 Amazon S3 퍼블릭 액세스 차단(BPA) 기능을 사용할 것을 권장합니다. 이렇게 하면 프라이빗 PKI의 세부 정보가 잠재적 공격자에게 불필요하게 노출되

는 것을 방지할 수 있습니다. BPA는 S3 모범 사례이며 새 버킷에서 기본적으로 활성화됩니다. 경우에 따라 추가 설정이 필요합니다. 자세한 내용은 CloudFront를 사용하여 S3 퍼블릭 액세스 차단(BPA) 활성화 단원을 참조하십시오.

Amazon EKS 애플리케이션 모범 사례

AWS Private CA 를 사용하여 X.509 인증서로 Amazon EKS를 프로비저닝하는 경우 Amazon EKS 모범 사례 가이드의 다중 테넌트 환경 보안을 위한 권장 사항을 따르세요. Kubernetes와 AWS Private CA 의 통합에 대한 일반 정보는 를 사용하여 Kubernetes 보호 AWS Private Certificate Authority 섹션을 참조하세요.

와 AWS Private CA 함께 사용 AWS SDK for Java

AWS Private Certificate Authority API를 사용하여 HTTP 요청을 전송하여 프로그래밍 방식으로 서비스와 상호 작용할 수 있습니다. 이 서비스는 HTTP 응답을 반환합니다. 자세한 내용은 AWS Private Certificate Authority API 참조를 참조하세요.

HTTP API 외에도 AWS SDKs 및 명령줄 도구를 사용하여와 상호 작용할 수 있습니다 AWS Private CA. HTTP API보다 이 방법을 사용하는 것이 좋습니다. 자세한 내용은 Amazon Web Services용 도구를 참조하세요. 다음 주제에서는 AWS SDK for Java를 사용하여 AWS Private CA API를 프로그래밍하는 방법을 보여 줍니다.

GetCertificateAuthorityCsr, GetCertificate, and DescribeCertificateAuthorityAuditReport 작업에는 waiter가 지원됩니다. waiter를 사용하여 특정 리소스의 존재 여부 또는 상태에 따라 코드의 진행을 제어 할 수 있습니다. 자세한 내용은 AWS 개발자 블로그의 다음 주제와 의 웨이터 AWS SDK for Java를 참조하세요.

AWS Private CA API 예제

다음 코드 예제에서는 AWS Private CA 에서 API 작업 및 데이터 유형을 선택하는 방법을 보여줍니다 AWS SDK for Java.

주제

- 프로그래밍 방식으로 루트 CA 생성 및 활성화
- 프로그래밍 방식으로 하위 CA 생성 및 활성화
- CreateCertificateAuthority
- Active Directory를 지원하기 위해 인증서 기관 생성 사용
- CreateCertificateAuthorityAuditReport
- CreatePermission
- DeleteCertificateAuthority
- DeletePermission
- DeletePolicy
- DescribeCertificateAuthority
- DescribeCertificateAuthorityAuditReport
- GetCertificate
- GetCertificateAuthorityCertificate

- GetCertificateAuthorityCsr
- GetPolicy
- ImportCertificateAuthorityCertificate
- IssueCertificate
- ListCertificateAuthorities
- ListPermissions
- ListTags
- PutPolicy
- RestoreCertificateAuthority
- RevokeCertificate
- TagCertificateAuthorities
- UntagCertificateAuthority
- UpdateCertificateAuthority
- 사용자 지정 주제 이름을 사용하여 CA 및 인증서 생성
- 사용자 지정 확장으로 인증서 생성

프로그래밍 방식으로 루트 CA 생성 및 활성화

이 Java 샘플은 다음 AWS Private CA API 작업을 사용하여 루트 CA를 활성화하는 방법을 보여줍니다.

- 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);
    }
}
```

프로그래밍 방식으로 하위 CA 생성 및 활성화

이 Java 샘플은 다음 AWS Private CA API 작업을 사용하여 하위 CA를 활성화하는 방법을 보여줍니다.

- 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

다음 Java 샘플은 CreateCerticateAuthority 작업을 사용하는 방법을 보여줍니다.

이 작업은 사설 하위 인증 기관(CA)을 생성합니다. CA 구성, 해지 구성, CA 유형 및 선택적 멱등성 토큰을 지정해야 합니다.

CA 구성은 다음을 지정합니다.

- CA 프라이빗 키를 생성하는 데 사용할 알고리즘 및 키 크기의 이름
- CA가 자체 인증서 서명 요청, CRLs 및 OCSP 응답에 서명하는 데 사용하는 서명 알고리즘의 유형입니다.
- X.500 주체 정보

CRL 구성은 다음을 지정합니다.

- CRL 만료 기간(일)(CRL의 유효 기간)
- CRL이 포함되어 있는 Amazon S3 버킷
- CA에서 발급한 인증서에 포함된 S3 버킷의 CNAME 별칭

성공하면 이 함수는 CA의 Amazon 리소스 이름(ARN)을 반환합니다.

```
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);
    }
}
```

다음과 유사하게 출력되어야 합니다.

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

Active Directory를 지원하기 위해 인증서 기관 생성 사용

다음 Java 샘플은 CreateCerticateAuthority 작업을 사용하여 Microsoft Active Directory(AD)의 Enterprise NTAuth 저장소에 설치할 수 있는 CA를 만드는 방법을 보여줍니다.

이 작업을 수행하면 사용자 지정 객체 식별자(OID) 를 사용하여 개인 루트 인증 기관CA)가 만들어집니다. 자세한 내용과 이에 상응하는 작업의 AWS CLI 예는 Active Directory 로그인을 위한 CA 생성을 참조하세요.

성공하면 이 함수는 CA의 Amazon 리소스 이름(ARN)을 반환합니다.

```
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);
    }
}
```

다음과 유사하게 출력되어야 합니다.

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

CreateCertificateAuthorityAuditReport

다음 Java 샘플은 CreateCertificateAuthorityAuditReport 작업을 사용하는 방법을 보여줍니다.

이 작업은 인증서가 발급 또는 해지될 때마다 나열되는 감사 보고서를 생성합니다. 이 보고서는 입력 시 지정한 Amazon S3 버킷에 저장됩니다. 30분에 한 번씩 새 보고서를 생성할 수 있습니다.

```
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);

   }
}
```

다음과 유사하게 출력되어야 합니다.

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

CreatePermission

다음 Java 샘플에서는 CreatePermission 작업을 사용하는 방법을 보여 줍니다.

작업은 프라이빗 CA의 액세스 권한을 지정된 AWS 서비스 보안 주체에 할당합니다. 서비스는 사설 CA에서 인증서를 생성 및 검색할 수 있는 권한을 부여 받을 수 있는 것은 물론이고, 사설 CA가 부여한 활성 권한을 나열할 수 있습니다. ACM을 통해 인증서를 자동으로 갱신하려면 CA에서 ACM 서비스 보안 주체(`ListPermissions`)에 가능한 모든 권한(`IssueCertificate`, `GetCertificate`, 및)을 할당해야 합니다`acm.amazonaws.com`. ListCertificateAuthorities 작업을 호출하여 CA의 ARN을 찾을 수 있습니다.

권한이 생성되면 ListPermissions 함수를 사용하여 해당 권한을 검사하거나 DeletePermissions 함수를 사용하여 권한을 삭제할 수 있습니다.

```
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

다음 Java 샘플은 DeleteCertificateAuthority 작업을 사용하는 방법을 보여줍니다.

이 작업은 CreateCertificateAuthority 작업을 사용하여 생성한 사설 CA(인증 기관)를 삭제합니다. DeleteCertificateAuthority 작업을 수행하려면 삭제할 CA의 ARN을 제공해야 합니다. ListCertificateAuthorities 작업을 호출하여 ARN을 찾을 수 있습니다. 상태가 CREATING 또는 PENDING_CERTIFICATE인 경우 프라이빗 CA를 즉시 삭제할 수 있습니다. 하지만 이미 인증서를 가져온 경우에는 즉시 삭제할 수 없습니다. 먼저 UpdateCertificateAuthority 작업을 호출하여 CA를 비활성화하고 Status 파라미터를 DISABLED로 설정해야 합니다. 그런 다음 DeleteCertificateAuthority 작업에서 PermanentDeletionTimeInDays 파라미터를 사용하여 7 ~ 30까지 일 수를 지정할 수 있습니다. 이 기간 동안 사설 CA를 disabled 상태로 복원할 수 있습니다. 기본적으로 PermanentDeletionTimeInDays 파라미터를 설정하지 않은 경우 복원 기간은 30일입니다. 이 기간이 만료되면 프라이빗 CA가 영구적으로 삭제되며 복원할 수 없습니다. 자세한 내용은 CA 복원 단원을 참조하십시오.

RestoreCertificateAuthority 작업을 사용하는 방법을 보여주는 Java 예제는 RestoreCertificateAuthority 섹션을 참조하세요.

```
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

다음 Java 샘플은 DeletePermission 작업을 사용하는 방법을 보여줍니다.

작업은 프라이빗 CA가 CreatePermissions 작업을 사용하여 AWS 서비스 보안 주체에게 위임한 권한을 삭제합니다. CreatePermissions ListCertificateAuthorities 작업을 호출하여 CA의 ARN을 찾을 수 있습니다. ListPermissions 함수를 호출하여 CA가 부여한 권한을 검사할 수 있습니다.

```
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

다음 Java 샘플은 DeletePolicy 작업을 사용하는 방법을 보여줍니다.

작업은 프라이빗 CA에 연결된 리소스 기반 정책을 삭제합니다. 리소스 기반 정책은 계정 간 CA 공유를 활성화하는 데 사용됩니다. ListCertificateAuthorities 작업을 호출하여 프라이빗 CA의 ARN을 찾을 수 있습니다.

관련 API 작업에는 PutPolicy 및 GetPolicy가 포함됩니다.

```
package com.amazonaws.samples;

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

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

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

public class DeletePolicy {

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

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

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

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

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

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

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

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

```
        }
    }
}
```

DescribeCertificateAuthority

다음 Java 샘플은 DescribeCertificateAuthority 작업을 사용하는 방법을 보여줍니다.

작업에는 프라이빗 인증 기관(CA)에 대한 정보가 나열됩니다. 프라이빗 CA의 ARN(Amazon 리소스 이름)을 지정해야 합니다. 출력에는 CA 상태가 포함됩니다. 이것은 다음 중 하나가 될 수 있습니다.

- CREATING – AWS Private CA 가 프라이빗 인증 기관을 생성하고 있습니다.
- PENDING_CERTIFICATE - 인증서가 보류 중입니다. 온프레미스 루트 또는 하위 CA를 사용하여 프라이빗 CA CSR에 서명한 후 이를 PCA로 가져와야 합니다.
- ACTIVE - 프라이빗 CA가 활성 상태입니다.
- DISABLED – 프라이빗 CA가 비활성화 되었습니다.
- EXPIRED - 프라이빗 CA 인증서가 만료되었습니다.
- FAILED - 프라이빗 CA를 생성할 수 없습니다.
- DELETED - 프라이빗 CA는 복원 기간 내에 있으며, 복원 기간이 지나면 영구적으로 삭제됩니다.

```
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

다음 Java 샘플은 DescribeCertificateAuthorityAuditReport 작업을 사용하는 방법을 보여줍니다.

이 작업은 CreateCertificateAuthorityAuditReport 작업을 호출하여 생성한 특정 감사 보고서에 대한 정보를 나열합니다. 감사 정보는 인증 기관(CA) 프라이빗 키를 사용할 때마다 생성됩니다. 프라이빗 키는 인증서를 발급하거나, CRL에 서명하거나, 인증서를 해지할 때 사용됩니다.

```
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);
   }
}
```

다음과 유사하게 출력되어야 합니다.

```
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

다음 Java 샘플은 GetCertificate 작업을 사용하는 방법을 보여줍니다.

이 작업은 사설 CA에서 인증서를 검색합니다. IssueCertificate 작업을 호출할 때 인증서의 ARN이 반환됩니다. 사설 CA의 ARN과 GetCertificate 작업을 호출할 때 발급된 인증서의 ARN을 모두 지정해야 합니다. 인증서가 ISSUED 상태에 있는 경우 인증서를 검색할 수 있습니다. CreateCertificateAuthorityAuditReport 작업을 호출하여 사설 CA에서 발급 및 해지된 모든 인증서에 대한 정보가 포함된 보고서를 생성할 수 있습니다.

```
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);
    }
}
```

출력은 지정한 인증 기관 (CA) 및 인증서에 대한 다음과 유사한 인증서 체인이어야 합니다.

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

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

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

GetCertificateAuthorityCertificate

다음 Java 샘플은 GetCertificateAuthorityCertificate 작업을 사용하는 방법을 보여줍니다.

이 작업은 사설 인증 기관(CA)을 위한 인증서와 인증서 체인을 검색합니다. 인증서와 체인은 모두 PEM 형식의 base64로 인코딩된 문자열입니다. 체인에는 CA 인증서가 포함되지 않습니다. 체인의 각 인증서는 이전 인증서에 서명합니다.

```
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);
   }
}
```

출력은 지정한 인증 기관(CA) 의 다음과 비슷한 인증서 및 체인이어야 합니다.

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

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

GetCertificateAuthorityCsr

다음 Java 샘플은 GetCertificateAuthorityCsr 작업을 사용하는 방법을 보여줍니다.

이 작업은 사설 인증 기관(CA)을 위한 인증서 서명 요청(CSR)을 검색합니다. CSR은 CreateCertificateAuthority 작업을 호출할 때 생성됩니다. CSR을 온프레미스 X.509 인프라로 가져와서

루트 또는 하위 CA를 사용해 서명합니다. 그런 다음 ImportCertificateAuthorityCertificate 작업을 호출하여 ACM PCA에 서명된 인증서를 다시 가져옵니다. CSR은 PEM 형식의 base64 인코딩 문자열로 반환됩니다.

```
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);
    }
}
```

지정한 CA(인증 기관)에 대한 출력은 다음과 유사해야 합니다. 인증서 서명 요청(CSR)은 PEM 형식으로 base64로 인코딩됩니다. CSR을 로컬 파일에 저장하고 온프레미스 X.509 인프라로 가져와서 루트 또는 하위 CA를 사용해 서명합니다.

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

GetPolicy

다음 Java 샘플은 GetPolicy 작업을 사용하는 방법을 보여줍니다.

이 작업은 프라이빗 CA에 연결된 리소스 기반 정책을 검색합니다. 리소스 기반 정책은 계정 간 CA 공유를 활성화하는 데 사용됩니다. ListCertificateAuthorities 작업을 호출하여 프라이빗 CA의 ARN을 찾을 수 있습니다.

관련 API 작업에는 PutPolicy 및 DeletePolicy가 포함됩니다.

```
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

다음 Java 샘플은 ImportCertificateAuthorityCertificate 작업을 사용하는 방법을 보여줍니다.

이 작업은 서명된 프라이빗 CA 인증서를 로 가져옵니다 AWS Private CA. 이 작업을 호출할 수 있으려면 먼저 CreateCertificateAuthority 작업을 호출하여 사설 CA를 생성해야 합니다. 그런 다음, GetCertificateAuthorityCsr 작업을 호출하여 인증서 서명 요청(CSR)을 생성해야 합니다. CSR을 온프레미스 CA로 가져간 다음 루트 인증서 또는 하위 인증서를 사용하여 서명합니다. 인증서 체인을 만들고 서명된 인증서와 인증서 체인을 작업 디렉터리에 복사합니다.

```
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

다음 Java 샘플은 IssueCertificate 함수를 사용하는 방법을 보여줍니다.

이 함수는 프라이빗 인증 기관(CA)을 사용하여 최종 엔터티 인증서를 발급합니다. 이 함수는 인증서의 Amazon 리소스 이름(ARN)을 반환합니다. GetCertificate를 호출하고 ARN을 지정하여 인증서를 검색할 수 있습니다.

> **Note**
>
> IssueCertificate 작업을 수행하려면 인증서 템플릿을 지정해야 합니다. 이 예제에서는 `EndEntityCertificate/V1` 템플릿을 사용합니다. 사용 가능한 모든 템플릿에 대한 자세한 내용은 AWS Private CA 인증서 템플릿 사용 섹션을 참조하세요.

```
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);
    }
}
```

다음과 유사하게 출력되어야 합니다.

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

ListCertificateAuthorities

다음 Java 샘플은 ListCertificateAuthorities 작업을 사용하는 방법을 보여줍니다.

이 작업은 CreateCertificateAuthority 작업을 사용하여 생성한 사설 CA를 나열합니다.

```
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());
   }
}
```

나열할 CA가 있는 경우에는 다음과 비슷하게 출력이 됩니다.

```
[{
 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

다음 Java 샘플에서는 ListPermissions 함수를 사용하는 방법을 보여 줍니다.

이 함수는 사설 CA가 할당한 권한(있는 경우)을 나열합니다. `ListPermissions`, `GetCertificate`및 `IssueCertificate`를 포함한 권한은 CreatePermission 작업을 사용하여 AWS 서비스 보안 주체에게 할당하고 DeletePermissions 작업을 사용하여 취소할 수 있습니다.

```
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);
   }
}
```

지정된 사설 CA가 서비스 보안 주체에 권한을 할당한 경우 출력은 다음과 유사해야 합니다.

```
[{
       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

다음 Java 샘플은 ListTags 작업을 사용하는 방법을 보여줍니다.

이 작업은 사설 CA에 연결되는 태그를 나열합니다. 태그는 CA를 식별 및 구성하는 데 사용할 수 있는 레이블입니다. 각 태그는 키와 값(선택사항)으로 구성됩니다. TagCertificateAuthority 작업을 호출하여 CA에 태그를 하나 이상 추가합니다. UntagCertificateAuthority 작업을 호출하여 태그를 제거합니다.

```
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);
    }
}
```

나열할 태그가 있는 경우에는 다음과 비슷하게 출력이 됩니다.

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

PutPolicy

다음 Java 샘플은 PutPolicy 작업을 사용하는 방법을 보여줍니다.

이 작업은 리소스 기반 정책을 프라이빗 CA에 연결하여 계정 간 공유를 가능하게 합니다. 정책에서 승인하면 다른 AWS 계정에 있는 보안 주체는 소유하지 않은 프라이빗 CA를 사용하여 프라이빗 최종 엔터티 인증서를 발급하고 갱신할 수 있습니다. ListCertificateAuthorities 작업을 호출하여 프라이빗 CA의 ARN을 찾을 수 있습니다. 정책의 예제는 리소스 기반 정책에 대한 AWS Private CA 지침을 참조하세요.

정책이 CA에 연결되면 GetPolicy 작업을 사용하여 정책을 검사하거나 DeletePolicy 작업을 사용하여 삭제할 수 있습니다.

```
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

다음 Java 샘플은 RestoreCertificateAuthority 작업을 사용하는 방법을 보여줍니다. 프라이빗 CA는 복원 기간 중 언제든지 복원할 수 있습니다. 현재 이 기간은 삭제일로부터 7~30일 동안 지속될 수 있으며 CA를 삭제할 때 정의할 수 있습니다. 자세한 내용은 CA 복원 단원을 참조하십시오. DeleteCertificateAuthority Java 예제도 참조하세요.

```
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

다음 Java 샘플은 RevokeCertificate 작업을 사용하는 방법을 보여줍니다.

이 작업은 IssueCertificate 작업을 호출하여 발급된 인증서를 해지합니다. 프라이빗 CA를 생성하거나 업데이트할 때 인증서 해지 목록(CRL)을 활성화한 경우 취소된 인증서에 대한 정보가 CRL에 포함됩니다.는 사용자가 지정한 Amazon S3 버킷에 CRL을 AWS Private CA 기록합니다. 자세한 내용은 CRL구성 구조를 참조하세요.

```
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

다음 Java 샘플은 TagCertificateAuthority 작업을 사용하는 방법을 보여줍니다.

이 작업은 사설 CA에 하나 이상의 태그를 추가합니다. 태그는 AWS 리소스를 식별하고 구성하는 데 사용할 수 있는 레이블입니다. 각 태그는 키와 값(선택사항)으로 구성됩니다. 이 작업을 호출할 때 사설 CA를 Amazon 리소스 이름(ARN)으로 지정합니다. 키-값 페어를 사용하여 태그를 지정합니다. 해당 CA의 구체적인 특성을 식별하기 위해 하나의 사설 CA에만 태그를 적용할 수 있습니다. 또는 이러한 CA 간의 공통 관계를 필터링하기 위해 여러 사설 CA에 동일한 태그를 적용할 수 있습니다. 하나 이상의 태그를 제거하려면 UntagCertificateAuthority 작업을 사용합니다. ListTags 작업을 호출하여 CA에 어떤 태그가 연결되는지 확인합니다.

```
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

다음 Java 샘플은 UntagCertificateAuthority 작업을 사용하는 방법을 보여줍니다.

이 작업은 사설 CA에서 하나 이상의 태그를 제거합니다. 태그는 키-값 쌍으로 이루어져 있습니다. 이 작업을 호출할 때 태그의 값 부분을 지정하지 않은 경우에는 값에 관계 없이 태그가 제거됩니다. 값을 지정하는 경우 태그가 지정된 값과 연결된 경우에만 태그가 제거됩니다. 사설 CA에 태그를 추가하려면 TagCertificateAuthority 작업을 사용하십시오. ListTags 작업을 호출하여 CA에 어떤 태그가 연결되는지 확인합니다.

```
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

다음 Java 샘플은 UpdateCertificateAuthority 작업을 사용하는 방법을 보여줍니다.

이 작업은 사설 CA의 상태 또는 구성을 업데이트합니다. 업데이트하려면 프라이빗 CA가 `ACTIVE` 또는 `DISABLED` 상태에 있어야 합니다. `ACTIVE` 상태에 있는 프라이빗 CA를 비활성화하거나 `DISABLED` 상태에 있는 CA를 다시 활성화할 수 있습니다.

```
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;
        }
    }
}
```

사용자 지정 주제 이름을 사용하여 CA 및 인증서 생성

CustomAttribute 객체를 사용하면 관리자가 사용자 지정 객체 식별자(OID)를 프라이빗 CA 및 인증서에 전달할 수 있습니다. 사용자 지정 OID를 사용하여 조직의 구조와 요구 사항을 반영하는 특수 주제 이름 계층 구조를 만들 수 있습니다. ApiPassthrough 템플릿 중 하나를 사용하여 사용자 지정된 인증서를 만들어야 합니다. 템플릿에 대한 자세한 내용은 AWS Private CA 템플릿 품종 단원을 참조하십시오. 사용자 지정 속성 사용에 대한 자세한 정보는 프라이빗 최종 엔터티 인증서 발급 및 에서 프라이빗 CA 생성 AWS Private CA 섹션을 참조하세요.

StandardAttributes를 CustomAttributes와 함께 사용할 수 없습니다. 하지만 표준 OID를 CustomAttributes의 일부로 전달할 수 있습니다. 기본 보안 주체 이름 OID는 다음 표에 나열되어 있습니다.

보안 주체 이름	객체 ID
국가	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
로컬리티	2.5.4.7
Organization	2.5.4.10
OrganizationalUnit	2.5.4.11
필명	2.5.4.65
SerialNumber	2.5.4.5
State	2.5.4.8
Surname	2.5.4.4
제목	2.5.4.12

주제

- CustomAttribute이 포함된 CA 생성
- CustomAttribute이 포함된 인증서 발급

CustomAttribute이 포함된 CA 생성

```
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);
    }
}
```

CustomAttribute이 포함된 인증서 발급

```
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);
    }
}
```

사용자 지정 확장으로 인증서 생성

CustomExtension 객체를 사용하면 관리자가 프라이빗 인증서에 사용자 지정 X.509 확장을 설정할 수 있습니다. APIPassthrough 템플릿 중 하나를 사용하여 사용자 지정 인증서를 생성해야 합니다. 템플릿에 대한 자세한 내용은 AWS Private CA 템플릿 품종 단원을 참조하십시오. 사용자 지정 확장에 대한 자세한 내용은 프라이빗 최종 엔터티 인증서 발급 섹션을 참조하세요.

주제

- NameConstraints 확장자를 사용하여 하위 CA를 활성화합니다.
- QC 명령문 확장이 포함된 인증서 발급

NameConstraints 확장자를 사용하여 하위 CA를 활성화합니다.

```
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 명령문 확장이 포함된 인증서 발급

```
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);
    }
}
```

Matter 인증서를 구현 AWS Private CA 하는 데 사용

AWS Private Certificate Authority API를 사용하여 Matter 연결 표준을 준수하는 인증서를 생성할 수 있습니다. Matter는 여러 엔지니어링 플랫폼에서 사물 인터넷(IoT) 장치의 보안 및 일관성을 개선하는 인증서 구성을 지정합니다. Matter에 대한 자세한 내용은 buildwithmatter.com을 참조하세요.

2023년 10월에 릴리스된 Matter 1.2는 인증서 해지 목록(CRLs)을 사용하여 DAC 해지를 지원합니다. 현재 Matter 표준을 준수하는 데 도움이 되도록 Matter 인증서를 발급하는 CAs에 대해 CRL 해지를 활성화하면 `CrlConfiguration` 객체의 `CrlDistributionPointExtensionConfiguration` 구조에서를 `OmitExtension`로 설정합니다`true`.

일반적으로 CAs 인증서 체인 검증을 수행하는 신뢰 당사자가 CRL을 가져오고 인증서 상태를 확인할 수 있도록 발급하는 인증서에 CRL 배포 지점(CDP)을 포함합니다. 여기서 CDP URI는 인증서에 기록되지 않습니다. 대신 사용자는 신뢰할 수 있는 Matter 데이터 스토어인 Matter Distributed Compliance Ledger(DCL)에서 CDPs를 가져옵니다. DACs를 검증할 때 검색할 수 있도록 CDP URI를 Matter DCL에 업로드해야 합니다. CDP URI 결정에 대한 자세한 내용은 섹션을 참조하세요CRL 배포 지점(CDP) URI 확인. Matter에 대한 자세한 내용은 Matter 표준 홈 페이지를 참조하세요.

주제

- 제품 인증 기관(PAA) 활성화
- 제품 증명 중간(PAI) 활성화
- 디바이스 인증 인증서(DAC) 생성
- 노드 운영 인증서(NOC)에 대한 루트 CA를 활성화합니다.
- 노드 운영 인증서(NOC)에 대한 하위 CA 활성화
- 노드 운영 인증서(NOC) 생성

제품 인증 기관(PAA) 활성화

이 Java 샘플은 RootCACertificate_APIPassthrough/V1 정의 템플릿을 사용하여 제품 증명을 위한 Matter Root CA(PAA) 인증서를 생성하고 설치하는 방법을 보여줍니다. PAAs의 경우 AuthorityKeyIdentifier(AKI) 확장은 선택 사항입니다. AKI를 설정하려면 Base64-encoded AKI 값을 생성하고 CustomExtension을 통해 전달해야 합니다.

이 예제에서는 다음 AWS Private CA API 작업을 호출합니다.

- CreateCertificateAuthority
- GetCertificateAuthorityCsr
- IssueCertificate
- GetCertificate
- ImportCertificateAuthorityCertificate

문제가 발생하는 경우 문제 해결 섹션의 AWS Private CA Matter 준수 인증서 오류 문제 해결을 참조하세요.

```
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);
    }
}
```

제품 증명 중간(PAI) 활성화

이 Java 샘플은 BlankSubordinateCACertificate_PathLen0_APIPassthrough/V1 정의 템플릿을 사용하여 제품 증명을 위한 Matter Subordinate CA(PAI) 인증서를 생성하고 설치하는 방법을 보여줍니다. Base64로 인코딩된 KeyUsage 값을 생성하여 CustomExtension.을 통해 전달해야 합니다.

이 예제에서는 다음 AWS Private CA API 작업을 호출합니다.

- CreateCertificateAuthority

- GetCertificateAuthorityCsr
- IssueCertificate
- GetCertificate
- ImportCertificateAuthorityCertificate
- GetCertificateAuthorityCertificate

문제가 발생하는 경우 문제 해결 섹션의 AWS Private CA Matter 준수 인증서 오류 문제 해결을 참조하세요.

```
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);
    }
}
```

디바이스 인증 인증서(DAC) 생성

이 Java 샘플은 BlankEndEntityCertificate_CriticalBasicConstraints_APIPassthrough/V1 템플릿을 사용하여 Matter 디바이스 증명 인증서를 생성하는 방법을 보여줍니다. Base64로 인코딩된 KeyUsage 값을 생성하여 CustomExtension.을 통해 전달해야 합니다.

이 예제에서는 다음 AWS Private CA API 작업을 호출합니다.

- IssueCertificate

문제가 발생하는 경우 문제 해결 섹션의 AWS Private CA Matter 준수 인증서 오류 문제 해결을 참조하세요.

```
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);
   }
}
```

노드 운영 인증서(NOC)에 대한 루트 CA를 활성화합니다.

이 Java 샘플은 RootCACertificate_APIPassthrough/V1 정의 템플릿을 사용하여 Matter Root CA 인증서를 생성하고 설치하여 NOCs를 발급하는 방법을 보여줍니다. NOC 루트 CA 인증서의 경우 AuthorityKeyIdentifier(AKI) 확장은 선택 사항입니다. AKI를 설정하려면 Base64-encoded AKI 값을 생성하고 CustomExtension을 통과해야 합니다.

이 예제에서는 다음 AWS Private CA API 작업을 호출합니다.

- CreateCertificateAuthority
- GetCertificateAuthorityCsr
- IssueCertificate
- GetCertificate
- ImportCertificateAuthorityCertificate

문제가 발생하는 경우 문제 해결 섹션의 AWS Private CA Matter 준수 인증서 오류 문제 해결을 참조하세요.

```
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);
    }
}
```

노드 운영 인증서(NOC)에 대한 하위 CA 활성화

이 Java 샘플은 BlankSubordinateCACertificate_PathLen0_APIPassthrough/V1 정의 템플릿을 사용하여 Matter Subordinate CA 인증서를 발급하고 설치하여 NOCs 발급하는 방법을 보여줍니다. Base64로 인코딩된 KeyUsage 값을 생성하여 CustomExtension.을 통해 전달해야 합니다.

이 예제에서는 다음 AWS Private CA API 작업을 호출합니다.

- CreateCertificateAuthority
- GetCertificateAuthorityCsr
- IssueCertificate
- GetCertificate

- ImportCertificateAuthorityCertificate
- GetCertificateAuthorityCertificate

문제가 발생하면 문제 해결 섹션AWS Private CA Matter 준수 인증서 오류 문제 해결의 섹션을 참조하세요.

```
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);
    }
}
```

노드 운영 인증서(NOC) 생성

이 Java 샘플은 BlankEndEntityCertificate_CriticalBasicConstraints_APIPassthrough/V1 템플릿을 사용하여 Matter 노드 운영 인증서를 생성하는 방법을 보여줍니다. Base64로 인코딩된 KeyUsage 값을 생성하여 CustomExtension.을 통해 전달해야 합니다.

이 예제에서는 다음 AWS Private CA API 작업을 호출합니다.

- IssueCertificate

문제가 발생하는 경우 문제 해결 섹션의 AWS Private CA Matter 준수 인증서 오류 문제 해결을 참조하세요.

```
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 를 사용하여 mDL 인증서 구현

AWS Private Certificate Authority API를 사용하여 모바일 운전면허증(mDL)에 대한 ISO/IEC 표준을 준수하는 인증서를 생성할 수 있습니다. 이 표준은 인증서 구성을 포함하여 모바일 디바이스와 관련하여 운전면허증을 구현하기 위한 인터페이스 사양을 설정합니다.

주제

- 발급 기관 인증 기관(IACA) 인증서 활성화
- 문서 서명자 인증서 생성

발급 기관 인증 기관(IACA) 인증서 활성화

이 Java 샘플은 BlankRootCACertificate_PathLen0_APIPassthrough/V1 정의 템플릿을 사용하여 ISO/IEC mDL 표준 준수 발급 기관 인증 기관(IACA) 인증서를 생성하고 설치하는 방법을 보여줍니다. `KeyUsage`, `IssuerAlternativeName` `CRLDistributionPoint`및에 대해 base64로 인코딩된 값을 생성하고를 통해 전달해야 합니다`CustomExtensions`.

> **Note**
>
> IACA 링크 인증서는 이전 IACA 루트 인증서에서 새 IACA 루트 인증서로의 신뢰 경로를 설정합니다. 발급 기관은 IACA 재키 프로세스 중에 IACA 링크 인증서를 생성하고 배포할 수 있습니다. `pathLen=0` 세트가 있는 IACA 루트 인증서를 사용하면 IACA 링크 인증서를 발급할 수 없습니다.

이 예제에서는 다음 AWS Private CA API 작업을 호출합니다.

- 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);
    }
}
```

문서 서명자 인증서 생성

이 Java 샘플은 BlankEndEntityCertificate_APIPassthrough/V1 템플릿을 사용하여 ISO/IEC mDL 표준 준수 문서 서명자 인증서를 생성하는 방법을 보여줍니다. KeyUsage, IssuerAlternativeName

CRLDistributionPoint 및에 대해 base64로 인코딩된 값을 생성하고를 통해 전달해야 합니다CustomExtensions.

이 예제에서는 다음 AWS Private CA API 작업을 호출합니다.

- 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);
    }
}
```

용 솔루션 설계 AWS Private CA

AWS Private CA 는 루트 인증 기관(CA)에서 하위 CAs를 거쳐 최종 엔터티 인증서로 확장되는 조직의 프라이빗 PKI(퍼블릭 키 인프라)에 대한 완전한 클라우드 기반 제어를 제공합니다. 안전하고 유지 관리 및 확장이 가능하며 조직의 요구 사항에 적합한 PKI를 위해서는 철저한 계획이 필수적입니다. 이 단원에서는 CA 계층 구조 설계, 사설 CA 및 사설 최종 엔터티 인증서 수명 주기의 관리, 보안을 위한 모범 사례 적용에 대한 지침을 제공합니다.

이 섹션에서는 사설 인증 기관(CA)을 생성하기 전에 AWS Private CA 사용을 준비하는 방법을 설명합니다. 또한 온라인 인증서 상태 프로토콜 (OCSP) 또는 인증서 해지 목록(CRL)을 통해 해지 지원을 추가하는 옵션에 대해서도 설명합니다.

또한 조직이가 아닌 온프레미스에서 프라이빗 루트 CA 자격 증명을 호스팅할지 여부를 결정해야 합니다 AWS. 이 경우 사용하기 전에 자체 관리형 프라이빗 PKI를 설정하고 보호해야 합니다 AWS Private CA. 이 시나리오에서는 외부의 상위 CA가 AWS Private CA 지원하는에 하위 CA를 생성합니다 AWS Private CA. 자세한 내용은 외부 상위 CA에서 서명한 하위 CA 인증서 설치를 참조하세요.

주제

- CA 계층 구조 설계
- 프라이빗 CA 수명 주기 관리
- AWS Private CA 인증서 해지 방법 계획
- AWS Private CA CA 모드 이해
- 의 복원력 계획 AWS Private CA

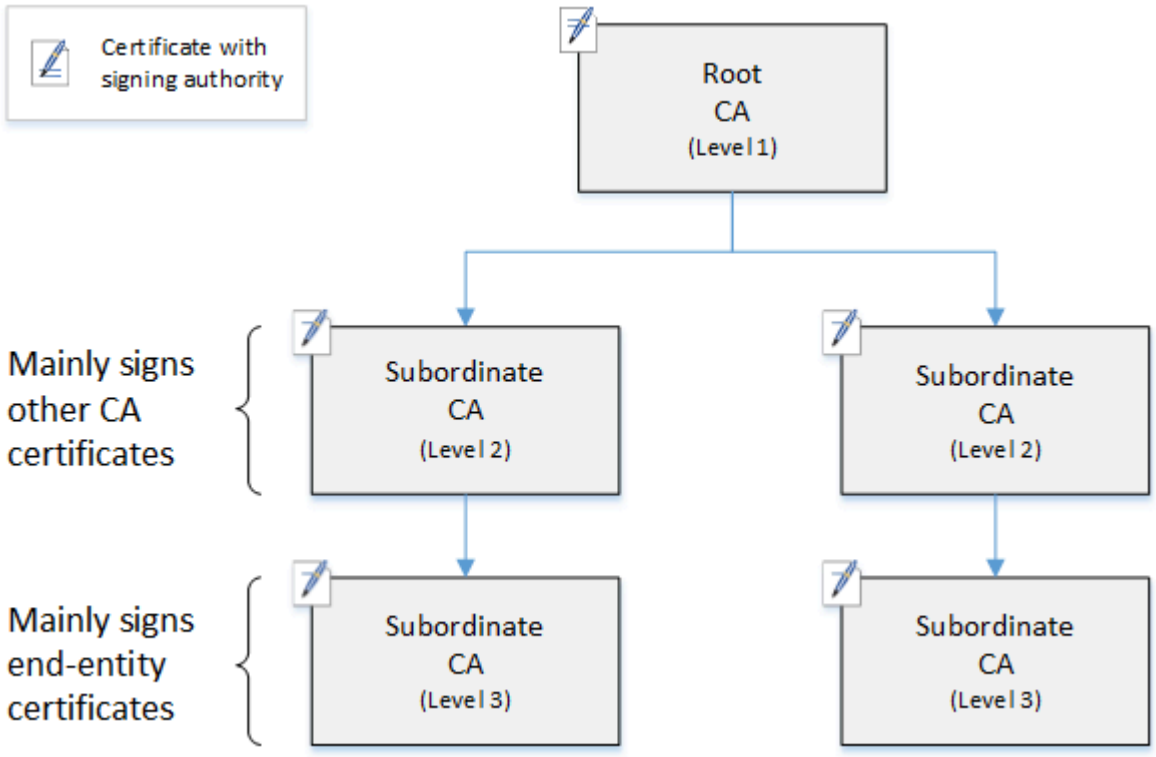
CA 계층 구조 설계

를 사용하면 최대 5개 수준의 인증 기관 계층을 생성할 AWS Private CA수 있습니다. 계층 트리 맨 위에 있는 루트 CA에는 원하는 수만큼 분기를 가질 수 있습니다. 루트 CA는 각 분기에 최대 4개의 하위 CA를 가질 수 있습니다. 또한 각각이 자체 루트를 가진 계층을 여러 개 만들 수 있습니다.

잘 설계된 CA 계층 구조는 다음과 같은 이점을 제공합니다.

- 각 CA에 적합한 세분화된 보안 제어
- 로드 밸런싱 및 보안 향상을 위한 관리 작업 분할
- 일상적 운영에 대한 신뢰가 제한적이고 해지 가능한 CA 사용
- 유효 기간 및 인증서 경로 제한

다음 다이어그램에서는 간단한 3단계 CA 계층 구조를 보여 줍니다.



트리의 각 CA에는 서명 권한이 있는 X.509 v3 인증서(펜 및 종이 아이콘으로 표시)가 지원됩니다. 이는 곧 CA가 해당 CA에 종속된 다른 인증서에 서명할 수 있다는 의미입니다. CA가 하위 레벨의 CA 인증서에 서명하면 서명된 인증서에 제한적이고 해지 가능한 권한이 부여됩니다. 레벨 1의 루트 CA는 레벨 2의 상위 레벨 하위 CA 인증서에 서명합니다. 이러한 CA는 최종 엔터티 인증서를 관리하는 퍼블릭 키 인프라(PKI) 관리자가 사용하는 레벨 3의 CA에 대한 인증서에 서명합니다.

CA 계층 구조의 보안은 트리 최상단에 가장 강력하게 구성되어 있어야 합니다. 이러한 구조는 루트 CA 인증서와 해당 프라이빗 키를 보호합니다. 루트 CA는 모든 하위 CA와 그 아래의 최종 엔터티 인증서에 대한 신뢰를 기반으로 합니다. 최종 엔터티 인증서의 손상으로 인해 지역화된 손상이 발생할 수 있지만, 루트가 손상되면 전체 PKI의 신뢰가 파괴됩니다. 루트 및 상위 레벨의 하위 CA는 자주 사용되지 않습니다(일반적으로 다른 CA 인증서에 서명하는 데 사용). 따라서 손상 위험을 낮출 수 있도록 인증서에 대해 엄격한 제어 및 감사가 이루어집니다. 계층 구조의 하위 레벨에서는 보안이 덜 제한적입니다. 이 접근 방법을 사용하면 사용자, 컴퓨터 호스트 및 소프트웨어 서비스에 대한 최종 엔터티 인증서를 발급하고 해지하는 일상적인 관리 작업을 수행할 수 있습니다.

> **Note**
>
> 루트 CA를 사용하여 하위 인증서에 서명하는 것은 몇 가지 상황에서만 발생하는 드문 경우입니다.

- PKI가 생성되는 경우
- 상위 레벨의 인증 기관을 교체해야 하는 경우
- 인증서 해지 목록(CRL) 또는 온라인 인증서 상태 프로토콜(OCSP) 응답자를 구성해야 하는 경우

루트 및 기타 상위 레벨 CA에는 매우 안전한 운영 프로세스와 액세스 제어 프로토콜이 필요합니다.

주제

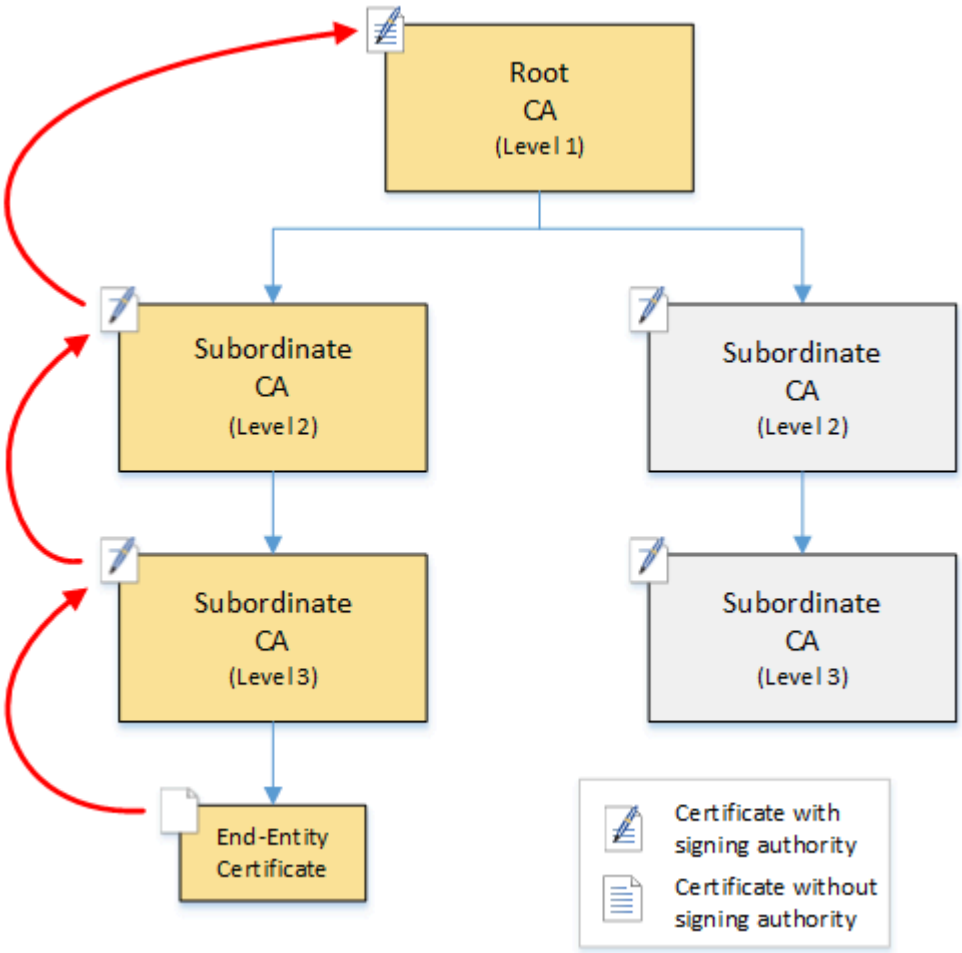
- 최종 엔터티 인증서 검증
- CA 계층 구조의 계획
- 인증 경로에서 길이 제약 조건 설정

최종 엔터티 인증서 검증

최종 엔터티 인증서는 하위 CA를 통해 루트 CA로 되돌아가는 인증 경로에서 신뢰를 파생시킵니다. 웹 브라우저 또는 다른 클라이언트는 최종 엔터티 인증서가 제공될 때 신뢰 체인을 구성하려고 시도합니다. 예를 들어, 인증서의 발급자 고유 이름 및 보안 주체 고유 이름이 발급 CA 인증서의 해당 필드와 일치하는지 확인할 수 있습니다. 클라이언트가 신뢰 저장소에 포함된 신뢰할 수 있는 루트에 도달할 때까지 계층 구조의 각 연속 레벨에서 일치 작업이 계속됩니다.

신뢰 저장소는 브라우저 또는 운영 체제에 포함된 신뢰할 수 있는 CA의 라이브러리입니다. 프라이빗 PKI의 경우, 조직의 IT 팀은 각 브라우저 또는 시스템이 이전에 사설 루트 CA를 신뢰 저장소에 추가했는지 확인해야 합니다. 그렇지 않으면 인증 경로의 유효성을 검사할 수 없으므로 클라이언트 오류가 발생합니다.

다음 그림은 최종 엔터티 X.509 인증서와 함께 표시될 때 브라우저가 따르는 유효성 검사 경로를 보여줍니다. 최종 엔터티 인증서에는 서명 권한이 없으며, 이러한 권한을 소유한 엔터티를 인증하는 역할만 합니다.



브라우저는 최종 엔터티 인증서를 검사합니다. 브라우저는 인증서가 하위 CA(레벨 3)의 서명을 신뢰 자격 증명으로 제공한다는 것을 확인합니다. 하위 CA의 인증서는 동일한 PEM 파일에 포함되어야 합니다. 또는 신뢰 체인을 구성하는 인증서가 포함된 별도의 파일에 있을 수도 있습니다. 이를 찾으면 브라우저는 하위 CA(레벨 3)의 인증서를 확인하고 하위 CA(레벨 2)의 서명을 제공하는지 확인합니다. 그러면 하위 CA(레벨 2)가 루트 CA(레벨 1)의 서명을 신뢰 자격 증명으로 제공합니다. 브라우저는 신뢰 저장소에 사전 설치된 사설 루트 CA 인증서의 복사본을 찾으면 최종 엔터티 인증서가 신뢰할 수 있는지 유효성 검사를 합니다.

일반적으로 브라우저는 인증서 해지 목록(CRL)과 비교하여 각 인증서를 확인합니다. 만료, 해지 또는 잘못 구성된 인증서는 거부되고 유효성 검사가 실패합니다.

CA 계층 구조의 계획

일반적으로 CA 계층 구조에는 조직의 구조가 반영되어야 합니다. 관리 및 보안 역할을 위임하는 데 필요한 레벨 이상의 경로 길이(즉, CA 레벨 수)를 목표로 합니다. 계층에 CA를 추가하면 인증 경로에서 인증서 수가 늘어나므로 유효성 검사 시간이 길어집니다. 경로 길이를 최소로 유지하면 최종 엔터티 인증서를 검증할 때 서버에서 클라이언트로 보내는 인증서 수도 줄어듭니다.

이론적으로 pathLenConstraint 파라미터가 없는 루트 CA는 하위 CAs. 하위 CA는 내부 구성에서 허용하는 수만큼 하위 하위 CAs를 가질 수 있습니다. AWS Private CA 관리형 계층 구조는 최대 5개 수준 깊이의 CA 인증 경로를 지원합니다.

잘 설계된 CA 구조에는 몇 가지 이점이 있습니다.

- 서로 다른 조직 단위에 대한 별도의 관리 제어
- 하위 CA에 대한 액세스 권한을 위임하는 기능
- 추가 보안 제어를 통해 상위 레벨의 CA를 보호하는 계층 구조

두 가지 공통 CA 구조가 이 모든 것을 수행합니다.

- 2개의 CA 레벨: 루트 CA 및 하위 CA

  이것은 루트 CA 및 하위 CA에 대해 별도의 관리, 제어 및 보안 정책을 허용하는 가장 간단한 CA 구조입니다. 하위 CA에 대해서는 보다 허용적으로 액세스를 허락하면서 루트 CA에 대해서는 제한적인 제어 및 정책을 유지할 수 있습니다. 후자는 최종 엔터티 인증서의 대량 발급에 사용됩니다.

- 3개의 CA 레벨: 루트 CA 및 두 계층의 하위 CA

  위와 유사하게 이러한 구조는 루트 CA를 하위 레벨의 CA 작업과 분리하기 위해 CA 계층을 추가합니다. 중간 CA 계층은 최종 엔터티 인증서의 발급을 수행하는 하위 CA에 서명하는 데만 사용됩니다.

덜 일반적인 CA 구조는 다음과 같습니다.

- 4개 이상의 CA 레벨

  레벨이 3개인 계층 구조보다 덜 일반적이지만 레벨이 4개 이상인 CA 계층 구조가 가능하며, 이러한 구조는 관리 위임을 허용하는 데 필요할 수 있습니다.
- 1개의 CA 레벨: 루트 CA만

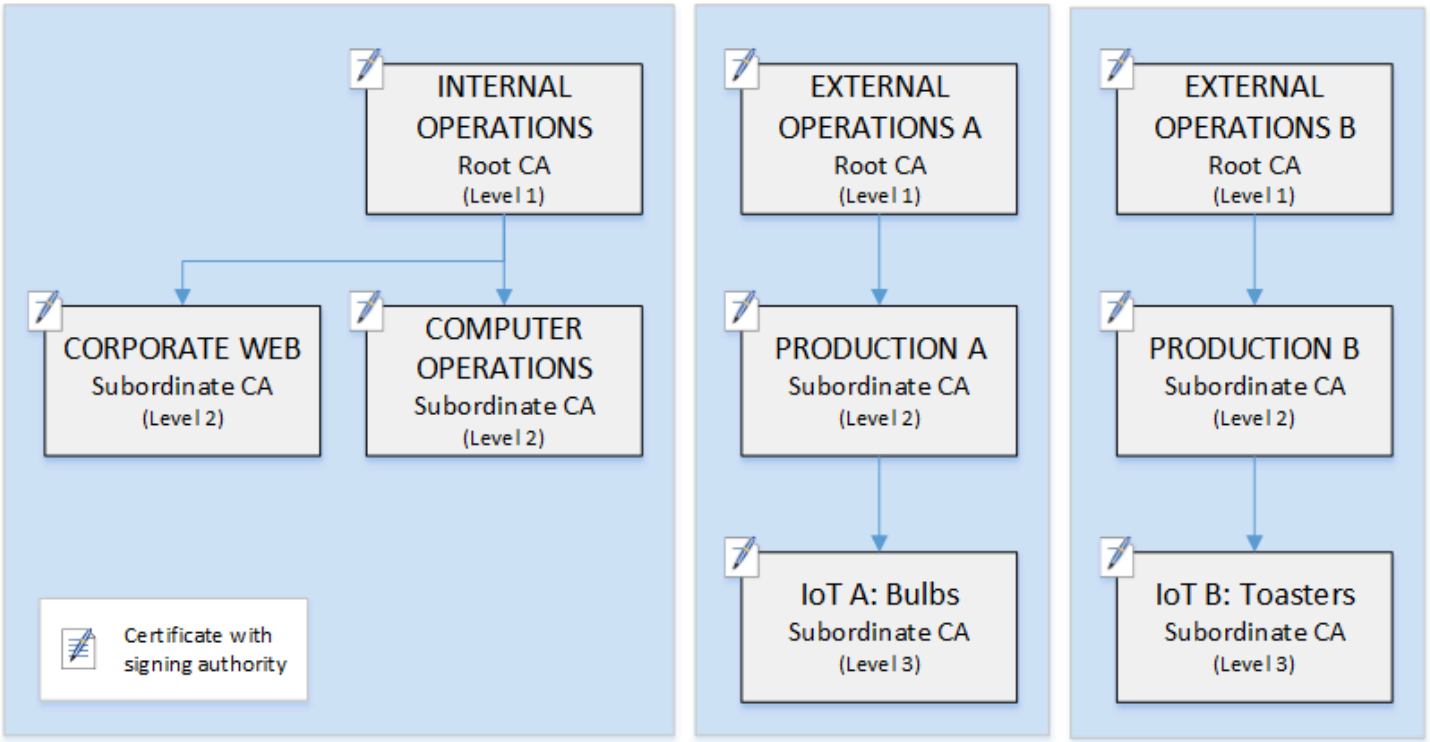
  이 구조는 보통 전체 신뢰 체인이 필요하지 않을 때 개발 및 테스트에 사용됩니다. 프로덕션 환경에서 사용되며 비정형적입니다. 또한 루트 CA와 최종 엔터티 인증서를 발급하는 CA에 대해 별도의 보안 정책을 유지 관리하는 모범 사례를 위반합니다.

  그러나 이미 루트 CA에서 직접 인증서를 발급하고 있는 경우 로 마이그레이션할 수 있습니다 AWS Private CA. 이렇게 하면 OpenSSL 또는 기타 소프트웨어로 관리되는 루트 CA를 사용하는 것보다 보안 및 제어 측면에서 이점이 있습니다.

제조업체용 프라이빗 PKI의 예

이 예에서는 어떤 가상의 기술 업체가 스마트 전구와 스마트 토스터라는 2개의 사물 인터넷(IoT) 제품을 제조합니다. 생산 중에 각 디바이스에 최종 엔터티 인증서가 발급되므로 인터넷을 통해 제조업체와 안전하게 통신할 수 있습니다. 또한 회사의 PKI는 내부 웹 사이트 및 금융 및 비즈니스 업무를 실행하는 다양한 자체 호스팅 컴퓨터 서비스를 포함해 컴퓨터 인프라를 보호합니다.

따라서 CA 계층 구조는 이러한 비즈니스의 관리 및 운영 측면을 면밀하게 모델링합니다.



이 계층 구조에는 내부 작업용 루트 1개와 외부 작업용 루트 2개(각 제품 라인마다 1개의 루트 CA) 등 3개의 루트가 포함되어 있습니다. 또한 내부 운영을 위한 CA 레벨 2개와 외부 운영을 위한 레벨 3개 등 여러 인증 경로 길이를 보여 줍니다.

분리된 루트 CA를 사용하고 외부 운영 측면에서 하위 CA 레이어를 추가하는 것은 비즈니스 및 보안 요구 사항을 충족하는 설계입니다. CA 트리가 여러 개 있는 PKI는 기업 조직 개편, 매각 또는 인수에 대해 미래에 대비할 수 있습니다. 변경 사항이 발생할 때 보호 부문에서 전체 루트 CA 계층을 완전히 이동시킬 수 있습니다. 그리고 하위 CA의 레벨이 2개이기 때문에 수 천 또는 수 백만 개의 제조 항목에 대한 인증서를 대량 서명하는 역할을 하는 레벨 3 CA로부터 루트 CA를 철저하게 분리할 수 있습니다.

내부 측면에서 기업 웹 및 내부 컴퓨터 작업은 레벨이 2개인 계층 구조를 완성합니다. 이러한 레벨들 덕분에 웹 관리자와 운영 엔지니어는 자체 작업 도메인에 대해 독립적으로 인증서 발급을 관리할 수 있습니다. PKI를 별도의 기능 도메인으로 구분하는 것이 보안 상 가장 바람직하며, 이렇게 하면 상호 영향

을 미칠 수 있는 손상으로부터 각자를 보호할 수 있습니다. 웹 관리자는 회사 전체에 웹 브라우저에서 사용할 최종 엔터티 인증서를 발급하여 내부 웹 사이트에서 통신을 인증하고 암호화합니다. 운영 엔지니어는 데이터 센터 호스트와 컴퓨터 서비스를 상호 인증하는 최종 엔터티 인증서를 발급합니다. 이 시스템은 중요한 데이터를 LAN에서 암호화하여 안전하게 유지합니다.

인증 경로에서 길이 제약 조건 설정

CA 계층의 구조는 각 인증서에 포함된 기본 제약 조건 확장에 의해 정의되고 적용됩니다. 확장은 두 가지 제약 조건을 정의합니다.

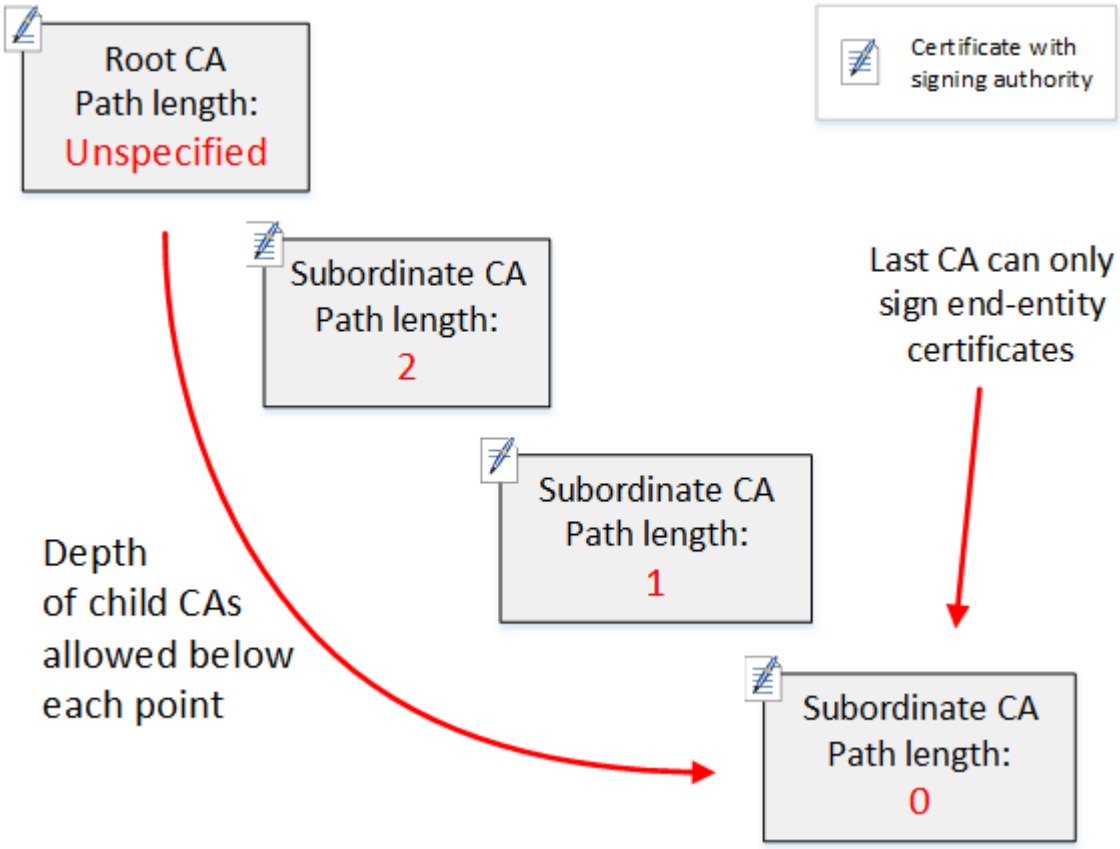
- `cA` - 인증서가 CA를 정의하는지 여부. 이 값이 false(기본값)이면 인증서는 최종 엔터티 인증서입니다.
- `pathLenConstraint` - 유효한 신뢰 체인에 존재할 수 있는 하위 수준 하위 CA의 최대 수. 최종 엔터티 인증서는 CA 인증서가 아니므로 계산에 포함되지 않습니다.

루트 CA 인증서에는 최대한의 유연성이 요구되며, 경로 길이 제약 조건이 포함되지 않습니다. 이렇게 하면 루트가 모든 길이의 인증 경로를 정의할 수 있습니다.

> **Note**
>
> AWS Private CA 는 인증 경로를 5개 수준으로 제한합니다.

하위 CA는 계층 구조 배치에서 위치와 원하는 기능에 따라 0 이상의 `pathLenConstraint` 값을 갖습니다. 예를 들어 CA가 3개인 계층 구조에서는 루트 CA에 대해 경로 제약 조건이 지정되지 않습니다. 첫 번째 하위 CA는 경로 길이가 1이므로 하위 CA에 서명할 수 있습니다. 이러한 각 하위 CA의 `pathLenConstraint` 값은 반드시 0이어야 합니다. 이는 최종 엔터티 인증서에 서명은 할 수 있지만 추가 CA 인증서를 발급할 수는 없다는 의미입니다. 새 CA를 생성할 수 있는 기능을 제한하는 것은 중요한 보안 제어책입니다.

다음 그림은 이렇게 제한된 권한이 계층 구조 아래로 전파되는 것을 보여줍니다.



이렇듯 레벨이 4개인 계층 구조에서는 루트가 제약되지 않습니다(항상 그렇듯이). 그러나 첫 번째 하위 CA의 `pathLenConstraint` 값은 2입니다. 이 값은 하위 CA의 깊이가 2개 레벨 이상이 되지 못하도록 제한합니다. 따라서 유효한 인증 경로의 경우, 제약 조건 값이 다음 두 레벨에서 0으로 감소해야 합니다. 웹 브라우저가 이렇듯 경로 길이가 4보다 큰 분기에서 최종 엔터티 인증서를 발견하면 유효성 검사가 실패합니다. 실수로 생성된 CA, 잘못 구성된 CA 또는 무단 발급의 결과로 이러한 인증서가 생성될 수 있습니다.

템플릿을 사용하여 경로 길이 관리

AWS Private CA 는 루트, 하위 및 최종 엔터티 인증서를 발급하기 위한 템플릿을 제공합니다. 이러한 템플릿은 경로 길이를 포함하여 기본 제약 조건 값에 대한 모범 사례를 캡슐화합니다. 템플릿에는 다음이 포함됩니다.

- RootCACertificate/V1
- SubordinateCACertificate_PathLen0/V1
- SubordinateCACertificate_PathLen1/V1
- SubordinateCACertificate_PathLen2/V1
- SubordinateCACertificate_PathLen3/V1

- EndEntityCertificate/V1

경로 길이를 발급 CA 인증서의 경로 길이보다 크거나 같게 해서 CA를 생성하려고 하면 `IssueCertificate` API가 오류를 반환합니다.

인증서 템플릿에 대한 자세한 내용은 AWS Private CA 인증서 템플릿 사용 단원을 참조하십시오.

를 사용하여 CA 계층 구조 설정 자동화 AWS CloudFormation

CA 계층 구조에 대한 설계를 결정했으면 AWS CloudFormation 템플릿을 사용하여 이를 테스트하고 프로덕션 환경에 배치할 수 있습니다. 이러한 템플릿의 예제는 CloudFormation 사용 설명서의 프라이빗 CA 계층 선언을 참조하세요.

프라이빗 CA 수명 주기 관리

CA 인증서는 수명(유효 기간)이 고정되어 있습니다. CA 인증서가 만료되면 CA 계층 구조에서 아래에 있는 하위 CA에서 직접 또는 간접적으로 발급한 모든 인증서가 유효하지 않게 됩니다. 미리 계획하면 CA 인증서 만료를 방지할 수 있습니다.

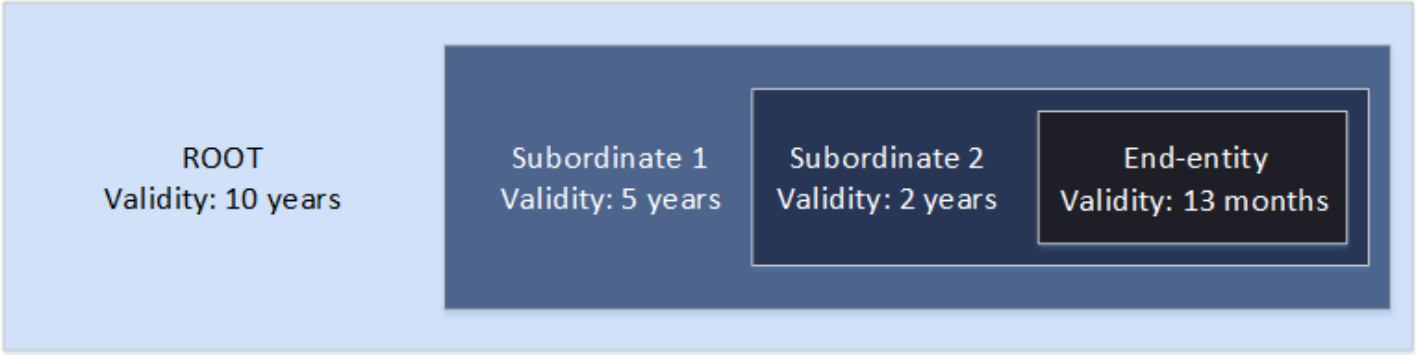
유효 기간 선택

X.509 인증서의 유효 기간은 필수적인 기본 인증서 필드입니다. 발급 CA가 인증서를 신뢰할 수 있음을 인증하는 기간 동안 해지를 금지하는 시간 범위를 결정합니다 (자체 서명된 루트 인증서는 자체 유효 기간을 인증).

AWS Private CA 및는 다음 제약 조건이 적용되는 인증서 유효 기간 구성을 AWS Certificate Manager 지원합니다.

- 에서 관리하는 인증서는 발급한 CA의 유효 기간보다 짧거나 같아 AWS Private CA 야 합니다. 즉, 하위 CA 및 최종 엔터티 인증서의 유효 기간이 상위 인증서보다 길 수 없습니다. `IssueCertificate` API를 사용하여 유효 기간이 상위 CA보다 크거나 같은 CA 인증서를 발급하려고 하는 시도는 실패합니다.
- AWS Certificate Manager (ACM이 프라이빗 키를 생성하는)에서 발급하고 관리하는 인증서의 유효 기간은 13개월(395일)입니다. ACM은 이러한 인증서의 갱신 프로세스를 관리합니다. AWS Private CA 를 사용하여 인증서를 직접 발급하는 경우 유효 기간을 선택할 수 있습니다.

다음 다이어그램은 중첩된 유효 기간의 일반적인 구성을 보여 줍니다. 루트 인증서는 유효 기간이 가장 길고 최종 엔터티 인증서는 유효 기간이 비교적 짧으며, 하위 CA는 중간 정도입니다.



CA 계층 구조를 계획할 때 CA 인증서의 최적 수명을 결정하십시오. 발급하려는 최종 엔터티 인증서의 원하는 수명에서 거슬러 내려가십시오.

최종 엔터티 인증서

최종 엔터티 인증서에는 사용 사례에 적합한 유효 기간이 있어야 합니다. 수명이 짧으면 프라이빗 키를 분실하거나 도난 당한 경우에 인증서 노출을 최소화할 수 있습니다. 그러나 수명이 짧으면 자주 갱신을 해야 합니다. 만료되는 인증서를 갱신하지 않으면 가동 중지가 발생할 수 있습니다.

최종 엔터티 인증서를 분산 사용하면 보안 위반이 발생할 경우 위치적으로 문제가 발생할 수 있습니다. 계획을 수립할 때는 갱신 및 배포 인증서, 손상된 인증서의 해지, 인증서를 사용하는 클라이언트에 해지 내용이 전파되는 속도를 고려해야 합니다.

ACM을 통해 발급된 최종 엔터티 인증서의 기본 유효 기간은 13개월(395일)입니다. 에서 `IssueCertificate` API를 사용하여 발급 CA보다 작은 유효 기간을 적용할 AWS Private CA수 있습니다.

하위 CA 인증서

하위 CA 인증서는 발급되는 인증서보다 유효 기간이 훨씬 길어야 합니다. CA 인증서의 유효 기간은 발급되는 하위 CA 인증서 또는 최종 엔터티 인증서의 유효 기간보다 2~5배 긴 것이 좋습니다. 예를 들어, 레벨이 2개인 CA 계층 구조(루트 CA 및 1개의 하위 CA)가 있다고 가정합니다. 유효 기간이 1년인 최종 엔터티 인증서를 발급하려는 경우, 하위 발급 CA의 유효 기간을 3년으로 구성할 수 있습니다. 이 기간은의 하위 CA 인증서에 대한 기본 유효 기간입니다 AWS Private CA. 루트 CA 인증서를 교체하지 않고 하위 CA 인증서를 변경할 수 있습니다.

루트 인증서

루트 CA 인증서를 변경하면 전체 PKI에 영향을 미치며, 모든 종속 클라이언트 운영 체제 및 브라우저 신뢰 저장소를 업데이트해야 합니다. 운영에 미치는 영향을 최소화하려면 루트 인증서에 대해 긴 유효 기간을 선택해야 합니다. 루트 인증서의 AWS Private CA 기본값은 10년입니다.

CA 승계 관리

이전 CA를 교체하거나 새 유효 기간으로 CA를 다시 발급하는 등 두 가지 방법으로 CA 승계를 관리할 수 있습니다.

이전 CA 교체

이전 CA를 교체하려면 새 CA를 생성하고 이를 동일한 상위 CA에 연결합니다. 그런 다음 새 CA에서 인증서를 발급합니다.

새 CA에서 발급된 인증서에는 새 CA 체인이 있습니다. 새 CA가 설정되면 이전 CA를 비활성화하여 새 인증서를 발급하지 못하도록 할 수 있습니다. 비활성화된 된 이전 CA는 CA에서 발급된 이전 인증서에 대한 해지를 지원하며, 그렇게 구성한 경우 OCSP 및/또는 인증서 해지 목록(CRL)을 통해 계속해서 인증서의 유효성을 검사합니다. 이전 CA에서 발급한 마지막 인증서가 만료되면 이전 CA를 삭제할 수 있습니다. CA에서 발급된 모든 인증서에 대한 감사 보고서를 생성하여 발급된 모든 인증서가 만료되었는지 확인할 수 있습니다. 이전 CA에 하위 CA가 있는 경우에는 하위 CA가 동시에 만료되거나 상위 CA보다 앞서 만료되기 때문에 CA를 교체해야 합니다. 교체가 필요한 계층 구조에서 최상위 CA를 교체하는 것부터 시작합니다. 그런 다음 각각의 후속 하위 레벨에서 새로운 대체 하위 CA를 생성합니다.

AWS 에서는 필요에 따라 CA 이름에 CAs 생성 식별자를 포함할 것을 권장합니다. 예를 들어, 첫 번째로 생성된 CA의 이름을 “회사 루트 CA”로 지정한다고 가정해 보겠습니다. 두 번째로 생성된 CA의 이름은 “회사 루트 CA G2”로 지정합니다. 이렇게 간단한 명명 규칙만으로도 두 CA가 모두 만료되지 않을 때의 혼란을 방지할 수 있습니다.

이러한 CA 승계 방법은 CA의 프라이빗 키를 교대로 사용한다는 점에서 선호됩니다. CA 키의 경우 프라이빗 키를 교대로 사용하는 것이 가장 좋습니다. 교대 빈도는 키 사용 빈도에 비례해야 합니다. 즉, 더 많은 인증서를 발급하는 CA는 더 자주 교대해야 합니다.

> **Note**
>
> CA를 교체한 경우에는 ACM을 통해 발급된 프라이빗 인증서를 갱신할 수 없습니다. 발급 및 갱신에 ACM을 사용하는 경우 CA 인증서를 다시 발급하여 CA의 유효 기간을 연장해야 합니다.

이전 CA 다시 발급

CA 만료가 가까워지면 수명을 연장하는 또 다른 방법은 CA 인증서를 새 만료 날짜로 재발급하는 것입니다. 재발급은 모든 CA 메타데이터를 그대로 유지하고 기존 프라이빗 및 퍼블릭 키를 보존합니다. 이 시나리오에서 CA에서 발급한 기존 인증서 체인과 만료되지 않은 최종 엔터티 인증서는 만료될 때까지

유효합니다. 또한 새 인증서 발급을 중단 없이 계속할 수 있습니다. 재발급한 인증서로 CA를 업데이트하려면 CA 인증서 설치에 설명된 일반적인 설치 절차를 따릅니다.

> **Note**
>
> 새 키 페어로 교체하면 보안상의 이점을 얻을 수 있으므로 인증서를 다시 발급하는 대신 만료되는 CA를 교체하는 것이 좋습니다.

CA 취소

기본 인증서를 취소하여 CA를 해지할 수 있습니다. 또한 CA에서 발급한 모든 인증서도 사실상 취소됩니다. 해지 정보는 OCSP 또는 CRL을 통해 클라이언트에 배포됩니다. 발급된 모든 최종 엔터티 및 하위 CA 인증서를 해지하려는 경우에만 CA 인증서를 해지해야 합니다.

AWS Private CA 인증서 해지 방법 계획

를 사용하여 프라이빗 PKI를 계획 AWS Private CA할 때는 엔드포인트의 프라이빗 키가 노출되는 경우와 같이 엔드포인트가 더 이상 발급된 인증서를 신뢰하지 않도록 하는 상황을 처리하는 방법을 고려해야 합니다. 이 문제에 대한 일반적인 접근 방식은 단기 인증서를 사용하거나 인증서 취소를 구성하는 것입니다. 단기 인증서는 몇 시간 또는 며칠 단위로 만료되므로 해지하는 것은 의미가 없으며, 엔드포인트에 해지를 알리는 데 걸리는 시간과 거의 같은 시간이 지나면 인증서가 유효하지 않게 됩니다. 이 섹션에서는 구성 및 모범 사례를 포함하여 AWS Private CA 고객을 위한 해지 옵션에 대해 설명합니다.

해지 방법을 찾는 고객은 온라인 인증서 상태 프로토콜(OCSP), 인증서 취소 목록(CRL) 또는 둘 다를 선택할 수 있습니다.

> **Note**
>
> 해지를 구성하지 않고 CA를 생성하는 경우 언제든지 나중에 구성할 수 있습니다. 자세한 내용은 에서 프라이빗 CA 업데이트 AWS Private Certificate Authority 단원을 참조하십시오.

- 온라인 인증서 상태 프로토콜(OCSP)

  AWS Private CA 는 고객이 인프라를 직접 운영할 필요 없이 인증서가 취소되었음을 엔드포인트에 알리는 완전 관리형 OCSP 솔루션을 제공합니다. 고객은 AWS Private CA 콘솔, API, CLI 또는를 통해 단일 작업으로 신규 또는 기존 CAs에서 OCSP를 활성화할 수 있습니다 CloudFormation. CRL은

엔드포인트에서 저장 및 처리되며 무효화될 수 있지만 OCSP 스토리지 및 처리 요구 사항은 응답자 백엔드에서 동기적으로 처리됩니다.

CA에 대해 OCSP를 활성화하면는 발급된 각 새 인증서의 권한 정보 액세스(AIA) 확장에 OCSP 응답기의 URL을 AWS Private CA 포함합니다. 확장을 사용하면 웹 브라우저와 같은 클라이언트가 응답자를 쿼리하고 최종 엔터티 또는 하위 CA 인증서를 신뢰할 수 있는지 여부를 확인할 수 있습니다. 응답자는 신뢰성을 보장하기 위해 암호로 서명된 상태 메시지를 반환합니다.

AWS Private CA OCSP 응답기는 RFC 5019를 준수합니다.

OCSP 고려 사항

- OCSP 상태 메시지는 발급 CA가 사용하도록 구성된 것과 동일한 서명 알고리즘을 사용하여 서명됩니다. AWS Private CA 콘솔에서 만든 CA는 기본적으로 SHA256WITHRSA 서명 알고리즘을 사용합니다. 지원되는 다른 알고리즘은 인증 기관 구성 API 설명서에서 찾을 수 있습니다.
- OCSP 응답자가 활성화된 경우 APIPassthrough 및 CSRPassthrough 인증서 템플릿은 AIA 확장과 함께 작동하지 않습니다.
- 관리형 OCSP 서비스의 엔드포인트는 공용 인터넷에서 액세스할 수 있습니다. OCSP를 원하지만 퍼블릭 엔드포인트는 원하지 않는 고객은 자체 OCSP 인프라를 운영해야 합니다.

- 인증서 해지 목록(CRL)

인증서 해지 목록(CRL)은 예정된 만료 날짜 이전에 취소된 인증서 목록이 포함된 파일입니다. CRL에는 더 이상 신뢰할 수 없어야 하는 인증서 목록, 해지 이유 및 기타 관련 정보가 포함되어 있습니다.

인증 기관(CA)을 구성할 때가 전체 또는 분할된 CRL을 AWS Private CA 생성할지 여부를 선택할 수 있습니다. 선택한 항목에 따라 인증 기관에서 발급하고 취소할 수 있는 최대 인증서 수가 결정됩니다. 자세한 내용은 AWS Private CA 할당량을 참조하세요.

CRL 고려 사항

- 메모리 및 대역폭 고려 사항: 로컬 다운로드 및 처리 요구 사항으로 인해 CRLs OCSP보다 더 많은 메모리가 필요합니다. 그러나 CRLs 연결당 상태를 확인하는 대신 취소 목록을 캐싱하여 OCSP에 비해 네트워크 대역폭을 줄일 수 있습니다. 특정 IoT 디바이스와 같이 메모리가 제한된 디바이스의 경우 분할된 CRLs을 사용하는 것이 좋습니다.
- CRL 유형 변경: 완료에서 분할된 CRL로 변경할 때 필요에 따라 새 파티션을 AWS Private CA 생성하고 원본을 포함한 모든 CRLs에 IDP 확장을 추가합니다. 파티셔닝됨에서 완료로 변경하면 단일 CRL만 업데이트되고 향후 이전 파티션과 연결된 인증서가 취소되는 것을 방지할 수 있습니다.

> **Note**
>
> OCSP와 CRL 모두 취소와 상태 변경 가능 시점 사이에 약간의 지연이 있습니다.
>
> - 인증서를 해지할 때 OCSP 응답에 새 상태가 반영되기까지 최대 60분이 걸릴 수 있습니다. 일반적으로 OCSP는 해지 정보의 빠른 배포를 지원하는 경향이 있는데, 이는 클라이언트가 며칠 동안 캐시할 수 있는 CRL과 달리 OCSP 응답은 일반적으로 클라이언트에 의해 캐시되지 않기 때문입니다.
> - CRL은 일반적으로 인증서가 해지된 후 약 30분 후에 업데이트됩니다. 어떤 이유로든 CRL 업데이트가 실패하면 AWS Private CA 는 15분마다 추가로 시도합니다.

해지 구성에 대한 일반 요구 사항

모든 해지 구성에는 다음 요구 사항이 적용됩니다.

- CRL 또는 OCSP를 비활성화하는 구성은 `Enabled=False` 파라미터만 포함해야 하며 `CustomCname` 또는 `ExpirationInDays` 등의 다른 파라미터가 포함된 경우 실패합니다.
- CRL 구성에서 `S3BucketName` 파라미터는 Amazon Simple Storage Service 버킷 명명 규칙을 준수해야 합니다.
- CRL 또는 OCSP에 대한 사용자 지정 표준 이름(CNAME) 파라미터가 포함된 구성은 CNAME에 특수 문자 사용에 대한 RFC7230 제한을 준수해야 합니다.
- CRL 또는 OCSP 구성에서 CNAME 파라미터 값에 'http://' 또는 'https://' 등의 프로토콜 접두사가 포함되지 않아야 합니다.

주제

- 에 대한 CRL 설정 AWS Private CA
- 에 대한 OCSP URL 사용자 지정 AWS Private CA

에 대한 CRL 설정 AWS Private CA

CA 생성 프로세스의 일부로 인증서 해지 목록(CRL)을 구성하려면 먼저 몇 가지 사전 설정이 필요할 수 있습니다. 이 섹션에서는 CRL이 연결된 CA를 만들기 전에 이해해야 하는 사전 요구 사항 및 옵션에 대해 설명합니다.

온라인 인증서 상태 프로토콜(OCSP)을 CRL의 대안 또는 보완으로 사용하는 방법에 대한 자세한 내용은 Certificate revocation options 및 에 대한 OCSP URL 사용자 지정 AWS Private CA을 참조하세요.

주제

- CRL 유형
- CRL 구조
- Amazon S3의 CRL에 대한 액세스 정책
- CloudFront를 사용하여 S3 퍼블릭 액세스 차단(BPA) 활성화
- CRL 배포 지점(CDP) URI 확인
-

CRL 유형

- 완료 - 기본 설정입니다.는 CA에서 발급하고 취소된 만료되지 않은 모든 인증서에 대해 분할되지 않은 단일 CRL 파일을 AWS Private CA 유지합니다. AWS Private CA 발급하는 각 인증서는 RFC 5280에 정의된 대로 CRL 배포 지점(CDP) 확장을 통해 특정 CRL에 바인딩됩니다. 전체 CRL이 활성화된 각 CA에 대해 최대 1백만 개의 프라이빗 인증서를 보유할 수 있습니다. 자세한 내용은 AWS Private CA 할당량을 참조하세요.
- 파티셔닝됨 - 전체 CRLs에 비해 파티셔닝된 CRLs 프라이빗 CA가 발급할 수 있는 인증서 수를 크게 늘리고 CAs를 자주 교체하지 않아도 됩니다.

> **Important**
>
> 분할된 CRLs 사용하는 경우 CRL의 관련 발급 배포 지점(IDP) URI가 인증서의 CDP URI와 일치하는지 확인하여 올바른 CRL을 가져왔는지 확인해야 합니다.는 IDP 확장을 중요한 것으로 AWS Private CA 표시하며, 클라이언트가 이를 처리할 수 있어야 합니다.

CRL 구조

각 CRL은 DER로 인코딩된 파일입니다. 파일을 다운로드하고 OpenSSL을 사용해 파일을 보려면 다음과 같은 명령을 사용하십시오.

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

CRL은 다음 형식을 사용합니다.

```
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**
>
> CRL은 해당 인증서를 참조하는 인증서가 발급된 후 Amazon S3에 배치됩니다. 그 전에는 Amazon S3 버킷에 `acm-pca-permission-test-key` 파일만 표시됩니다.

Amazon S3의 CRL에 대한 액세스 정책

CRL을 생성하려는 경우에 저장할 Amazon S3 버킷을 준비해야 합니다.는 지정한 Amazon S3 버킷에 CRL을 AWS Private CA 자동으로 저장하고 주기적으로 업데이트합니다. 자세한 내용은 버킷 생성 섹션을 참조하세요.

S3 버킷은 첨부된 IAM 권한 정책으로 보호되어야 합니다. 권한 있는 사용자 및 서비스 보안 주체는가 버킷에 객체를 배치할 수 AWS Private CA 있는 `Put` 권한과 객체를 검색할 수 있는 `Get` 권한이 필요합니다.

> **Note**
>
> IAM 정책 구성은 AWS 리전 관련에 따라 달라집니다. 리전은 다음 두 가지 범주로 나뉩니다.
>
> - 기본 지원 리전 - 모든에 대해 기본적으로 활성화된 리전입니다 AWS 계정.
> - 기본 비활성화 리전 - 기본적으로 비활성화되지만 고객이 수동으로 활성화할 수 있는 리전입니다.
>
> 자세한 내용과 기본 비활성화 리전 목록은 AWS 리전관리를 참조하세요. IAM과 관련된 서비스 보안 주체에 대한 설명은 옵트인 리전의AWS 서비스 보안 주체를 참조하세요.
> CRL을 인증서 취소 방법으로 구성하면 AWS Private CA 는 CRL을 생성하여 S3 버킷에 게시합니다. S3 버킷에는 AWS Private CA 서비스 보안 주체가 버킷에 쓸 수 있도록 허용하는 IAM 정책이 필요합니다. 서비스 보안 주체의 이름은 사용하는 리전에 따라 다르며 모든 가능성이 지원되는 것은 아닙니다.

PCA	S3	서비스 보안 주체
둘 다 같은 리전에 있습니다.		`acm-pca.amazonaws.com`
활성화됨	활성화됨	`acm-pca.amazonaws.com`

PCA	S3	서비스 보안 주체
비활성	활성화됨	acm-pca.*Region*.amazonaws.com
활성화됨	비활성화됨	지원되지 않음

기본 정책은 CA에 SourceArn 제한을 적용하지 않습니다. 특정 AWS 계정과 특정 프라이빗 CA 모두에 대한 액세스를 제한하는 다음과 같은 덜 허용적인 정책을 적용하는 것이 좋습니다. 또는 aws:SourceOrgID 조건 키를 사용하여의 특정 조직에 대한 액세스를 제한할 수 있습니다 AWS Organizations. 버킷 정책에 대한 자세한 내용은 Amazon Simple Storage Service의 버킷 정책을 참조하세요.

기본 정책을 허용하도록 선택한 경우 나중에 언제든지 수정할 수 있습니다.

CloudFront를 사용하여 S3 퍼블릭 액세스 차단(BPA) 활성화

새 Amazon S3 버킷은 기본적으로 퍼블릭 액세스 차단(BPA) 기능이 활성화된 상태로 구성됩니다. Amazon S3 보안 모범 사례에 포함된 BPA는 고객이 S3 버킷의 객체 및 전체 버킷에 대한 액세스를 세밀하게 조정하는 데 사용할 수 있는 액세스 제어 세트입니다. BPA가 활성 상태이고 올바르게 구성된 경우 승인되고 인증된 AWS 사용자만 버킷과 해당 콘텐츠에 액세스할 수 있습니다.

AWS 에서는 잠재적 공격자에게 민감한 정보가 노출되지 않도록 모든 S3 버킷에서 BPA를 사용할 것을 권장합니다. 그러나 PKI 클라이언트가 퍼블릭 인터넷을 통해(즉, AWS 계정에 로그인하지 않은 상태에서) CRLs을 검색하는 경우 추가 계획이 필요합니다. 이 섹션에서는 콘텐츠 전송 네트워크(CDN)인 Amazon CloudFront를 사용하여 S3 버킷에 대한 인증된 클라이언트 액세스를 요구하지 않고 CRL을 제공하도록 프라이빗 PKI 솔루션을 구성하는 방법을 설명합니다.

> **Note**
>
> CloudFront를 사용하면 AWS 계정에 추가 비용이 발생합니다. 자세한 내용은 Amazon CloudFront 요금을 참조하세요.
> BPA가 활성화된 S3 버킷에 CRL을 저장하기로 선택하고 CloudFront를 사용하지 않는 경우, PKI 클라이언트가 CRL에 액세스할 수 있도록 다른 CDN 솔루션을 구축해야 합니다.

BPA에 대한 CloudFront 설정

프라이빗 S3 버킷에 액세스할 수 있고 인증되지 않은 클라이언트에게 CRL을 제공할 수 있는 CloudFront 배포를 생성합니다.

CRL에 대한 CloudFront 배포를 구성하는 방법

1. Amazon CloudFront 개발자 안내서의 배포 생성에 나와 있는 절차를 사용하여 새 CloudFront 배포를 생성합니다.

   절차를 완료하는 동안 다음 설정을 적용합니다.

   - 원본 도메인 이름에서 S3 버킷을 선택합니다.
   - 버킷 액세스 제한에서 예를 선택합니다.
   - 원본 액세스 ID에 대한 새 ID 생성을 선택합니다.
   - 버킷에 대한 읽기 권한 부여에서 예, 버킷 정책 업데이트를 선택합니다.

   > **Note**
   >
   > 이 절차에서 CloudFront는 버킷 객체에 액세스할 수 있도록 버킷 정책을 수정합니다. `crl` 폴더 아래의 객체에만 액세스할 수 있도록 이 정책을 편집해 보세요.

2. 배포가 초기화되면 CloudFront 콘솔에서 해당 도메인 이름을 찾아 다음 절차를 위해 저장합니다.

   > **Note**
   >
   > us-east-1이 아닌 리전에서 S3 버킷을 새로 생성한 경우 CloudFront를 통해 게시된 애플리케이션에 액세스할 때 HTTP 307 임시 리디렉션 오류가 발생할 수 있습니다. 버킷 주소가 전파되는 데 몇 시간이 걸릴 수 있습니다.

CA에서 BPA을 설정합니다.

새 CA를 구성하는 동안 CloudFront 배포에 별칭을 포함합니다.

CloudFront용 CNAME을 사용하여 CA를 구성하는 방법

- 에서 프라이빗 CA 생성 AWS Private CA를 사용하여 CA를 생성합니다.

절차를 수행할 때 해지 파일 `revoke_config.txt`에는 비공개 CRL 객체를 지정하고 CloudFront의 배포 엔드포인트에 URL을 제공하는 다음 행이 포함되어야 합니다.

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

이후에 이 CA로 인증서를 발급하면 인증서에 다음과 같은 블록이 포함됩니다.

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

> **Note**
>
> 이 CA에서 발급한 이전 인증서가 있는 경우 해당 CA는 CRL에 액세스할 수 없습니다.

CRL 배포 지점(CDP) URI 확인

워크플로에서 CRL 배포 지점(CDP) URI를 사용해야 하는 경우 해당 인증서에서 CRL URI를 사용하는 인증서를 발급하거나 다음 방법을 사용할 수 있습니다. 이는 전체 CRLs에서만 작동합니다. 분할된 CRLs 임의 GUID가 추가됩니다.

S3 버킷을 CA의 CRL 배포 지점(CDP)으로 사용하는 경우 CDP URI는 다음 형식 중 하나일 수 있습니다.

- `http://amzn-s3-demo-bucket.s3.region-code.amazonaws.com/crl/CA-ID.crl`
- `http://s3.region-code.amazonaws.com/amzn-s3-demo-bucket/crl/CA-ID.crl`

사용자 지정 CNAME으로 CA를 구성한 경우 CDP URI에는 예를 들어 CNAME이 포함됩니다. `http://alternative.example.com/crl/CA-ID.crl`

기본적으로는 리전별 IPv4-only `amazonaws.com` 엔드포인트를 사용하여 CDP 확장을 AWS Private CA 작성합니다. IPv6를 통해 CRLs 사용하려면 다음 단계 중 하나를 수행하여 CDPs가 S3의 듀얼 스택 엔드포인트를 가리키는 URLs로 작성되도록 합니다.

- CRL 사용자 지정 이름을 S3 듀얼 스택 엔드포인트 도메인으로 설정합니다. 예: *bucketname*.s3.dualstack.*region-code*.amazonaws.com
- 관련 S3 듀얼 스택 엔드포인트를 가리키는 자체 CNAME DNS 레코드를 설정한 다음 CRL 사용자 지정 이름으로 사용합니다.

에 대한 OCSP URL 사용자 지정 AWS Private CA

> **Note**
>
> 이 주제는 브랜딩 또는 기타 목적으로 온라인 인증서 상태 프로토콜(OCSP) 응답자 엔드포인트의 퍼블릭 URL을 사용자 지정하려는 고객을 위한 것입니다. AWS Private CA 관리형 OCSP의 기본 구성을 사용하려는 경우이 주제를 건너뛰고 취소 구성의 구성 지침을 따를 수 있습니다.

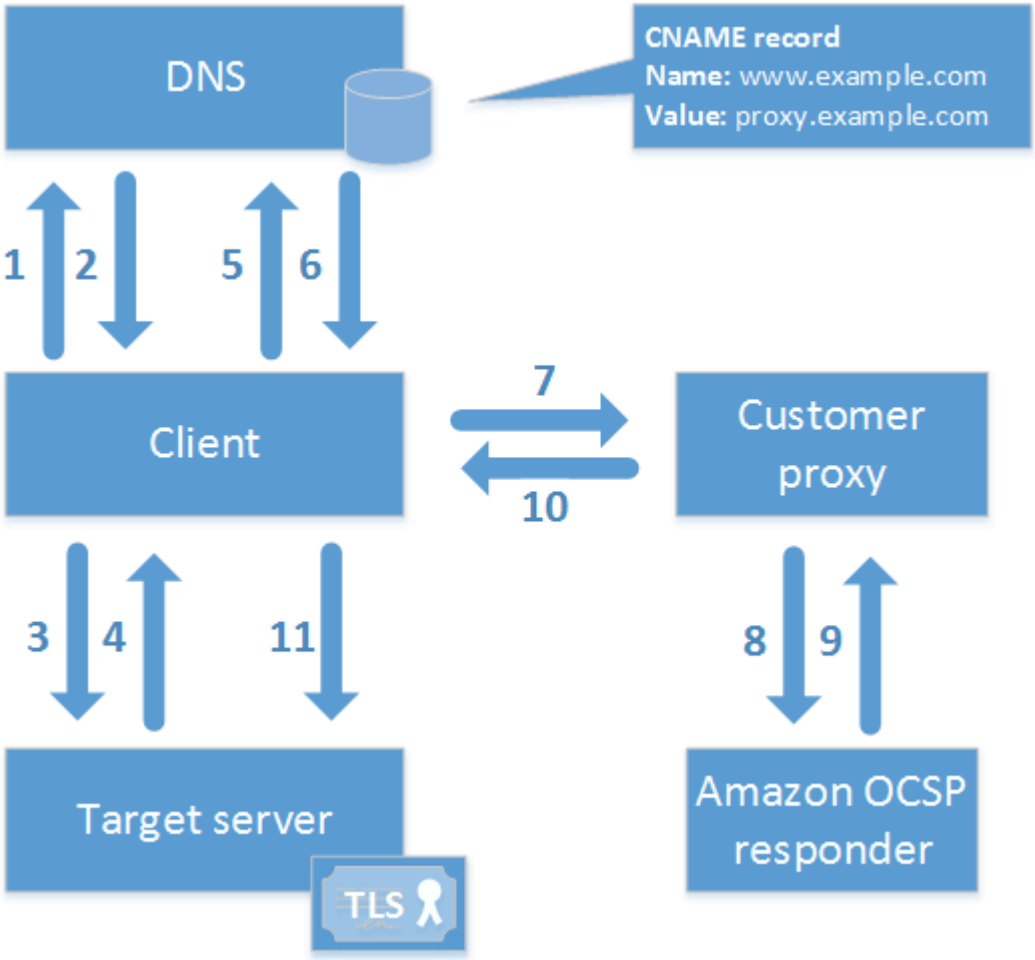
기본적으로 OCSP를 활성화하면 발급하는 AWS Private CA각 인증서에는 AWS OCSP 응답기의 URL이 포함됩니다. 이렇게 하면 암호로 안전한 연결을 요청하는 클라이언트가 OCSP 유효성 검사 쿼리를 AWS에 직접 보낼 수 있습니다. 하지만 경우에 따라서는 AWS에 대한 OCSP 쿼리를 제출하면서 인증서에 다른 URL을 지정하는 것이 더 나을 수도 있습니다.

> **Note**
>
> OCSP의 대안 또는 보완책으로 CRL (인증서 취소 목록) 을 사용하는 방법에 대한 자세한 내용은 해지 구성 및 인증서 취소 목록(CRL) 계획을 참조하세요.

OCSP의 사용자 지정 URL 구성에는 세 가지 요소가 포함됩니다.

- CA 구성 - 에서 프라이빗 CA 생성 AWS Private CA의 예제 2: OCSP와 사용자 지정 CNAME이 활성화된 CA 생성에 설명된 대로 CA에 대한 `RevocationConfiguration`의 사용자 지정 OCSP URL을 지정합니다.
- DNS - 도메인 구성에 CNAME 레코드를 추가하여 인증서에 나타나는 URL을 프록시 서버 URL에 매핑합니다. 자세한 설명은 에서 프라이빗 CA 생성 AWS Private CA에서 예제 2: OCSP와 사용자 지정 CNAME이 활성화된 CA 생성 섹션을 참조하세요.
- 전달 프록시 서버 - 수신한 OCSP 트래픽을 AWS OCSP 응답자에게 투명하게 전달할 수 있는 프록시 서버를 설정합니다.

다음 다이어그램은 이러한 요소가 함께 작동하는 방식을 보여줍니다.



다이어그램에 표시된 대로 사용자 지정된 OCSP 검증 프로세스에는 다음 단계가 포함됩니다.

1. 클라이언트는 대상 도메인의 DNS를 쿼리합니다.
2. 클라이언트는 대상 IP를 수신합니다.
3. 클라이언트는 타겟과 TCP 연결을 엽니다.
4. 클라이언트는 대상 TLS 인증서를 받습니다.
5. 클라이언트는 인증서에 나열된 OCSP 도메인에 대해 DNS를 쿼리합니다.
6. 클라이언트는 프록시 IP를 받습니다.
7. 클라이언트가 OCSP 쿼리를 프록시로 보냅니다.
8. 프록시는 OCSP 응답자에게 쿼리를 전달합니다.
9. 응답자는 인증서 상태를 프록시에 반환합니다.
10. 프록시는 인증서 상태를 클라이언트에 전달합니다.

11.인증서가 유효하면 클라이언트가 TLS 핸드셰이크를 시작합니다.

> **Tip**
>
> 이 예제는 위에서 설명한 대로 CA를 구성한 후 Amazon CloudFront와 Amazon Route 53을 사용하여 구현할 수 있습니다.
>
> 1. CloudFront에서 배포를 생성하고 다음과 같이 구성합니다.
>   - 사용자 지정 CNAME과 일치하는 대체 이름을 생성합니다.
>   - 인증서를 여기에 바인딩합니다.
>   - 를 오리진`ocsp.acm-pca.<region>.amazonaws.com`으로 설정합니다.
>   - IPv6 연결을 사용하려면 듀얼 스택 엔드포인트를 사용합니다. `acm-pca-ocsp.<region>.api.aws`
>   - `Managed-CachingDisabled` 정책을 배포합니다.
>   - 뷰어 프로토콜 정책을 HTTP 및 HTTPS로 설정합니다.
>   - 허용된 HTTP 방법을 GET, HEAD, OPTIONS, PUT, POST, PATCH, DELETE로 설정합니다.
> 2. Route 53에서 사용자 지정 CNAME을 CloudFront 배포의 URL에 매핑하는 DNS 레코드를 생성합니다.

IPv6를 통한 OCSP 사용

기본 AWS Private CA OCSP 응답기 URL은 IPv4-only입니다. IPv6를 통해 OCSP를 사용하려면 CA에 대한 사용자 지정 OCSP URL을 구성합니다. URL은 다음 중 하나일 수 있습니다.

- 형식을 취하는 듀얼 스택 PCA OCSP 응답자의 FQDN `acm-pca-ocsp.region-name.api.aws`
- 위에서 설명한 대로 듀얼 스택 OCSP 응답기를 가리키도록 구성한 CNAME 레코드입니다.

AWS Private CA CA 모드 이해

AWS Private CA 는 두 모드 중 하나로 인증 기관(CA) 생성을 지원합니다. 범용 인증서와 단기 인증서 모드는 CA에서 발급한 인증서의 허용되는 유효 기간에 영향을 미칩니다.

> **Note**
>
> AWS Private CA 는 루트 CA 인증서에 대한 유효성 검사를 수행하지 않습니다.

범용(기본값)

이 모드를 사용하면 CA가 유효 기간에 상관없이 인증서를 발급할 수 있습니다. 대부분의 애플리케이션은 이 유형의 인증서를 사용합니다. 일반적으로 CA는 해지 메커니즘도 지정합니다.

수명이 짧은 인증서

이 모드는 최대 유효 기간이 7일인 인증서를 독점적으로 발급하는 CA를 정의합니다. 이러한 단기 인증서는 매우 빨리 만료되므로 해지 메커니즘을 마련하지 않고도 배포할 수 있습니다. 일부 애플리케이션의 경우 해지로 인한 네트워크 및 처리 오버헤드가 발생하는 것보다 단기 인증서를 자주 배포하는 것이 더 합리적입니다.

수명이 짧은 인증서는 인증서 계층 구조의 마지막 CA여야 합니다. 프라이빗 CA는 7일마다 갱신해야 하므로 상당한 오버헤드가 발생합니다.

수명이 짧은 인증서 모드CAs는 범용 CAs. 자세한 내용은 AWS Private Certificate Authority 요금을 참조하세요.

단기 인증서를 발급하는 CA를 생성하려면 CA 생성을 위한 CA 생성 절차를 사용하여 `UsageMode` 파라미터를 단기 인증서로 설정합니다.

> **Note**
>
> AWS Certificate Manager 는 수명이 짧은 모드의 프라이빗 CA에서 서명한 인증서를 발급할 수 없습니다.

다음 AWS 서비스에서는 단기 인증서 사용을 지원합니다.

- Amazon AppStream
- Amazon WorkSpaces

의 복원력 계획 AWS Private CA

AWS 글로벌 인프라는 AWS 리전 및 가용 영역을 중심으로 구축됩니다. AWS 리전은 물리적으로 분리되고 격리된 여러 가용 영역을 제공하며,이 가용 영역은 지연 시간이 짧고 처리량이 높으며 중복성이 높은 네트워킹과 연결됩니다. 가용 영역을 사용하면 중단 없이 영역 간에 자동으로 장애 극복 조치가 이루어지는 애플리케이션 및 데이터베이스를 설계하고 운영할 수 있습니다. 가용 영역은 기존의 단일 또는 다중 데이터 센터 인프라보다 가용성, 내결함성, 확장성이 뛰어납니다.

AWS 리전 및 가용 영역에 대한 자세한 내용은 AWS 글로벌 인프라를 참조하세요.

이중화 및 재해 복구

CA 계층 구조를 계획할 때 중복성과 DR을 고려합니다. AWS Private CA 는 여러 리전에서 사용할 수 있으므로 여러 리전에서 중복 CAs를 생성할 수 있습니다. 이 AWS Private CA 서비스는 가용성이 99.9%인 서비스 수준 계약(SLA)으로 운영됩니다. 이중화와 재해 복구를 위해 고려할 수 있는 접근 방식은 최소 두 가지입니다. 루트 CA 또는 가장 높은 레벨의 하위 CA에서 이중화 구성할 수 있습니다. 각 접근법에는 장단점이 있습니다.

1. 중복 및 재해 복구를 위해 서로 다른 두 AWS 리전에 두 개의 루트 CAs를 생성할 수 있습니다. 이 구성을 사용하면 각 루트 CA가 AWS 리전에서 독립적으로 작동하여 단일 리전 재해 발생 시 사용자를 보호합니다. 그러나 이중화된 루트 CA를 생성하면 운영 복잡성이 증가합니다. 두 루트 CA 인증서를 모두 사용자 환경의 브라우저 및 운영 체제의 신뢰 저장소에 배포해야 합니다.
2. 또한 중복 하위 CA를 생성하여 각 AWS 리전에 배포하고 이를 단일 AWS 리전의 동일한 고유 루트 CA에 체인으로 연결할 수 있습니다. 이 접근 방식의 이점은 사용자 환경의 신뢰 저장소에 단일 루트 CA 인증서만 배포하면 된다는 것입니다. 제한 사항은 루트 CA가 있는 AWS 리전에 영향을 미치는 재해 발생 시 중복 루트 CA가 없다는 것입니다.

의 인증 기관 AWS Private CA

를 사용하면 조직의 내부 사용을 위해 루트 및 하위 인증 기관(CAs)의 완전히 AWS 호스팅된 계층 구조를 생성할 AWS Private Certificate Authority수 있습니다. 인증서 해지를 관리하기 위해 온라인 인증서 상태 프로토콜(OCSP), 인증서 해지 목록(CRLs) 또는 둘 다를 활성화할 수 있습니다.는 CA 인증서, CRLs 및 OCSP 응답을 AWS Private CA 저장하고 관리하며 루트 기관의 프라이빗 키는에서 안전하게 저장됩니다 AWS.

> **Note**
>
> 의 OCSP 구현은 OCSP 요청 확장을 지원하지 AWS Private CA 않습니다. 여러 인증서가 포함된 OCSP 배치 쿼리를 제출하면 AWS OCSP 응답기는 대기열의 첫 번째 인증서만 처리하고 다른 인증서는 삭제합니다. 해지가 OCSP 응답에 표시되려면 최대 1시간이 걸릴 수 있습니다.

AWS Management Console, AWS CLI및 AWS Private CA API를 AWS Private CA 사용하여에 액세스할 수 있습니다. 다음 주제는 콘솔 및 CLI를 사용하는 방법을 보여줍니다. API에 대해 자세히 알아보려면 AWS Private Certificate Authority API 참조 섹션을 참조하세요. API 사용 방법을 보여주는 Java 예제는 와 AWS Private CA 함께 사용 AWS SDK for Java 섹션을 참조하세요.

활성 프라이빗 CA를 생성하고 이에 대한 액세스를 구성한 후에 설명된 대로 인증서를 발급하고 검색할 수 있습니다에서 인증서 발급 및 관리 AWS Private CA.

주제

- 를 사용하도록 설정 AWS Private CA
- 에서 프라이빗 CA 생성 AWS Private CA
- CA 인증서 설치
- 프라이빗 CA에 대한 액세스 제어
- 프라이빗 CAs 나열
- 프라이빗 CA 보기
- 프라이빗 CA에 대한 태그 추가
- AWS Private CA CA 상태 이해
- 에서 프라이빗 CA 업데이트 AWS Private Certificate Authority
- 프라이빗 CA 삭제
- 프라이빗 CA 복원

- 외부에서 서명된 사설 CA 인증서 사용

를 사용하도록 설정 AWS Private CA

아직 Amazon Web Services(AWS) 고객이 아닌 경우 AWS Private CA을 사용하려면 가입해야 합니다. 계정은 사용 가능한 모든 서비스에 자동으로 액세스할 수 있지만, 사용하는 서비스에 대해서만 요금이 청구됩니다.

주제

- 에 가입 AWS 계정
- 관리자 액세스 권한이 있는 사용자 생성
- 설치 AWS Command Line Interface

에 가입 AWS 계정

이 없는 경우 다음 단계를 AWS 계정완료하여 생성합니다.

에 가입하려면 AWS 계정

1. https://portal.aws.amazon.com/billing/signup을 엽니다.
2. 온라인 지시 사항을 따르세요.

   등록 절차 중 전화 또는 텍스트 메시지를 받고 전화 키패드로 확인 코드를 입력하는 과정이 있습니다.

   에 가입하면 AWS 계정AWS 계정 루트 사용자이 생성됩니다. 루트 사용자에게는 계정의 모든 AWS 서비스 및 리소스에 액세스할 권한이 있습니다. 보안 모범 사례는 사용자에게 관리 액세스 권한을 할당하고, 루트 사용자만 사용하여 루트 사용자 액세스 권한이 필요한 작업을 수행하는 것입니다.

AWS 는 가입 프로세스가 완료된 후 확인 이메일을 보냅니다. 언제든지 https://aws.amazon.com/으로 이동하고 내 계정을 선택하여 현재 계정 활동을 확인하고 계정을 관리할 수 있습니다.

관리자 액세스 권한이 있는 사용자 생성

에 가입한 후 일상적인 작업에 루트 사용자를 사용하지 않도록 관리 사용자를 AWS 계정보호 AWS IAM Identity Center, AWS 계정 루트 사용자활성화 및 생성합니다.

보안 AWS 계정 루트 사용자

1. 루트 사용자를 선택하고 AWS 계정 이메일 주소를 입력하여 계정 소유자AWS Management Console로에 로그인합니다. 다음 페이지에서 비밀번호를 입력합니다.

   루트 사용자를 사용하여 로그인하는 데 도움이 필요하면 AWS 로그인 사용 설명서의 루트 사용자로 로그인을 참조하세요.
2. 루트 사용자의 다중 인증(MFA)을 활성화합니다.

   지침은 IAM 사용 설명서의 AWS 계정 루트 사용자(콘솔)에 대한 가상 MFA 디바이스 활성화를 참조하세요.

관리자 액세스 권한이 있는 사용자 생성

1. IAM Identity Center를 활성화합니다.

   지침은 AWS IAM Identity Center 사용 설명서의 AWS IAM Identity Center설정을 참조하세요.
2. IAM Identity Center에서 사용자에게 관리 액세스 권한을 부여합니다.

   를 자격 증명 소스 IAM Identity Center 디렉터리 로 사용하는 방법에 대한 자습서는 사용 AWS IAM Identity Center 설명서의 기본값으로 사용자 액세스 구성을 IAM Identity Center 디렉터리 참조하세요.

관리 액세스 권한이 있는 사용자로 로그인

- IAM IDentity Center 사용자로 로그인하려면 IAM Identity Center 사용자를 생성할 때 이메일 주소로 전송된 로그인 URL을 사용합니다.

  IAM Identity Center 사용자를 사용하여 로그인하는 데 도움이 필요하면 사용 설명서의 AWS 액세스 포털에 로그인을 참조하세요. AWS 로그인

추가 사용자에게 액세스 권한 할당

1. IAM Identity Center에서 최소 권한 적용 모범 사례를 따르는 권한 세트를 생성합니다.

   지침은 AWS IAM Identity Center 사용 설명서의 Create a permission set를 참조하세요.
2. 사용자를 그룹에 할당하고, 그룹에 Single Sign-On 액세스 권한을 할당합니다.

   지침은 AWS IAM Identity Center 사용 설명서의 그룹 추가를 참조하세요.

설치 AWS Command Line Interface

를 설치하지 AWS CLI 않았지만 사용하려면의 지침을 따르세요AWS Command Line Interface. 이 가이드에서는 엔드포인트, 리전 및 인증 세부 정보를 구성했다고 가정하고 샘플 명령에서 이러한 파라미터를 생략합니다.

에서 프라이빗 CA 생성 AWS Private CA

이 섹션의 절차를 사용하여 루트 CA 또는 하위 CA 중 하나를 생성할 수 있으므로 조직의 요구 사항과 일치하는 트러스트 관계에 대해 감사 가능한 계층을 만들 수 있습니다. AWS Management Console,의 PCA 부분 AWS CLI또는를 사용하여 CA를 생성할 수 있습니다 AWS CloudFormation.

이미 생성한 CA의 구성을 업데이트하는 방법에 대한 자세한 내용은 에서 프라이빗 CA 업데이트 AWS Private Certificate Authority 섹션을 참조하세요.

CA를 사용하여 사용자, 디바이스 및 애플리케이션의 최종 엔터티 인증서에 서명하는 방법은 프라이빗 최종 엔터티 인증서 발급 섹션을 참조하세요.

> **Note**
>
> 계정에는 사설 CA를 생성한 시점부터 각 사설 CA에 대한 월별 요금이 청구됩니다.
> 최신 AWS Private CA 요금 정보는 AWS Private Certificate Authority 요금을 참조하세요. AWS 요금 계산기를 사용하여 비용을 추정할 수도 있습니다.

주제

- 프라이빗 CA 생성을 위한 CLI 예제

Console

콘솔을 사용하여 프라이빗 API를 생성하는 방법

1. AWS Management Console을 사용하여 프라이빗 CA를 생성하려면 다음 단계를 완료합니다.

   콘솔을 사용하여 시작하기

   AWS 계정에 로그인하고에서 AWS Private CA 콘솔을 엽니다**https://console.aws.amazon.com/acm-pca/home**.

- 프라이빗 CA가 없는 리전에서 콘솔을 여는 경우 소개 페이지가 나타납니다. 프라이빗 CA 생성을 선택합니다.
- CA를 이미 생성한 리전에서 콘솔을 여는 경우 CA 목록이 포함된 프라이빗 인증 기관 페이지가 열립니다. CA 생성을 선택합니다.

2. 모드 옵션에서 CA에서 발급하는 인증서의 만료 모드를 선택합니다.

- 범용 - 모든 만료 날짜로 구성할 수 있는 인증서를 발급합니다. 기본값입니다.
- 단기 인증서 - 최대 유효 기간이 7일인 인증서를 발급합니다. 경우에 따라 짧은 유효 기간이 해지 메커니즘을 대체할 수 있습니다.

3. 콘솔의 유형 옵션 섹션에서 만들려는 프라이빗 인증 기관의 유형을 선택합니다.

- 루트를 선택하면 새 CA 계층이 설정됩니다. 이 CA는 자체 서명된 인증서에 의해 지원됩니다. 이는 계층의 다른 CA 및 최종 엔터티 인증서에 대한 궁극적 서명 기관 역할을 합니다.
- 하위를 선택하면 계층에서 위에 있는 상위 CA가 서명해야 하는 CA가 생성됩니다. 하위 CA는 일반적으로 다른 하위 CA를 생성하거나 사용자, 컴퓨터 및 애플리케이션에 최종 엔터티 인증서를 발급하는 데 사용됩니다.

> **Note**
>
> AWS Private CA 는 하위 CA의 상위 CA도 호스팅되는 경우 자동 서명 프로세스를 제공합니다 AWS Private CA. 사용할 상위 CA를 선택하기만 하면 됩니다.
> 외부 신뢰 서비스 공급자가 하위 CA에 서명해야 할 수도 있습니다. 그렇다면는 서명된 CA 인증서를 얻기 위해 다운로드하고 사용해야 하는 인증서 서명 요청(CSR)을 AWS Private CA 제공합니다. 자세한 내용은 외부 상위 CA에서 서명한 하위 CA 인증서 설치 단원을 참조하십시오.

4. 보안 주체 고유 이름 옵션에서 프라이빗 CA의 보안 주체 이름을 구성합니다. 다음 옵션 중 하나 이상의 값을 입력해야 합니다.

- 조직(O) - 예: 회사 이름
- 조직 단위(OU) - 예: 회사 내 부서
- 국가 이름(C) - 두 글자로 된 국가 코드
- 주 또는 도 이름 - 주 또는 도의 전체 이름

- 지역 이름 - 도시 이름
- 일반 이름(CN) - CA를 식별하기 위해 사람이 읽을 수 있는 문자열입니다.

> **Note**
>
> 발급 시 APIPassthrough 템플릿을 적용하여 인증서의 보안 주체 이름을 추가로 사용자 지정할 수 있습니다. 자세한 정보와 상세한 예제는 APIPassThrough 템플릿을 사용하여 사용자 지정 보안 주체 이름이 포함된 인증서를 발급합니다. 섹션을 참조하세요.

백업 인증서는 자체 서명되므로 사설 CA에 제공하는 보안 주체 정보는 공용 CA에 포함된 정보보다 더 적을 수 있습니다. 보안 주체 고유 이름을 구성하는 각 값에 대한 자세한 내용은 RFC 5280을 참조하세요.

5. 키 알고리즘 옵션에서 키 알고리즘과 알고리즘 강도를 선택합니다. 기본값은 RSA 2048입니다. 다음 알고리즘 중에서 선택할 수 있습니다.

   - ML-DSA-44
   - ML-DSA-65
   - ML-DSA-87
   - RSA 2048
   - RSA 3072
   - RSA 4096
   - ECDSA P256
   - ECDSA P384
   - ECDSA P521

6. 인증서 취소 옵션에서 인증서를 사용하는 클라이언트와 해지 상태를 공유하는 두 가지 방법 중 하나를 선택할 수 있습니다.

   - CRL 배포 활성화
   - OCSP 활성화

   CA에 대해 이러한 해지 옵션 중 하나를 구성하거나, 둘 다 구성하거나, 둘 다 구성하지 않을 수 있습니다. 선택 사항이긴 하지만 관리형 해지가 모범 사례로 권장됩니다. 이 단계를 완료하기

전에 각 방법의 장점, 필요할 수 있는 예비 설정 및 추가 해지 기능에 대한 정보는 AWS Private CA 인증서 해지 방법 계획섹션을 참조하세요.

> **Note**
>
> 해지를 구성하지 않고 CA를 생성하는 경우 언제든지 나중에 구성할 수 있습니다. 자세한 내용은 에서 프라이빗 CA 업데이트 AWS Private Certificate Authority 단원을 참조하십시오.

인증서 해지 옵션을 구성하려면 다음 단계를 수행합니다.

a. 인증서 취소 옵션에서 CRL 배포 활성화를 선택합니다.

b. S3 버킷 URI의 목록에서 기존 버킷을 선택합니다.

   기존 버킷을 지정할 때 계정 및 버킷에 대해 BPA가 비활성화되어 있는지 확인해야 합니다. 그렇지 않으면 CA를 생성하는 작업이 실패합니다. CA가 성공적으로 생성된 경우에도 정책을 수동으로 연결해야 CRL 생성을 시작할 수 있습니다. Amazon S3의 CRL에 대한 액세스 정책에 설명된 정책 패턴 중 하나를 사용합니다. 자세한 내용은 Amazon S3 콘솔을 사용하여 버킷 정책 추가를 참조하세요.

c. 추가 구성 옵션을 보려면 CRL 설정을 확장합니다.

   - 파티셔닝 활성화를 선택하여 CRLs의 파티셔닝을 활성화합니다. 파티셔닝을 활성화하지 않으면 CA에 취소된 인증서의 최대 수가 적용됩니다. 자세한 내용은 AWS Private Certificate Authority 할당량을 참조하세요. 분할된 CRLs. ???
   - 사용자 지정 CRL 이름을 추가하여 Amazon S3 버킷에 대한 별칭을 생성합니다. 이 이름은 RFC 5280에서 정의한 “CRL 배포 지점” 확장에 따라 CA에서 발급한 인증서에 포함되어 있습니다. IPv6를 통해 CRLs 사용하려면 IPv6를 통해 CRLs 사용에 설명된 대로 버킷의 듀얼 스택 S3 엔드포인트로 설정합니다.
   - 사용자 지정 경로를 추가하여 Amazon S3 버킷의 파일 경로에 대한 DNS 별칭을 생성합니다.
   - CRL이 유효한 상태로 유지되는 일수를 입력합니다. 기본값은 7일입니다. 온라인 CRL의 경우, 유효 기간은 일반적으로 2~7일입니다. AWS Private CA 에서는 지정된 기간의 1/2 시점에 CRL 재생성을 시도합니다.

7. 인증서 해지 옵션에서 OCSP 켜기를 선택합니다.

- 사용자 지정 OCSP 엔드포인트 - 옵션 필드에서 Amazon 이외의 OCSP 엔드포인트에 대한 정규화된 도메인 이름(FQDN)을 제공할 수 있습니다. IPv6를 통해 OCSP를 사용하려면 IPv6를 통해 OCSP 사용에 설명된 대로이 필드를 듀얼 스택 엔드포인트로 설정합니다.

  이 필드에 FQDN을 제공하면는 AWS OCSP 응답기의 기본 URL 대신 발급된 각 인증서의 권한 정보 액세스 확장에 FQDN을 AWS Private CA 삽입합니다. 엔드포인트는 사용자 지정 FQDN이 포함된 인증서를 받으면 해당 주소를 쿼리하여 OCSP 응답을 요청합니다. 이 메커니즘이 작동하려면 다음 두 가지 추가 조치를 취해야 합니다.

  - 프록시 서버를 사용하여 사용자 지정 FQDN에 도착하는 트래픽을 AWS OCSP 응답기로 전달합니다.
  - 해당 CNAME 레코드를 DNS 데이터베이스에 추가합니다.

  > **Tip**
  >
  > 사용자 지정 CNAME을 사용하여 완전한 OCSP 솔루션을 구현하는 방법에 대한 자세한 내용은 에 대한 OCSP URL 사용자 지정 AWS Private CA 섹션을 참조하세요.

  예를 들어, 다음은 Amazon Route 53에 표시되는 사용자 지정 OCSP에 대한 CNAME 레코드입니다.

레코드 이름	Type	라우팅 정책	차별화 요소	값/트래픽 라우팅 대상
alternative.example.com	CNAME	간편함	-	proxy.example.com

  > **Note**
  >
  > CNAME 값에 'http://' 또는 'https://'와 같은 프로토콜 접두사가 포함되면 안 됩니다.

8. 선택에 따라 태그 추가 아래에서 인증서에 태그를 지정할 수 있습니다. 태그는 AWS 리소스를 식별하고 구성하기 위한 메타데이터 역할을 하는 키-값 페어입니다. AWS Private CA 태그 파라미터 목록과 생성 후 CAs에 태그를 추가하는 방법에 대한 지침은 섹션을 참조하세요프라이빗 CA에 대한 태그 추가.

   > **Note**
   >
   > 생성 절차 중에 프라이빗 CA에 태그를 첨부하려면 CA 관리자가 먼저 인라인 IAM 정책을 `CreateCertificateAuthority` 작업에 연결하고 태그 지정을 명시적으로 허용해야 합니다. 자세한 내용은 생성 시 태그 지정: CA를 만들 때 CA에 태그 첨부 단원을 참조하십시오.

9. CA 권한 옵션에서 선택적으로 AWS Certificate Manager 서비스 보안 주체에게 자동 갱신 권한을 위임할 수 있습니다. ACM은 이 권한이 부여된 경우에만 이 CA에서 생성한 프라이빗 최종 엔터티 인증서를 자동으로 갱신할 수 있습니다. 언제든지 AWS Private CA CreatePermission API 또는 create-permission CLI 명령을 사용하여 갱신 권한을 할당할 수 있습니다.

   기본적으로 이러한 권한을 사용하도록 설정되어 있습니다.

   > **Note**
   >
   > AWS Certificate Manager 는 단기 인증서의 자동 갱신을 지원하지 않습니다.

10. 요금에서 프라이빗 CA의 요금을 이해했는지 확인하세요.

    > **Note**
    >
    > 최신 AWS Private CA 요금 정보는 AWS Private Certificate Authority 요금을 참조하세요. AWS 요금 계산기를 사용하여 비용을 추정할 수도 있습니다.

11. 입력한 모든 정보가 정확한지 확인한 후 CA 생성을 선택합니다. CA의 세부 정보 페이지가 열리고 상태가 보류 중인 인증서로 표시됩니다.

> **Note**
>
> 세부 정보 페이지에서 작업, CA 인증서 설치를 선택하여 CA 구성을 완료하거나 나중에 프리이빗 인증 기관 목록으로 돌아가서 해당 경우에 적용되는 설치 절차를 완료할 수 있습니다.
>
> - 루트 CA 인증서 설치
> - 에서 호스팅하는 하위 CA 인증서 설치 AWS Private CA
> - 외부 상위 CA에서 서명한 하위 CA 인증서 설치

CLI

프라이빗 CA를 만들려면 create-certificate-authority 명령을 사용합니다. CA 구성(알고리즘 및 보안 주체 이름 정보 포함), 해지 구성(OCSP 및/또는 CRL을 사용하려는 경우), CA 유형(루트 또는 하위)을 지정해야 합니다. 구성 및 해지 구성 세부 정보는 명령에 인수로 제공하는 두 파일에 포함되어 있습니다. 선택적으로 CA 사용 모드(표준 또는 단기 인증서 발급용)를 구성하고, 태그를 첨부하고, 멱등성 토큰을 제공할 수도 있습니다.

CRL을 구성하는 경우 create-certificate-authority 명령을 실행하기 전에 안전한 Amazon S3 버킷이 있어야 합니다. 자세한 내용은 Amazon S3의 CRL에 대한 액세스 정책 단원을 참조하십시오.

CA 구성 파일은 다음 정보를 지정합니다.

- 알고리즘의 이름
- CA 프라이빗 키를 생성하는 데 사용할 키 크기
- CA가 자체 인증서 서명 요청, CRLs 및 OCSP 응답에 서명하는 데 사용하는 서명 알고리즘의 유형입니다.
- X.500 보안 주체 정보

OCSP의 해지 구성은 다음 정보를 사용하여 `OcspConfiguration` 객체를 정의합니다.

- `Enabled` 플래그를 “true”로 설정합니다.
- (옵션) `OcspCustomCname`의 값으로 선언된 사용자 지정 CNAME입니다.

CRL의 해지 구성은 다음 정보로 `CrlConfiguration` 객체를 정의합니다.

- `Enabled` 플래그를 “true”로 설정합니다.
- CRL 만료 기간(일)(CRL의 유효 기간).
- CRL이 포함되어 있는 Amazon S3 버킷입니다.
- (선택 사항) CRL에 공개적으로 액세스할 수 있는지 여부를 결정하는 S3ObjectAcl 값입니다. 여기에 제시된 예제에서는 공개 액세스가 차단됩니다. 자세한 내용은 CloudFront를 사용하여 S3 퍼블릭 액세스 차단(BPA) 활성화 단원을 참조하십시오.
- (옵션) CA에서 발급한 인증서에 포함된 S3 버킷의 CNAME 별칭입니다. CRL에 공개적으로 액세스할 수 없는 경우 이는 Amazon CloudFront와 같은 배포 메커니즘을 가리킵니다.
- (선택 사항) 다음 정보가 포함된 `CrlDistributionPointExtensionConfiguration` 객체입니다.
  - "true" 또는 "false"로 설정된 `OmitExtension` 플래그입니다. 이렇게 하면 CDP 확장의 기본값을 CA에서 발급한 인증서에 쓸지 여부가 제어됩니다. CDP 확장에 대한 자세한 내용은 섹션을 참조하세요CRL 배포 지점(CDP) URI 확인. OmitExtension이 "true"인 경우 CustomCname을 설정할 수 없습니다.
- (선택 사항) S3 버킷의 CRL에 대한 사용자 지정 경로입니다.
- (선택 사항) CRL이 완료 또는 분할될지 여부를 결정하는 CrlType 값입니다. 입력하지 않으면 CRL이 기본적으로 완료됩니다.

> **Note**
>
> `OcspConfiguration` 객체와 `CrlConfiguration` 객체를 모두 정의하여 동일한 CA에서 두 개의 해지 메커니즘을 모두 활성화할 수 있습니다. --revocation-configuration 파라미터를 제공하지 않으면 기본적으로 두 메커니즘 모두 비활성화됩니다. 나중에 해지 유효성 검사 지원이 필요한 경우 CA(CLI) 업데이트 섹션을 참조하세요.

CLI 예제는 다음 섹션을 참조하세요.

프라이빗 CA 생성을 위한 CLI 예제

다음 예에서는 유효한 기본 리전, 엔드포인트 및 자격 증명을 사용하여 `.aws` 구성 디렉터리를 설정했다고 가정합니다. AWS CLI 환경 구성에 대한 자세한 내용은 구성 및 자격 증명 파일 설정을 참조하세요. 가독성을 높이기 위해 예제 명령에서 CA 구성 및 해지 입력을 JSON 파일로 제공합니다. 필요에 따라 예제 파일을 수정하여 사용하세요.

달리 명시되지 않는 한 모든 예제는 다음 `ca_config.txt` 구성 파일을 사용합니다.

파일: ca_config.txt

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

예제 1: OCSP가 활성화된 CA 생성

이 예제에서 해지 파일은 AWS Private CA 응답기를 사용하여 인증서 상태를 확인하는 기본 OCSP 지원을 활성화합니다.

파일: OCSP용 revoke_config.txt

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

명령

```
$ 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
```

이 명령이 제대로 실행되면 새 CA의 Amazon 리소스 이름(ARN)을 출력합니다.

```
{
```

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

명령

```
$ 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
```

이 명령이 제대로 실행되면 CA의 Amazon 리소스 이름(ARN)을 출력합니다.

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

CA의 구성을 검사하려면 다음 명령을 사용합니다.

```
$ 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
```

이 설명에 다음 사항이 포함되어 있어야 합니다.

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

예제 2: OCSP와 사용자 지정 CNAME이 활성화된 CA 생성

이 예제에서 해지 파일은 사용자 지정된 OCSP 지원을 활성화합니다. `OcspCustomCname` 파라미터는 정규화된 도메인 이름(FQDN)을 값으로 사용합니다.

이 필드에 FQDN을 제공하면는 AWS OCSP 응답기의 기본 URL 대신 발급된 각 인증서의 권한 정보 액세스 확장에 FQDN을 AWS Private CA 삽입합니다. 엔드포인트는 사용자 지정 FQDN이 포함된 인증서를 받으면 해당 주소를 쿼리하여 OCSP 응답을 요청합니다. 이 메커니즘이 작동하려면 다음 두 가지 추가 조치를 취해야 합니다.

- 프록시 서버를 사용하여 사용자 지정 FQDN에 도착하는 트래픽을 AWS OCSP 응답기로 전달합니다.
- 해당 CNAME 레코드를 DNS 데이터베이스에 추가합니다.

> **Tip**
>
> 사용자 지정 CNAME을 사용하여 완전한 OCSP 솔루션을 구현하는 방법에 대한 자세한 내용은 에 대한 OCSP URL 사용자 지정 AWS Private CA 섹션을 참조하세요.

예를 들어, 다음은 Amazon Route 53에 표시되는 사용자 지정 OCSP에 대한 CNAME 레코드입니다.

레코드 이름	Type	라우팅 정책	차별화 요소	값/트래픽 라우팅 대상
alternative.example.com	CNAME	간편함	-	proxy.example.com

> **Note**
>
> CNAME 값에 'http://' 또는 'https://'와 같은 프로토콜 접두사가 포함되면 안 됩니다.

파일: OCSP용 revoke_config.txt

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

명령

```
$ 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
```

이 명령이 제대로 실행되면 CA의 Amazon 리소스 이름(ARN)을 출력합니다.

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

CA의 구성을 검사하려면 다음 명령을 사용합니다.

```
$ 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
```

이 설명에 다음 사항이 포함되어 있어야 합니다.

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

예제 3: CRL이 첨부된 CA 생성

이 예제에서는 해지 구성이 CRL 파라미터를 정의합니다.

파일: revoke_config.txt

```
{
```

```
   "CrlConfiguration":{
      "Enabled":true,
      "ExpirationInDays":7,
      "S3BucketName":"amzn-s3-demo-bucket"
   }
}
```

명령

```
$ 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
```

이 명령이 제대로 실행되면 CA의 Amazon 리소스 이름(ARN)을 출력합니다.

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

CA의 구성을 검사하려면 다음 명령을 사용합니다.

```
$ 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
```

이 설명에 다음 사항이 포함되어 있어야 합니다.

```
"RevocationConfiguration": {
   ...
   "CrlConfiguration": {
      "Enabled": true,
      "ExpirationInDays": 7,
      "S3BucketName": "amzn-s3-demo-bucket"
   },
   ...
}
```

예제 4: CRL이 첨부되고 사용자 지정 CNAME이 활성화된 CA 생성

이 예제에서 해지 구성은 사용자 지정 CNAME이 포함된 CRL 파라미터를 정의합니다.

파일: revoke_config.txt

```
{
   "CrlConfiguration":{
      "Enabled":true,
      "ExpirationInDays":7,
      "CustomCname": "alternative.example.com",
      "S3BucketName":"amzn-s3-demo-bucket"
   }
}
```

명령

```
$ 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
```

이 명령이 제대로 실행되면 CA의 Amazon 리소스 이름(ARN)을 출력합니다.

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

CA의 구성을 검사하려면 다음 명령을 사용합니다.

```
$ 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
```

이 설명에 다음 사항이 포함되어 있어야 합니다.

```
"RevocationConfiguration": {
   ...
```

```
   "CrlConfiguration": {
      "Enabled": true,
      "ExpirationInDays": 7,
      "CustomCname": "alternative.example.com",
      "S3BucketName": "amzn-s3-demo-bucket",
   ...
   }
}
```

예제 5: CA 생성 및 사용 모드 지정

이 예제에서는 CA를 생성할 때 CA 사용 모드를 지정합니다. 지정하지 않을 경우 사용 모드 파라미터의 기본값은 GENERAL_PURPOSE입니다. 이 예제에서는 파라미터가 SHORT_LIVED_CERTICATION으로 설정되어 있습니다. 즉, CA는 최대 유효 기간이 7일인 인증서를 발급합니다. 해지를 구성하는 것이 불편한 상황에서는 손상된 단기 인증서가 정상 작업의 일부로 빠르게 만료됩니다. 따라서 이 예제 CA에는 해지 메커니즘이 없습니다.

> **Note**
>
> AWS Private CA 는 루트 CA 인증서에 대한 유효성 검사를 수행하지 않습니다.

```
$ 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
```

다음 describe-certificate-authority 명령과 같이에서 명령을 사용하여 결과 CA에 대한 세부 정보를 AWS CLI 표시합니다.

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

예제 6: Active Directory 로그인을 위한 CA 생성

Microsoft Active Directory(AD)의 Enterprise NTAuth 저장소에서 사용하기에 적합한 프라이빗 CA를 만들 수 있으며, 이 저장소에서 카드 로그온 또는 도메인 컨트롤러 인증서를 발급할 수 있습니다. CA 인증서를 AD로 가져오는 방법에 대한 자세한 내용은 타사 인증 기관(CA) 인증서를 Enterprise NAuth 저장소로 가져오는 방법을 참조하세요.

Microsoft certutil 도구를 사용하면 -dspublish 옵션을 호출하여 AD에 CA 인증서를 게시할 수 있습니다. certutil을 사용하여 AD에 게시된 인증서는 전체 포리스트에서 신뢰됩니다. 그룹 정책을 사용하여 전체 포리스트의 하위 집합(예: 단일 도메인 또는 도메인의 컴퓨터 그룹)으로 신뢰를 제한할 수도 있습니다. 로그온이 제대로 작동하려면 발급 CA도 NTAuth 저장소에 게시해야 합니다. 자세한 내용은 그룹 정책을 사용하여 클라이언트 컴퓨터에 인증서 배포를 참조하세요.

이 예제에서는 다음 `ca_config_AD.txt` 구성 파일을 사용합니다.

파일: 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"
         }
      ]
   }
}
```

명령

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

이 명령이 제대로 실행되면 CA의 Amazon 리소스 이름(ARN)을 출력합니다.

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

CA의 구성을 검사하려면 다음 명령을 사용합니다.

```
$ 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
```

이 설명에 다음 사항이 포함되어 있어야 합니다.

```
...

"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"
      }
   ]
}
...
```

예제 7: 발급된 인증서에서 생략된 CRL 및 CDP 확장이 연결된 Matter CA 생성

Matter 스마트 홈 표준에 대한 인증서를 발급하는 데 적합한 프라이빗 CA를 생성할 수 있습니다. 이 예제에서의 CA 구성은 공급업체 ID(VID)가 FFF1로 설정된 Matter Product Attestation Authority(PAA)를 `ca_config_PAA.txt` 정의합니다.

파일: 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"
      }
    ]
  }
}
```

취소 구성은 CRLs 활성화하고 발급된 인증서에서 기본 CDP URL을 생략하도록 CA를 구성합니다.

파일: revoke_config.txt

```
{
   "CrlConfiguration":{
      "Enabled":true,
      "ExpirationInDays":7,
      "S3BucketName":"amzn-s3-demo-bucket",
   "CrlDistributionPointExtensionConfiguration":{
  "OmitExtension":true
   }
   }
}
```

명령

```
$ 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
```

이 명령이 제대로 실행되면 CA의 Amazon 리소스 이름(ARN)을 출력합니다.

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

CA의 구성을 검사하려면 다음 명령을 사용합니다.

```
$ 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
```

이 설명에 다음 사항이 포함되어 있어야 합니다.

```
"RevocationConfiguration": {
   ...
   "CrlConfiguration": {
      "Enabled": true,
      "ExpirationInDays": 7,
      "S3BucketName": "amzn-s3-demo-bucket",
   "CrlDistributionPointExtensionConfiguration":{
  "OmitExtension":true
   }
   },
   ...
}
...
```

CA 인증서 설치

사설 CA 인증서를 생성 및 설치하려면 다음 절차를 수행하십시오. 그러면 CA를 사용할 준비가 완료됩니다.

AWS Private CA 는 CA 인증서 설치를 위한 세 가지 시나리오를 지원합니다.

- 에서 호스팅하는 루트 CA에 대한 인증서 설치 AWS Private CA
- 상위 기관이 AWS Private CA에 의해 호스팅되는 하위 CA 인증서 설치
- 상위 기관이 외부에서 호스팅되는 하위 CA 인증서 설치

다음 단원에서는 각 시나리오에 대한 절차에 대해 설명합니다. 콘솔 절차가 콘솔 페이지 사설 CA에서 시작됩니다.

호환되는 서명 알고리즘

CA 인증서에 대한 서명 알고리즘 지원은 상위 CA의 서명 알고리즘과 AWS 리전에 따라 달라집니다. 다음 제약 조건은 콘솔과 AWS CLI 작업 모두에 적용됩니다.

- RSA 키 알고리즘이 있는 상위 CA는 다음 서명 알고리즘으로 인증서를 발급할 수 있습니다.
  - SHA256 RSA

- SHA384 RSA
- SHA512 RSA

- 레거시에서 EDCSA 키 알고리즘 AWS 리전이 있는 상위 CA는 다음 서명 알고리즘으로 인증서를 발급할 수 있습니다.
  - SHA256 ECDSA
  - SHA384 ECDSA
  - SHA512 ECDSA

  레거시에는 다음이 AWS 리전 포함됩니다.

리전 이름	지리적 위치
eu-north-1	유럽(스톡홀름)
me-south-1	Middle East (Bahrain)
ap-south-1	아시아 태평양(뭄바이)
eu-west-3	유럽(파리)
us-east-2	미국 동부(오하이오)
af-south-1	아프리카(케이프타운)
eu-west-1	유럽(아일랜드)
eu-central-1	유럽(프랑크푸르트)
sa-east-1	남아메리카(상파울루)
ap-east-1	아시아 태평양(홍콩)

리전 이름	지리적 위치
us-east-1	미국 동부(버지니아 북부)
ap-northeast-2	아시아 태평양(서울)
eu-west-2	유럽(런던)
ap-northeast-1	아시아 태평양(도쿄)
us-gov-east-1	AWS GovCloud(미국 동부)
us-gov-west-1	AWS GovCloud(미국 서부)
us-west-2	미국 서부(오리건)
us-west-1	미국 서부(캘리포니아 북부)
ap-southeast-1	아시아 태평양(싱가포르)
ap-southeast-2	아시아 태평양(시드니)

- 레거시가 아닌 경우 EDCSA에 다음 AWS 리전규칙이 적용됩니다.
  - EC_prime256v1 서명 알고리즘을 사용하는 상위 CA는 ECDSA P256으로 인증서를 발급할 수 있습니다.
  - EC_secp384r1 서명 알고리즘을 사용하는 상위 CA는 ECDSA P384로 인증서를 발급할 수 있습니다.
- 모든에서 EDCSA에 다음 AWS 리전규칙이 적용됩니다.
  - EC_secp521r1 서명 알고리즘이 있는 상위 CA는 ECDSA P521로 인증서를 발급할 수 있습니다.

루트 CA 인증서 설치

AWS Management Console 또는에서 루트 CA 인증서를 설치할 수 있습니다 AWS CLI.

프라이빗 루트 CA(콘솔)에 대한 인증서를 생성 및 설치하는 방법

1. (옵션) CA의 세부 정보 페이지에 아직 접속하지 않은 경우 https://console.aws.amazon.com/acm-pca/home에서 AWS Private CA 콘솔을 엽니다. 프라이빗 인증 기관 페이지에서 보류 중인 인증서 또는 활성 상태인 루트 CA를 선택합니다.
2. 작업, CA 인증서 설치를 선택하여 루트 CA 인증서 설치 페이지를 엽니다.
3. 루트 CA 인증서 파라미터 지정에서 다음과 같은 인증서 파라미터를 지정합니다.

   - 유효성 - CA 인증서의 만료 날짜 및 시간을 지정합니다. 루트 CA 인증서의 AWS Private CA 기본 유효 기간은 10년입니다.
   - 서명 알고리즘 - 루트 CA가 새 인증서를 발급할 때 사용할 서명 알고리즘을 지정합니다. 사용 가능한 옵션은 CA를 생성하는 AWS 리전 에 따라 다릅니다. 자세한 내용은 호환되는 서명 알고리즘, 에서 지원되는 암호화 알고리즘 AWS Private Certificate Authority 및 CertificateAuthorityConfiguration의 SigningAlgorithm을 참조하세요.
     - SHA256 RSA
     - SHA384 RSA
     - SHA512 RSA

   설정이 올바른지 검토한 다음 확인 및 install. AWS Private CA exports a CSR for your CA를 선택하고 루트 CA 인증서 템플릿을 사용하여 인증서를 생성한 AWS Private CA 다음 자체 서명 루트 CA 인증서를 가져옵니다.
4. CA의 세부 정보 페이지 상단에 설치 상태(성공 또는 실패)가 표시됩니다. 설치가 성공하면 새로 완료된 루트 CA의 일반 창에 활성 상태가 표시됩니다.

프라이빗 루트 CA에 대한 인증서를 생성 및 설치하는 방법(AWS CLI)

1. 인증서 서명 요청(CSR)을 생성합니다.

```
$ 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
```

base64 형식으로 인코딩된 결과 파일 ca.csr인 PEM 파일은 다음과 같이 표시됩니다.

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

OpenSSL을 사용하여 CSR의 내용을 보고 확인할 수 있습니다.

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

그러면 다음과 유사한 출력이 생성됩니다.

```
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. 이전 단계의 CSR을 `--csr` 파라미터의 인수로 사용하여 루트 인증서를 발급합니다.

> **Note**
>
> AWS CLI 버전 1.6.3 이상을 사용하는 경우 필요한 입력 파일을 지정할 `fileb://` 때 접두사를 사용합니다. 이렇게 AWS Private CA 하면가 Base64-encoded 데이터를 올바르게 구문 분석할 수 있습니다.

```
$ 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. 루트 인증서를 검색합니다.

```
$ 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
```

base64 형식으로 인코딩된 결과 파일 `cert.pem`인 PEM 파일은 다음과 같이 표시됩니다.

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

OpenSSL을 사용하여 인증서의 내용을 보고 확인할 수 있습니다.

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

그러면 다음과 유사한 출력이 생성됩니다.

```
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. 루트 CA 인증서를 가져와서 CA에 설치합니다.

> **Note**
>
> AWS CLI 버전 1.6.3 이상을 사용하는 경우 필요한 입력 파일을 지정할 `fileb://` 때 접두사를 사용합니다. 이렇게 AWS Private CA 하면가 Base64-encoded 데이터를 올바르게 구문 분석할 수 있습니다.

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

CA의 새 상태를 검사합니다.

```
$ 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
```

이제 상태가 ACTIVE로 표시됩니다.

```
{
    "CertificateAuthority": {
        "Arn": "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
        "CreatedAt": "2021-03-05T14:24:12.867000-08:00",
        "LastStateChangeAt": "2021-03-08T12:37:14.235000-08:00",
        "Type": "ROOT",
        "Serial": "serial_number",
        "Status": "ACTIVE",
        "NotBefore": "2021-03-08T07:46:27-08:00",
        "NotAfter": "2022-03-08T08:46:27-08:00",
        "CertificateAuthorityConfiguration": {
            "KeyAlgorithm": "RSA_2048",
            "SigningAlgorithm": "SHA256WITHRSA",
            "Subject": {
                "Country": "US",
                "Organization": "Example Corp",
                "OrganizationalUnit": "Sales",
                "State": "WA",
                "CommonName": "www.example.com",
                "Locality": "Seattle"
            }
        },
        "RevocationConfiguration": {
            "CrlConfiguration": {
                "Enabled": true,
```

```
                "ExpirationInDays": 7,
                "CustomCname": "alternative.example.com",
                "S3BucketName": "amzn-s3-demo-bucket"
            },
            "OcspConfiguration": {
                "Enabled": false
            }
        }
    }
}
```

에서 호스팅하는 하위 CA 인증서 설치 AWS Private CA

AWS Management Console 를 사용하여 호스팅된 하위 CA에 대한 인증서를 생성하고 설치할 수 있습니다 AWS Private CA .

AWS Private CA 호스팅된 하위 CA에 대한 인증서를 생성하고 설치하려면

1. (옵션) CA의 세부 정보 페이지에 아직 접속하지 않은 경우 https://console.aws.amazon.com/acm-pca/home에서 AWS Private CA 콘솔을 엽니다. 프라이빗 인증 기관 페이지에서 보류 중인 인증서 또는 활성 상태인 하위 CA를 선택합니다.
2. 작업, CA 인증서 설치를 선택하여 하위 CA 인증서 설치 페이지를 엽니다.
3. 하위 CA 인증서 설치 페이지의 CA 유형 선택에서에서 관리하는 인증서를 설치AWS Private CA하도록 선택합니다 AWS Private CA.
4. 상위 CA 선택 아래의 상위 프라이빗 CA 목록에서 CA를 선택합니다. 이 목록은 다음 기준을 충족하는 CA를 표시하도록 필터링됩니다.

   - 해당 CA에 대한 사용 권한이 있습니다.
   - CA는 자체적으로 서명하지 않습니다.
   - CA의 상태가 ACTIVE입니다.
   - CA 모드는 GENERAL_PURPOSE입니다.
5. 하위 CA 인증서 파라미터 지정에서 다음과 같은 인증서 파라미터를 지정합니다.

   - 유효성 - CA 인증서의 만료 날짜 및 시간을 지정합니다.
   - 서명 알고리즘 - 루트 CA가 새 인증서를 발급할 때 사용할 서명 알고리즘을 지정합니다. 옵션은 다음과 같습니다.
     - SHA256 RSA
     - SHA384 RSA

- SHA512 RSA

- 경로 길이 - 새 인증서에 서명할 때 하위 CA가 추가할 수 있는 신뢰 계층 수입니다. 경로 길이가 0(기본값)이면 CA 인증서가 아닌 최종 엔터티 인증서만 생성될 수 있음을 의미합니다. 경로 길이가 1 이상이면 하위 CA가 인증서를 발급하여 추가할 하위 CA를 생성할 수 있습니다.
- 템플릿 ARN - 이 CA 인증서에 대한 구성 템플릿의 ARN을 표시합니다. 지정된 경로 길이를 변경하면 템플릿이 변경됩니다. CLI issue-certificate 명령 또는 API IssueCertificate 작업을 사용하여 인증서를 생성하는 경우 ARN을 수동으로 지정해야 합니다. 사용 가능한 CA 인증서 템플릿에 대한 자세한 내용은 AWS Private CA 인증서 템플릿 사용 단원을 참조하십시오.

6. 설정이 올바른지 검토한 다음 확인 및 install. AWS Private CA exports a CSR을 선택하고 하위 CA 인증서 템플릿을 사용하여 인증서를 생성한 다음 선택한 상위 CA로 인증서에 서명합니다. AWS Private CA 그런 다음는 서명된 하위 CA 인증서를 가져옵니다.
7. CA의 세부 정보 페이지 상단에 설치 상태(성공 또는 실패)가 표시됩니다. 설치가 성공하면 새로 완료된 하위 CA의 일반 창에 활성 상태가 표시됩니다.

외부 상위 CA에서 서명한 하위 CA 인증서 설치

에 설명된 대로 하위 프라이빗 CA를 생성한 후 외부 서명 기관에서 서명한 CA 인증서를 설치하여 활성화에서 프라이빗 CA 생성 AWS Private CA할 수 있습니다. 외부 CA로 하위 CA 인증서를 서명하려면 먼저 외부 신뢰 서비스 공급자를 서명 기관으로 설정하거나 타사 공급자를 사용하도록 설정해야 합니다.

> **Note**
>
> 외부 신뢰 서비스 공급자를 생성하거나 얻는 절차는 이 설명서에서는 다루지 않습니다.

하위 CA를 만들고 외부 서명 기관에 액세스할 수 있게 되면 다음 작업을 완료합니다.

1. 에서 인증서 서명 요청(CSR)을 가져옵니다 AWS Private CA.
2. CSR을 외부 서명 기관에 제출하고 모든 체인 인증서와 함께 서명된 CA 인증서를 받습니다.
3. CA 인증서를 가져오고에 연결하여 하위 CA AWS Private CA 를 활성화합니다.

자세한 절차는 외부에서 서명된 사설 CA 인증서 사용 섹션을 참조하세요.

프라이빗 CA에 대한 액세스 제어

의 프라이빗 CA에 필요한 권한이 있는 모든 사용자는 해당 CA를 사용하여 다른 인증서에 서명할 AWS Private CA 수 있습니다. CA 소유자는 인증서를 발급하거나 인증서를 발급하는 데 필요한 권한을 동일한에 있는 AWS Identity and Access Management (IAM) 사용자에게 위임할 수 있습니다 AWS 계정. 다른 AWS 계정에 상주하는 사용자는 CA 소유자가 리소스 기반 정책을 통해 승인한 경우 인증서를 발급할 수도 있습니다.

권한 있는 사용자는 단일 계정이든 교차 계정이든 인증서를 발급할 때 AWS Private CA 또는 AWS Certificate Manager 리소스를 사용할 수 있습니다. AWS Private CA IssueCertificate API 또는 issue-certificate CLI 명령에서 발급된 인증서는 관리되지 않습니다. 이러한 인증서는 대상 장치에 수동으로 설치해야 하며 만료 시 수동으로 갱신해야 합니다. ACM 콘솔, ACM RequestCertificate API 또는 request-certificate CLI 명령에서 발급한 인증서가 관리됩니다. 이러한 인증서는 ACM과 통합된 서비스에 쉽게 설치할 수 있습니다. CA 관리자가 허용하고 발급자 계정에 ACM에 대한 서비스 연결 역할이 있는 경우 관리형 인증서는 만료 시 자동으로 갱신됩니다.

주제

- IAM 사용자를 위한 단일 계정 권한 생성
- 교차 계정 액세스를 위한 정책 연결

IAM 사용자를 위한 단일 계정 권한 생성

CA 관리자(즉, CA 소유자)와 인증서 발급자가 단일 AWS 계정에 상주하는 경우 권한이 제한된 AWS Identity and Access Management (IAM) 사용자를 생성하여 발급자와 관리자 역할을 분리하는 것이 가장 좋습니다. 예제 권한과 AWS Private CA함께에서 IAM을 사용하는 방법에 대한 자세한 내용은 섹션을 참조하세요용 Identity and Access Management(IAM) AWS Private Certificate Authority.

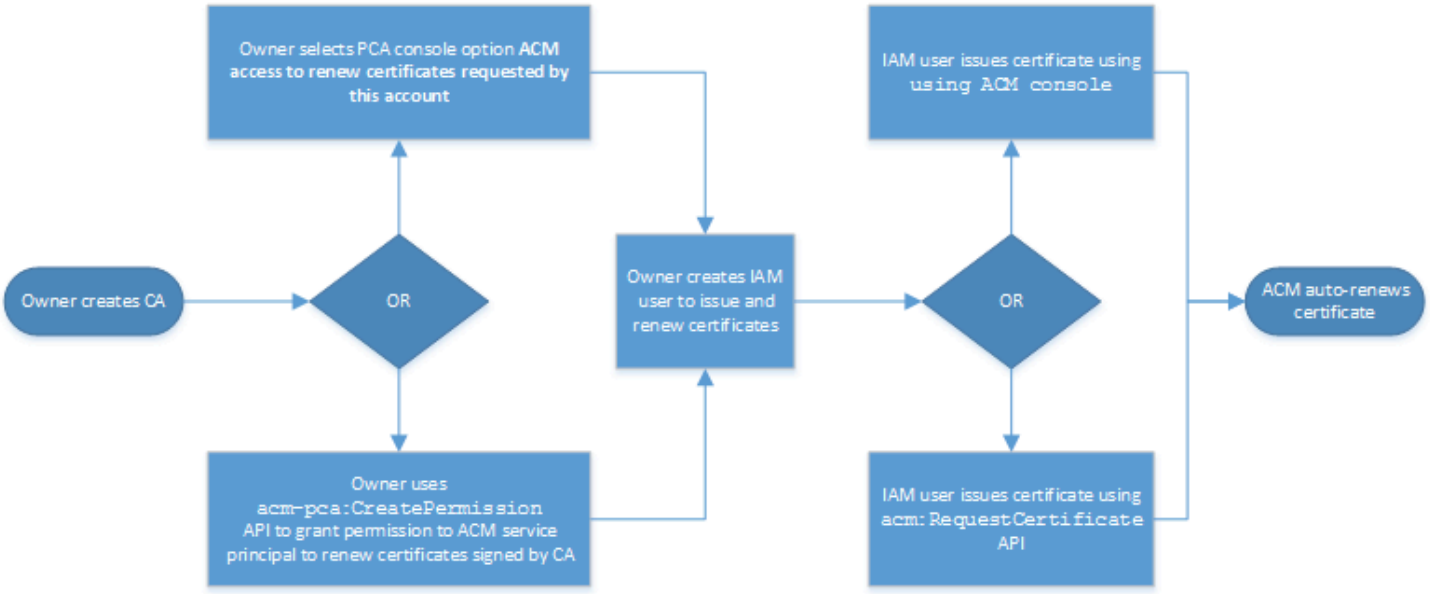
단일 계정 사례 1: 비관리형 인증서 발급

이 경우 계정 소유자는 프라이빗 CA를 만든 다음 프라이빗 CA에서 서명한 인증서를 발급할 권한이 있는 IAM 사용자를 생성합니다. IAM 사용자는 `IssueCertificate` API를 호출하여 인증서를 발급합니다 AWS Private CA .

이러한 방식으로 발급된 인증서는 관리되지 않으므로 관리자가 인증서를 내보내서 사용하려는 장치에 설치해야 합니다. 또한 만료되면 수동으로 갱신해야 합니다. 이 API를 사용하여 인증서를 발급하려면 OpenSSL 또는 유사한 프로그램에 AWS Private CA 의해 외부에서 생성되는 인증서 서명 요청(CSR) 및 키 페어가 필요합니다. 자세한 내용은 `IssueCertificate` 설명서를 참조하세요.

단일 계정 사례 2: ACM을 통한 관리형 인증서 발급

이 두 번째 경우에는 ACM과 PCA의 API 작업이 모두 포함됩니다. 계정 소유자는 이전과 마찬가지로 프라이빗 CA 및 IAM 사용자를 생성합니다. 그러면 계정 소유자는 ACM 서비스 보안 주체에게 이 CA가 서명한 모든 인증서를 자동으로 갱신할 수 있는 권한을 부여합니다. IAM 사용자는 다시 인증서를 발급하지만, 이번에는 CSR 및 키 생성을 처리하는 ACM `RequestCertificate` API를 호출하여 인증서를 발급합니다. 인증서가 만료되면 ACM은 갱신 워크플로를 자동화합니다.



계정 소유자는 CA를 생성하는 동안이나 이후에 또는 PCA `CreatePermission` API를 사용하는 동안 관리 콘솔을 통해 갱신 권한을 부여할 수 있습니다. 이 워크플로에서 생성된 관리형 인증서는 ACM과 통합된 AWS 서비스와 함께에서 사용할 수 있습니다.

다음 섹션에는 갱신 권한을 부여하는 절차가 나와 있습니다.

ACM에 인증서 갱신 권한 할당

AWS Certificate Manager (ACM)의 관리형 갱신 로그인을 사용하면 퍼블릭 인증서와 프라이빗 인증서 모두에 대한 인증서 갱신 프로세스를 자동화할 수 있습니다. ACM이 프라이빗 CA에서 생성한 인증서를 자동으로 갱신하려면 CA 자체에서 ACM 서비스 보안 주체에게 가능한 모든 권한을 부여해야 합니다. ACM에 대해 이러한 갱신 권한이 없는 경우 CA 소유자(또는 승인된 대리인)는 만료 시 각 프라이빗 인증서를 수동으로 재발급해야 합니다

> **Important**
>
> 갱신 권한을 할당하는 이러한 절차는 CA 소유자와 인증서 발급자가 동일한 AWS 계정에 있는 경우에만 적용됩니다. 계정 간 시나리오에 대해서는 교차 계정 액세스를 위한 정책 연결 섹션을 참조하세요.

갱신 권한은 사설 CA를 생성하는 동안 위임을 하거나, CA가 ACTIVE 상태에 있는 한 이후에 언제든지 변경할 수 있습니다.

AWS Private CA 콘솔, AWS Command Line Interface (AWS CLI) 또는 AWS Private CA API에서 사설 CA 권한을 관리할 수 있습니다.

프라이빗 CA 권한을 ACM에 할당하는 방법(콘솔)

1. AWS 계정에 로그인하고 https://console.aws.amazon.com/acm-pca/home AWS Private CA 콘솔을 엽니다.
2. 프라이빗 인증 기관 페이지의 목록에서 프라이빗 CA를 선택합니다.
3. 작업, CA 권한 구성을 선택합니다.
4. ACM 액세스를 승인하여 이 계정에서 요청한 인증서를 갱신을 선택합니다.
5. 저장을 선택합니다.

에서 ACM 권한을 관리하려면 AWS Private CA (AWS CLI)

create-permission 명령을 사용하여 ACM에 권한을 할당합니다. ACM이 인증서를 자동으로 갱신하려면 필요한 권한(IssueCertificate, GetCertificate, 및 ListPermissions)을 할당해야 합니다.

```
$ 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
```

list-permissions 명령을 사용하여 CA에서 위임한 권한을 나열합니다.

```
$ aws acm-pca list-permissions \
```

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

delete-permission 명령을 사용하여 CA가 AWS 서비스 보안 주체에 할당한 권한을 취소합니다.

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

교차 계정 액세스를 위한 정책 연결

CA 관리자와 인증서 발급자가 서로 다른 AWS 계정에 있는 경우 CA 관리자는 CA 액세스를 공유해야 합니다. CA에 리소스 기반 정책을 연결하여 이 작업을 수행합니다. 이 정책은 AWS 계정 소유자, IAM 사용자, AWS Organizations ID 또는 조직 단위 ID일 수 있는 특정 보안 주체에 발급 권한을 부여합니다.

CA 관리자는 다음과 같은 방식으로 정책을 연결하고 관리할 수 있습니다.

- 관리 콘솔에서 계정 간에 AWS 리소스를 공유하는 표준 방법인 AWS Resource Access Manager (RAM)을 사용합니다. 의 CA 리소스를 다른 계정의 보안 주체 AWS RAM 와 공유하면 필요한 리소스 기반 정책이 CA에 자동으로 연결됩니다. RAM에 대한 자세한 내용은 AWS RAM 사용 설명서를 참조하세요.

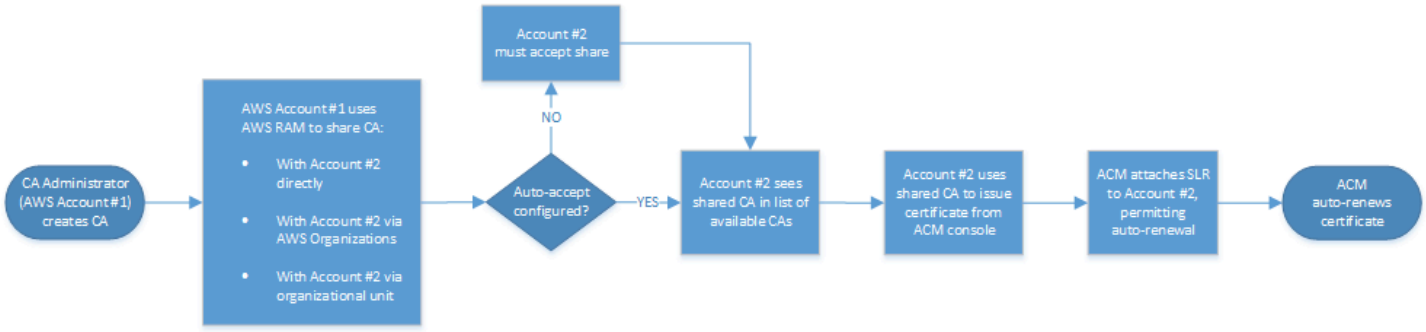
  > **Note**
  >
  > CA를 선택한 다음 작업, 리소스 공유 관리를 선택하여 RAM 콘솔을 쉽게 열 수 있습니다.

- 프로그래밍 방식으로 PCA API PutPolicy, GetPolicy, 및 DeletePolicy를 사용합니다.
- AWS CLI에서 수동으로 PCA 명령 put-policy, get-policy, 및 delete-policy를 사용합니다.

콘솔 방식에만 RAM 액세스가 필요합니다.

교차 계정 사례 1: 콘솔에서 관리형 인증서 발급

이 경우 CA 관리자는 AWS Resource Access Manager (AWS RAM)를 사용하여 CA 액세스를 다른 AWS 계정과 공유하므로 해당 계정에서 관리형 ACM 인증서를 발급할 수 있습니다. 다이어그램은가 CA를 계정과 직접 공유하거나 계정이 멤버인 AWS Organizations ID를 통해 간접적으로 AWS RAM 공유할 수 있음을 보여줍니다.

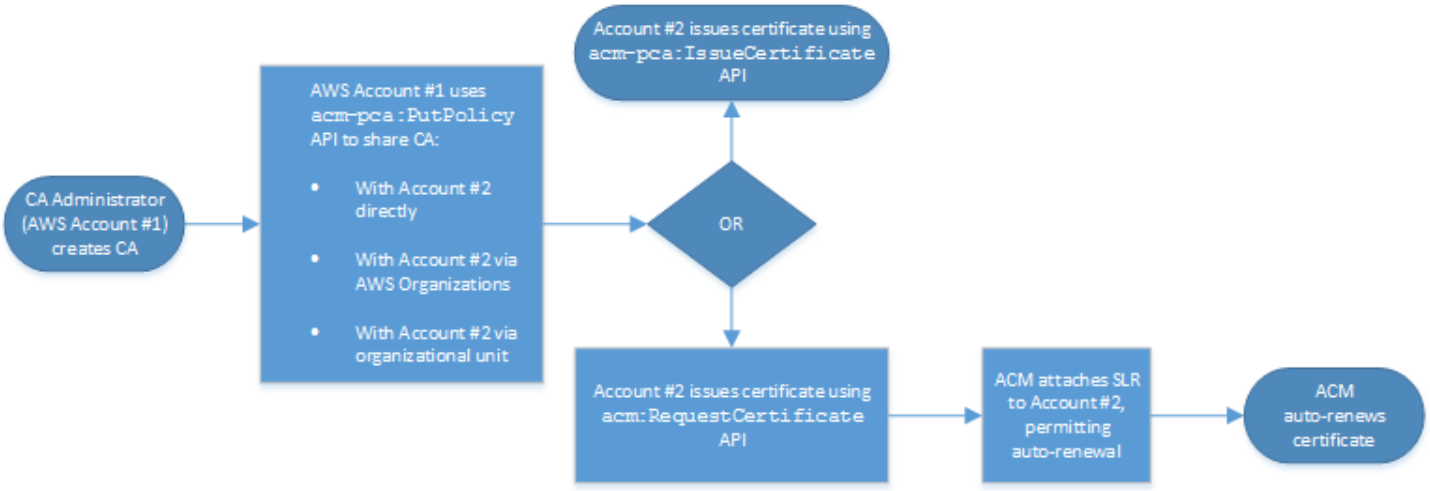


RAM이 리소스를 공유한 후 수신자 보안 주체 AWS Organizations는 리소스를 수락해야 리소스가 적용됩니다. 수신자는 제공된 공유를 자동으로 수락 AWS Organizations 하도록를 구성할 수 있습니다.

> **Note**
>
> 수신자 계정은 ACM에서 자동 갱신을 구성할 책임이 있습니다. 일반적으로 공유 CA를 처음 사용하는 경우 ACM은 AWS Private CA에 대한 무인 인증서 호출을 허용하는 서비스 연결 역할을 설치합니다. 이것이 실패하는 경우(일반적으로 권한 누락으로 인해) CA의 인증서는 자동으로 갱신되지 않습니다. ACM 사용자만 문제를 해결할 수 있으며 CA 관리자는 해결할 수 없습니다. 자세한 내용은 ACM에서 서비스 연결 역할(SLR) 사용을 참조하세요.

교차 계정 사례 2: API 또는 CLI를 사용하여 관리형 및 비관리형 인증서 발급

이 두 번째 사례는 AWS Certificate Manager 및 AWS Private CA API를 사용하여 가능한 공유 및 발급 옵션을 보여줍니다. 이러한 모든 작업은 해당 AWS CLI 명령을 사용하여 수행할 수도 있습니다.



이 예시에서는 API 작업을 직접 사용하므로 인증서 발급자는 두 가지 API 작업 중 하나를 선택하여 인증서를 발급할 수 있습니다. PCA API 작업 `IssueCertificate`을 수행하면 관리되지 않는 인증서가

생성되며 이 인증서는 자동으로 갱신되지 않으므로 내보내 수동으로 설치해야 합니다. ACM API 작업 RequestCertificate를 사용하면 ACM 통합 서비스에 쉽게 설치할 수 있고 자동으로 갱신되는 관리형 인증서가 생성됩니다.

> **Note**
>
> 수신자 계정은 ACM에서 자동 갱신을 구성할 책임이 있습니다. 일반적으로 공유 CA를 처음 사용하는 경우 ACM은 AWS Private CA에 대한 무인 인증서 호출을 수행할 수 있는 서비스 연결 역할을 설치합니다. 일반적으로 권한 누락으로 인해 실패할 경우 CA의 인증서는 자동으로 갱신되지 않으며 CA 관리자가 아닌 ACM 사용자만 문제를 해결할 수 있습니다. 자세한 내용은 ACM에서 서비스 연결 역할(SLR) 사용을 참조하세요.

프라이빗 CAs 나열

AWS Private CA 콘솔 또는를 사용하여 소유하거나 액세스할 수 있는 프라이빗 CAs AWS CLI 나열할 수 있습니다.

콘솔을 사용하여 사용 가능한 CA를 나열하는 방법

1. AWS 계정에 로그인하고 https://console.aws.amazon.com/acm-pca/home AWS Private CA 콘솔을 엽니다.
2. 프라이빗 인증 기관 목록의 정보를 검토합니다. 오른쪽 상단의 페이지 번호를 사용하여 여러 페이지의 CA를 탐색할 수 있습니다. 각 CA는 각 CA에 대해 다음 열 중 일부 또는 전체가 표시되는 행을 차지합니다.

- 제목 – CA에 대한 고유 이름 정보 요약입니다.
- Id – CA의 32바이트 16진수 고유 식별자입니다.
- 상태 - CA 상태. 가능한 값은 생성 중, 인증서 보류 중, 활성, 삭제됨, 비활성화됨, 만료됨 및 실패함입니다.
- 유형 - CA 유형. 가능한 값은 루트 및 하위 값입니다.
- 모드 – CA의 모드입니다. 가능한 값은 범용 인증서(모든 만료 날짜로 구성할 수 있는 인증서 발급)와 단기 인증서(최대 유효 기간이 7일인 인증서 발급)입니다. 경우에 따라 짧은 유효 기간이 해지 메커니즘을 대체할 수 있습니다. 기본값은 범용입니다.
- 소유자 - CA를 소유한 AWS 계정입니다. 이 계정은 사용자 계정이거나 CA 관리 권한을 위임받은 계정일 수 있습니다.

- 키 알고리즘 - CA에서 지원하는 퍼블릭 키 알고리즘입니다. 가능한 값은 ML_DSA_44, ML_DSA_65, ML_DSA_87, RSA_2048, RSA_3072, RSA_4096, EC_prime256v1, EC_secp384r1 및 EC_secp521r1입니다.
- 서명 알고리즘 - CA에서 인증서 요청에 서명하는 데 사용하는 알고리즘입니다. (인증서가 발급될 때 서명하는 데 사용되는 `SigningAlgorithm` 파라미터와 혼동하지 마십시오.) 가능한 값은 ML_DSA_44, ML_DSA_65, ML_DSA_87, SHA256WITHRSA, SHA384WITHRSA, SHA512WITHRSA, SHA256WITHECDSA, SHA384WITHECDSA 및 SHA512WITHECDSA.

> **Note**
>
> 콘솔의 오른쪽 상단에 있는 설정 아이콘을 선택하여 표시할 열과 기타 설정을 사용자 지정할 수 있습니다.

를 사용하여 사용 가능한 CAs를 나열하려면 AWS CLI

list-certificate-authorities 명령을 사용하여 다음 예제와 같이 사용 가능한 CA를 나열합니다.

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

이 명령은 다음과 유사한 정보를 반환합니다.

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

프라이빗 CA 보기

ACM 콘솔 또는를 사용하여 프라이빗 CA AWS CLI 에 대한 자세한 메타데이터를 보고 필요에 따라 여러 값을 변경할 수 있습니다. CA 업데이트에 대한 자세한 내용은 에서 프라이빗 CA 업데이트 AWS Private Certificate Authority 섹션을 참조하세요.

콘솔에서 인증서 세부 정보를 보는 방법

1. AWS 계정에 로그인하고 https://console.aws.amazon.com/acm-pca/home AWS Private CA 콘솔을 엽니다.
2. 프라이빗 인증 기관 목록을 검토합니다. 오른쪽 상단의 페이지 번호를 사용하여 여러 페이지의 CA를 탐색할 수 있습니다.
3. 나열된 CA에 대한 자세한 메타데이터를 표시하려면 검사하려는 CA 옆의 라디오 버튼을 선택합니다. 그러면 다음과 같은 탭 뷰가 있는 세부 정보 창이 열립니다.

   - 제목 - CA의 고유 이름에 대한 정보. 자세한 내용은 Subject distinguished name 단원을 참조하십시오. 표시된 필드는 다음과 같습니다.
     - 제목 - 제공된 이름 정보 필드 요약
     - 조직(O) - 예: 회사 이름
     - 조직 단위(OU) - 예: 회사 내 부서
     - 국가 이름(C) - 두 글자로 된 국가 코드
     - 주 또는 도 이름 - 주 또는 도의 전체 이름
     - 지역 이름 - 도시 이름
     - 일반 이름(CN) - CA를 식별하기 위해 사람이 읽을 수 있는 문자열입니다.
   - CA 인증서 탭 - CA 인증서의 유효성에 대한 정보
     - 유효 기한 - CA 인증서가 유효할 때까지의 날짜 및 시간

    - 만료까지 남은 일수 - 만료될 때까지의 기간(일)
- 해지 구성 탭 - 인증서 해지 옵션에 대한 현재 선택 항목 편집을 선택하여 업데이트합니다.
    - 인증서 해지 목록 (CRL) 배포 - 활성화 또는 비활성화 상태
    - 온라인 인증서 상태 프로토콜(OCSP) - 활성화 또는 비활성화 상태
- 권한 탭 - AWS Certificate Manager (ACM)을 통해 이 CA에 대해 현재 선택한 인증서 갱신 권한입니다. 편집을 선택하여 업데이트합니다.
- 갱신을 위한 ACM 승인 - 승인된 상태 또는 승인되지 않은 상태
- 태그 탭 - 이 CA에 현재 할당되어 있는 사용자 지정 가능한 레이블입니다. 업데이트할 태그 관리를 선택합니다.
- 리소스 공유 탭 - AWS Resource Access Manager (RAM)을 통해이 CA에 대한 리소스 공유의 현재 할당입니다. 업데이트할 리소스 공유 관리를 선택합니다.
    - 이름 – 리소스 공유의 이름
    - 상태 - 리소스 공유의 상태

4. 검사하려는 CA의 ID 필드를 선택하여 일반 창을 엽니다. CA의 32바이트 16진수로 된 고유 식별자가 맨 위에 표시됩니다. 이 창은 다음과 같은 추가 정보를 제공합니다.

    - 상태 - CA 상태. 가능한 값은 생성 중, 인증서 보류 중, 활성, 삭제됨, 비활성화됨, 만료됨 및 실패함입니다.
    - ARN – CA의 Amazon 리소스 이름입니다.
    - 소유자 - CA를 소유한 AWS 계정입니다. 이 계정은 사용자 계정(본인)이거나 CA 관리 권한을 위임받은 계정일 수 있습니다.
    - CA 유형 - CA의 유형입니다. 가능한 값은 루트 및 하위 값입니다.
    - 생성 날짜 – 이 CA가 생성된 날짜와 시간입니다.
    - 만료 날짜 - CA 인증서가 만료되는 날짜 및 시간입니다.
    - 모드 – CA의 모드입니다. 가능한 값은 범용 인증서(모든 만료 날짜로 구성할 수 있는 인증서)와 단기 인증서(최대 유효 기간이 7일인 인증서)입니다. 경우에 따라 짧은 유효 기간이 해지 메커니즘을 대체할 수 있습니다. 기본값은 범용입니다.
    - 키 알고리즘 - CA에서 지원하는 퍼블릭 키 알고리즘입니다. 가능한 값은 ML-DSA-44, ML-DSA-65, ML-DSA-87, RSA 2048, RSA 3072, RSA 4096, ECDSA P256, ECDSA P384 및 ECDSA P521입니다.
    - 서명 알고리즘 - CA가 자체 인증서 서명 요청, CRLs 및 OCSP 응답에 서명하는 데 사용하는 알고리즘입니다(IssueCertificate API에 사용되는 `SigningAlgorithm` 파라미터와 혼동해서

는 안 됨). 가능한 값은 ML-DSA-44, ML-DSA-65, ML-DSA-87, SHA256 RSA, SHA384 RSA 및 SHA512 RSA, SHA256 ECDSA, SHA384 ECDSA, SHA512 ECDSA입니다.

- 키 스토리지 보안 표준 - FIPS(연방 정보 처리 표준) 규정 준수 수준입니다. FIPS 140-2 레벨 2 이상, FIPS 140-2 레벨 3 이상, CCPC 레벨 1 이상 중에서 선택할 수 있습니다. 이 파라미터는 AWS 리전에 따라 다릅니다.

> **Note**
>
> 2023년 1월 26일부터 AWS 프라이빗 CA는 FIPS PUB 140-2 레벨 3을 준수하는 하드웨어 보안 모듈(HSMs 사용하여 중국 외 리전의 모든 CA 프라이빗 키를 보호합니다.

를 사용하여 CA 세부 정보를 보고 수정하려면 AWS CLI

다음 명령에 표시된 대로 AWS CLI 의 describe-certificate-authority 명령을 사용하여 CA에 대한 세부 정보를 표시합니다.

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

이 명령은 다음과 유사한 정보를 반환합니다.

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

명령줄에서 프라이빗 CA를 업데이트하는 방법에 대한 자세한 내용은 CA(CLI) 업데이트 섹션을 참조하세요.

프라이빗 CA에 대한 태그 추가

태그는 AWS 리소스를 식별 및 구성하기 위한 메타데이터로 작동하는 단어나 구문입니다. 각 태그는 키와 값으로 구성됩니다. AWS Private CA 콘솔, AWS Command Line Interface (AWS CLI) 또는 PCA API를 사용하여 프라이빗 CAs.

언제든지 프라이빗설 CA에 대한 사용자 지정 태그를 추가하거나 제거할 수 있습니다. 예를 들어, CA가 어떤 환경을 대상으로 하는지 식별하기 위해 `Environment=Prod` 또는 `Environment=Beta`와 같은 키-값 페어로 프라이빗 CA에 태그를 지정할 수 있습니다. 자세한 내용은 프라이빗 CA 생성을 참조하세요.

> **Note**
>
> 생성 절차 중에 프라이빗 CA에 태그를 첨부하려면 CA 관리자가 먼저 인라인 IAM 정책을 `CreateCertificateAuthority` 작업에 연결하고 태그 지정을 명시적으로 허용해야 합니다. 자세한 내용은 생성 시 태그 지정: CA를 만들 때 CA에 태그 첨부 단원을 참조하십시오.

다른 AWS 리소스도 태그 지정을 지원합니다. 동일한 태그를 다른 리소스에 할당하여 해당 리소스의 관련된 것을 나타낼 수 있습니다. 예를 들어, CA, Elastic Load Balancing 로드 밸런서 및 기타 관련 리소스에 대한 `Website=example.com`과 같은 태그를 할당할 수 있습니다. AWS 리소스 태그 지정에 대한 자세한 내용은 Amazon EC2 사용 설명서의 Amazon EC2 리소스 태그 지정을 참조하세요. Amazon EC2

AWS Private CA 태그에는 다음과 같은 기본 제한이 적용됩니다.

- 프라이빗 CA당 최대 태그 수는 50개입니다.
- 태그 키의 최대 길이는 128자입니다.
- 태그 값의 최대 길이는 256자입니다.

- 태그 키와 값에는 A-Z, a-z 및 .:+=@_%-(하이픈)이 포함될 수 있습니다.
- 태그 키와 값은 대/소문자를 구분합니다.
- `aws:` 및 `rds:` 접두사는 AWS 용으로 예약되어 있습니다. 따라서 키가 `aws:` 또는 `rds:`로 시작하는 태그는 추가, 편집 또는 삭제할 수 없습니다. `aws:` 및 `rds:`로 시작하는 기본 태그는 tags-per-resource 할당량으로 계산되지 않습니다.
- 태깅 스키마를 여러 서비스와 리소스에서 사용하려는 경우, 서비스마다 허용되는 문자에 대한 제한이 다를 수 있음에 유의하십시오. 해당 서비스에 대한 문서를 참조하세요.
- AWS Private CA 태그는 의 리소스 그룹 및 태그 편집기에서 사용할 수 없습니다 AWS Management Console.

AWS Private CA 콘솔, AWS Command Line Interface (AWS CLI) 또는 AWS Private CA API에서 사설 CA에 태그를 지정할 수 있습니다.

사설 CA에 태그를 지정하는 방법(콘솔)

1. AWS 계정에 로그인하고 https://console.aws.amazon.com/acm-pca/home AWS Private CA 콘솔을 엽니다.
2. 프라이빗 인증 기관 페이지의 목록에서 프라이빗 CA를 선택합니다.
3. 목록 아래의 세부 정보 영역에서 태그 탭을 선택합니다. 기존 태그의 목록이 표시됩니다.
4. 태그 관리를 선택합니다.
5. 새로운 태그 추가를 선택합니다.
6. 키 및 값 페어를 입력합니다.
7. 저장을 선택합니다.

사설 CA에 태그를 지정하는 방법(AWS CLI)

tag-certificate-authority 명령을 사용하여 프라이빗 CA에 태그를 추가할 수 있습니다.

```
$ 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 명령을 사용하여 프라이빗 CA의 태그를 나열합니다.

```
$ aws acm-pca list-tags \
```

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

untag-certificate-authority 명령을 사용하여 프라이빗 CA에서 태그를 제거합니다.

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

AWS Private CA CA 상태 이해

사용자 작업 또는 경우에 따라 서비스 작업의 AWS Private CA 결과로 관리되는 CA의 상태입니다. 예를 들어 CA 상태가 만료되면 변경됩니다. CA 관리자가 사용할 수 있는 상태 옵션은 CA의 현재 상태에 따라 다릅니다.

AWS Private CA 는 다음 상태 값을 보고할 수 있습니다. 이 표에는 각 상태에서 사용할 수 있는 CA 기능이 나와 있습니다.

> **Note**
>
> DELETED 및 FAILED를 제외한 모든 상태 값에 대해 CA에 대한 요금이 청구됩니다.

Status	인증서 발급	OCSP로 인증서 검증	CRL 생성	감사 생성	CA 인증서를 업데이트할 수 있습니다	인증서를 해지할 수 있습니다	CA 요금이 청구됩니다
CREATING - CA가 생성되고 있습니다.	아니요	아니요	아니요	아니요	아니요	아니요	예
PENDING_CERTIFICATE - CA가 생성되었으며 작동	아니요	아니요	아니요	아니요	아니요	아니요	예

Status	인증서 발급	OCSP로 인증서 검증	CRL 생성	감사 생성	CA 인증서를 업데이트할 수 있습니다	인증서를 해지할 수 있습니다	CA 요금이 청구됩니다
하려면 인증서가 필요합니다.*							
ACTIVE	예	예	예	예	예	예	예
DISABLED - CA를 수동으로 비활성화했습니다.	아니요	예	예	예	아니요	예	예
EXPIRED - CA 인증서가 만료되었습니다.**	아니요	아니요	아니요	아니요	예	아니요	예
FAILED	CreateCertificateAuthority 작업이 실패했습니다. 이는 네트워크 중단, 백엔드 AWS 장애 또는 기타 오류로 인해 발생할 수 있습니다. 실패한 CA는 복구가 불가능합니다. CA를 삭제하고 새 CA를 생성합니다.						아니요
DELETED	CA가 복원 기간 내에 있으며, 복원 기간은 7~30일입니다. 이 기간이 지나면 영구적으로 삭제됩니다. • 상태가 DELETED이고 인증서가 만료된 CA에서 RestoreCertificateAuthority API를 호출하면 CA가 EXPIRED으로 설정됩니다. • CA를 삭제하는 방법에 대한 자세한 내용은 프라이빗 CA 삭제 단원을 참조하십시오.						아니요

활성화를 완료하려면 CSR을 생성하고, CA에서 서명된 CA 인증서를 가져오고, 인증서를 로 가져와야 합니다 AWS Private CA. CSR은 새 CA(자체 서명용)에 제출하거나 온프레미스 루트 또는 하위 CA에 제출할 수 있습니다. 자세한 내용은 CA 인증서 설치 단원을 참조하십시오.

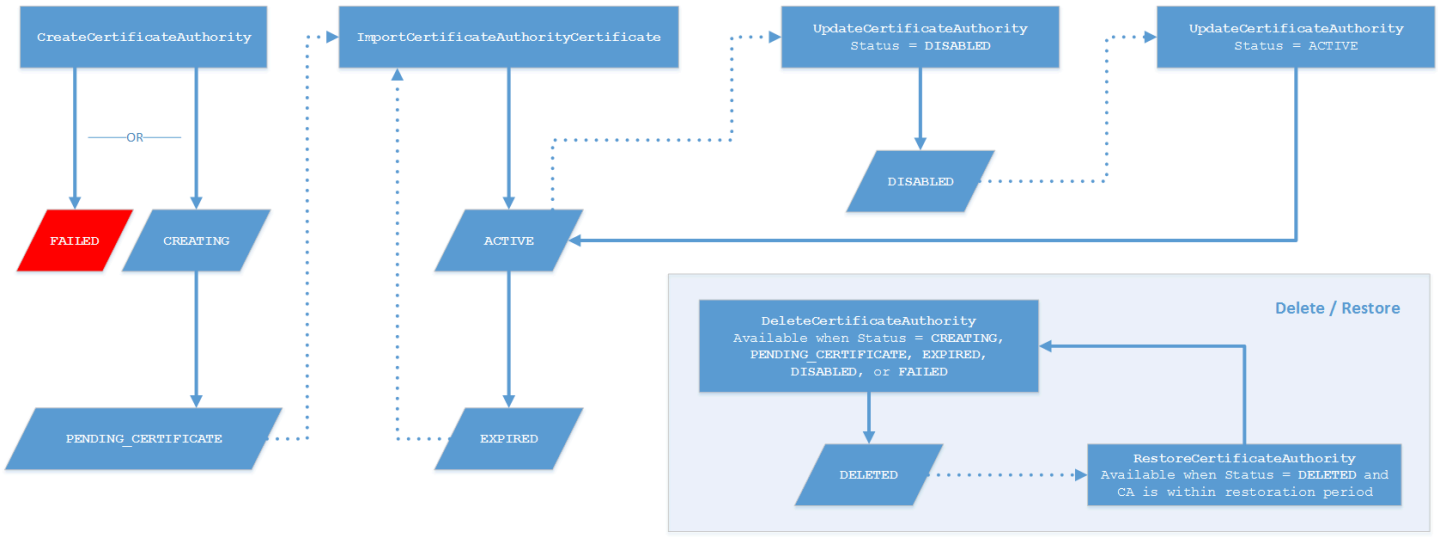
만료된 CA의 상태를 직접 변경할 수 없습니다. CA에 대한 새 인증서를 가져오는 경우는 인증서가 만료되기 DISABLED 전에 로 설정되지 ACTIVE 않은 한 상태를 로 AWS Private CA 다시 설정합니다.

만료된 CA 인증서에 대한 추가 고려 사항:

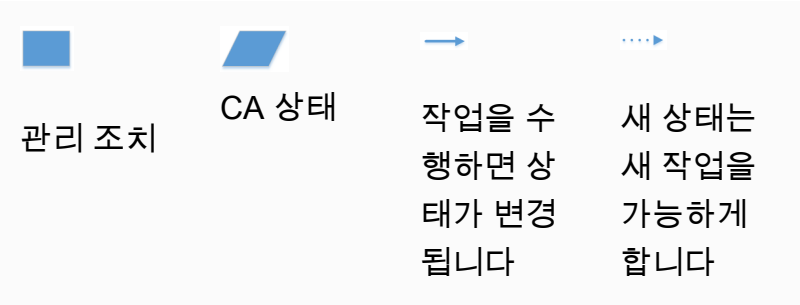
- CA 인증서는 자동 갱신되지 않습니다. 를 통한 갱신 자동화에 대한 자세한 내용은 섹션을 AWS Certificate Manager참조하세요ACM에 인증서 갱신 권한 할당.
- 만료된 CA로 새 인증서를 발급하려고 하면 IssueCertificate API가 InvalidStateException를 반환합니다. 만료된 루트 CA는 새 하위 인증서를 발급하기 앞서 새 루트 CA 인증서에 자체 서명해야 합니다.
- The ListCertificateAuthorities 및 DescribeCertificateAuthority API는 CA 상태가 ACTIVE 또는 DISABLED로 설정되어 있는지 여부에 관계없이 CA 인증서가 만료된 경우에 EXPIRED 상태를 반환합니다. 그러나 만료된 CA가 DELETED로 설정된 경우 반환된 상태는 DELETED입니다.
- UpdateCertificateAuthority API는 만료된 CA의 상태를 업데이트할 수 없습니다.
- RevokeCertificate API를 사용하여 CA 인증서를 포함하여 만료된 모든 인증서를 해지할 수 없습니다.

CA 상태와 CA 수명 주기 간의 관계

다음 다이어그램에서는 CA 상태와 관리 작업의 상호 작용으로 CA 수명 주기를 보여 줍니다.



다이어그램 키



다이어그램 상단의 관리 작업은 AWS Private CA 콘솔, CLI 또는 API를 통해 적용됩니다. 이러한 관리 작업에서는 CA를 생성, 활성화, 만료 및 갱신합니다. CA 상태는 수동 작업 또는 자동 업데이트에 대한 응답으로 변경됩니다(실선으로 표시됨). 대부분의 경우 새 상태는 CA 관리자가 적용할 수 있는 새로운 가능 작업(점선으로 표시)으로 이어집니다. 오른쪽 아래의 삽입 그림에는 삭제 및 복원 작업을 허용하는 가능한 상태 값이 표시됩니다.

에서 프라이빗 CA 업데이트 AWS Private Certificate Authority

프라이빗 CA의 상태를 업데이트하거나 CA를 생성한 후 해지 구성을 변경할 수 있습니다. 이 항목에서는 CA에 대한 콘솔 및 CLI 업데이트의 예제와 함께 CA 상태 및 CA 수명 주기에 대한 세부 정보를 제공합니다.

CA 업데이트(콘솔)

다음 절차는 AWS Management Console를 사용하여 기존 CA 구성을 업데이트하는 방법을 보여줍니다.

CA 상태 업데이트(콘솔)

이 예제에서는 활성화된 CA의 상태가 비활성화로 변경됩니다.

CA의 상태를 업데이트하는 방법

1. AWS 계정에 로그인하고 https://console.aws.amazon.com/acm-pca/home AWS Private CA 콘솔을 엽니다.
2. 프라이빗 인증 기관 페이지의 목록에서 현재 활성화된 프라이빗 CA를 선택합니다.
3. 작업 메뉴에서 비활성화를 선택하여 프라이빗 CA를 비활성화합니다.

CA의 해지 구성 업데이트(콘솔)

예를 들어 OCSP 또는 CRL 지원을 추가 또는 제거하거나 설정을 수정하여 사설 CA의 해지 구성을 업데이트할 수 있습니다.

> **Note**
>
> CA의 해지 구성을 변경해도 이미 발급된 인증서에는 영향을 주지 않습니다. 관리형 해지가 제대로 작동하려면 이전 인증서를 다시 발급해야 합니다.

다음과 같은 설정을 변경할 수 있습니다.

- OCSP 용 및 사용 중지
- 사용자 지정 OCSP 정규화된 도메인 이름(FQDN)을 활성화하거나 비활성화합니다.
- FQDN을 변경합니다.

CRL의 경우 다음 설정 중 하나를 선택할 수 있습니다.

- CRL 유형(전체 또는 분할됨)
- 사설 CA가 CRL을 생성하는지 여부
- CRL이 만료되기까지 남은 일수. 는 지정한 일수의 1/2로 CRL을 재생성하기 AWS Private CA 시작합니다.
- CRL이 저장되는 Amazon S3 버킷의 이름입니다.
- Amazon S3 버킷의 이름을 공개적으로 볼 수 없도록 숨기는 별칭입니다.

> **Important**
>
> 위의 파라미터를 변경하면 부정적인 영향을 미칠 수 있습니다. CRL 생성 비활성화, 유효 기간 변경, 사설 CA를 프로덕션 환경에 배치한 후 S3 버킷 변경 등을 예제로 들 수 있습니다. 이러한 변경으로 인해 CRL 및 현재 CRL 구성에 의존하는 기존 인증서가 손상될 수 있습니다. 이전 별칭이 올바른 버킷에 연결되어 있는 한 별칭을 안전하게 변경할 수 있습니다.

해지 설정을 업데이트하는 방법

1. AWS 계정에 로그인하고 https://console.aws.amazon.com/acm-pca/home AWS Private CA 콘솔을 엽니다.
2. 프라이빗 인증 기관 페이지의 목록에서 프라이빗 CA를 선택합니다. 그러면 CA의 세부 정보 패널이 열립니다.
3. 해지 구성 탭을 선택한 다음 편집을 선택합니다.
4. 인증서 해지 옵션에는 두 가지 옵션이 표시됩니다.

   - CRL 배포 활성화
   - OCSP 활성화

   CA에 대해 이러한 해지 옵션을 둘 중 하나 또는 둘 다 구성하지 않을 수 있습니다. 선택 사항이긴 하지만 관리형 해지가 모범 사례로 권장됩니다. 이 단계를 완료하기 전에 각 방법의 장점, 필요할 수 있는 예비 설정 및 추가 해지 기능에 대한 정보는 AWS Private CA 인증서 해지 방법 계획 섹션을 참조하세요.

CRL을 구성하는 방법

1. CRL 배포 활성화를 선택합니다.
2. CRL 항목을 위한 Amazon S3 버킷을 만들려면 새 S3 버킷 생성을 선택합니다. 고유한 버킷 이름을 제공합니다. (버킷에 대한 경로를 포함할 필요는 없습니다.) 그렇지 않으면 이 옵션을 선택하지 않고 S3 버킷 이름 목록에서 기존 버킷을 선택합니다.

   새 버킷을 생성하는 경우는 필요한 액세스 정책을 AWS Private CA 생성하고 연결합니다. 기존 버킷을 사용하기로 결정한 경우 먼저 액세스 정책을 추가해야 CRL 생성을 시작할 수 있습니다. Amazon S3의 CRL에 대한 액세스 정책에 설명된 정책 패턴 중 하나를 사용합니다. 정책 연결에 대한 자세한 내용은 Amazon S3 콘솔을 사용하여 버킷 정책 추가를 참조하세요.

   > **Note**
   >
   > AWS Private CA 콘솔을 사용하는 경우 다음 두 조건이 모두 적용되는 경우 CA 생성 시도가 실패합니다.
   >
   > - Amazon S3 버킷 또는 계정에 퍼블릭 액세스 차단 설정을 적용하고 있습니다.
   > - Amazon S3 버킷을 자동으로 생성 AWS Private CA 하도록 요청했습니다.

이 경우 콘솔은 기본적으로 공개적으로 액세스할 수 있는 버킷을 만들려고 시도하지만 Amazon S3는 이 작업을 거부합니다. 이 경우 Amazon S3 설정을 확인하합니다. 자세한 내용은 Amazon S3 스토리지에 대한 퍼블릭 액세스 차단을 참조하세요.

3. 추가 구성 옵션을 보려면 고급을 확장합니다.

   - 파티셔닝 활성화를 선택하여 CRLs의 파티셔닝을 활성화합니다. 파티셔닝을 활성화하지 않으면 CA에 AWS Private Certificate Authority 할당량에 표시된 취소된 인증서의 최대 수가 적용됩니다. 분할된 CRLs. ???
   - 사용자 지정 CRL 이름을 추가하여 Amazon S3 버킷에 대한 별칭을 생성합니다. 이 이름은 RFC 5280에서 정의한 “CRL 배포 지점” 확장에 따라 CA에서 발급한 인증서에 포함되어 있습니다. IPv6를 통해 CRLs 사용하려면 IPv6를 통해 CRLs 사용에 설명된 대로 버킷의 듀얼 스택 S3 엔드포인트로 설정합니다.
   - 사용자 지정 경로를 추가하여 Amazon S3 버킷의 파일 경로에 대한 DNS 별칭을 생성합니다.
   - CRL이 유효하게 유지되는 일수를 입력합니다. 기본값은 7일입니다. 온라인 CRL의 경우, 유효 기간은 일반적으로 2~7일입니다. AWS Private CA 에서는 지정된 기간의 1/2 시점에 CRL 재생성을 시도합니다.

4. 작업을 마쳤으면 변경 사항 저장을 선택합니다.

OCSP를 구성하는 방법

1. 인증서 해지 페이지에서 OCSP 활성화를 선택합니다.
2. (옵션) 사용자 지정 OCSP 엔드포인트 필드에 OCSP 엔드포인트에 대한 정규화된 도메인 이름(FQDN)을 제공합니다. IPv6를 통해 OCSP를 사용하려면 IPv6를 통해 OCSP 사용에 설명된 대로 이 필드를 듀얼 스택 엔드포인트로 설정합니다.

   이 필드에 FQDN을 제공하면는 AWS OCSP 응답기의 기본 URL 대신 발급된 각 인증서의 권한 정보 액세스 확장에 FQDN을 AWS Private CA 삽입합니다. 엔드포인트는 사용자 지정 FQDN이 포함된 인증서를 받으면 해당 주소를 쿼리하여 OCSP 응답을 요청합니다. 이 메커니즘이 작동하려면 다음 두 가지 추가 조치를 취해야 합니다.

   - 프록시 서버를 사용하여 사용자 지정 FQDN에 도착하는 트래픽을 AWS OCSP 응답기로 전달합니다.
   - 해당 CNAME 레코드를 DNS 데이터베이스에 추가합니다.

> **Tip**
>
> 사용자 지정 CNAME을 사용하여 완전한 OCSP 솔루션을 구현하는 방법에 대한 자세한 내용은 에 대한 OCSP URL 사용자 지정 AWS Private CA 섹션을 참조하세요.

예를 들어, 다음은 Amazon Route 53에 표시되는 사용자 지정 OCSP에 대한 CNAME 레코드입니다.

레코드 이름	Type	라우팅 정책	차별화 요소	값/트래픽 라우팅 대상
alternative.example.com	CNAME	간편함	-	proxy.example.com

> **Note**
>
> CNAME 값에 'http://' 또는 'https://'와 같은 프로토콜 접두사가 포함되면 안 됩니다.

3. 작업을 마쳤으면 변경 사항 저장을 선택합니다.

CA(CLI) 업데이트

다음 절차는 AWS CLI를 사용하여 기존 CA의 상태 및 해지 구성을 업데이트하는 방법을 보여줍니다.

> **Note**
>
> CA의 해지 구성을 변경해도 이미 발급된 인증서에는 영향을 주지 않습니다. 관리형 해지가 제대로 작동하려면 이전 인증서를 다시 발급해야 합니다.

프라이빗 CA의 상태를 업데이트하는 방법(AWS CLI)

update-certificate-authority 명령을 사용합니다.

이는 상태 DISABLED를 ACTIVE로 설정하려는 기존 CA가 있을 때 유용합니다. 시작하려면 다음 명령을 사용하여 CA의 초기 상태를 확인합니다.

```
$ 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
```

결과로 다음과 유사한 출력이 반환됩니다.

```
{
    "CertificateAuthority": {
        "Arn": "arn:aws:acm-pca:us-east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566",
        "CreatedAt": "2021-03-05T14:24:12.867000-08:00",
        "LastStateChangeAt": "2021-03-08T13:17:40.221000-08:00",
        "Type": "ROOT",
        "Serial": "serial_number",
        "Status": "DISABLED",
        "NotBefore": "2021-03-08T07:46:27-08:00",
        "NotAfter": "2022-03-08T08:46:27-08:00",
        "CertificateAuthorityConfiguration": {
            "KeyAlgorithm": "RSA_2048",
            "SigningAlgorithm": "SHA256WITHRSA",
            "Subject": {
                "Country": "US",
                "Organization": "Example Corp",
                "OrganizationalUnit": "Sales",
                "State": "WA",
                "CommonName": "www.example.com",
                "Locality": "Seattle"
            }
        },
        "RevocationConfiguration": {
            "CrlConfiguration": {
                "Enabled": true,
                "ExpirationInDays": 7,
                "CustomCname": "alternative.example.com",
                "S3BucketName": "amzn-s3-demo-bucket"
   },
            "OcspConfiguration": {
                "Enabled": false
            }
```

```
        }
    }
}
```

다음 명령은 프라이빗 CA의 상태를 ACTIVE로 설정합니다. 이는 CA에 유효한 인증서가 설치된 경우에만 가능합니다.

```
$ 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"
```

CA의 새 상태를 검사합니다.

```
$ 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
```

이제 상태가 ACTIVE로 표시됩니다.

```
{
    "CertificateAuthority": {
        "Arn": "arn:aws:acm-pca:us-east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566",
        "CreatedAt": "2021-03-05T14:24:12.867000-08:00",
        "LastStateChangeAt": "2021-03-08T13:23:09.352000-08:00",
        "Type": "ROOT",
        "Serial": "serial_number",
        "Status": "ACTIVE",
        "NotBefore": "2021-03-08T07:46:27-08:00",
        "NotAfter": "2022-03-08T08:46:27-08:00",
        "CertificateAuthorityConfiguration": {
            "KeyAlgorithm": "RSA_2048",
            "SigningAlgorithm": "SHA256WITHRSA",
            "Subject": {
                "Country": "US",
                "Organization": "Example Corp",
                "OrganizationalUnit": "Sales",
                "State": "WA",
                "CommonName": "www.example.com",
                "Locality": "Seattle"
```

```
            }
        },
        "RevocationConfiguration": {
            "CrlConfiguration": {
                "Enabled": true,
                "ExpirationInDays": 7,
                "CustomCname": "alternative.example.com",
                "S3BucketName": "amzn-s3-demo-bucket"
            },
            "OcspConfiguration": {
                "Enabled": false
            }
        }
    }
}
```

해지 메커니즘이 구성되어 있지 않은 활성 CA가 있기도 합니다. 인증서 취소 목록(CRL) 을 사용하기 시작하려면 다음 절차를 사용합니다.

기존 CA에 CRL을 추가하는 방법(AWS CLI)

1. CA의 현재 상태를 검사하려면 다음 명령을 사용합니다.

```
$ 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
```

출력은 CA가 상태 ACTIVE는 있지만 CRL을 사용하도록 구성되지 않았음을 확인합니다.

```
{
    "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 구성 매개 변수를 정의하는 `revoke_config.txt`과 같은 이름을 가진 파일을 만들고 저장합니다.

```
{
   "CrlConfiguration":{
      "Enabled": true,
      "ExpirationInDays": 7,
      "S3BucketName": "amzn-s3-demo-bucket"
   }
}
```

> **Note**
>
> CRLs을 활성화하도록 Matter 디바이스 증명 CA를 업데이트할 때 현재 Matter 표준을 준수하도록 발급된 인증서에서 CDP 확장을 생략하도록 구성해야 합니다. 이렇게 하려면 아래 그림과 같이 CRL 구성 파라미터를 정의합니다.
>
> ```
> {
>    "CrlConfiguration":{
>       "Enabled": true,
> ```

```
      "ExpirationInDays": 7,
      "S3BucketName": "amzn-s3-demo-bucket"
      "CrlDistributionPointExtensionConfiguration":{
         "OmitExtension": true
      }
   }
}
```

3. update-certificate-authority 명령과 해지 구성 파일을 사용하여 CA를 업데이트합니다.

```
$ 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. CA의 상태를 다시 검사합니다.

```
$ 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
```

출력을 통해 CA가 이제 CRL을 사용하도록 구성되었음을 확인할 수 있습니다.

```
{
    "CertificateAuthority": {
        "Arn": "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
        "CreatedAt": "2021-03-08T14:36:26.449000-08:00",
        "LastStateChangeAt": "2021-03-08T14:50:52.224000-08:00",
        "Type": "ROOT",
        "Serial": "serial_number",
        "Status": "ACTIVE",
        "NotBefore": "2021-03-08T13:46:50-08:00",
        "NotAfter": "2022-03-08T14:46:50-08:00",
        "CertificateAuthorityConfiguration": {
            "KeyAlgorithm": "RSA_2048",
            "SigningAlgorithm": "SHA256WITHRSA",
            "Subject": {
                "Country": "US",
                "Organization": "Example Corp",
                "OrganizationalUnit": "Sales",
```

```
                "State": "WA",
                "CommonName": "www.example.com",
                "Locality": "Seattle"
            }
        },
        "RevocationConfiguration": {
            "CrlConfiguration": {
                "Enabled": true,
                "ExpirationInDays": 7,
                "S3BucketName": "amzn-s3-demo-bucket",
            },
            "OcspConfiguration": {
                "Enabled": false
            }
        }
    }
}
```

이전 절차와 같이 CRL을 활성화하는 대신 OCSP 취소 지원을 추가해야 하는 경우도 있습니다. 이 경우 다음 단계를 사용합니다.

기존 CA에 OCSP 지원을 추가하는 방법(AWS CLI)

1. OCSP 파라미터를 정의하는 `revoke_config.txt`과 같은 이름을 사용하여 파일을 만들고 저장합니다.

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

2. update-certificate-authority 명령과 해지 구성 파일을 사용하여 CA를 업데이트합니다.

```
$ 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. CA의 상태를 다시 검사합니다.

```
$ 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
```

출력은 CA가 이제 OCSP를 사용하도록 구성되어 있음을 확인합니다.

```
{
    "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**
>
> CA에서 CRL 및 OCSP 지원을 모두 구성할 수도 있습니다.

프라이빗 CA 삭제

AWS Management Console 또는에서 프라이빗 CA를 AWS CLI 영구적으로 삭제할 수 있습니다. 예를 들어 새로운 프라이빗 키를 가진 신규 CA로 이를 대체하기 위해 삭제를 원할 수 있습니다. CA를 안전하게 삭제하려면 다음과 같이 하십시오.

1. 대체 CA를 생성합니다.
2. 새 프라이빗 CA를 운영 상태로 전환한 후에는 이전 프라이빗 CA를 비활성화하되 즉시 삭제하지는 마십시오.
3. 기존 CA에서 발급한 인증서가 모두 만료될 때까지 이전 CA를 비활성화된 상태로 유지하세요.
4. 이전 CA를 삭제합니다.

AWS Private CA 는 삭제 요청을 처리하기 전에 발급된 모든 인증서가 만료되었는지 확인하지 않습니다. 감사 보고서를 생성하여 어떤 인증서가 만료되었는지 확인할 수 있습니다. CA가 비활성화되어 있는 동안에는 인증서를 해지할 수 있지만 새 인증서를 발급할 수는 없습니다.

발급한 인증서가 모두 만료되기 전에 프라이빗 CA를 삭제해야 하는 경우 CA 인증서도 해지하는 것이 좋습니다. CA 인증서는 상위 CA의 CRL에 나열되며 클라이언트는 프라이빗 CA를 신뢰하지 않게 됩니다.

> **Important**
>
> `PENDING_CERTIFICATE`, `CREATING`, `EXPIRED`, `DISABLED` 또는 `FAILED` 상태인 경우 사설 CA를 삭제할 수 있습니다. `ACTIVE` 상태인 CA를 삭제하려면 먼저 비활성화를 해야 합니다. 그렇지 않으면 삭제 요청으로 인해 예외가 발생합니다. `PENDING_CERTIFICATE` 또는 `DISABLED` 상태의 프라이빗 CA를 삭제하는 경우 복원 기간을 7~30일로 설정할 수 있으며 기본값은 30일입니다. 이 기간 동안 상태는 `DELETED`로 설정되고 CA를 복원할 수 있습니다. `CREATING` 또는 `FAILED` 상태에 있는 동안 삭제된 프라이빗 CA에는 복원 기간이 할당되어 있지 않으므로 복원할 수 없습니다. 자세한 내용은 프라이빗 CA 복원 단원을 참조하십시오.

삭제한 프라이빗 CA에 대해서는 요금이 부과되지 않습니다. 그러나 프라이빗 CA를 복원하면 삭제부터 복원까지의 기간에 대한 요금이 청구됩니다. 자세한 내용은 에 대한 요금 AWS Private Certificate Authority 단원을 참조하십시오.

사설 CA를 삭제하는 방법(콘솔)

1. AWS 계정에 로그인하고 https://console.aws.amazon.com/acm-pca/home AWS Private CA 콘솔을 엽니다.
2. 프라이빗 인증 기관 페이지의 목록에서 프라이빗 CA를 선택합니다.
3. CA가 ACTIVE 상태인 경우 먼저 CA를 사용하지 않도록 설정해야 합니다. 작업 메뉴에서 비활성화를 선택합니다. 메시지가 표시되면 위험을 이해하고 계속합니다를 선택합니다.
4. ACTIVE 상태가 아닌 CA의 경우 작업, 삭제를 선택합니다.
5. CA가 DISABLED, EXPIRED, 또는 PENDING_CERTIFICATE 상태인 경우 CA 삭제 페이지에서 복원 기간을 7~30일로 지정할 수 있습니다. 프라이빗 CA가 이러한 상태 중 하나가 아닌 경우 나중에 복원할 수 없으며 삭제는 영구적입니다.
6. 삭제를 선택합니다.
7. 프라이빗 CA를 삭제하려는 것이 확실하다면 메시지가 표시되면 영구 삭제를 선택합니다. 프라이빗 CA의 상태가 DELETED로 변경됩니다. 하지만 복원 기간이 끝나기 전에 프라이빗 CA를 복원할 수 있습니다. DELETED 상태의 프라이빗 CA 복원 기간을 확인하려면 DescribeCerticateAuthority 또는 ListCertificateAuthorities API 작업을 호출합니다.

사설 CA를 삭제하는 방법(AWS CLI)

delete-certificate-authority 명령을 사용하여 프라이빗 CA를 삭제합니다.

```
$ 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
```

프라이빗 CA 복원

삭제 시 지정한 복원 기간 내에 CA가 남아 있는 한 삭제된 프라이빗 CA를 복원할 수 있습니다. 복원 기간은 7-30일입니다. 이 기간이 끝나면 프라이빗 CA는 영구적으로 삭제됩니다. 자세한 내용은 프라이빗 CA 삭제 단원을 참조하십시오. 영구적으로 삭제된 프라이빗 CA는 복원할 수 없습니다.

> **Note**
>
> 삭제한 프라이빗 CA에 대해서는 요금이 부과되지 않습니다. 그러나 프라이빗 CA를 복원하면 삭제부터 복원까지의 기간에 대한 요금이 청구됩니다. 자세한 내용은 에 대한 요금 AWS Private Certificate Authority 단원을 참조하십시오.

프라이빗 CA 복원(콘솔)

를 사용하여 프라이빗 CA를 복원 AWS Management Console 할 수 있습니다.

프라이빗 CA를 복원하는 방법(콘솔)

1. AWS 계정에 로그인하고 https://console.aws.amazon.com/acm-pca/home AWS Private CA 콘솔을 엽니다.
2. 프라이빗 인증 기관 페이지의 목록에서 삭제한 프라이빗 CA를 선택합니다.
3. 작업 메뉴에서 복원을 선택합니다.
4. CA 복원 페이지에서 복원을 다시 선택합니다.
5. 성공하면 프라이빗 CA의 상태가 삭제 전 상태로 설정됩니다. 작업, 활성화 및 활성화를 다시 선택하여 상태를 `ACTIVE`로 변경합니다. 사설 CA가 삭제 당시 `PENDING_CERTIFICATE` 상태였던 경우에는 활성화에 앞서 먼저 CA 인증서를 사설 CA로 가져와야 합니다.

프라이빗 CA 복원(AWS CLI)

restore-certificate-authority 명령을 사용하여 `DELETED` 상태에 있는 삭제된 프라이빗 CA를 복원할 수 있습니다. 다음 단계에서는 프라이빗 CA를 삭제, 복원한 다음 다시 활성화하는 데 필요한 전체 프로세스에 대해 설명합니다.

프라이빗 CA를 삭제, 복원 및 다시 활성화하는 방법(AWS CLI)

1. 프라이빗 CA를 삭제합니다.

   delete-certificate-authority 명령을 실행하여 프라이빗 CA를 삭제합니다. 프라이빗 CA의 상태가 `DISABLED` 또는 `PENDING_CERTIFICATE` 인 경우 `--permanent-deletion-time-in-days` 파라미터를 설정하여 프라이빗 CA의 복원 기간을 7~30일로 지정할 수 있습니다. 복원 기간을 지정하지 않으면 기본값은 30일입니다. 성공할 경우 이 명령은 프라이빗 CA의 상태를 `DELETED`로 설정합니다.

> **Note**
>
> 복원할 수 있으려면 삭제 시점의 프라이빗 CA 상태가 DISABLED 또는 PENDING_CERTIFICATE 상태여야 합니다.

```
$ 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. 프라이빗 CA를 복원합니다.

   restore-certificate-authority 명령을 실행하여 프라이빗 CA를 복원합니다. delete-certificate-authority 명령으로 설정한 복원 기간이 만료되기 전에 명령을 실행해야 합니다. 이 명령이 제대로 실행되면 프라이빗 CA 상태를 삭제 전 상태로 설정합니다.

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

3. 프라이빗 ACTIVE CA를 만듭니다.

   update-certificate-authority 명령을 실행하여 프라이빗 CA의 상태를 ACTIVE로 변경합니다.

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

외부에서 서명된 사설 CA 인증서 사용

프라이빗 CA 계층 구조의 신뢰 루트가 외부의 CA여야 하는 경우 자체 루트 CA를 생성하고 자체 서명 AWS Private CA할 수 있습니다. 또는 조직에서 운영하는 외부 사설 CA에서 서명한 사설 CA 인증서를 얻을 수 있습니다. 소스가 무엇이든이 외부에서 얻은 CA를 사용하여가 AWS Private CA 관리하는 프라이빗 하위 CA 인증서에 서명할 수 있습니다.

> **Note**
>
> 외부 신뢰 서비스 공급자를 생성하거나 얻는 절차는 이 설명서에서는 다루지 않습니다.

에서 외부 상위 CA AWS Private CA 를 사용하면 RFC 5280의 이름 제약 조건에 정의된 대로 CA 이름 제약 조건을 적용할 수 있습니다. 이름 제약 조건을 사용하면 CA 관리자가 인증서에서 보안 주체 이름을 제한할 수 있습니다.

외부 CA를 사용하여 사설 하위 CA 인증서에 서명하려는 경우 AWS Private CA에서 작동하는 CA를 갖기 전에 완료해야 할 세 가지 작업이 있습니다.

1. 인증서 서명 요청(CSR)을 생성합니다.
2. CSR을 외부 서명 기관에 제출하고 서명된 인증서 및 인증서 체인과 함께 반환합니다.
3. 서명된 인증서를에 설치합니다 AWS Private CA.

다음 절차에서는 AWS Management Console 또는 AWS CLI를 사용하여 이러한 작업을 완료하는 방법을 설명합니다.

외부에서 서명된 CA 인증서를 가져와 설치하는 방법(콘솔)

1. (옵션) CA의 세부 정보 페이지에 아직 접속하지 않은 경우 https://console.aws.amazon.com/acm-pca/home에서 AWS Private CA 콘솔을 엽니다. 프라이빗 인증 기관 페이지에서 인증서 보류 중, 활성, 비활성화 또는 만료됨 상태의 하위 CA를 선택합니다.
2. 작업, CA 인증서 설치를 선택하여 하위 CA 인증서 설치 페이지를 엽니다.
3. 하위 CA 인증서 설치 페이지의 CA 유형 선택에서 외부 프라이빗 CA를 선택합니다.
4. 이 CA의 CSR에서 콘솔에는 CSR의 Base64로 인코딩된 ASCII 텍스트가 표시됩니다. 복사 버튼을 사용하여 텍스트를 복사하거나 CSR을 파일로 내보내기를 선택하여 로컬에 저장할 수 있습니다.

   > **Note**
   >
   > 복사 및 붙여넣을 때 CSR 텍스트의 정확한 형식을 보존해야 합니다.

5. 오프라인 단계를 즉시 수행하여 외부 서명 기관으로부터 서명된 인증서를 받을 수 없는 경우, 서명된 인증서와 인증서 체인을 보유하고 나면 페이지를 닫고 해당 페이지로 돌아갈 수 있습니다.

   그렇지 않으면 준비가 되었으면 다음 중 하나를 수행합니다.

- 인증서 본문과 인증서 체인의 Base64로 인코딩된 ASCII 텍스트를 해당 텍스트 상자에 붙여넣습니다.
- 업로드를 선택하여 로컬 파일의 인증서 본문과 인증서 체인을 해당 텍스트 상자로 로드합니다.

6. 확인 및 설치를 선택합니다.

외부에서 서명된 CA 인증서(CLI)를 가져와 설치하는 방법

1. get-certificate-authority-csr 명령을 사용하여 프라이빗 CA에 대한 인증서 서명 요청(CSR)을 검색할 수 있습니다. CSR을 디스플레이로 보내려면 `--output text` 옵션을 사용하여 각 줄 끝에서 CR/LF 문자를 제거합니다. CSR을 파일로 보내려면 리디렉션 옵션(>) 다음에 파일 이름을 입력합니다.

```
$ 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
```

CSR을 로컬 파일로 저장한 후 다음 OpenSSL 명령을 사용하여 CSR을 검사할 수 있습니다.

```
openssl req -in path_to_CSR_file -text -noout
```

이 명령은 다음과 비슷한 출력을 생성합니다. CA 확장은 TRUE이고, 이는 CSR이 CA 인증서에 대한 것임을 나타냅니다.

```
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. CSR을 외부 서명 기관에 제출하고 Base64 PEM으로 인코딩된 서명된 인증서 및 인증서 체인이 포함된 파일을 가져오십시오.

3. import-certificate-authority-certificate 명령을 사용하여 사설 CA 인증서 파일과 체인 파일을 AWS Private CA로 가져옵니다.

```
$ 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
```

에서 인증서 발급 및 관리 AWS Private CA

프라이빗 인증 기관(CA)을 생성 및 활성화하고 이에 대한 액세스를 구성한 후 사용자 또는 권한 있는 사용자가 인증서를 발급하고 관리할 수 있습니다. CA에 대한 AWS Identity and Access Management (IAM) 정책을 아직 설정하지 않은 경우이 가이드의 Identity and Access Management 섹션에서 정책을 구성하는 방법에 대해 자세히 알아볼 수 있습니다. 단일 계정 및 교차 계정 시나리오에서 CA 액세스를 구성하는 방법에 대한 자세한 내용은 프라이빗 CA에 대한 액세스 제어 섹션을 참조하세요.

주제

- 프라이빗 최종 엔터티 인증서 발급
- 프라이빗 인증서 검색
- 프라이빗 인증서 나열
- 프라이빗 인증서 및 해당 보안 키 내보내기
- 프라이빗 인증서 취소
- 갱신된 인증서 내보내기 자동화
- AWS Private CA 인증서 템플릿 사용

프라이빗 최종 엔터티 인증서 발급

프라이빗 CA를 사용하면 AWS Certificate Manager (ACM) 또는에서 프라이빗 최종 엔터티 인증서를 요청할 수 있습니다 AWS Private CA. 두 가지 서비스의 기능은 다음 표에서 비교가 되어 있습니다.

기능	ACM	AWS Private CA
최종 엔터티 인증서 발급	✓ (RequestCertificate 또는 콘솔 사용)	✓ (IssueCertificate 사용)
로드 밸런서 및 인터넷 연결 AWS 서비스와의 연결	✓	지원되지 않음
관리형 인증서 갱신	✓	ACM을 통해 간접적으로 지원됩니다.
콘솔 지원	✓	지원되지 않음

기능	ACM	AWS Private CA
API 지원	✓	✓
CLI 지원	✓	✓

는 인증서를 AWS Private CA 생성할 때 인증서 유형과 경로 길이를 지정하는 템플릿을 따릅니다. 인증서를 생성하는 API 또는 CLI 문에 템플릿 ARN이 제공되지 않으면 EndEntityCertificate/V1 템플릿이 기본적으로 적용됩니다. 사용 가능한 인증서 템플릿에 대한 자세한 내용은 AWS Private CA 인증서 템플릿 사용 단원을 참조하십시오 .

ACM 인증서는 퍼블릭 신뢰를 중심으로 설계되었지만는 프라이빗 PKI의 요구 사항을 AWS Private CA 충족합니다. 따라서 ACM에서 허용하지 않는 방식으로 AWS Private CA API 및 CLI를 사용하여 인증서를 구성할 수 있습니다. 여기에는 다음이 포함됩니다.

- 보안 주체 이름으로 인증서를 생성합니다.
- 지원되는 프라이빗 키 알고리즘 및 키 길이 사용
- 지원되는 서명 알고리즘 사용.
- 프라이빗 CA 및 프라이빗 인증서의 유효 기간 지정

를 사용하여 프라이빗 TLS 인증서를 생성한 후 ACM으로 가져와 지원되는 AWS 서비스와 함께 사용할 AWS Private CA수 있습니다.

> **Note**
>
> 아래 절차, issue-certificate 명령 사용 또는 IssueCertificate API 작업을 사용하여 만든 인증서는 AWS외부에서 사용하기 위해 직접 내보낼 수 없습니다. 하지만 프라이빗 CA를 사용하여 ACM을 통해 발급된 인증서에 서명할 수 있으며, 이러한 인증서를 해당 비밀 키와 함께 내보낼 수 있습니다. 자세한 내용은 ACM 사용 설명서의 프라이빗 인증서 요청 및 프라이빗 인증서 내보내기를 참조하세요.

표준 인증서 발급(AWS CLI)

AWS Private CA CLI 명령 issue-certificate 또는 API 작업 IssueCertificate를 사용하여 최종 엔터티 인증서를 요청할 수 있습니다. 이 명령을 사용하려면 인증서를 발급하는 데 사용할 프라이빗 CA의

Amazon 리소스 이름(ARN)이 필요합니다. 또한 OpenSSL과 같은 프로그램을 사용하여 인증서 서명 요청(CSR)을 생성해야 합니다.

AWS Private CA API 또는를 사용하여 프라이빗 인증서를 AWS CLI 발급하는 경우 인증서는 관리되지 않습니다. 즉, ACM 콘솔, ACM CLI 또는 ACM API를 사용하여 인증서를 보거나 내보낼 수 없으며 인증서가 자동으로 갱신되지 않습니다. 하지만 PCA get-certificate 명령을 사용하여 인증서 세부 정보를 검색할 수 있으며 CA를 소유한 경우 감사 보고서를 생성할 수 있습니다.

인증서 생성 시 고려 사항

- RFC 5280에 따라, 제공하는 도메인 이름(일반 이름)의 길이는 마침표를 포함하여 64 옥텟(자)을 초과할 수 없습니다. 더 긴 도메인 이름을 추가하려면 최대 253옥텟 길이의 이름을 지원하는 보안 주체 대체 이름 필드에 해당 이름을 지정하십시오.
- AWS CLI 버전 1.6.3 이상을 사용하는 경우 CSRs과 같은 base64 인코딩 입력 파일을 지정할 `fileb://` 때 접두사를 사용합니다. 이렇게 AWS Private CA 하면가 데이터를 올바르게 구문 분석할 수 있습니다.

다음 OpenSSL 명령은 인증서에 대한 CSR 및 개인 키를 생성합니다.

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

다음과 같이 CSR의 콘텐츠를 검사할 수 있습니다.

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

결과 출력은 다음과 같은 축약된 예제와 비슷해야 합니다.

```
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
```

다음 명령은 인증서를 생성합니다. 템플릿이 지정되지 않았으므로 기본적으로 기본 최종 엔터티 인증서가 발급됩니다.

```
$ 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"
```

발급된 인증서의 ARN이 반환됩니다.

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

> **Note**
>
> AWS Private CA 는 issue-certificate 명령을 수신할 때 일련 번호가 있는 ARN을 즉시 반환합니다. 하지만 인증서 처리는 비동기적으로 이루어지므로 여전히 실패할 수 있습니다. 이 경우 새 ARN을 사용하는 get-certificate 명령도 실패합니다.

APIPassThrough 템플릿을 사용하여 사용자 지정 보안 주체 이름이 포함된 인증서를 발급합니다.

이 예제에서는 사용자 지정된 보안 주체 이름 요소가 포함된 인증서가 발급됩니다. 표준 인증서 발급(AWS CLI)와 같은 CSR을 제공하는 것 외에도 issue-certificate 명령에 두 개의 추가 인수, 즉 APIPassThrough 템플릿의 ARN과 사용자 지정 속성 및 해당 객체 식별자(OID)를 지정하는 JSON 구성 파일을 전달합니다. `CustomAttributes`와 함께 `StandardAttributes`를 사용할 수는 없지만 표준 OID를 `CustomAttributes`의 일부로 전달할 수는 있습니다. 기본 보안 주체 이름 OID는 다음 표에 나열되어 있습니다(RFC 4519 및 글로벌 OID 참조 데이터베이스의 정보).

보안 주체 이름	약어	객체 ID
countryName	c	2.5.4.6
commonName	cn	2.5.4.3
dnQualifier[고유 이름 한정자]		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
serialNumber		2.5.4.5
st [state]		2.5.4.8
surname	sn	2.5.4.4
제목		2.5.4.12

보안 주체 이름	약어	객체 ID
domainComponent	dc	0.9.2342.19200300.100.1.25
userid		0.9.2342.19200300.100.1.1

샘플 구성 파일 `api_passthrough_config.txt`에는 다음 코드가 들어 있습니다.

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

인증서를 발급하려면 다음 명령을 사용합니다.

```
$ 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
```

발급된 인증서의 ARN이 반환됩니다.

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

다음과 같이 로컬에서 인증서를 검색합니다.

```
$ 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
```

OpenSSL을 사용하여 인증서의 내용을 검사할 수 있습니다.

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

> **Note**
>
> 발급한 각 인증서에 사용자 지정 특성을 전달하는 프라이빗 CA를 만들 수도 있습니다.

APIPassThrough 템플릿을 사용하여 사용자 지정 확장이 포함된 인증서를 발급합니다.

이 예제에서는 사용자 지정 확장이 포함된 인증서가 발급됩니다. 이를 위해서는 APIPassThrough 템플릿의 ARN, 사용자 지정 확장을 지정하는 JSON 구성 파일, 표준 인증서 발급(AWS CLI)에 표시된 것과 같은 CSR이라는 세 가지 인수를 issue-certificate 명령에 전달해야 합니다.

샘플 구성 파일 `api_passthrough_config.txt`에는 다음 코드가 들어 있습니다.

```
{
  "Extensions": {
    "CustomExtensions": [
      {
        "ObjectIdentifier": "2.5.29.30",
        "Value": "MBWgEzARgg8ucGVybWl0dGVkLnRlc3Q=",
        "Critical": true
      }
```

```
    ]
  }
}
```

사용자 지정 인증서는 다음과 같이 발급됩니다.

```
$ 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
```

발급된 인증서의 ARN이 반환됩니다.

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

다음과 같이 로컬에서 인증서를 검색합니다.

```
$ 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
```

OpenSSL을 사용하여 인증서의 내용을 검사할 수 있습니다.

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

프라이빗 인증서 검색

AWS Private CA API 및를 사용하여 프라이빗 인증서를 발급 AWS CLI 할 수 있습니다. 이 경우 AWS CLI 또는 AWS Private CA API를 사용하여 해당 인증서를 검색할 수 있습니다. ACM을 사용하여 프라

이빗 CA를 생성하고 인증서를 요청한 경우 ACM을 사용하여 인증서와 암호화된 프라이빗 키를 내보내야 합니다. 자세한 내용은 프라이빗 인증서 내보내기를 참조하세요.

엔드 엔터티 인증서를 검색하는 방법

get-certificate AWS CLI 명령을 사용하여 프라이빗 최종 엔터티 인증서를 검색합니다. 또한 GetCertificate API 작업을 사용할 수도 있습니다. sed와 유사한 파서인 jq를 사용하여 출력 형식을 지정하는 것이 좋습니다.

> **Note**
>
> 인증서를 해지하려면 get-certificate 명령을 사용하여 16진수 형식으로 일련 번호를 검색할 수 있습니다. 감사 보고서를 생성하여 16진수 일련 번호를 검색할 수도 있습니다. 자세한 내용은 프라이빗 CA에서 감사 보고서 사용 단원을 참조하십시오.

```
$ 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'
```

이 명령은 인증서 및 인증서 체인을 다음 표준 형식으로 출력합니다.

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

CA 인증서를 가져오는 방법

AWS Private CA API 및를 사용하여 프라이빗 CA에 대한 인증 기관(CA) 인증서를 AWS CLI 검색할 수 있습니다. get-certificate-authority-certificate 명령을 실행합니다. GetCertificateAuthorityCertificate 작업을 호출할 수도 있습니다. sed와 유사한 파서인 jq를 사용하여 출력 형식을 지정하는 것이 좋습니다.

```
$ 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'
```

이 명령은 CA 인증서를 다음과 같은 표준 형식으로 출력합니다.

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

프라이빗 인증서 나열

프라이빗 인증서를 나열하려면 감사 보고서를 생성하여 해당 S3 버킷에서 검색하고 필요에 따라 보고서 내용을 분석합니다. AWS Private CA 감사 보고서 작성에 대한 자세한 내용은 프라이빗 CA에서 감사 보고서 사용 섹션을 참조하세요. S3 버킷에서 객체를 검색하는 방법에 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 객체 다운로드를 참조하세요.

다음 예제에서는 감사 보고서를 생성하고 유용한 데이터를 찾기 위해 이를 파싱하는 방법을 보여줍니다. 결과는 JSON으로 포맷되며 데이터는 sed와 유사한 파서인 jq를 사용하여 필터링됩니다.

1. 감사 보고서 생성.

다음 명령은 지정된 CA에 대한 감사 보고서를 생성합니다.

```
$ 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
```

성공할 경우 명령이 새 감사 보고서의 ID와 위치를 반환합니다.

```
{
   "AuditReportId":"audit_report_ID",
   "S3Key":"audit-report/CA_ID/audit_report_ID.json"
}
```

2. 감사 보고서를 검색하고 형식을 지정합니다.

이 명령은 감사 보고서를 검색하고, 그 내용을 표준 출력으로 표시하고, 2020-12-01 이후에 발급된 인증서만 표시하도록 결과를 필터링합니다.

```
$ 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")'
```

반환된 항목은 다음과 유사합니다.

```
{
   "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. 감사 보고서를 로컬에 저장합니다.

여러 쿼리를 수행하려는 경우 감사 보고서를 로컬 파일에 저장하는 것이 편리합니다.

```
$ aws s3api get-object \
     --region region \
     --bucket bucket_name \
     --key audit-report/CA_ID/audit_report_ID.json > my_local_audit_report.json
```

이전과 동일한 필터를 사용해도 동일한 결과가 출력됩니다.

```
$ 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. 날짜 범위 내 쿼리

다음과 같이 날짜 범위 내에 발급된 인증서를 쿼리할 수 있습니다.

```
$ cat my_local_audit_report.json | jq '.[] | select(.issuedAt >= "2020-11-01"
 and .issuedAt <= "2020-11-10")'
```

필터링된 콘텐츠는 표준 출력에 표시됩니다.

```
{
   "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. 지정된 템플릿에 따라 인증서를 검색합니다.

다음 명령은 템플릿 ARN을 사용하여 보고서 콘텐츠를 필터링합니다.

```
$ cat my_local_audit_report.json | jq '.[] | select(.templateArn == "arn:aws:acm-
pca:::template/RootCACertificate/V1")'
```

출력에는 일치하는 인증서 레코드가 표시됩니다.

```
{
   "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. 해지된 인증서를 필터링합니다.

해지된 인증서를 모두 찾으려면 다음 명령을 사용합니다.

```
$ cat my_local_audit_report.json | jq '.[] | select(.revokedAt != null)'
```

해지된 인증서는 다음과 같이 표시됩니다.

```
{
   "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. 정규식을 사용하여 필터링합니다.

다음 명령은 문자열 “leaf”가 포함된 보안 주체 이름을 검색합니다.

```
$ cat my_local_audit_report.json | jq '.[] | select(.subject|test("leaf"))'
```

일치하는 인증서 레코드는 다음과 같이 반환됩니다.

```
{
   "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"
}
```

프라이빗 인증서 및 해당 보안 키 내보내기

AWS Private CA 는 서명 및 발급한 프라이빗 인증서를 직접 내보낼 수 없습니다. 그러나 AWS Certificate Manager 를 사용하여 암호화된 보안 암호 키와 함께 이러한 인증서를 내보낼 수 있습니다. 그러면 인증서는 개인 PKI 어디에나 배포할 수 있도록 완전히 이동이 가능합니다. 자세한 내용은 AWS Certificate Manager 사용 설명서의 프라이빗 인증서 내보내기를 참조하세요.

추가 이점으로는 ACM 콘솔, ACM API의 `RequestCertificate` 작업 또는의 ACM 섹션에 있는 request-certificate 명령을 사용하여 발급된 프라이빗 인증서에 대한 관리형 갱신을 AWS Certificate Manager 제공합니다 AWS CLI. 갱신에 대한 자세한 내용은 프라이빗 PKI의 인증서 갱신을 참조하세요.

프라이빗 인증서 취소

revoke-certificate AWS CLI 명령 또는 RevokeCertificate API 작업을 사용하여 AWS Private CA 인증서를 취소할 수 있습니다. 예를 들어 비밀 키가 손상되거나 관련 도메인이 유효하지 않게 되는 경우 예정된 만료 전에 인증서를 해지해야 할 수 있습니다. 해지가 유효하려면 인증서를 사용하는 클라이언트가 보안 네트워크 연결을 구축하려고 할 때마다 해지 상태를 확인할 방법이 필요합니다.

AWS Private CA 는 해지 상태 확인을 지원하는 두 가지 완전 관리형 메커니즘인 온라인 인증서 상태 프로토콜(OCSP)과 인증서 해지 목록(CRLs 제공합니다. OCSP를 사용하면 클라이언트가 상태를 실시간으로 반환하는 신뢰할 수 있는 해지 데이터베이스를 쿼리합니다. 클라이언트는 CRL을 사용하여 인증서를 정기적으로 다운로드하고 저장하는 해지된 인증서 목록과 대조합니다. 클라이언트는 취소된 인증서를 수락하지 않습니다.

OCSP와 CRL은 모두 인증서에 내장된 검증 정보를 기반으로 합니다. 따라서 발급 CA는 발급 전에 이러한 메커니즘 중 하나 또는 둘 다를 지원하도록 구성해야 합니다. 를 통한 관리형 해지 선택 및 구현에 대한 자세한 내용은 섹션을 AWS Private CA참조하세요AWS Private CA 인증서 해지 방법 계획.

취소된 인증서는 항상 AWS Private CA 감사 보고서에 기록됩니다.

> **Note**
>
> 교차 계정 호출자의 경우 AWSRAMRevokeCertificateCertificateAuthority 권한이 있는 공유가 필요합니다. 해지 권한은에 포함되지 않습니다AWSRAMDefaultPermissionCertificateAuthority. 계정 간 발급자가 해지할 수 있도록 하려면 CA 관리자가 동일한 CA를 가리키는 RAM 공유 두 개를 만들어야 합니다.
>
> 1. AWSRAMRevokeCertificateCertificateAuthority 권한의 공유.
> 2. AWSRAMDefaultPermissionCertificateAuthority 권한의 공유.

인증서를 해지하는 방법

개인 PKI 인증서를 해지하려면 RevokeCertificate API 작업 또는 revoke-certificate 명령을 사용합니다. 일련 번호는 16진수 형식이어야 합니다. get-certificate 명령을 호출하여 이 일련 번호를 검색할 수 있습니다. revoke-certificate 명령은 응답을 반환하지 않습니다.

```
$ 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"
```

해지된 인증서 및 OCSP

인증서를 해지할 때 OCSP 응답에 새 상태가 반영되기까지 최대 60분이 걸릴 수 있습니다. 일반적으로 OCSP는 해지 정보의 빠른 배포를 지원하는 경향이 있는데, 이는 클라이언트가 며칠 동안 캐시할 수 있는 CRL과 달리 OCSP 응답은 일반적으로 클라이언트에 의해 캐시되지 않기 때문입니다.

CRL의 해지된 인증서

CRL은 일반적으로 인증서가 해지된 후 약 30분 후에 업데이트됩니다. 어떤 이유로든 CRL 업데이트가 실패하면 AWS Private CA 는 15분마다 추가로 시도합니다.

Amazon CloudWatch를 사용하면 CRLGenerated 및 MisconfiguredCRLBucket 지표에 대한 경보를 생성할 수 있습니다. 자세한 내용은 지원되는 CloudWatch 지표를 참조하십시오. CRL 생성 및 구성에 대한 자세한 내용은 에 대한 CRL 설정 AWS Private CA 섹션을 참조하세요.

다음 예제에서는 인증서 취소 목록(CRL)에 있는 취소된 인증서를 보여줍니다.

```
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
```

감사 보고서에서 해지된 인증서

해지된 인증서를 포함한 모든 인증서는 프라이빗 CA의 감사 보고서에 포함됩니다. 다음 예제에서는 발급된 인증서 하나와 해지된 인증서 하나가 포함된 감사 보고서를 보여줍니다. 자세한 내용은 프라이빗 CA에서 감사 보고서 사용 단원을 참조하십시오.

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


갱신된 인증서 내보내기 자동화

AWS Private CA 를 사용하여 CA를 생성할 때 해당 CA를 로 가져오 AWS Certificate Manager 고 ACM 이 인증서 발급 및 갱신을 관리하도록 할 수 있습니다. 갱신되는 인증서가 통합 서비스에 연결되어 있는 경우 서비스는 새 인증서를 원활하게 적용합니다. 그러나 원래 PKI 환경의 다른 곳(예: 온프레미스 서버 또는 기기)에서 사용하기 위해 인증서를 내보낸 경우에는 갱신 후 다시 내보내야 합니다.

Amazon EventBridge와 AWS Lambda를 사용하여 ACM 내보내기 프로세스를 자동화하는 샘플 솔루션은 갱신된 인증서 내보내기 자동화를 참조하세요.

AWS Private CA 인증서 템플릿 사용

AWS Private CA 는 구성 템플릿을 사용하여 CA 인증서와 최종 엔터티 인증서를 모두 발급합니다. PCA 콘솔에서 CA 인증서를 발급하면 적절한 루트 또는 하위 CA 인증서 템플릿이 자동으로 적용됩니다.

CLI 또는 API를 사용하여 인증서를 발급하는 경우 템플릿 ARN을 `IssueCertificate` 작업에 대한 파라미터로 제공할 수 있습니다. ARN을 제공하지 않으면 기본적으로 `EndEntityCertificate/V1` 템플릿이 적용됩니다. 자세한 내용은 IssueCertificate API 및 issue-certificate 명령 설명서를 참조하세요.

> **Note**
>
> AWS Certificate Manager 프라이빗 CA에 대한 교차 계정 공유 액세스 권한이 있는 (ACM) 사용자는 CA에서 서명한 관리형 인증서를 발급할 수 있습니다. 교차 계정 발급자는 리소스 기반 정책의 제약을 받으며 다음과 같은 최종 엔터티 인증서 템플릿에만 액세스할 수 있습니다.
>
> - EndEntityCertificate/V1
> - EndEntityClientAuthCertificate/V1
> - EndEntityServerAuthCertificate/V1
> - BlankEndEntityCertificate_APIPassthrough/V1
> - BlankEndEntityCertificate_APICSRPassthrough/V1
> - SubordinateCACertificate_PathLen0/V1
>
> 자세한 내용은 리소스 기반 정책 단원을 참조하십시오.

주제

- AWS Private CA 템플릿 품종
- AWS Private CA 템플릿 작업 순서
- AWS Private CA 템플릿 정의

AWS Private CA 템플릿 품종

AWS Private CA 는 네 가지 유형의 템플릿을 지원합니다.

- 기본 템플릿

  패스스루 파라미터가 허용되지 않는 사전 정의된 템플릿.

- CSRPassthrough 템플릿

  CSR 패스스루를 허용하여 해당 기본 템플릿 버전을 확장하는 템플릿. 인증서를 발급하는 데 사용되는 CSR의 확장은 발급된 인증서에 복사됩니다. CSR에 템플릿 정의와 충돌하는 확장 값이 포함된 경우 템플릿 정의가 항상 우선 순위가 더 높습니다. 우선 순위에 대한 자세한 내용은 AWS Private CA 템플릿 작업 순서 섹션을 참조하세요.

- APIPassthrough 템플릿

  API 패스스루를 허용하여 해당 기본 템플릿 버전을 확장하는 템플릿. 관리자 또는 다른 중간 시스템에 알려진 동적 값은 인증서를 요청하는 보안 주체가 알지 못할 수도 있고, 템플릿에서 정의하지 못할 수도 있고, CSR에서 사용할 수 없을 수도 있습니다. 하지만 CA 관리자는 Active Directory와 같은 다른 데이터 소스에서 추가 정보를 검색하여 요청을 완료할 수 있습니다. 예를 들어 컴퓨터가 어떤 조직 단위에 속해 있는지 모르는 경우 관리자는 Active Directory에서 정보를 조회하고 JSON 구조에 정보를 포함시켜 인증서 요청에 해당 정보를 추가할 수 있습니다.

  `IssueCertificate` 작업의 `ApiPassthrough` 파라미터 값이 발급된 인증서에 복사됩니다. `ApiPassthrough` 파라미터에 템플릿 정의와 충돌하는 정보가 포함된 경우 템플릿 정의가 항상 더 높은 우선 순위를 가집니다. 우선 순위에 대한 자세한 내용은 AWS Private CA 템플릿 작업 순서 섹션을 참조하세요.

- APICSRPassthrough 템플릿

  API와 CSR 패스스루를 모두 허용하여 해당 기본 템플릿 버전을 확장하는 템플릿. 인증서 발급에 사용된 CSR의 확장은 발급된 인증서에 복사되고 `IssueCertificate` 작업의 `ApiPassthrough` 파라미터 값도 복사됩니다. 템플릿 정의, API 패스스루 값 및 CSR 패스스루 확장이 충돌하는 경우 템플릿 정의의 우선 순위가 가장 높고, API 패스스루 값, CSR 패스스루 확장 순입니다. 우선 순위에 대한 자세한 내용은 AWS Private CA 템플릿 작업 순서 섹션을 참조하세요.

아래 표에는에서 지원하는 모든 템플릿 유형이 정의 링크 AWS Private CA 와 함께 나열되어 있습니다.

> **Note**
>
> GovCloud 리전의 템플릿 ARNs에 대한 자세한 내용은 AWS GovCloud (US) 사용 설명서AWS Private Certificate Authority의 섹션을 참조하세요.

기본 템플릿

템플릿 이름	템플릿 ARN	인증서 유형
CodeSigningCertificate/V1	arn:aws:acm-pca:::template/CodeSigningCertificate/V1	코드 서명
EndEntityCertificate/V1	arn:aws:acm-pca:::template/EndEntityCertificate/V1	최종 엔터티
EndEntityClientAuthCertificate/V1	arn:aws:acm-pca:::template/EndEntityClientAuthCertificate/V1	최종 엔터티
EndEntityServerAuthCertificate/V1	arn:aws:acm-pca:::template/EndEntityServerAuthCertificate/V1	최종 엔터티
OCSPSigningCertificate/V1	arn:aws:acm-pca:::template/OCSPSigningCertificate/V1	OCSP 서명
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

템플릿 이름	템플릿 ARN	인증서 유형
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 템플릿

템플릿 이름	템플릿 ARN	인증서 유형
BlankEndEntityCertificate_CSRPassthrough/V1	arn:aws:acm-pca:::template/BlankEndEntityCertificate_CSRPassthrough/V1	최종 엔터티
BlankEndEntityCertificate_CriticalBasicConstraints_CSRPassthrough/V1	arn:aws:acm-pca:::template/BlankEndEntityCertificate_CriticalBasicConstraints_CSRPassthrough/V1	최종 엔터티
BlankSubordinateCACertificate_PathLen0_CSRPassthrough/V1	arn:aws:acm-pca:::template/BlankSubordinateCACertificate_PathLen0_CSRPassthrough/V1	CA
BlankSubordinateCACertificate_PathLen1_CSRPassthrough/V1	arn:aws:acm-pca:::template/BlankSubordinateCACertifica	CA

템플릿 이름	템플릿 ARN	인증서 유형
	te_PathLen1_CSRPas sthrough/V1	
BlankSubordinateCACertifica te_PathLen2_CSRPassthrough/V1	arn:aws:acm-pca::: template/BlankSubo rdinateCACertifica te_PathLen2_CSRPas sthrough/V1	CA
BlankSubordinateCACertifica te_PathLen3_CSRPassthrough/V1	arn:aws:acm-pca::: template/BlankSubo rdinateCACertifica te_PathLen3_CSRPas sthrough/V1	CA
CodeSigningCertificate_CSRP assthrough/V1	arn:aws:acm-pca::: template/CodeSigni ngCertificate_CSRP assthrough/V1	코드 서명
EndEntityCertificate_CSRPas sthrough/V1	arn:aws:acm-pca::: template/EndEntity Certificate_CSRPas sthrough/V1	최종 엔터티
EndEntityClientAuthCertific ate_CSRPassthrough/V1	arn:aws:acm-pca::: template/EndEntity ClientAuthCertific ate_CSRPassthrough/V1	최종 엔터티
EndEntityServerAuthCertific ate_CSRPassthrough/V1	arn:aws:acm-pca::: template/EndEntity ServerAuthCertific ate_CSRPassthrough/V1	최종 엔터티

템플릿 이름	템플릿 ARN	인증서 유형
OCSPSigningCertificate_CSRPassthrough/V1	arn:aws:acm-pca:::template/OCSPSigningCertificate_CSRPassthrough/V1	OCSP 서명
SubordinateCACertificate_PathLen0_CSRPassthrough/V1	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen0_CSRPassthrough/V1	CA
SubordinateCACertificate_PathLen1_CSRPassthrough/V1	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen1_CSRPassthrough/V1	CA
SubordinateCACertificate_PathLen2_CSRPassthrough/V1	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen2_CSRPassthrough/V1	CA
SubordinateCACertificate_PathLen3_CSRPassthrough/V1	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen3_CSRPassthrough/V1	CA

APIPassthrough 템플릿

템플릿 이름	템플릿 ARN	인증서 유형
BlankEndEntityCertificate_APIPassthrough/V1	arn:aws:acm-pca:::template/BlankEndE	최종 엔터티

템플릿 이름	템플릿 ARN	인증서 유형
	ntityCertificate_APIPassthrough/V1	
BlankEndEntityCertificate_CriticalBasicConstraints_APIPassthrough/V1	arn:aws:acm-pca:::template/BlankEndEntityCertificate_CriticalBasicConstraints_APIPassthrough/V1	최종 엔터티
CodeSigningCertificate_APIPassthrough/V1	arn:aws:acm-pca:::template/CodeSigningCertificate_APIPassthrough/V1	코드 서명
EndEntityCertificate_APIPassthrough/V1	arn:aws:acm-pca:::template/EndEntityCertificate_APIPassthrough/V1	최종 엔터티
EndEntityClientAuthCertificate_APIPassthrough/V1	arn:aws:acm-pca:::template/EndEntityClientAuthCertificate_APIPassthrough/V1	최종 엔터티
EndEntityServerAuthCertificate_APIPassthrough/V1	arn:aws:acm-pca:::template/EndEntityServerAuthCertificate_APIPassthrough/V1	최종 엔터티
OCSPSigningCertificate_APIPassthrough/V1	arn:aws:acm-pca:::template/OCSPSigningCertificate_APIPassthrough/V1	OCSP 서명

템플릿 이름	템플릿 ARN	인증서 유형
RootCACertificate_APIPassthrough/V1	arn:aws:acm-pca:::template/RootCACertificate_APIPassthrough/V1	CA
BlankRootCACertificate_APIPassthrough/V1	arn:aws:acm-pca:::template/BlankRootCACertificate_APIPassthrough/V1	CA
BlankRootCACertificate_PathLen0_APIPassthrough/V1	arn:aws:acm-pca:::template/BlankRootCACertificate_PathLen0_APIPassthrough/V1	CA
BlankRootCACertificate_PathLen1_APIPassthrough/V1	arn:aws:acm-pca:::template/BlankRootCACertificate_PathLen1_APIPassthrough/V1	CA
BlankRootCACertificate_PathLen2_APIPassthrough/V1	arn:aws:acm-pca:::template/BlankRootCACertificate_PathLen2_APIPassthrough/V1	CA
BlankRootCACertificate_PathLen3_APIPassthrough/V1	arn:aws:acm-pca:::template/BlankRootCACertificate_PathLen3_APIPassthrough/V1	CA
SubordinateCACertificate_PathLen0_APIPassthrough/V1	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen0_APIPassthrough/V1	CA

템플릿 이름	템플릿 ARN	인증서 유형
BlankSubordinateCACertificate_PathLen0_APIPassthrough/V1	arn:aws:acm-pca:::template/BlankSubordinateCACertificate_PathLen0_APIPassthrough/V1	CA
SubordinateCACertificate_PathLen1_APIPassthrough/V1	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen1_APIPassthrough/V1	CA
BlankSubordinateCACertificate_PathLen1_APIPassthrough/V1	arn:aws:acm-pca:::template/BlankSubordinateCACertificate_PathLen1_APIPassthrough/V1	CA
SubordinateCACertificate_PathLen2_APIPassthrough/V1	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen2_APIPassthrough/V1	CA
BlankSubordinateCACertificate_PathLen2_APIPassthrough/V1	arn:aws:acm-pca:::template/BlankSubordinateCACertificate_PathLen2_APIPassthrough/V1	CA
SubordinateCACertificate_PathLen3_APIPassthrough/V1	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen3_APIPassthrough/V1	CA

템플릿 이름	템플릿 ARN	인증서 유형
BlankSubordinateCACertificate_PathLen3_APIPassthrough/V1	arn:aws:acm-pca:::template/BlankSubordinateCACertificate_PathLen3_APIPassthrough/V1	CA

APICSRPassthrough 템플릿

템플릿 이름	템플릿 ARN	인증서 유형
BlankEndEntityCertificate_APICSRPassthrough/V1	arn:aws:acm-pca:::template/BlankEndEntityCertificate_APICSRPassthrough/V1	최종 엔터티
BlankEndEntityCertificate_CriticalBasicConstraints_APICSRPassthrough/V1	arn:aws:acm-pca:::template/BlankEndEntityCertificate_CriticalBasicConstraints_APICSRPassthrough/V1	최종 엔터티
CodeSigningCertificate_APICSRPassthrough/V1	arn:aws:acm-pca:::template/CodeSigningCertificate_APICSRPassthrough/V1	코드 서명
EndEntityCertificate_APICSRPassthrough/V1	arn:aws:acm-pca:::template/EndEntityCertificate_APICSRPassthrough/V1	최종 엔터티
EndEntityClientAuthCertificate_APICSRPassthrough/V1	arn:aws:acm-pca:::template/EndEntity	최종 엔터티

템플릿 이름	템플릿 ARN	인증서 유형
	ClientAuthCertificate_APICSRPassthrough/V1	
EndEntityServerAuthCertificate_APICSRPassthrough/V1	arn:aws:acm-pca:::template/EndEntityServerAuthCertificate_APICSRPassthrough/V1	최종 엔터티
OCSPSigningCertificate_APICSRPassthrough/V1	arn:aws:acm-pca:::template/OCSPSigningCertificate_APICSRPassthrough/V1	OCSP 서명
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

템플릿 이름	템플릿 ARN	인증서 유형
BlankSubordinateCACertificate_PathLen1_APICSRPassthrough/V1	arn:aws:acm-pca:::template/BlankSubordinateCACertificate_PathLen1_APICSRPassthrough/V1	CA
SubordinateCACertificate_PathLen2_APICSRPassthrough/PathLen3_APIPassthroughV1	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen2_APICSRPassthrough/V1	CA
BlankSubordinateCACertificate_PathLen2_APICSRPassthrough/V1	arn:aws:acm-pca:::template/BlankSubordinateCACertificate_PathLen2_APICSRPassthrough/V1	CA
SubordinateCACertificate_PathLen3_APICSRPassthrough/V1	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen3_APICSRPassthrough/V1	CA
BlankSubordinateCACertificate_PathLen3_APICSRPassthrough/V1	arn:aws:acm-pca:::template/BlankSubordinateCACertificate_PathLen3_APICSRPassthrough/V1	CA

AWS Private CA 템플릿 작업 순서

발급된 인증서에 포함된 정보는 템플릿 정의, API 패스스루, CSR 패스스루, CA 구성의 네 가지 소스에서 가져올 수 있습니다.

API 패스스루 값은 API 패스스루 또는 APICSR 패스스루 템플릿을 사용하는 경우에만 적용됩니다. CSR 패스스루는 CSR 패스스루 또는 APICSR 패스스루 템플릿을 사용하는 경우에만 적용됩니다. 이러한 정보 소스가 서로 충돌하는 경우 일반적으로 일반적인 규칙이 적용됩니다. 각 확장 값에 대해 템플릿 정의의 우선 순위가 가장 높고, API 패스스루 값, CSR 패스스루 확장이 그 뒤를 잇습니다.

예시

1. EndEntityClientAuthCertificate_APIPassthrough 템플릿 정의는 값이 “TLS 웹 서버 인증, TLS 웹 클라이언트 인증인 확장 키 사용 확장을 정의합니다. ExtendedKeyUsage에 대한 `ApiPassthrough` 값이 CSR 또는 `IssueCertificate` `ApiPassthrough` 파라미터에 정의된 경우 템플릿 정의가 우선하므로 ExtendedKeyUsage 값이 무시되고, 템플릿이 CSR 패스스루 유형이 아니므로 ExtendedKeyUsage 값의 CSR 값은 무시됩니다.

   > **Note**
   >
   > 하지만 템플릿 정의는 CSR의 다른 값(예: 보안 주체 및 보안 주체 대체 이름)을 덮어씁니다. 템플릿 정의가 항상 가장 높은 우선 순위를 갖기 때문에 템플릿이 CSR 패스스루 유형이 아니더라도 이러한 값은 여전히 CSR에서 가져옵니다.

2. EndEntityClientAuthCertificate_APICSRPassthrough의 템플릿 정의는 보안 주체 대체 이름(SAN) 확장을 API 또는 CSR에서 복사한 것으로 정의합니다. CSR에 SAN 확장을 정의하고 `IssueCertificate` `ApiPassthrough` 파라미터에 제공하는 경우 API 패스스루 값이 CSR 패스스루 값보다 우선하므로 API 패스스루 값이 우선합니다.

AWS Private CA 템플릿 정의

다음 섹션에서는 지원되는 AWS Private CA 인증서 템플릿에 대한 구성 세부 정보를 제공합니다.

BlankEndEntityCertificate_APIPassthrough/V1 정의

빈 최종 엔터티 인증서 템플릿을 사용하면 X.509 Basic 제약 조건만 있는 최종 엔터티 인증서를 발급할 수 있습니다. 이는에서 발급 AWS Private CA 할 수 있는 가장 간단한 최종 엔터티 인증서이지만 API 구조를 사용하여 사용자 지정할 수 있습니다. 기본 제약 조건 확장은 인증서가 CA 인증서인지 여부를 정의합니다. 빈 최종 엔터티 인증서 템플릿은 CA 인증서가 아닌 최종 엔터티 인증서가 발급되도록 기본 제약 조건에 FALSE 값을 적용합니다.

빈 패스스루 템플릿을 사용하여 키 사용(KU) 및 확장 키 사용(EKU)에 대한 특정 값이 필요한 스마트 카드 인증서를 발급할 수 있습니다. 예를 들어 확장 키 사용에는 클라이언트 인증 및 스마트 카드 로그온

이 필요할 수 있으며, 키 사용에는 디지털 서명, 거부 방지 및 키 암호화가 필요할 수 있습니다. 다른 패스스루 템플릿과 달리 빈 최종 엔터티 인증서 템플릿은 KU 및 EKU 확장을 구성할 수 있습니다. 여기서 KU는 지원되는 9개의 값(digitalSignature, nonRepudiation, keyEncipherment, dataEncipherment, keyAgreement, keyCertSign, cRLSign, encipherOnly, decipherOnly) 중 하나일 수 있고, EKU는 지원되는 값(serverAuth, clientAuth, codesigning, emailProtection, timestamping, OCSPSigning)과 사용자 지정 확장일 수 있습니다.

BlankEndEntityCertificate_APIPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[API 또는 CSR을 통한 패스스루]
제목	[API 또는 CSR을 통한 패스스루]
기본 제약 조건	CA:FALSE
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
CRL 배포 지점*	[CA 구성에서의 패스스루]

* CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 템플릿에 포함됩니다.

BlankEndEntityCertificate_APICSRPassthrough/V1 정의

빈 템플릿에 대한 일반적인 정보는 BlankEndEntityCertificate_APIPassthrough/V1 정의 섹션을 참조하세요.

BlankEndEntityCertificate_APICSRPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[API 또는 CSR을 통한 패스스루]
제목	[API 또는 CSR을 통한 패스스루]
기본 제약 조건	CA:FALSE
인증 기관 키 식별자	[CA 인증서에서의 SKI]

X509v3 파라미터	값
보안 주체 키 식별자	[CSR에서 파생됨]
CRL 배포 지점*	[CA 구성 또는 CSR에서의 패스스루]

* CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 템플릿에 포함됩니다.

BlankEndEntityCertificate_CriticalBasicConstraints_APICSRPassthrough/V1 정의

빈 템플릿에 대한 일반적인 정보는 BlankEndEntityCertificate_APIPassthrough/V1 정의 섹션을 참조하세요.

BlankEndEntityCertificate_CriticalBasicConstraints_APICSRPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[API 또는 CSR을 통한 패스스루]
제목	[API 또는 CSR을 통한 패스스루]
기본 제약 조건	중요, CA:FALSE
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
CRL 배포 지점*	[CA 구성, API 또는 CSR에서의 패스스루]

* CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 템플릿에 포함됩니다.

BlankEndEntityCertificate_CriticalBasicConstraints_APIPassthrough/V1 definition

빈 템플릿에 대한 일반적인 정보는 BlankEndEntityCertificate_APIPassthrough/V1 정의 섹션을 참조하세요.

BlankEndEntityCertificate_CriticalBasicConstraints_APIPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[API 또는 CSR을 통한 패스스루]

X509v3 파라미터	값
제목	[API 또는 CSR을 통한 패스스루]
기본 제약 조건	중요, CA:FALSE
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
CRL 배포 지점*	[CA 구성 또는 API에서의 패스스루]

* CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 템플릿에 포함됩니다.

BlankEndEntityCertificate_CriticalBasicConstraints_CSRPassthrough/V1 정의

빈 템플릿에 대한 일반적인 정보는 BlankEndEntityCertificate_APIPassthrough/V1 정의 섹션을 참조하세요.

BlankEndEntityCertificate_CriticalBasicConstraints_CSRPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[API 또는 CSR을 통한 패스스루]
제목	[API 또는 CSR을 통한 패스스루]
기본 제약 조건	중요, CA:FALSE
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
CRL 배포 지점*	[CA 구성 또는 CSR에서의 패스스루]

* CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 템플릿에 포함됩니다.

BlankEndEntityCertificate_CSRPassthrough/V1 정의

빈 템플릿에 대한 일반적인 정보는 BlankEndEntityCertificate_APIPassthrough/V1 정의 섹션을 참조하세요.

BlankEndEntityCertificate_CSRPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[CSR에서의 패스스루]
제목	[CSR에서의 패스스루]
기본 제약 조건	CA:FALSE
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
CRL 배포 지점*	[CA 구성 또는 CSR에서의 패스스루]

* CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 템플릿에 포함됩니다.

BlankSubordinateCACertificate_PathLen0_CSRPassthrough/V1 정의

빈 템플릿에 대한 일반적인 정보는 BlankEndEntityCertificate_APIPassthrough/V1 정의 섹션을 참조하세요.

BlankSubordinateCACertificate_PathLen0_CSRPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[CSR에서의 패스스루]
제목	[CSR에서의 패스스루]
기본 제약 조건	중요, `CA:TRUE, pathlen: 0`
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
CRL 배포 지점*	[CA 구성 또는 CSR에서의 패스스루]

* CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 템플릿에 포함됩니다.

BlankSubordinateCACertificate_PathLen0_APICSRPassthrough/V1 정의

빈 템플릿에 대한 일반적인 정보는 BlankEndEntityCertificate_APIPassthrough/V1 정의 섹션을 참조하세요.

BlankSubordinateCACertificate_PathLen0_APICSRPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[API 또는 CSR을 통한 패스스루]
제목	[API 또는 CSR을 통한 패스스루]
기본 제약 조건	중요, `CA:TRUE, pathlen: 0`
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
CRL 배포 지점*	[CA 구성 또는 CSR에서의 패스스루]

* CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 템플릿에 포함됩니다.

BlankSubordinateCACertificate_PathLen0_APIPassthrough/V1 정의

빈 템플릿에 대한 일반적인 정보는 BlankEndEntityCertificate_APIPassthrough/V1 정의 섹션을 참조하세요.

BlankSubordinateCACertificate_PathLen0_APIPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[API 또는 CSR을 통한 패스스루]
제목	[API 또는 CSR을 통한 패스스루]
기본 제약 조건	중요, `CA:TRUE, pathlen: 0`
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]

X509v3 파라미터	값
CRL 배포 지점*	[CA 구성에서의 패스스루]

BlankSubordinateCACertificate_PathLen1_APIPassthrough/V1 정의

빈 템플릿에 대한 일반적인 정보는 BlankEndEntityCertificate_APIPassthrough/V1 정의 섹션을 참조하세요.

BlankSubordinateCACertificate_PathLen1_APIPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[API 또는 CSR을 통한 패스스루]
제목	[API 또는 CSR을 통한 패스스루]
기본 제약 조건	중요, `CA:TRUE`, `pathlen: 1`
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
CRL 배포 지점*	[CA 구성에서의 패스스루]

* CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 템플릿에 포함됩니다.

BlankSubordinateCACertificate_PathLen1_CSRPassthrough/V1 정의

빈 템플릿에 대한 일반적인 정보는 BlankEndEntityCertificate_APIPassthrough/V1 정의 섹션을 참조하세요.

BlankSubordinateCACertificate_PathLen1_CSRPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[CSR에서의 패스스루]
제목	[CSR에서의 패스스루]
기본 제약 조건	중요, `CA:TRUE`, `pathlen: 1`

X509v3 파라미터	값
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
CRL 배포 지점*	[CA 구성 또는 CSR에서의 패스스루]

* CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 템플릿에 포함됩니다.

BlankSubordinateCACertificate_PathLen1_APICSRPassthrough/V1 정의

빈 템플릿에 대한 일반적인 정보는 BlankEndEntityCertificate_APIPassthrough/V1 정의 섹션을 참조하세요.

BlankSubordinateCACertificate_PathLen1_APICSRPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[API 또는 CSR을 통한 패스스루]
제목	[API 또는 CSR을 통한 패스스루]
기본 제약 조건	중요, `CA:TRUE`, `pathlen: 1`
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
CRL 배포 지점*	[CA 구성 또는 CSR에서의 패스스루]

* CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 템플릿에 포함됩니다.

BlankSubordinateCACertificate_PathLen2_APIPassthrough/V1 정의

빈 템플릿에 대한 일반적인 정보는 BlankEndEntityCertificate_APIPassthrough/V1 정의 섹션을 참조하세요.

BlankSubordinateCACertificate_PathLen2_APIPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[API 또는 CSR을 통한 패스스루]
제목	[API 또는 CSR을 통한 패스스루]
기본 제약 조건	중요, `CA:TRUE, pathlen: 2`
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
CRL 배포 지점*	[CA 구성에서의 패스스루]

* CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 템플릿에 포함됩니다.

BlankSubordinateCACertificate_PathLen2_CSRPassthrough/V1 정의

빈 템플릿에 대한 일반적인 정보는 BlankEndEntityCertificate_APIPassthrough/V1 정의 섹션을 참조하세요.

BlankSubordinateCACertificate_PathLen2_CSRPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[CSR에서의 패스스루]
제목	[CSR에서의 패스스루]
기본 제약 조건	중요, `CA:TRUE, pathlen: 2`
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
CRL 배포 지점*	[CA 구성 또는 CSR에서의 패스스루]

* CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 템플릿에 포함됩니다.

BlankSubordinateCACertificate_PathLen2_APICSRPassthrough/V1 정의

빈 템플릿에 대한 일반적인 정보는 BlankEndEntityCertificate_APIPassthrough/V1 정의 섹션을 참조하세요.

BlankSubordinateCACertificate_PathLen2_APICSRPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[API 또는 CSR을 통한 패스스루]
제목	[API 또는 CSR을 통한 패스스루]
기본 제약 조건	중요, `CA:TRUE, pathlen: 2`
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
CRL 배포 지점*	[CA 구성 또는 CSR에서의 패스스루]

* CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 템플릿에 포함됩니다.

BlankSubordinateCACertificate_PathLen3_APIPassthrough/V1 정의

빈 템플릿에 대한 일반적인 정보는 BlankEndEntityCertificate_APIPassthrough/V1 정의 섹션을 참조하세요.

BlankSubordinateCACertificate_PathLen3_APIPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[API 또는 CSR을 통한 패스스루]
제목	[API 또는 CSR을 통한 패스스루]
기본 제약 조건	중요, `CA:TRUE, pathlen: 3`
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]

X509v3 파라미터	값
CRL 배포 지점*	[CA 구성에서의 패스스루]

* CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 템플릿에 포함됩니다.

BlankSubordinateCACertificate_PathLen3_CSRPassthrough/V1 정의

빈 템플릿에 대한 일반적인 정보는 BlankEndEntityCertificate_APIPassthrough/V1 정의 섹션을 참조하세요.

BlankSubordinateCACertificate_PathLen3_CSRPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[CSR에서의 패스스루]
제목	[CSR에서의 패스스루]
기본 제약 조건	중요, `CA:TRUE, pathlen: 3`
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
CRL 배포 지점*	[CA 구성 또는 CSR에서의 패스스루]

* CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 템플릿에 포함됩니다.

BlankSubordinateCACertificate_PathLen3_APICSRPassthrough/V1 정의

빈 템플릿에 대한 일반적인 정보는 BlankEndEntityCertificate_APIPassthrough/V1 정의 섹션을 참조하세요.

BlankSubordinateCACertificate_PathLen3_APICSRPassthrough

X509v3 파라미터	값
보안 주체 대체 이름	[API 또는 CSR을 통한 패스스루]
제목	[API 또는 CSR을 통한 패스스루]

X509v3 파라미터	값
기본 제약 조건	중요, `CA:TRUE, pathlen: 3`
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
CRL 배포 지점*	[CA 구성 또는 CSR에서의 패스스루]

* CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 템플릿에 포함됩니다.

CodeSigningCertificate/V1 정의

이 템플릿은 코드 서명에 대한 인증서를 생성하는 데 사용됩니다. 프라이빗 CA 인프라를 기반으로 하는 모든 코드 서명 솔루션 AWS Private CA 에서의 코드 서명 인증서를 사용할 수 있습니다. 예를 들어 용 코드 서명을 사용하는 고객은를 사용하여 코드 서명 인증서를 생성하고 로 AWS Private CA 가져올 AWS IoT 수 있습니다 AWS Certificate Manager. 자세한 내용은 코드 서명이란 무엇입니까 AWS IoT? 및 코드 서명 인증서 가져오기 및 가져오기를 참조하세요.

CodeSigningCertificate/V1

X509v3 파라미터	값
보안 주체 대체 이름	[CSR에서의 패스스루]
제목	[CSR에서의 패스스루]
기본 제약 조건	`CA:FALSE`
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
키 사용	중요, 디지털 서명
확장 키 사용	중요, 코드 서명
CRL 배포 지점*	[CA 구성에서의 패스스루]

*CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 템플릿에 포함됩니다.

CodeSigningCertificate_APICSRPassthrough/V1 정의

이 템플릿은 코드 디자인 인증서/v1을 확장하여 API 및 CSR 패스스루 값을 지원합니다.

CodeSigningCertificate_APICSRPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[API 또는 CSR을 통한 패스스루]
제목	[API 또는 CSR을 통한 패스스루]
기본 제약 조건	CA:FALSE
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
키 사용	중요, 디지털 서명
확장 키 사용	중요, 코드 서명
CRL 배포 지점*	[CA 구성 또는 CSR에서의 패스스루]

* CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 템플릿에 포함됩니다.

CodeSigningCertificate_APIPassthrough/V1 정의

이 템플릿은 한 가지 차이점이 있는 CodeSigningCertificate 템플릿과 동일합니다.이 템플릿에서는 템플릿에 확장이 지정되지 않은 경우 API를 통해 인증서로 추가 확장을 AWS Private CA 전달합니다. 템플릿에 지정된 확장은 항상 API의 확장보다 우선합니다.

CodeSigningCertificate_APIPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[API 또는 CSR을 통한 패스스루]
제목	[API 또는 CSR을 통한 패스스루]

X509v3 파라미터	값
기본 제약 조건	CA:FALSE
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
키 사용	중요, 디지털 서명
확장 키 사용	중요, 코드 서명
CRL 배포 지점*	[CA 구성에서의 패스스루]

* CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 템플릿에 포함됩니다.

CodeSigningCertificate_CSRPassthrough/V1 정의

이 템플릿은 한 가지 차이점이 있는 CodeSigningCertificate 템플릿과 동일합니다.이 템플릿에서는 템플릿에 확장이 지정되지 않은 경우 인증서 서명 요청(CSR)의 추가 확장을 인증서로 AWS Private CA 전달합니다. 템플릿에 지정된 확장은 항상 CSR의 확장보다 우선합니다.

CodeSigningCertificate_CSRPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[CSR에서의 패스스루]
제목	[CSR에서의 패스스루]
기본 제약 조건	CA:FALSE
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
키 사용	중요, 디지털 서명
확장 키 사용	중요, 코드 서명
CRL 배포 지점*	[CA 구성 또는 CSR에서의 패스스루]

*CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 템플릿에 포함됩니다.

EndEntityCertificate/V1 정의

이 템플릿은 운영 체제 또는 웹 서버와 같은 최종 엔터티에 대한 인증서를 생성하는 데 사용됩니다.

EndEntityCertificate/V1

X509v3 파라미터	값
보안 주체 대체 이름	[CSR에서의 패스스루]
제목	[CSR에서의 패스스루]
기본 제약 조건	CA:FALSE
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
키 사용	중요, 디지털 서명, 키 암호화
확장 키 사용	TLS 웹 서버 인증, TLS 웹 클라이언트 인증
CRL 배포 지점*	[CA 구성에서의 패스스루]

*CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 템플릿에 포함됩니다.

EndEntityCertificate_APICSRPassthrough/V1 정의

이 템플릿은 최종 엔터티 인증서/v1을 확장하여 API 및 CSR 패스스루 값을 지원합니다.

EndEntityCertificate_APICSRPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[API 또는 CSR을 통한 패스스루]
제목	[API 또는 CSR을 통한 패스스루]
기본 제약 조건	CA:FALSE

X509v3 파라미터	값
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
키 사용	중요, 디지털 서명, 키 암호화
확장 키 사용	TLS 웹 서버 인증, TLS 웹 클라이언트 인증
CRL 배포 지점*	[CA 구성 또는 CSR에서의 패스스루]

* CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 템플릿에 포함됩니다.

EndEntityCertificate_APIPassthrough/V1 정의

이 템플릿은 한 가지 차이점이 있는 `EndEntityCertificate` 템플릿과 동일합니다.이 템플릿에서는 템플릿에 확장이 지정되지 않은 경우 API를 통해 인증서로 추가 확장을 AWS Private CA 전달합니다. 템플릿에 지정된 확장은 항상 API의 확장보다 우선합니다.

EndEntityCertificate_APIPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[API 또는 CSR을 통한 패스스루]
제목	[API 또는 CSR을 통한 패스스루]
기본 제약 조건	CA:FALSE
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
키 사용	중요, 디지털 서명, 키 암호화
확장 키 사용	TLS 웹 서버 인증, TLS 웹 클라이언트 인증
CRL 배포 지점*	[CA 구성에서의 패스스루]

* CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 템플릿에 포함됩니다.

EndEntityCertificate_CSRPassthrough/V1 정의

이 템플릿은 한 가지 차이점이 있는 `EndEntityCertificate` 템플릿과 동일합니다.이 템플릿에서는 템플릿에 확장이 지정되지 않은 경우 인증서 서명 요청(CSR)의 추가 확장을 인증서로 AWS Private CA 전달합니다. 템플릿에 지정된 확장은 항상 CSR의 확장보다 우선합니다.

EndEntityCertificate_CSRPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[CSR에서의 패스스루]
제목	[CSR에서의 패스스루]
기본 제약 조건	CA:FALSE
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
키 사용	중요, 디지털 서명, 키 암호화
확장 키 사용	TLS 웹 서버 인증, TLS 웹 클라이언트 인증
CRL 배포 지점*	[CA 구성 또는 CSR에서의 패스스루]

*CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 템플릿에 포함됩니다.

EndEntityClientAuthCertificate/V1 정의

이 템플릿은 TLS 웹 클라이언트 인증으로 제한하는 확장 키 사용 값에서만 `EndEntityCertificate`과 다릅니다.

EndEntityClientAuthCertificate/V1

X509v3 파라미터	값
보안 주체 대체 이름	[CSR에서의 패스스루]
제목	[CSR에서의 패스스루]
기본 제약 조건	CA:FALSE

X509v3 파라미터	값
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
키 사용	중요, 디지털 서명, 키 암호화
확장 키 사용	TLS 웹 클라이언트 인증
CRL 배포 지점*	[CA 구성 또는 CSR에서의 패스스루]

*CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 템플릿에 포함됩니다.

EndEntityClientAuthCertificate_APICSRPassthrough/V1 정의

이 템플릿은 최종 엔터티 클라이언트 인증 인증서/v1을 확장하여 API 및 CSR 패스스루 값을 지원합니다.

EndEntityClientAuthCertificate_APICSRPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[API 또는 CSR을 통한 패스스루]
제목	[API 또는 CSR을 통한 패스스루]
기본 제약 조건	CA:FALSE
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
키 사용	중요, 디지털 서명, 키 암호화
확장 키 사용	TLS 웹 클라이언트 인증
CRL 배포 지점*	[CA 구성 또는 CSR에서의 패스스루]

* CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 템플릿에 포함됩니다.

EndEntityClientAuthCertificate_APIPassthrough/V1 정의

이 템플릿은 한 가지 차이점을 제외하고 `EndEntityClientAuthCertificate` 템플릿과 동일합니다. 이 템플릿에서는 템플릿에 확장이 지정되지 않은 경우 API를 통해 추가 확장을 인증서로 AWS Private CA 전달합니다. 템플릿에 지정된 확장은 항상 API의 확장보다 우선합니다.

EndEntityClientAuthCertificate_APIPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[API 또는 CSR을 통한 패스스루]
제목	[API 또는 CSR을 통한 패스스루]
기본 제약 조건	CA:FALSE
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
키 사용	중요, 디지털 서명, 키 암호화
확장 키 사용	TLS 웹 클라이언트 인증
CRL 배포 지점*	[CA 구성에서의 패스스루]

* CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 템플릿에 포함됩니다.

EndEntityClientAuthCertificate_CSRPassthrough/V1 정의

이 템플릿은 한 가지 차이점을 제외하고 `EndEntityClientAuthCertificate` 템플릿과 동일합니다. 이 템플릿에서는 템플릿에 확장이 지정되지 않은 경우 인증서 서명 요청(CSR)의 추가 확장을 인증서로 AWS Private CA 전달합니다. 템플릿에 지정된 확장은 항상 CSR의 확장보다 우선합니다.

EndEntityClientAuthCertificate_CSRPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[CSR에서의 패스스루]
제목	[CSR에서의 패스스루]

X509v3 파라미터	값
기본 제약 조건	CA:FALSE
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
키 사용	중요, 디지털 서명, 키 암호화
확장 키 사용	TLS 웹 클라이언트 인증
CRL 배포 지점*	[CA 구성 또는 CSR에서의 패스스루]

*CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 템플릿에 포함됩니다.

EndEntityServerAuthCertificate/V1 정의

이 템플릿은 TLS 웹 서버 인증으로 제한하는 확장 키 사용 값에서만 `EndEntityCertificate`과 다릅니다.

EndEntityServerAuthCertificate/V1

X509v3 파라미터	값
보안 주체 대체 이름	[CSR에서의 패스스루]
제목	[CSR에서의 패스스루]
기본 제약 조건	CA:FALSE
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
키 사용	중요, 디지털 서명, 키 암호화
확장 키 사용	TLS 웹 서버 인증
CRL 배포 지점*	[CA 구성에서의 패스스루]

*CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 템플릿에 포함됩니다.

EndEntityServerAuthCertificate_APICSRPassthrough/V1 정의

이 템플릿은 최종 엔터티 서버 인증 인증서/v1을 확장하여 API 및 CSR 패스스루 값을 지원합니다.

EndEntityServerAuthCertificate_APICSRPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[API 또는 CSR을 통한 패스스루]
제목	[API 또는 CSR을 통한 패스스루]
기본 제약 조건	CA:FALSE
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
키 사용	중요, 디지털 서명, 키 암호화
확장 키 사용	TLS 웹 서버 인증
CRL 배포 지점*	[CA 구성 또는 CSR에서의 패스스루]

* CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 템플릿에 포함됩니다.

EndEntityServerAuthCertificate_APIPassthrough/V1 정의

이 템플릿은 한 가지 차이점을 제외하고 `EndEntityServerAuthCertificate` 템플릿과 동일합니다. 이 템플릿에서는 템플릿에 확장이 지정되지 않은 경우 API를 통해 추가 확장을 인증서로 AWS Private CA 전달합니다. 템플릿에 지정된 확장은 항상 API의 확장보다 우선합니다.

EndEntityServerAuthCertificate_APIPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[API 또는 CSR을 통한 패스스루]
제목	[API 또는 CSR을 통한 패스스루]

X509v3 파라미터	값
기본 제약 조건	CA:FALSE
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
키 사용	중요, 디지털 서명, 키 암호화
확장 키 사용	TLS 웹 서버 인증
CRL 배포 지점*	[CA 구성에서의 패스스루]

* CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 템플릿에 포함됩니다.

EndEntityServerAuthCertificate_CSRPassthrough/V1 정의

이 템플릿은 한 가지 차이점을 제외하고 `EndEntityServerAuthCertificate` 템플릿과 동일합니다. 이 템플릿에서는 템플릿에 확장이 지정되지 않은 경우 인증서 서명 요청(CSR)의 추가 확장을 인증서로 AWS Private CA 전달합니다. 템플릿에 지정된 확장은 항상 CSR의 확장보다 우선합니다.

EndEntityServerAuthCertificate_CSRPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[CSR에서의 패스스루]
제목	[CSR에서의 패스스루]
기본 제약 조건	CA:FALSE
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
키 사용	중요, 디지털 서명, 키 암호화
확장 키 사용	TLS 웹 서버 인증
CRL 배포 지점*	[CA 구성 또는 CSR에서의 패스스루]

*CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 템플릿에 포함됩니다.

OCSPSigningCertificate/V1 정의

이 템플릿은 OCSP 응답에 서명하기 위한 인증서를 생성하는 데 사용됩니다. 확장 키 사용 값이 코드 서명 대신 OCSP 서명을 지정한다는 점을 제외하면 `CodeSigningCertificate` 템플릿과 동일합니다.

OCSPSigningCertificate/V1

X509v3 파라미터	값
보안 주체 대체 이름	[CSR에서의 패스스루]
제목	[CSR에서의 패스스루]
기본 제약 조건	`CA:FALSE`
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
키 사용	중요, 디지털 서명
확장 키 사용	중요, OCSP 서명
CRL 배포 지점*	[CA 구성에서의 패스스루]

*CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 템플릿에 포함됩니다.

OCSPSigningCertificate_APICSRPassthrough/V1 정의

이 템플릿은 OCSP 서명 인증서/v1을 확장하여 API 및 CSR 패스스루 값을 지원합니다.

OCSPSigningCertificate_APICSRPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[API 또는 CSR을 통한 패스스루]
제목	[API 또는 CSR을 통한 패스스루]

X509v3 파라미터	값
기본 제약 조건	CA:FALSE
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
키 사용	중요, 디지털 서명
확장 키 사용	중요, OCSP 서명
CRL 배포 지점*	[CA 구성 또는 CSR에서의 패스스루]

* CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 템플릿에 포함됩니다.

OCSPSigningCertificate_APIPassthrough/V1 정의

이 템플릿은 한 가지 차이점을 제외하고 OCSPSigningCertificate 템플릿과 동일합니다. 이 템플릿에서는 템플릿에 확장이 지정되지 않은 경우 API를 통해 추가 확장을 인증서로 AWS Private CA 전달합니다. 템플릿에 지정된 확장은 항상 API의 확장보다 우선합니다.

OCSPSigningCertificate_APIPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[API 또는 CSR을 통한 패스스루]
제목	[API 또는 CSR을 통한 패스스루]
기본 제약 조건	CA:FALSE
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
키 사용	중요, 디지털 서명
확장 키 사용	중요, OCSP 서명
CRL 배포 지점*	[CA 구성에서의 패스스루]

* CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 템플릿에 포함됩니다.

OCSPSigningCertificate_CSRPassthrough/V1 정의

이 템플릿은 한 가지 차이점을 제외하고 `OCSPSigningCertificate` 템플릿과 동일합니다. 이 템플릿에서는 템플릿에 확장이 지정되지 않은 경우 인증서 서명 요청(CSR)의 추가 확장을 인증서로 AWS Private CA 전달합니다. 템플릿에 지정된 확장은 항상 CSR의 확장보다 우선합니다.

OCSPSigningCertificate_CSRPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[CSR에서의 패스스루]
제목	[CSR에서의 패스스루]
기본 제약 조건	`CA:FALSE`
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
키 사용	중요, 디지털 서명
확장 키 사용	중요, OCSP 서명
CRL 배포 지점*	[CA 구성 또는 CSR에서의 패스스루]

*CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 템플릿에 포함됩니다.

RootCACertificate/V1 정의

이 템플릿은 자체 서명된 루트 CA 인증서를 발급하는 데 사용됩니다. CA 인증서에는 CA 인증서를 발급하는 데 사용할 수 있음을 지정하도록 CA 필드가 `TRUE`로 설정된 중요한 기본 제약 조건 확장이 포함됩니다. 템플릿은 경로 길이(pathLenConstraint)를 지정하지 않습니다. 이렇게 하면 향후 계층 구조 확장이 방해될 수 있기 때문입니다. CA 인증서가 TLS 클라이언트 또는 서버 인증서로 사용되는 막기 위해 확장 키 사용은 제외됩니다. 자체 서명된 인증서를 해지할 수 없으므로 CRL 정보가 지정되지 않았습니다.

RootCACertificate/V1

X509v3 파라미터	값
보안 주체 대체 이름	[CSR에서의 패스스루]
제목	[CSR에서의 패스스루]
기본 제약 조건	중요, CA:TRUE
보안 주체 키 식별자	[CSR에서 파생됨]
키 사용	중요, 디지털 서명, keyCertSign, CRL 서명
CRL 배포 지점	해당 사항 없음

RootCACertificate_APIPassthrough/V1 정의

이 템플릿은 루트 CA 인증서/v1을 확장하여 API 패스스루 값을 지원합니다.

RootCACertificate_APIPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[API 또는 CSR을 통한 패스스루]
제목	[API 또는 CSR을 통한 패스스루]
기본 제약 조건	중요, CA:TRUE
인증 기관 키 식별자	[API에서의 패스스루]
보안 주체 키 식별자	[CSR에서 파생됨]
키 사용	중요, 디지털 서명, keyCertSign, CRL 서명
CRL 배포 지점*	해당 사항 없음

BlankRootCACertificate_APIPassthrough/V1 정의

빈 루트 인증서 템플릿을 사용하면 X.509 기본 제약 조건만 있는 루트 인증서를 발급할 수 있습니다. 이는에서 발급 AWS Private CA 할 수 있는 가장 간단한 루트 인증서이지만 API 구조를 사용하여 사용자 지정할 수 있습니다. 기본 제약 조건 확장은 인증서가 CA 인증서인지 여부를 정의합니다. 빈 루트 인증서 템플릿은 기본 제약 조건에 TRUE 대해 값을 적용하여 루트 CA 인증서가 발급되도록 합니다.

빈 패스스루 루트 템플릿을 사용하여 키 사용(KU)에 특정 값이 필요한 루트 인증서를 발급할 수 있습니다. 예를 들어 키 사용에 keyCertSign 및가 필요할 수 cRLSign있지만는 필요하지 않습니다digitalSignature. 비어 있지 않은 다른 루트 패스스루 인증서 템플릿과 달리 빈 루트 인증서 템플릿은 KU 확장의 구성을 허용합니다. 여기서 KU는 9개의 지원되는 값 (digitalSignature, , nonRepudiation, keyEncipherment, dataEncipherment, keyAgreement, keyCertSigncRLSign, , encipherOnly,) 중 하나일 수 있습니다decipherOnly.

BlankRootCACertificate_APIPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[API 또는 CSR을 통한 패스스루]
제목	[API 또는 CSR을 통한 패스스루]
기본 제약 조건	중요, CA:TRUE
보안 주체 키 식별자	[CSR에서 파생됨]

BlankRootCACertificate_PathLen0_APIPassthrough/V1 정의

빈 루트 CA 템플릿에 대한 일반적인 내용은 섹션을 참조하세요???.

BlankRootCACertificate_PathLen0_APIPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[API 또는 CSR을 통한 패스스루]
제목	[API 또는 CSR을 통한 패스스루]
기본 제약 조건	중요, CA:TRUE, pathlen: 0

X509v3 파라미터	값
보안 주체 키 식별자	[CSR에서 파생됨]

BlankRootCACertificate_PathLen1_APIPassthrough/V1 정의

빈 루트 CA 템플릿에 대한 일반적인 내용은 섹션을 참조하세요???.

BlankRootCACertificate_PathLen1_APIPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[API 또는 CSR을 통한 패스스루]
제목	[API 또는 CSR을 통한 패스스루]
기본 제약 조건	중요, `CA:TRUE`, `pathlen: 1`
보안 주체 키 식별자	[CSR에서 파생됨]

BlankRootCACertificate_PathLen2_APIPassthrough/V1 정의

빈 루트 CA 템플릿에 대한 일반적인 내용은 섹션을 참조하세요???.

BlankRootCACertificate_PathLen2_APIPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[API 또는 CSR을 통한 패스스루]
제목	[API 또는 CSR을 통한 패스스루]
기본 제약 조건	중요, `CA:TRUE`, `pathlen: 2`
보안 주체 키 식별자	[CSR에서 파생됨]

BlankRootCACertificate_PathLen3_APIPassthrough/V1 정의

빈 루트 CA 템플릿에 대한 일반적인 내용은 섹션을 참조하세요???.

BlankRootCACertificate_PathLen3_APIPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[API 또는 CSR을 통한 패스스루]
제목	[API 또는 CSR을 통한 패스스루]
기본 제약 조건	중요, `CA:TRUE`, `pathlen: 3`
보안 주체 키 식별자	[CSR에서 파생됨]

SubordinateCACertificate_PathLen0/V1 정의

이 템플릿은 경로 길이가 인 하위 CA 인증서를 발급하는 데 사용됩니다`0`. CA 인증서에는 CA 인증서를 발급하는 데 사용할 수 있음을 지정하도록 CA 필드가 `TRUE`로 설정된 중요한 기본 제약 조건 확장이 포함됩니다. 확장 키 사용은 포함되지 않으므로 CA 인증서가 TLS 클라이언트 또는 서버 인증서로 사용되지 않습니다.

인증 경로에 대한 자세한 내용은 인증 경로에서 길이 제약 조건 설정을 참조하십시오.

SubordinateCACertificate_PathLen0/V1

X509v3 파라미터	값
보안 주체 대체 이름	[CSR에서의 패스스루]
제목	[CSR에서의 패스스루]
기본 제약 조건	중요, `CA:TRUE`, `pathlen: 0`
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
키 사용	중요, 디지털 서명, `keyCertSign` , CRL 서명
CRL 배포 지점*	[CA 구성에서의 패스스루]

*CRL 배포 지점은 CA가 CRL 생성이 활성화된 상태로 구성된 경우에만이 템플릿으로 발급된 인증서에 포함됩니다.

SubordinateCACertificate_PathLen0_APICSRPassthrough/V1 정의

이 템플릿은 SubordinateCACertificate_PathLen0/V1을 확장하여 API 및 CSR 패스스루 값을 지원합니다.

SubordinateCACertificate_PathLen0_APICSRPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[API 또는 CSR을 통한 패스스루]
제목	[API 또는 CSR을 통한 패스스루]
기본 제약 조건	중요, `CA:TRUE`, `pathlen: 0`
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
키 사용	중요, 디지털 서명, `keyCertSign` , CRL 서명
CRL 배포 지점*	[CA 구성 또는 CSR에서의 패스스루]

* CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 템플릿에 포함됩니다.

SubordinateCACertificate_PathLen0_APIPassthrough/V1 정의

이 템플릿은 SubordinateCACertificate_PathLen0/V1을 확장하여 API 패스스루 값을 지원합니다.

SubordinateCACertificate_PathLen0_APIPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[API 또는 CSR을 통한 패스스루]
제목	[API 또는 CSR을 통한 패스스루]
기본 제약 조건	중요, `CA:TRUE`, `pathlen: 0`
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]

X509v3 파라미터	값
키 사용	중요, 디지털 서명, `keyCertSign` , CRL 서명
CRL 배포 지점*	[CA 구성에서의 패스스루]

* CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 템플릿에 포함됩니다.

SubordinateCACertificate_PathLen0_CSRPassthrough/V1 정의

이 템플릿은 한 가지 차이점이 있는 `SubordinateCACertificate_PathLen0` 템플릿과 동일합니다.이 템플릿에서는 템플릿에 확장이 지정되지 않은 경우 인증서 서명 요청(CSR)의 추가 확장을 인증서로 AWS Private CA 전달합니다. 템플릿에 지정된 확장은 항상 CSR의 확장보다 우선합니다.

> **Note**
>
> 사용자 지정 추가 확장이 포함된 CSR은 AWS Private CA외부에서 생성해야 합니다.

SubordinateCACertificate_PathLen0_CSRPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[CSR에서의 패스스루]
제목	[CSR에서의 패스스루]
기본 제약 조건	중요, `CA:TRUE`, `pathlen: 0`
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
키 사용	중요, 디지털 서명, `keyCertSign` , CRL 서명
CRL 배포 지점*	[CA 구성 또는 CSR에서의 패스스루]

*CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 이 템플릿으로 발급된 인증서에 포함됩니다.

SubordinateCACertificate_PathLen1/V1 정의

이 템플릿은 경로 길이가 인 하위 CA 인증서를 발급하는 데 사용됩니다1. CA 인증서에는 CA 인증서를 발급하는 데 사용할 수 있음을 지정하도록 CA 필드가 TRUE로 설정된 중요한 기본 제약 조건 확장이 포함됩니다. 확장 키 사용은 포함되지 않으므로 CA 인증서가 TLS 클라이언트 또는 서버 인증서로 사용되지 않습니다.

인증 경로에 대한 자세한 내용은 인증 경로에서 길이 제약 조건 설정을 참조하십시오.

SubordinateCACertificate_PathLen1/V1

X509v3 파라미터	값
보안 주체 대체 이름	[CSR에서의 패스스루]
제목	[CSR에서의 패스스루]
기본 제약 조건	중요, `CA:TRUE`, `pathlen: 1`
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
키 사용	중요, 디지털 서명, `keyCertSign` , CRL 서명
CRL 배포 지점*	[CA 구성에서의 패스스루]

*CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 이 템플릿으로 발급된 인증서에 포함됩니다.

SubordinateCACertificate_PathLen1_APICSRPassthrough/V1 정의

이 템플릿은 SubordinateCACertificate_PathLen1/V1을 확장하여 API 및 CSR 패스스루 값을 지원합니다.

SubordinateCACertificate_PathLen1_APICSRPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[API 또는 CSR을 통한 패스스루]
제목	[API 또는 CSR을 통한 패스스루]

X509v3 파라미터	값
기본 제약 조건	중요, `CA:TRUE, pathlen: 1`
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
키 사용	중요, 디지털 서명, `keyCertSign` , CRL 서명
CRL 배포 지점*	[CA 구성 또는 CSR에서의 패스스루]

* CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 템플릿에 포함됩니다.

SubordinateCACertificate_PathLen1_APIPassthrough/V1 정의

이 템플릿은 SubordinateCACertificate_PathLen0/V1을 확장하여 API 패스스루 값을 지원합니다.

SubordinateCACertificate_PathLen1_APIPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[API 또는 CSR을 통한 패스스루]
제목	[API 또는 CSR을 통한 패스스루]
기본 제약 조건	중요, `CA:TRUE, pathlen: 1`
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
키 사용	중요, 디지털 서명, `keyCertSign` , CRL 서명
CRL 배포 지점*	[CA 구성에서의 패스스루]

* CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 템플릿에 포함됩니다.

SubordinateCACertificate_PathLen1_CSRPassthrough/V1 정의

이 템플릿은 한 가지 차이점이 있는 SubordinateCACertificate_PathLen1 템플릿과 동일합니다.이 템플릿에서는 템플릿에 확장이 지정되지 않은 경우 인증서 서명 요청(CSR)의 추가 확장을 인증서로 AWS Private CA 전달합니다. 템플릿에 지정된 확장은 항상 CSR의 확장보다 우선합니다.

> **Note**
>
> 사용자 지정 추가 확장이 포함된 CSR은 AWS Private CA외부에서 생성해야 합니다.

SubordinateCACertificate_PathLen1_CSRPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[CSR에서의 패스스루]
제목	[CSR에서의 패스스루]
기본 제약 조건	중요, CA:TRUE, pathlen: 1
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
키 사용	중요, 디지털 서명, keyCertSign , CRL 서명
CRL 배포 지점*	[CA 구성 또는 CSR에서의 패스스루]

*CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 이 템플릿으로 발급된 인증서에 포함됩니다.

SubordinateCACertificate_PathLen2/V1 정의

이 템플릿은 경로 길이가 2인 하위 CA 인증서를 발급하는 데 사용됩니다. CA 인증서에는 CA 인증서를 발급하는 데 사용할 수 있음을 지정하도록 CA 필드가 TRUE로 설정된 중요한 기본 제약 조건 확장이 포함됩니다. 확장 키 사용은 포함되지 않으므로 CA 인증서가 TLS 클라이언트 또는 서버 인증서로 사용되지 않습니다.

인증 경로에 대한 자세한 내용은 인증 경로에서 길이 제약 조건 설정을 참조하십시오.

SubordinateCACertificate_PathLen2/V1

X509v3 파라미터	값
보안 주체 대체 이름	[CSR에서의 패스스루]
제목	[CSR에서의 패스스루]
기본 제약 조건	중요, `CA:TRUE`, `pathlen: 2`
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
키 사용	중요, 디지털 서명, `keyCertSign` , CRL 서명
CRL 배포 지점*	[CA 구성에서의 패스스루]

*CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 이 템플릿으로 발급된 인증서에 포함됩니다.

SubordinateCACertificate_PathLen2_APICSRPassthrough/V1 정의

이 템플릿은 SubordinateCACertificate_PathLen2/V1을 확장하여 API 및 CSR 패스스루 값을 지원합니다.

SubordinateCACertificate_PathLen2_APICSRPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[API 또는 CSR을 통한 패스스루]
제목	[API 또는 CSR을 통한 패스스루]
기본 제약 조건	중요, `CA:TRUE`, `pathlen: 2`
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
키 사용	중요, 디지털 서명, `keyCertSign` , CRL 서명

X509v3 파라미터	값
CRL 배포 지점*	[CA 구성 또는 CSR에서의 패스스루]

* CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 템플릿에 포함됩니다.

SubordinateCACertificate_PathLen2_APIPassthrough/V1 정의

이 템플릿은 SubordinateCACertificate_PathLen2/V1을 확장하여 API 패스스루 값을 지원합니다.

SubordinateCACertificate_PathLen2_APIPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[API 또는 CSR을 통한 패스스루]
제목	[API 또는 CSR을 통한 패스스루]
기본 제약 조건	중요, `CA:TRUE`, `pathlen: 2`
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
키 사용	중요, 디지털 서명, `keyCertSign` , CRL 서명
CRL 배포 지점*	[CA 구성에서의 패스스루]

* CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 템플릿에 포함됩니다.

SubordinateCACertificate_PathLen2_CSRPassthrough/V1 정의

이 템플릿은 한 가지 차이점이 있는 `SubordinateCACertificate_PathLen2` 템플릿과 동일합니다.이 템플릿에서는 템플릿에 확장이 지정되지 않은 경우 인증서 서명 요청(CSR)의 추가 확장을 인증서로 AWS Private CA 전달합니다. 템플릿에 지정된 확장은 항상 CSR의 확장보다 우선합니다.

> **Note**
>
> 사용자 지정 추가 확장이 포함된 CSR은 AWS Private CA외부에서 생성해야 합니다.

SubordinateCACertificate_PathLen2_CSRPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[CSR에서의 패스스루]
제목	[CSR에서의 패스스루]
기본 제약 조건	중요, `CA:TRUE`, `pathlen: 2`
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
키 사용	중요, 디지털 서명, `keyCertSign` , CRL 서명
CRL 배포 지점*	[CA 구성 또는 CSR에서의 패스스루]

*CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 이 템플릿으로 발급된 인증서에 포함됩니다.

SubordinateCACertificate_PathLen3/V1 정의

이 템플릿은 경로 길이가 3인 하위 CA 인증서를 발급하는 데 사용됩니다. CA 인증서에는 CA 인증서를 발급하는 데 사용할 수 있음을 지정하도록 CA 필드가 `TRUE`로 설정된 중요한 기본 제약 조건 확장이 포함됩니다. 확장 키 사용은 포함되지 않으므로 CA 인증서가 TLS 클라이언트 또는 서버 인증서로 사용되지 않습니다.

인증 경로에 대한 자세한 내용은 인증 경로에서 길이 제약 조건 설정을 참조하십시오.

SubordinateCACertificate_PathLen3/V1

X509v3 파라미터	값
보안 주체 대체 이름	[CSR에서의 패스스루]
제목	[CSR에서의 패스스루]
기본 제약 조건	중요, `CA:TRUE`, `pathlen: 3`
인증 기관 키 식별자	[CA 인증서에서의 SKI]

X509v3 파라미터	값
보안 주체 키 식별자	[CSR에서 파생됨]
키 사용	중요, 디지털 서명, `keyCertSign` , CRL 서명
CRL 배포 지점*	[CA 구성에서의 패스스루]

*CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 이 템플릿으로 발급된 인증서에 포함됩니다.

SubordinateCACertificate_PathLen3_APICSRPassthrough/V1 정의

이 템플릿은 SubordinateCACertificate_PathLen3/V1을 확장하여 API 및 CSR 패스스루 값을 지원합니다.

SubordinateCACertificate_PathLen3_APICSRPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[API 또는 CSR을 통한 패스스루]
제목	[API 또는 CSR을 통한 패스스루]
기본 제약 조건	중요, `CA:TRUE`, `pathlen: 3`
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
키 사용	중요, 디지털 서명, `keyCertSign` , CRL 서명
CRL 배포 지점*	[CA 구성 또는 CSR에서의 패스스루]

* CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 템플릿에 포함됩니다.

SubordinateCACertificate_PathLen3_APIPassthrough/V1 정의

이 템플릿은 SubordinateCACertificate_PathLen3/V1을 확장하여 API 패스스루 값을 지원합니다.

SubordinateCACertificate_PathLen3_APIPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[API 또는 CSR을 통한 패스스루]
제목	[API 또는 CSR을 통한 패스스루]
기본 제약 조건	중요, `CA:TRUE`, `pathlen: 3`
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
키 사용	중요, 디지털 서명, `keyCertSign` , CRL 서명
CRL 배포 지점*	[CA 구성에서의 패스스루]

* CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 템플릿에 포함됩니다.

SubordinateCACertificate_PathLen3_CSRPassthrough/V1 정의

이 템플릿은 한 가지 차이점이 있는 `SubordinateCACertificate_PathLen3` 템플릿과 동일합니다.이 템플릿에서는 템플릿에 확장이 지정되지 않은 경우 인증서 서명 요청(CSR)의 추가 확장을 인증서로 AWS Private CA 전달합니다. 템플릿에 지정된 확장은 항상 CSR의 확장보다 우선합니다.

> **Note**
>
> 사용자 지정 추가 확장이 포함된 CSR은 AWS Private CA외부에서 생성해야 합니다.

SubordinateCACertificate_PathLen3_CSRPassthrough/V1

X509v3 파라미터	값
보안 주체 대체 이름	[CSR에서의 패스스루]
제목	[CSR에서의 패스스루]
기본 제약 조건	중요, `CA:TRUE`, `pathlen: 3`

X509v3 파라미터	값
인증 기관 키 식별자	[CA 인증서에서의 SKI]
보안 주체 키 식별자	[CSR에서 파생됨]
키 사용	중요, 디지털 서명, `keyCertSign` , CRL 서명
CRL 배포 지점*	[CA 구성 또는 CSR에서의 패스스루]

*CRL 배포 지점은 CA가 CRL 생성을 사용하도록 구성된 경우에만 이 템플릿으로 발급된 인증서에 포함됩니다.

의 보안 AWS Private Certificate Authority

의 클라우드 보안 AWS 이 최우선 순위입니다. AWS 고객은 보안에 가장 민감한 조직의 요구 사항을 충족하도록 구축된 데이터 센터 및 네트워크 아키텍처의 이점을 누릴 수 있습니다.

보안은 AWS 와 사용자 간의 공동 책임입니다. 공동 책임 모델은 이 사항을 클라우드의 보안 및 클라우드 내 보안으로 설명합니다.

- 클라우드 보안 - AWS 는 클라우드에서 AWS AWS 서비스를 실행하는 인프라를 보호할 책임이 있습니다. AWS 또한는 안전하게 사용할 수 있는 서비스를 제공합니다. 타사 감사자는 AWS 규정 준수 프로그램 일환으로 보안의 효과를 정기적으로 테스트하고 확인합니다. 에 적용되는 규정 준수 프로그램에 대한 자세한 내용은 규정 준수 AWS 서비스 프로그램 제공 범위규정 준수 AWS Private Certificate Authority참조하세요.
- 클라우드의 보안 - 사용자의 책임은 사용하는 AWS 서비스에 따라 결정됩니다. 또한 귀하는 귀사의 데이터 민감도, 귀사의 요구 사항, 관련 법률 및 규정을 비롯한 기타 요소에 대해서도 책임이 있습니다.

이 설명서는를 사용할 때 공동 책임 모델을 적용하는 방법을 이해하는 데 도움이 됩니다 AWS Private CA. 다음 주제에서는 보안 및 규정 준수 목표를 충족하도록 AWS Private CA 를 구성하는 방법을 보여줍니다. 또한 AWS Private CA 리소스를 모니터링하고 보호하는 데 도움이 AWS 서비스 되는 다른를 사용하는 방법을 알아봅니다.

주제

- 용 Identity and Access Management(IAM) AWS Private Certificate Authority
- 프라이빗 CA에 대한 교차 계정 액세스의 보안 모범 사례
- 의 데이터 보호 AWS Private Certificate Authority
- 에 대한 규정 준수 검증 AWS Private Certificate Authority
- 의 인프라 보안 AWS Private Certificate Authority
- AWS Private Certificate Authority 고객 CP/CPS 프레임워크

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

에 액세스하려면가 요청을 인증하는 데 사용할 AWS 수 있는 자격 증명이 AWS Private CA 필요합니다. 다음 주제에서는 AWS Identity and Access Management (IAM)를 사용하여 CA에 액세스할 수 있는 보안 주체를 제어함으로써 Private Certificate Authority(CA)를 보호하는 방법을 설명합니다.

에서 작업하는 AWS Private CA기본 리소스는 인증 기관(CA)입니다. 사용자가 소유 또는 제어하는 모든 사설 CA는 다음과 같은 형식을 가진 Amazon 리소스 이름(ARN)으로 식별됩니다.

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

리소스 소유자는 AWS 리소스가 생성되는 AWS 계정의 보안 주체 엔터티입니다. 다음 예에서는 이 계정의 작동 방식을 설명합니다.

- 의 자격 증명을 사용하여 프라이빗 CA를 AWS 계정 루트 사용자 생성하는 경우 AWS 계정이 CA를 소유합니다.

  > **Important**
  > - 를 사용하여 CA AWS 계정 루트 사용자 를 생성하는 것은 권장하지 않습니다. CAs
  > - 에 액세스할 때마다 다중 인증(MFA)을 사용하는 것이 좋습니다 AWS Private CA.

- AWS 계정에서 IAM 사용자를 생성하는 경우 해당 사용자에게 프라이빗 CA를 생성할 수 있는 권한을 부여할 수 있습니다. 하지만 해당 사용자가 속한 계정이 CA를 소유합니다.
- AWS 계정에서 IAM 역할을 생성하고 프라이빗 CA를 생성할 수 있는 권한을 부여하는 경우 해당 역할을 수임할 수 있는 사람은 누구나 CA를 생성할 수 있습니다. 하지만 역할이 속한 계정은 프라이빗 CA를 소유하게 됩니다.

권한 정책은 누가 무엇에 액세스할 수 있는지를 나타냅니다. 다음 단원에서는 권한 정책을 생성하는 데 사용할 수 있는 옵션에 대해 설명합니다.

> **Note**
>
> 이 설명서에서는 컨텍스트에서 IAM 사용에 대해 설명합니다 AWS Private CA. IAM 서비스에 대한 자세한 내용은 다루지 않습니다. IAM 설명서 전체 내용은 IAM 사용 설명서를 참조하세요. IAM 정책 구문과 설명에 대한 자세한 내용은 AWS IAM 정책 참조를 참조하세요.

AWS Private CA API 작업 및 권한

IAM 자격 증명에 연결할 액세스 제어 및 권한 정책(자격 증명 기반 정책)을 설정할 때 다음 표를 참조로 사용합니다. 표의 첫 번째 열에는 각 AWS Private CA API 작업이 나열됩니다. 정책의 `Action` 요소에 작업을 지정합니다. 다음과 같이 남은 열에서 정보를 추가로 제공합니다.

AWS Private CA API 작업	필수 권한	리소스
CreateCertificateAuthority	`acm-pca:CreateCertificateAuthority` `acm-pca:TagCertificateAuthority` (태그가 있는 CA를 생성할 때만 필요)	`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/11223344-1234-1122-2233-112233445566`
DeleteCertificateAuthority	`acm-pca:DeleteCertificateAuthority`	`arn:aws:acm-pca:us-east-1:111122223`

AWS Private CA API 작업	필수 권한	리소스
		333 :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*
DescribeCertificateAuthorityAuditReport	acm-pca:DescribeCertificateAuthorityAuditReport	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*

AWS Private CA API 작업	필수 권한	리소스
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*
ImportCertificateAuthorityCertificate	acm-pca:ImportCertificateAuthorityCertificate	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*

AWS Private CA API 작업	필수 권한	리소스
IssueCertificate	acm-pca:IssueCertificate	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*
ListCertificateAuthorities	acm-pca:ListCertificateAuthorities	해당 사항 없음
ListPermissions	acm-pca:ListPermissions	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*
ListTags	acm-pca:ListTags	해당 사항 없음
PutPolicy	acm-pca:PutPolicy	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*
RevokeCertificate	acm-pca:RevokeCertificate	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*

AWS Private CA API 작업	필수 권한	리소스
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*

액세스 권한을 제공하려면 사용자, 그룹 또는 역할에 권한을 추가하세요.

- 의 사용자 및 그룹 AWS IAM Identity Center:

  권한 세트를 생성합니다. AWS IAM Identity Center 사용자 안내서에서 권한 세트 생성의 지침을 따릅니다.

- ID 제공업체를 통해 IAM에서 관리되는 사용자:

  ID 페더레이션을 위한 역할을 생성합니다. IAM 사용자 설명서의 Create a role for a third-party identity provider (federation)의 지침을 따릅니다.

- IAM 사용자:
  - 사용자가 맡을 수 있는 역할을 생성합니다. IAM 사용자 설명서에서 Create a role for an IAM user의 지침을 따릅니다.

- (권장되지 않음) 정책을 사용자에게 직접 연결하거나 사용자를 사용자 그룹에 추가합니다. IAM 사용 설명서에서 사용자(콘솔)에 권한 추가의 지침을 따릅니다.

AWS 관리형 정책

AWS Private CA 에는 관리자, 사용자 및 감사자를 위한 AWS Private CA 사전 정의된 AWS 관리형 정책 세트가 포함되어 있습니다. 이러한 정책을 이해하면 고객 관리형 정책을 구현하는 데 도움이 될 수 있습니다.

세부 정보 및 샘플 정책 코드를 보려면 아래 나열된 정책 중 하나를 선택합니다.

AWSPrivateCAFullAccess

무제한 관리 제어를 부여합니다.

정책 세부 정보의 JSON 목록은 AWSPrivateCAFullAccess를 참조하세요.

AWSPrivateCAReadOnly

읽기 전용 API 작업으로 제한된 액세스 권한을 부여합니다.

정책 세부 정보의 JSON 목록은 AWSPrivateCAReadOnly를 참조하세요.

AWSPrivateCAPrivilegedUser

CA 인증서를 발급하고 취소할 수 있는 권한을 부여합니다. 이 정책에는 다른 관리 기능이 없으며 최종 엔터티 인증서를 발급할 수 있는 기능이 없습니다. 권한은 사용자 정책과 상호 배타적입니다.

정책 세부 정보의 JSON 목록은 AWSPrivateCAPrivilegedUser를 참조하세요.

AWSPrivateCAUser

최종 엔터티 인증서를 발급하고 취소할 수 있는 기능을 부여합니다. 이 정책에는 관리 기능이 없으며 CA 인증서를 발급할 수 있는 기능이 없습니다. 권한은 PrivilegedUser 정책과 상호 배타적입니다.

정책 세부 정보의 JSON 목록은 AWSPrivateCAUser를 참조하세요.

AWSPrivateCAAuditor

읽기 전용 API 작업에 대한 액세스 권한과 CA 감사 보고서 생성 권한을 부여합니다.

정책 세부 정보의 JSON 목록은 AWSPrivateCAAuditor를 참조하세요.

AWSPrivateCAConnectorForKubernetesPolicy

Kubernetes용 AWS Private CA 커넥터에 대한 필수 권한을 부여합니다.

정책 세부 정보의 JSON 목록은 AWSPrivateCAConnectorForKubernetesPolicy를 참조하세요.

에 대한 AWS 관리형 정책 업데이트 AWS Private CA

다음 표에서는 서비스가 이러한 변경 사항을 추적하기 시작한 AWS Private CA 이후의 AWS 관리형 정책 업데이트에 대한 세부 정보를 봅니다. 의 모든 변경 사항에 대한 자동 알림을 받으려면 페이지에서 RSS 피드를 AWS Private CA구독하세요문서 기록.

관리형 정책 변경

변경	설명	Date
새 정책: AWS PrivateCA ConnectorForKubernetesPolicy	Kubernetes용 AWS Private CA 커넥터와 함께 사용할 수 있도록 도입된 새로운 관리형 정책입니다.	2025년 5월 19일
AWS PrivateCAPrivilegedUser 및 AWS PrivateCAUser - 업데이트된 정책	`StringLike` 로 `ArnLike`대체되고 `StringNotLike` 로 대체됩니다`ArnNotLike` . 와일드카드를 로 포함하도록 템플릿 ARN`arn:aws:acm-pca:::template` 을 업데이트했습니다`arn:aws:acm-pca:*:*:template` .	2025년 1월 22일
새 정책 이름: • `AWS PrivateCA FullAccess` • `AWS PrivateCA ReadOnly` • `AWS PrivateCA PrivilegedUser`	정책 이름 접두사가 `AWS CertificateManager PrivateCA` 에서 `AWS PrivateCA` 로 변경되었습니다. 기능은 변경되지 않습니다.	2023년 2월 13일

변경	설명	Date
• AWS PrivateCAAuditor • AWS PrivateCAUser		

고객 관리형 정책

를 AWS포함하여와 상호 작용하는 AWS 계정 루트 사용자 데를 사용하지 않는 것이 가장 좋습니다 AWS Private CA. 대신 AWS Identity and Access Management (IAM)를 사용하여 IAM 사용자, IAM 역할 또는 페더레이션 사용자를 생성합니다. 관리자 그룹을 만들고 여기에 자신을 추가합니다. 그런 다음 관리자로 로그인합니다. 필요에 따라 그룹에 사용자를 더 추가합니다.

사용자에게 할당할 수 있는 고객 관리형 IAM 정책을 생성하는 것도 좋은 방법입니다. 고객 관리형 정책은 AWS 계정에서 생성하여 여러 사용자, 그룹 및 역할에 연결할 수 있는 독립적인 자격 증명 기반 정책입니다. 이러한 정책은 사용자가 지정한 AWS Private CA 작업만 수행하도록 제한합니다.

사용자는 아래의 고객 관리형 정책 예를 사용하여 CA 감사 보고서를 생성할 수 있습니다. 이 정책은 예제이며, 원하는 AWS Private CA 작업을 선택할 수 있습니다. 더 많은 예시는 인라인 정책를 참조합니다.

고객 관리형 정책을 생성하는 방법

1. AWS 관리자의 자격 증명을 사용하여 IAM 콘솔에 로그인합니다.
2. 콘솔의 탐색 창에서 정책을 선택합니다.
3. 정책 생성을 선택합니다.
4. JSON 탭을 선택합니다.
5. 다음 정책을 복사하여 편집기에 붙여 넣습니다.

JSON

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

```
        }
    ]
}
```

6. 정책 검토를 선택합니다.
7. Name에 `PcaListPolicy`를 입력합니다.
8. (선택 사항) 설명을 입력합니다.
9. 정책 생성을 선택합니다.

관리자는 모든 IAM 사용자에게 정책을 연결하여 해당 사용자가 수행할 수 있는 AWS Private CA 작업을 제한할 수 있습니다. 권한 정책을 적용하는 방법은 IAM 사용 설명서의 IAM 사용자 권한 변경을 참조하세요.

인라인 정책

인라인 정책은 자신이 생성 및 관리하며, 사용자, 그룹 또는 역할에 직접 포함되는 정책입니다. 다음 정책 예제에서는 AWS Private CA 작업을 수행할 권한을 할당하는 방법을 보여줍니다. 인라인 정책에 대한 일반적인 내용은 IAM 사용 설명서의 인라인 정책 작업을 참조하십시오. AWS Management Console, AWS Command Line Interface (AWS CLI) 또는 IAM API를 사용하여 인라인 정책을 생성하고 포함할 수 있습니다.

> **Important**
>
> 에 액세스할 때마다 다중 인증(MFA)을 사용하는 것이 좋습니다 AWS Private CA.

주제

- 프라이빗 CA 목록
- 프라이빗 CA 인증서 검색
- 프라이빗 CA 인증서 가져오기
- 프라이빗 CA 삭제
- 생성 시 태그 지정: CA를 만들 때 CA에 태그 첨부
- 생성 시 태그 지정: 제한된 태그 지정
- 태그를 사용한 프라이빗 CA에 대한 액세스 제어
- 에 대한 읽기 전용 액세스 AWS Private CA

- 에 대한 전체 액세스 AWS Private CA

프라이빗 CA 목록

다음 정책을 통해 사용자는 계정의 모든 프라이빗 CA를 나열할 수 있습니다.

JSON

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

프라이빗 CA 인증서 검색

다음 정책은 사용자가 특정 프라이빗 CA 인증서를 검색하도록 허용합니다.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":{
      "Effect":"Allow",
      "Action":"acm-pca:GetCertificateAuthorityCertificate",
      "Resource":"arn:aws:acm-pca:us-east-1:123456789012:certificate-
authority/CA_ID/certificate/certificate_ID"
   }
}
```

프라이빗 CA 인증서 가져오기

다음 정책은 사용자가 프리이빗 CA 인증서를 가져오도록 허용합니다.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":{
      "Effect":"Allow",
      "Action":"acm-pca:ImportCertificateAuthorityCertificate",
      "Resource":"arn:aws:acm-pca:us-east-1:123456789012:certificate-
authority/CA_ID/certificate/certificate_ID"
   }
}
```

프라이빗 CA 삭제

다음 정책은 사용자가 특정 프라이빗 CA 인증서를 삭제하도록 허용합니다.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":{
      "Effect":"Allow",
      "Action":"acm-pca:DeleteCertificateAuthority",
      "Resource":"arn:aws:acm-pca:us-east-1:123456789012:certificate-
authority/CA_ID/certificate/certificate_ID"   }
}
```

생성 시 태그 지정: CA를 만들 때 CA에 태그 첨부

다음 정책은 사용자가 CA 생성 중에 태그를 적용할 수 있도록 허용합니다.

JSON

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

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

생성 시 태그 지정: 제한된 태그 지정

다음 태그 생성 시 태그 지정 정책은 CA를 생성하는 동안 키-값 페어 Environment=Prod를 사용하지 못하도록 합니다. 다른 키-값 페어로 태그를 지정할 수 있습니다.

JSON

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

태그를 사용한 프라이빗 CA에 대한 액세스 제어

다음 정책은 키-값 페어 Environment=PreProd인 CA에만 액세스를 허용합니다. 또한 새 CA에 이 태그를 포함해야 합니다.

JSON

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

에 대한 읽기 전용 액세스 AWS Private CA

다음 정책은 사용자가 사설 CA를 설명 및 나열하고 사설 CA 인증서 및 인증서 체인을 검색하도록 허용합니다.

JSON

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

```
            "acm-pca:DescribeCertificateAuthorityAuditReport",
            "acm-pca:ListCertificateAuthorities",
            "acm-pca:ListTags",
            "acm-pca:GetCertificateAuthorityCertificate",
            "acm-pca:GetCertificateAuthorityCsr",
            "acm-pca:GetCertificate"
        ],
        "Resource":"*"
    }
}
```

에 대한 전체 액세스 AWS Private CA

다음 정책은 사용자가 모든 AWS Private CA 작업을 수행하도록 허용합니다.

JSON

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

프라이빗 CA에 대한 교차 계정 액세스의 보안 모범 사례

AWS Private CA 관리자는 다른의 보안 주체(사용자, 역할 등)와 CA를 공유할 수 있습니다. AWS 계정공유가 수신되고 수락되면 보안 주체는 CA를 사용하여 AWS Private CA 또는 AWS Certificate Manager 리소스를 사용하여 최종 엔터티 인증서를 발급할 수 있습니다. 보안 주체는 CA를 사용하여를 사용하여 하위 CA 인증서를 발급할 수 있습니다 AWS Private CA.

> **Important**
>
> 교차 계정 시나리오에서 발급된 인증서와 관련된 요금은 인증서를 발급한 AWS 계정에 청구됩니다.

CA에 대한 액세스를 공유하기 위해 AWS Private CA 관리자는 다음 방법 중 하나를 선택할 수 있습니다.

- AWS Resource Access Manager (RAM)을 사용하여 CA를 다른 계정의 보안 주체 또는와 리소스로 공유합니다 AWS Organizations. RAM은 계정 간에 AWS 리소스를 공유하는 표준 방법입니다. RAM에 대한 자세한 내용은 AWS RAM 사용 설명서를 참조하세요. AWS Organizations에 대한 자세한 내용은 AWS Organizations 사용 설명서를 참조하세요.
- AWS Private CA API 또는 CLI를 사용하여 리소스 기반 정책을 CA에 연결하여 다른 계정의 보안 주체에 액세스 권한을 부여합니다. 자세한 내용은 리소스 기반 정책 단원을 참조하십시오.

이 가이드의 프라이빗 CA에 대한 액세스 제어 섹션에서는 단일 계정 시나리오와 교차 계정 시나리오 모두에서 CA에 대한 액세스 권한을 부여하는 워크플로를 제공합니다.

리소스 기반 정책

리소스 기반 정책은 사용자 ID 또는 역할 대신 리소스(이 경우 프라이빗 CA)에 생성하여 수동으로 연결하는 권한 정책입니다. 또는 자체 정책을 생성하는 대신 AWS 관리형 정책을 사용할 수 있습니다 AWS Private CA. 를 사용하여 리소스 기반 정책을 적용 AWS RAM 하면 AWS Private CA 관리자는 CA에 대한 액세스를 다른 AWS 계정의 사용자와 직접 또는를 통해 공유할 수 있습니다 AWS Organizations. 또는 AWS Private CA 관리자는 PCA APIs PutPolicy, GetPolicy 및 DeletePolicy 또는 해당 AWS CLI 명령 put-policy, get-policy 및 delete-policy를 사용하여 리소스 기반 정책을 적용하고 관리할 수 있습니다.

리소스 기반 정책에 대한 일반적인 내용은 자격 증명 기반 정책 및 리소스 기반 정책 및 정책을 사용한 액세스 제어를 참조하세요.

에 대한 AWS 관리형 리소스 기반 정책 목록을 보려면 AWS Resource Access Manager 콘솔에서 관리형 권한 라이브러리로 AWS Private CA이동하여 CertificateAuthority를 검색합니다. 다른 정책과 마찬가지로 정책을 적용하기 전에 테스트 환경에 정책을 적용하여 요구 사항을 충족하는지 확인하는 것이 좋습니다.

AWS Certificate Manager 프라이빗 CA에 대한 교차 계정 공유 액세스 권한이 있는 (ACM) 사용자는 CA에서 서명한 관리형 인증서를 발급할 수 있습니다. 교차 계정 발급자는 리소스 기반 정책의 제약을 받으며 다음과 같은 최종 엔터티 인증서 템플릿에만 액세스할 수 있습니다.

- EndEntityCertificate/V1
- EndEntityClientAuthCertificate/V1
- EndEntityServerAuthCertificate/V1
- BlankEndEntityCertificate_APIPassthrough/V1
- BlankEndEntityCertificate_APICSRPassthrough/V1
- SubordinateCACertificate_PathLen0/V1

정책 예시

이 섹션에서는 다양한 요구 사항에 맞는 교차 계정 정책의 예제를 제공합니다. 모든 경우에 다음 명령 패턴을 사용하여 정책을 적용합니다.

```
$ 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
```

관리자는 CA의 ARN을 지정하는 것 외에도 CA에 대한 액세스 권한을 부여할 계정 ID 또는 AWS Organizations ID를 제공합니다 AWS . 다음 각 정책의 JSON은 가독성을 위해 파일 형식으로 지정되지만 인라인 CLI 인수로 제공될 수도 있습니다.

> **Note**
>
> 아래에 표시된 JSON 리소스 기반 정책의 구조를 정확히 따라야 합니다. 보안 주체의 ID 필드 (AWS 계정 번호 또는 AWS 조직 ID)와 CA ARNs만 고객이 구성할 수 있습니다.

1. File: policy1.json – Sharing access to a CA with a user in a different account

   *555555555555*을 CA를 공유하는 AWS 계정 ID로 바꿉니다.

   리소스 ARN의 경우 다음을 고유한 값으로 바꿉니다.

   - *aws* - 파티션입니다 AWS . 예: , awsaws-us-govaws-cn, 등

- *us-east-1* - AWS 와 같이 리소스를 사용할 수 있는 리전입니다us-west-1.
- *111122223333* - 리소스 소유자의 AWS 계정 ID입니다.
- *11223344-1234-1122-2233-112233445566* - 인증 기관의 리소스 ID입니다.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [{
            "Sid": "ExampleStatementID",
            "Effect": "Allow",
            "Principal": {
                "AWS": "555555555555"
            },
            "Action": [
                "acm-pca:DescribeCertificateAuthority",
                "acm-pca:GetCertificate",
                "acm-pca:GetCertificateAuthorityCertificate",
                "acm-pca:ListPermissions",
                "acm-pca:ListTags"
            ],
            "Resource": "arn:aws:acm-pca:us-east-1:123456789012:certificate-
authority/CA_ID"
        },
        {
            "Sid": "ExampleStatementID2",
            "Effect": "Allow",
            "Principal": {
                "AWS": "555555555555"
            },
            "Action": [
                "acm-pca:IssueCertificate"
            ],
            "Resource": "arn:aws:acm-pca:us-east-1:123456789012:certificate-
authority/CA_ID",
            "Condition": {
                "StringEquals": {
                    "acm-pca:TemplateArn": "arn:aws:acm-pca:::template/
EndEntityCertificate/V1"
                }
            }
        }
    ]
```

```
}
```

2. 파일: policy2.json –를 통해 CA에 대한 액세스 공유 AWS Organizations

*o-a1b2c3d4z5*를 ID로 바꿉니다 AWS Organizations .

리소스 ARN의 경우 다음을 고유한 값으로 바꿉니다.

- *aws* - 파티션입니다 AWS . 예: , awsaws-us-govaws-cn, 등
- *us-east-1* - AWS 와 같이 리소스를 사용할 수 있는 리전입니다us-west-1.
- *111122223333* - 리소스 소유자의 AWS 계정 ID입니다.
- *11223344-1234-1122-2233-112233445566* - 인증 기관의 리소스 ID입니다.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ExampleStatementID3",
            "Effect": "Allow",
            "Principal": "*",
            "Action": "acm-pca:IssueCertificate",
            "Resource":"arn:aws:acm-pca:us-east-1:123456789012:certificate-
authority/CA_ID",
            "Condition": {
                "StringEquals": {
                    "acm-pca:TemplateArn": "arn:aws:acm-pca:::template/
EndEntityCertificate/V1",
                    "aws:PrincipalOrgID": "o-a1b2c3d4z5"
                },
                "StringNotEquals": {
                    "aws:PrincipalAccount": "111122223333"
                }
            }
        },
        {
            "Sid": "ExampleStatementID4",
            "Effect": "Allow",
            "Principal": "*",
            "Action": [
                "acm-pca:DescribeCertificateAuthority",
```

```
                "acm-pca:GetCertificate",
                "acm-pca:GetCertificateAuthorityCertificate",
                "acm-pca:ListPermissions",
                "acm-pca:ListTags"
            ],
            "Resource":"arn:aws:acm-pca:us-east-1:123456789012:certificate-
authority/CA_ID",
            "Condition": {
                "StringEquals": {
                    "aws:PrincipalOrgID": "o-a1b2c3d4z5"
                },
                "StringNotEquals": {
                    "aws:PrincipalAccount": "111122223333"
                }
            }
        }
    ]
}
```

의 데이터 보호 AWS Private Certificate Authority

AWS 공동 책임 모델의 데이터 보호에 적용됩니다 AWS Private Certificate Authority. 이 모델에 설명된 대로 AWS 는 모든를 실행하는 글로벌 인프라를 보호할 책임이 있습니다 AWS 클라우드. 사용자는 이 인프라에 호스팅되는 콘텐츠에 대한 통제 권한을 유지할 책임이 있습니다. 사용하는 AWS 서비스 의 보안 구성과 관리 태스크에 대한 책임도 사용자에게 있습니다. 데이터 프라이버시에 관한 자세한 내용은 데이터 프라이버시 FAQ를 참조하세요. 유럽의 데이터 보호에 대한 자세한 내용은 AWS 보안 블로그의 AWS 공동 책임 모델 및 GDPR 블로그 게시물을 참조하세요.

데이터 보호를 위해 자격 증명을 보호하고 AWS 계정 AWS IAM Identity Center 또는 AWS Identity and Access Management (IAM)를 사용하여 개별 사용자를 설정하는 것이 좋습니다. 이렇게 하면 개별 사용자에게 자신의 직무를 충실히 이행하는 데 필요한 권한만 부여됩니다. 또한 다음과 같은 방법으로 데이터를 보호하는 것이 좋습니다.

- 각 계정에 다중 인증(MFA)을 사용합니다.
- SSL/TLS를 사용하여 AWS 리소스와 통신합니다. TLS 1.2는 필수이며 TLS 1.3을 권장합니다.
- 를 사용하여 API 및 사용자 활동 로깅을 설정합니다 AWS CloudTrail. CloudTrail 추적을 사용하여 AWS 활동을 캡처하는 방법에 대한 자세한 내용은 AWS CloudTrail 사용 설명서의 CloudTrail 추적 작업을 참조하세요.
- 내의 모든 기본 보안 제어와 함께 AWS 암호화 솔루션을 사용합니다 AWS 서비스.

- Amazon S3에 저장된 민감한 데이터를 검색하고 보호하는 데 도움이 되는 Amazon Macie와 같은 고급 관리형 보안 서비스를 사용합니다.
- 명령줄 인터페이스 또는 API를 AWS 통해에 액세스할 때 FIPS 140-3 검증 암호화 모듈이 필요한 경우 FIPS 엔드포인트를 사용합니다. 사용 가능한 FIPS 엔드포인트에 대한 자세한 내용은 연방 정보 처리 표준(FIPS) 140-3을 참조하세요.

고객의 이메일 주소와 같은 기밀 정보나 중요한 정보는 태그나 이름 필드와 같은 자유 형식 텍스트 필드에 입력하지 않는 것이 좋습니다. 여기에는 AWS Private CA 또는 기타 AWS 서비스 에서 콘솔, API AWS CLI또는 AWS SDKs를 사용하여 작업하는 경우가 포함됩니다. 이름에 사용되는 태그 또는 자유 형식 텍스트 필드에 입력하는 모든 데이터는 청구 또는 진단 로그에 사용될 수 있습니다. 외부 서버에 URL을 제공할 때 해당 서버에 대한 요청을 검증하기 위해 자격 증명을 URL에 포함해서는 안 됩니다.

AWS Private CA 프라이빗 키의 스토리지 및 보안 규정 준수

프라이빗 CAs의 프라이빗 키는 AWS 관리형 하드웨어 보안 모듈(HSMs)에 저장됩니다. HSMs은 암호화 모듈에 대한 FIPS PUB 140-2 레벨 3 보안 요구 사항을 준수합니다.

Active Directory용 AWS Private CA 커넥터의 데이터 암호화

AWS Private CA AD용 커넥터는 커넥터, 템플릿, 디렉터리 등록, 서비스 보안 주체 이름 및 템플릿 그룹 액세스 제어 항목에 대한 고객 구성 데이터를 저장합니다. 이 데이터는 전송 및 저장 시 암호화됩니다. AD용 커넥터를 통해 발급된 인증서에 대한 정보는 AWS Private CA API의 GetCertificate 작업을 사용하여 검색할 수 있습니다. 발급된 인증서 또는 인증서를 요청하는 클라이언트 또는 시스템에 대한 정보는에서 저장하지 않습니다 AWS.

에 대한 규정 준수 검증 AWS Private Certificate Authority

타사 감사자는 여러 규정 준수 프로그램의 AWS Private Certificate Authority 일환으로의 보안 및 AWS 규정 준수를 평가합니다. 여기에는 SOC, PCI, FedRAMP, HIPAA 등이 포함됩니다.

특정 규정 준수 프로그램 범위의 AWS 서비스 목록은 규정 준수 AWS 프로그램 제공 범위 내 서비스규정 준수 서비스를 참조하세요. 일반 정보는 AWS 규정 준수 프로그램.

를 사용하여 타사 감사 보고서를 다운로드할 수 있습니다 AWS Artifact. 자세한 내용은 에서 보고서 다운로드 AWS Artifact에서 .

사용 시 규정 준수 책임은 데이터의 민감도, 회사의 규정 준수 목표 및 관련 법률과 규정에 따라 결정됩니다. AWS Private CA 는 규정 준수를 지원할 다음과 같은 리소스를 AWS 제공합니다.

- Amazon S3 버킷을 암호화해야 하는 조직을 위해 다음 주제에서는 AWS Private CA 자산을 수용하도록 암호화를 구성하는 방법에 대해 설명합니다.
  - 감사 보고서 암호화
  - CRL 암호화
- 보안 및 규정 준수 빠른 시작 안내서 – 이 배포 안내서에서는 아키텍처 고려 사항에 관해 설명하고 AWS에서 보안 및 규정 준수에 중점을 둔 기본 환경을 배포하기 위한 단계를 제공합니다.
- HIPAA 보안 및 규정 준수를 위한 설계 백서 -이 백서에서는 기업이 AWS 를 사용하여 HIPAA 준수 애플리케이션을 생성하는 방법을 설명합니다.
- AWS 규정 준수 리소스 -이 워크북 및 가이드 모음은 업계 및 위치에 적용될 수 있습니다.
- AWS Config 개발자 안내서의 규칙을 사용하여 리소스 평가 -이 AWS Config 서비스는 리소스 구성이 내부 관행, 업계 지침 및 규정을 얼마나 잘 준수하는지 평가합니다.
- AWS Security Hub CSPM -이 AWS 서비스는 보안 업계 표준 및 모범 사례 준수 여부를 확인하는 데 도움이 AWS 되는 내 보안 상태에 대한 포괄적인 보기를 제공합니다.

프라이빗 CA에서 감사 보고서 사용

프라이빗 CA가 발급 또는 취소한 모든 인증서를 나열하는 감사 보고서를 만들 수 있습니다. 보고서는 입력 시 지정한 신규 또는 기존 S3 버킷에 저장됩니다.

감사 보고서에 암호화 보호를 추가하는 방법에 대한 자세한 내용은 감사 보고서 암호화 단원을 참조하십시오 .

감사 보고서 파일은 다음과 같은 경로와 파일 이름을 가집니다. Amazon S3 버킷의 ARN은 `amzn-s3-demo-bucket`에 대한 값입니다. `CA_ID`는 발급 CA의 고유 식별자입니다. `UUID`은 감사 보고서의 고유 식별자입니다.

```
amzn-s3-demo-bucket/audit-report/CA_ID/UUID.[json|csv]
```

30분마다 새 보고서를 생성하여 버킷에서 다운로드할 수 있습니다. 다음 예제에서는 CSV로 구분된 보고서를 보여 줍니다.

```
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
```

다음 예제에서는 JSON 형식의 보고서를 보여줍니다.

```
[
   {
      "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**
>
> 가 인증서를 AWS Certificate Manager 갱신하면 프라이빗 CA 감사 보고서가 `requestedByServicePrincipal` 필드를 로 채웁니다`acm.amazonaws.com`. 이는 AWS Certificate Manager 서비스가 고객을 대신하여 AWS Private CA API의 `IssueCertificate` 작업을 호출하여 인증서를 갱신했음을 나타냅니다.

감사 보고서를 위한 Amazon S3 버킷 준비

> **Important**
>
> AWS Private CA 는 Amazon S3 객체 잠금 사용을 지원하지 않습니다. 버킷에서 객체 잠금을 활성화하면 AWS Private CA 가 버킷에 감사 보고서를 작성할 수 없습니다.

감사 보고서를 저장하려면 Amazon S3 버킷을 준비해야 합니다. 자세한 내용은 S3 버킷을 어떻게 생성합니까?를 참조하세요.

S3 버킷은가 지정한 S3 버킷에 AWS Private CA 액세스하고 쓸 수 있도록 허용하는 권한 정책에 의해 보호되어야 합니다. 권한 있는 사용자 및 서비스 보안 주체는가 버킷에 객체를 배치할 수 AWS Private CA 있는 `Put` 권한과 객체를 검색할 수 있는 `Get` 권한이 필요합니다. 아래 표시된 정책을 적용하여 프라이빗 CA의 AWS 계정과 ARN 모두에 대한 액세스를 제한하는 것이 좋습니다. 또는 aws:SourceOrgID 조건 키를 사용하여의 특정 조직에 대한 액세스를 제한할 수 있습니다 AWS Organizations. 버킷 정책에 대한 자세한 내용은 Amazon Simple Storage Service의 버킷 정책을 참조하세요.

감사 보고서 생성

콘솔이나 AWS CLI에서 감사 보고서를 생성할 수 있습니다.

감사 보고서를 생성하는 방법(콘솔)

1. AWS 계정에 로그인하고 https://console.aws.amazon.com/acm-pca/home AWS Private CA 콘솔을 엽니다.
2. 프라이빗 인증 기관 페이지 목록에서 프라이빗 CA를 선택합니다.
3. 작업 메뉴에서 감사 보고서 생성을 선택합니다.

4. 감사 보고서 대상 아래의 새 S3 버킷을 생성하시겠습니까?에서 예를 선택하고 고유한 버킷 이름을 입력하거나, 아니요를 선택하고 목록에서 기존 버킷을 선택합니다.

   예를 선택하면가 기본 정책을 AWS Private CA 생성하여 버킷에 연결합니다. 기본 정책에는 특정 AWS 계정에 대한 액세스를 제한하는 `aws:SourceAccount` 조건 키가 포함되어 있습니다. 액세스를 추가로 제한하려면 이전 예제와 같이 정책에 다른 조건 키를 추가할 수 있습니다.

   아니요를 선택한 경우 감사 보고서를 생성하기 전에 먼저 정책을 버킷에 연결해야 합니다. 감사 보고서를 위한 Amazon S3 버킷 준비에 설명된 정책 패턴을 사용합니다. 정책 연결에 대한 자세한 내용은 Amazon S3 콘솔을 사용하여 버킷 정책 추가를 참조하세요.
5. 출력 형식에서 JavaScript 객체 표기법의 경우 JSON을 선택하고 쉼표로 구분된 값의 경우 CSV를 선택합니다.
6. 감사 보고서 생성을 선택합니다.

감사 보고서를 생성하는 방법(AWS CLI)

1. 사용할 S3 버킷이 아직 없다면 새로 생성합니다.
2. 버킷에 정책 연결 감사 보고서를 위한 Amazon S3 버킷 준비에 설명된 정책 패턴을 사용합니다. 정책 연결에 대한 자세한 내용은 Amazon S3 콘솔을 사용하여 버킷 추가를 참조하세요.
3. create-certificate-authority-audit-report 명령을 사용하여 감사 보고서를 생성하고 준비된 S3 버킷에 배치합니다.

```
$ aws acm-pca create-certificate-authority-audit-report \
--certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566 \
--s3-bucket-name amzn-s3-demo-bucket \
--audit-report-response-format JSON
```

감사 보고서 검색

검사를 위해 감사 보고서를 검색하려면 Amazon S3 콘솔, API, CLI 또는 SDK를 사용합니다. 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 객체 다운로드를 참조하세요.

감사 보고서 암호화

선택적으로 감사 보고서가 포함된 Amazon S3 버킷에서 암호화를 구성할 수 있습니다.는 S3의 자산에 대해 두 가지 암호화 모드를 AWS Private CA 지원합니다.

- Amazon S3 관리형 AES-256 키를 사용한 자동 서버 측 암호화.
- 사양에 맞게 AWS KMS key 구성된 AWS Key Management Service 및를 사용한 고객 관리형 암호화.

> **Note**
>
> AWS Private CA 는 S3에서 자동으로 생성된 기본 KMS 키 사용을 지원하지 않습니다.

다음 절차에서는 각 암호화 옵션을 설정하는 방법에 대해 설명합니다.

자동 암호화를 구성하려면

S3 서버 측 암호화를 활성화하려면 다음 단계를 수행하십시오.

1. https://console.aws.amazon.com/s3/에서 S3 콘솔을 엽니다.
2. 버킷 테이블에서 AWS Private CA 자산을 보관할 버킷을 선택합니다.
3. 버킷의 페이지에서 속성 탭을 선택합니다.
4. 기본 암호화 카드를 선택합니다.
5. 활성화를 선택합니다.
6. Amazon S3 키(SSE-S3)를 선택합니다.
7. 변경 사항 저장(Save Changes)을 선택합니다.

사용자 지정 암호화를 구성하려면

사용자 지정 키를 사용하여 암호화를 활성화하려면 다음 단계를 수행하십시오.

1. https://console.aws.amazon.com/s3/에서 S3 콘솔을 엽니다.
2. 버킷 테이블에서 AWS Private CA 자산을 보관할 버킷을 선택합니다.
3. 버킷의 페이지에서 속성 탭을 선택합니다.
4. 기본 암호화 카드를 선택합니다.
5. 활성화를 선택합니다.
6. AWS Key Management Service 키(SSE-KMS)를 선택합니다.
7. AWS KMS 키에서 선택 또는 AWS KMS key ARN 입력을 선택합니다.
8. 변경 사항 저장(Save Changes)을 선택합니다.

9. (옵션) KMS 키가 아직 없는 경우 다음 AWS CLI create-key 명령을 사용하여 생성합니다.

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

이 명령의 출력 화면에는 KMS 키의 키 ID와 Amazon 리소스 이름(ARN)이 포함됩니다. 다음은 예제 출력입니다.

```
{
    "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. 다음 단계를 사용하여 AWS Private CA 서비스 보안 주체에게 KMS 키를 사용할 수 있는 권한을 부여합니다. 모든 KMS 키는 기본적으로 비공개입니다. 따라서 리소스 소유자만 KMS 키를 사용해서 데이터를 암호화 및 해독할 수 있습니다. 그러나 리소스 소유자가 원한다면 다른 사용자 및 리소스에 KMS 키에 대한 액세스 권한을 부여할 수 있습니다. 서비스 보안 주체는 KMS 키가 저장된 위치와 동일한 리전에 있어야 합니다.

    a. 먼저 다음 get-key-policy 명령을 사용하는 `policy.json`로 KMS 키에 대한 기본 정책을 저장합니다.

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

    b. 텍스트 편집기에서 `policy.json` 파일을 엽니다. 다음 정책 설명 중 하나를 선택하여 기존 정책에 추가합니다.

    Amazon S3 버킷 키가 활성화된 경우 다음 명령문을 사용합니다.

    ```
    {
       "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 버킷 키가 비활성화된 경우 다음 명령문을 사용합니다.

```
{
   "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. 마지막으로 아래 put-key-policy 명령을 사용하여 업데이트된 정책을 적용합니다.

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

의 인프라 보안 AWS Private Certificate Authority

관리형 서비스인는 AWS 글로벌 네트워크 보안으로 보호 AWS Private Certificate Authority 됩니다. AWS 보안 서비스 및가 인프라를 AWS 보호하는 방법에 대한 자세한 내용은 AWS 클라우드 보안을 참조하세요. 인프라 보안 모범 사례를 사용하여 환경을 설계하려면 보안 원칙 AWS Well-Architected Framework의 인프라 보호를 참조하세요 AWS .

AWS 게시된 API 호출을 사용하여 네트워크를 AWS Private CA 통해에 액세스합니다. 클라이언트는 다음을 지원해야 합니다.

- Transport Layer Security(TLS). TLS 1.2는 필수이며 TLS 1.3을 권장합니다.
- DHE(Ephemeral Diffie-Hellman) 또는 ECDHE(Elliptic Curve Ephemeral Diffie-Hellman)와 같은 완전 전송 보안(PFS)이 포함된 암호 제품군. Java 7 이상의 최신 시스템은 대부분 이러한 모드를 지원합니다.

AWS Private CA VPC 엔드포인트(AWS PrivateLink)

인터페이스 VPC 엔드포인트를 구성 AWS Private CA 하여 VPC와 간에 프라이빗 연결을 생성할 수 있습니다. 인터페이스 엔드포인트는 AWS Private CA API 작업에 비공개로 액세스하기 위한 기술AWS PrivateLink인 로 구동됩니다.는 개방형 인터넷에 노출되지 않도록 VPC와 AWS Private CA Amazon 네트워크 간의 모든 네트워크 트래픽을 AWS PrivateLink 라우팅합니다. 각 VPC 엔드포인트는 하나 이상의 탄력적 네트워크 인터페이스 및 VPC 서브넷의 프라이빗 IP 주소로 표현됩니다.

인터페이스 VPC 엔드포인트는 인터넷 게이트웨이, NAT 디바이스, VPN 연결 또는 Direct Connect 연결 AWS Private CA 없이 VPC를에 직접 연결합니다. VPC에 있는 인스턴스는 퍼블릭 IP 주소가 없어도 AWS Private CA API와 통신할 수 있습니다.

VPC를 AWS Private CA 통해를 사용하려면 VPC 내에 있는 인스턴스에서 연결해야 합니다. 또는 AWS Virtual Private Network (Site-to-Site VPN) 또는를 사용하여 프라이빗 네트워크를 VPC에 연결할 수 있습니다 Direct Connect. 에 대한 자세한 내용은 Amazon VPC 사용 설명서의 VPN 연결을 Site-to-Site VPN참조하세요. https://docs.aws.amazon.com/vpc/latest/userguide/vpn-connections.html Direct Connect에 대한 자세한 내용은 Direct Connect 사용 설명서의 연결 생성을 참조하세요.

AWS Private CA 에서는를 사용할 필요가 AWS PrivateLink없지만 추가 보안 계층으로 사용하는 것이 좋습니다. AWS PrivateLink 및 VPC 엔드포인트에 대한 자세한 내용은를 통한 서비스 액세스를 참조하세요 AWS PrivateLink.

AWS Private CA VPC 엔드포인트에 대한 고려 사항

에 대한 인터페이스 VPC 엔드포인트를 설정하기 전에 다음 고려 사항에 유의 AWS Private CA하세요.

- AWS Private CA 는 일부 가용 영역에서 VPC 엔드포인트를 지원하지 않을 수 있습니다. VPC 엔드포인트를 생성할 때는 먼저 관리 콘솔에서 지원을 확인하세요. 지원되지 않는 가용 영역은 “이 가용 영역에서 서비스가 지원되지 않음”으로 표시됩니다.
- VPC 엔드포인트는 교차 리전 요청을 지원하지 않습니다. API 호출을 AWS Private CA(으)로 발행할 계획인 동일 리전에서 엔드포인트를 생성해야 합니다.
- VPC 엔드포인트는 Amazon Route 53을 통해 Amazon이 제공하는 DNS만 지원합니다. 자신의 DNS를 사용하는 경우에는 조건적인 DNS 전송을 사용할 수 있습니다. 자세한 내용은 Amazon VPC 사용 설명서의 DHCP 옵션 세트를 참조하세요.
- VPC 엔드포인트에 연결된 보안 그룹은 VPC의 프라이빗 서브넷에서 443 포트로 들어오는 연결을 허용해야 합니다.

AWS Private CA API는 현재 다음에서 VPC 엔드포인트를 지원합니다 AWS 리전.

- 미국 동부(오하이오)
- 미국 동부(버지니아 북부)
- 미국 서부(캘리포니아 북부)
- 미국 서부(오리건)
- 아프리카(케이프타운)
- 아시아 태평양(홍콩)
- 아시아 태평양(하이데라바드)
- 아시아 태평양(자카르타)
- 아시아 태평양(멜버른)
- 아시아 태평양(뭄바이)
- 아시아 태평양(오사카)
- 아시아 태평양(서울)
- 아시아 태평양(싱가포르)

- 아시아 태평양(시드니)
- 아시아 태평양(도쿄)
- 캐나다(중부)
- 캐나다 서부(캘거리)
- 유럽(프랑크푸르트)
- 유럽(아일랜드)
- 유럽(런던)
- 유럽(밀라노)
- 유럽(파리)
- 유럽(스페인)
- 유럽(스톡홀름)
- 유럽(취리히)
- 이스라엘(텔아비브)
- Middle East (Bahrain)
- 중동(UAE)
- 남아메리카(상파울루)

에 대한 VPC 엔드포인트 생성 AWS Private CA

https://console.aws.amazon.com/vpc/ 또는에서 VPC 콘솔을 사용하여 AWS Private CA 서비스에 대한 VPC 엔드포인트를 생성할 수 있습니다 AWS Command Line Interface. 자세한 내용은 Amazon VPC 사용 설명서의 인터페이스 엔드포인트 생성 절차를 참조하세요.는 VPC 내의 모든 API 작업에 대한 호출을 AWS Private CA 지원합니다.

엔드포인트에서 프라이빗 DNS 호스트 이름을 활성화하면 기본 AWS Private CA 엔드포인트가 VPC 엔드포인트로 확인됩니다. 기본 서비스 엔드포인트의 전체 목록은 서비스 엔드포인트 및 할당량을 참조하십시오.

프라이빗 DNS 호스트 이름을 활성화하지 않은 경우, Amazon VPC는 다음 형식으로 사용할 수 있는 DNS 엔드포인트를 제공합니다.

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

> **Note**
>
> 값 ##은 미국 동부(오하이오) AWS 리전과 AWS Private CA같이에서 지원하는 리전us-east-2의 리전 식별자를 나타냅니다. AWS Private CA목록은 AWS Certificate Manager 프라이빗 인증 기관 엔드포인트 및 할당량을 참조하세요.

자세한 내용은 Amazon AWS Private CA VPC 사용 설명서의 VPC 엔드포인트(AWS PrivateLink)를 참조하세요.

에 대한 VPC 엔드포인트 정책 생성 AWS Private CA

에 대한 Amazon VPC 엔드포인트 정책을 생성 AWS Private CA 하여 다음을 지정할 수 있습니다.

- 작업을 수행할 수 있는 보안 주체.
- 수행할 수 있는 작업
- 작업을 수행할 수 있는 리소스

자세한 내용은 Amazon VPC 가이드의 VPC 엔드포인트를 사용하여 서비스에 대한 액세스 제어를 참조하세요.

예 - AWS Private CA 작업에 대한 VPC 엔드포인트 정책

엔드포인트에 연결되면 다음 정책은 모든 보안 주체에 대해 AWS Private CA 작업 , `IssueCertificate`, `DescribeCertificateAuthority`, `GetCertificate`, `GetCertificateAuthorityCertificate`및에 대한 액세스 권한을 부여합니다`ListPermissions`. `ListTags` 각 스탠자의 리소스는 사설 CA입니다. 첫 번째 스탠자는 지정된 사설 CA 및 인증서 템플릿을 사용하여 최종 엔터티 인증서의 생성을 승인합니다. 사용 중인 템플릿을 제어하지 않으려는 경우에는 `Condition` 섹션이 필요하지 않습니다. 그러나 이 옵션을 제거하면 모든 보안 주체가 CA 인증서와 최종 엔터티 인증서를 생성할 수 있습니다.

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

듀얼 스택 엔드포인트 지원

AWS Private Certificate Authority 는 IPv4 및 IPv6 클라이언트를 모두 지원하는 듀얼 스택 퍼블릭 엔드포인트를 제공합니다. 듀얼 스택 엔드포인트를 사용하면 클라이언트가 IPv4 또는 IPv6 주소를 AWS Private CA 사용하여와 통신할 수 있습니다. Active Directory의 AWS Private CA 경우 및 SCEP용 AWS Private CA 커넥터는 듀얼 스택 엔드포인트도 지원합니다.

의 AWS Private CA 듀얼 스택 퍼블릭 엔드포인트는 IPv4 및 IPv6 클라이언트를 모두 `https://acm-pca.your-region.api.aws` 지원합니다. AWS Private CA 는를 사용하여 가상 프라이빗 클라우드(VPC)에서 IPv4 및 IPv6를 통해 비공개로 액세스할 수도 있습니다 AWS PrivateLink. 에 대한 프라이빗 인터페이스 VPC 엔드포인트 생성에 대한 자세한 내용은 섹션을 AWS Private CA참조하세요AWS Private CA VPC 엔드포인트(AWS PrivateLink).

자세한 정보는 다음 자료를 참조하세요.

- VPC 및 서브넷의 IP 주소 지정
- VPC에 대한 IPv6 지원

IAM 및에서 IPv6 주소 사용 AWS Private CA

IPv6를 AWS Private Certificate Authority 통해에 액세스하기 전에 IP 주소 제한이 포함된 IAM 정책이 IPv6 주소 범위를 포함하도록 업데이트되었는지 확인합니다. IP 기반 정책이 IPv6 주소를 처리하도록 업데이트되지 않은 경우, 클라이언트가 IPv6을 사용할 때 액세스 권한을 잘못 잃거나 얻을 수 있습니다. AWS Private CA 및 듀얼 스택 지원에 대한 자세한 내용은 섹션을 참조하세요듀얼 스택 엔드포인트 지원.

> **Important**
>
> 이러한 문은 어떤 작업도 허용하지 않습니다. 이러한 문은 특정 작업을 허용하는 다른 문과 함께 사용하세요.

다음 문은 IPv4 주소 `192.0.2.*` 범위에서 시작된 요청에 대한 모든 AWS Private CA 권한에 대한 액세스를 명시적으로 거부합니다. 이 범위를 벗어나는 모든 IP 주소는 명시적으로 거부된 AWS Private CA 권한이 아닙니다. 모든 IPv6 주소가 거부된 범위를 벗어나므로이 문은 IPv6 주소에 대한 AWS Private CA 권한을 명시적으로 거부하지 않습니다.

```
{
    "Sid": "DenyPrivateCAPermissions",
    "Effect": "Deny",
    "Action": [
        "acm-pca:*"
    ],
    "Resource": "*",
    "Condition": {
        "NotIpAddress": {
            "aws:SourceIp": [
                "192.0.2.0/24"
            ]
        }
    }
```

```
}
```

다음 예제와 같이 IPv4(`192.0.2.0/24`) 및 IPv6(`2001:db8::/32`) 주소 범위를 모두 거부하도록 `Condition` 요소를 수정할 수 있습니다.

```
{
    "Sid": "DenyPrivateCAPermissions",
    "Effect": "Deny",
    "Action": [
        "acm-pca:*"
    ],
    "Resource": "*",
    "Condition": {
        "NotIpAddress": {
            "aws:SourceIp": [
                "192.0.2.0/24",
                "2001:db8::/32"
            ]
        }
    }
}
```

AWS Private Certificate Authority 고객 CP/CPS 프레임워크

AWS Private Certificate Authority 는 온프레미스 CA 운영에 대한 투자 및 유지 관리 비용 없이 루트 및 하위 CAs를 포함한 인증 기관(CA) 계층 구조를 생성할 수 있는 인프라 서비스를 제공합니다. AWS Private CA 를 사용하여 CA 계층 구조를 생성하는 경우 사용자와 간에 공동 책임이 있습니다 AWS Private CA. 공동 책임 모델은 서비스가 AWS 운영되는 시설의 물리적 보안을 운영, 관리 및 제어함에 따라 운영 부담을 줄이는 데 도움이 될 수 있습니다. 인증 기관의 책임과 관리(CA 리소스 생성 및 삭제, 트러스트 앵커 배포, PKI 계층 생성, 인증 정책 및 관행, CA 간 공유를 허용하거나 거부하기 위한 구성 AWS 계정, 템플릿 사용에 대한 정책, 감사, 업무 분리를 포함한 액세스 제어, 기타 CA 구성 및 정책 포함)를 맡습니다. 사용한 서비스, 해당 서비스를 IT 환경에 통합, 관련 법률 및 규정에 따라 책임이 다르므로 선택하는 서비스를 신중하게 고려해야 합니다. 자세한 내용은 AWS 클라우드 보안 공동 책임 모델을 참조하세요.

프라이빗 인증 기관에 대한 인증서 정책(CP) 또는 인증 관행 문(CPS)을 생성하는 것은 퍼블릭 키 인프라(PKI)를 관리하는 데 중요한 부분입니다. CP는 PKI에 대한 모든 요구 사항/규칙을 정의하고 CPS는 CP 요구 사항을 충족하는 방법을 설명합니다. 사용자는 PKI의 인증 기관으로 CP 및 CPS를 생성할 책임이 있습니다.는 AWS 시스템 및 조직 제어(SOC) 2 보고서와 같은 AWS 제어 및 규정 준수 문서를

AWS Private CA 제공합니다.를 사용하여 CP 및 CPS를 생성하고 필요에 따라 제어 평가 및 확인 절차를 수행할 수 있습니다. AWS SOC 보고서는가 주요 규정 준수 제어 및 목표를 AWS 달성하는 방법을 보여주는 독립적인 타사 검사 보고서입니다. 보고서의 목적은 사용자와 감사자가 운영 및 규정 준수를 지원하기 위해 설정된 AWS 제어를 이해하는 데 도움이 되는 것입니다.

이 문서는 RFC 3647에 부합하는 프레임워크를 제공하여 CP 및 CPS를 작성하는 데 도움이 되며 사용자와 간의 공동 책임을 식별합니다 AWS Private CA. AWS Private CA 가 규정 준수 책임이 있는 CP/CPS 요구 사항의 섹션은 "공유" 또는 "AWS Private Certificate Authority"로 식별되며, 관련 "보충 정보"가 관련 CP/CPS 요구 사항을 AWS Private CA 충족하는 방법을 이해하는 데 도움이 됩니다. 예를 들어 요구 사항 5(4.5.1)는 AWS Private CA 책임이며 AWS , SOC 2 보고서의 섹션 D.6에서 해당 제어 언어를 찾아 CP/CPS를 완료하는 데 도움을 받을 수 있습니다. AWS SOC 보고서 및 SOC 보고서에 대한 액세스를 요청하는 방법에 대한 자세한 내용은 SOC FAQs 페이지를 참조하십시오.

CP/CPS 요구 사항 및 책임

CP/CPS 요구 사항	책임	추가 정보
1. Introduction (All)	You	개요, 문서 이름 및 식별, PKI 참가자, 인증서 사용, 정책 관리, PKI와 관련된 정의 및 약어를 문서화하는 것은 사용자의 책임입니다.
2. Publication and Repository Responsibilities (All)	You	PKI와 관련된 정의를 문서화하는 것은 사용자의 책임입니다.
3. Identification and Authentication (All)	You	인증서 발급 전에 CA 또는 등록 기관(RA)에 최종 사용자 인증서 신청자의 자격 증명 및/또는 기타 속성을 인증하는 데 사용되는 절차를 문서화하는 것은 사용자의 책임입니다.
4. Certificate Life-Cycle Operational Requirements (4.4.1 — 4.4.6, 4.4.9 — 4.4.11)	Shared	인증서의 수명 주기와 관련하여 CA, 주체 CAs, RAs, 구독자 또는 기타 참가자를 발급할 때 적용되는 요구 사항을 지정할 책임은 사용자에게 있습니다.

CP/CPS 요구 사항	책임	추가 정보
		AWS Private CA 는 해지 상태 확인을 지원하는 두 가지 완전 관리형 메커니즘, 즉 4.4.9 및 4.4.10를 충족하는 데 도움이 되는 온라인 인증서 상태 프로토콜(OCSP) 및 인증서 해지 목록(CRLs)을 제공합니다.
4. Certificate Life-Cycle Operational Requirements (4.4.7, 4.4.8, 4.4.12)	N/A	AWS Private CA 는 인증서 재키, 인증서 수정 또는 키 에스크로 및 복구를 지원하지 않습니다.
5. Facility, Management, and Operational Controls (4.5.1)	AWS Private CA	AWS Private CA SOC 2 Type 2 보고서의 범위 내에 있는이 섹션의 요구 사항을 충족하는 데 도움이 되는 액세스 제어를 상속합니다(섹션 D.6 물리적 보안 및 환경 보호 참조). **Note** 사용자는 AWS 환경에서 내보내거나 전송한 CA 데이터의 물리적 보안 및 데이터 분류에 대한 책임이 있지만, 저장된 CA 데이터의 물리적 보안에는 책임이 없습니다 AWS.

CP/CPS 요구 사항	책임	추가 정보
5. Facility, Management, and Operational Controls (4.5.2)	Shared	PKI 환경 운영에 대한 신뢰할 수 있는 역할을 정의하는 것과 관련된이 섹션의 요구 사항을 충족하는 것은 사용자의 책임입니다. AWS Private CA 는 암호화 모듈의 물리적 액세스와 관련된 신뢰할 수 있는 역할을 유지합니다.
5. Facility, Management, and Operational Controls (4.5.3)	Shared	신뢰할 수 있는 사람의 신원 조회, 교육 및 징계 조치 절차와 관련된이 섹션의 요구 사항을 충족할 책임은 사용자에게 있습니다. AWS Private CA SOC 2 Type 2 보고서의 범위에 속하는 AWS 직원에 대한 신원 조회, 교육 및 징계 조치 절차와 관련된 제어를 상속합니다(섹션 A. 정책, A.1 제어 환경, B. 커뮤니케이션 및 D.1 보안 조직 및 D.2 직원 사용자 액세스 참조).

CP/CPS 요구 사항	책임	추가 정보
5. Facility, Management, and Operational Controls (4.5.4)	Shared	CloudTrail 및 감사 보고 로그와 CloudWatch 알림을 활성화, 보존 구성 및 보호할 책임은 사용자에게 있습니다. 또한 사용자는 로그 처리 절차를 생성하고 이 섹션의 요구 사항을 충족하는 AWS Private CA 서비스 사용에 대한 취약성 평가를 수행할 책임이 있습니다. 로그 가용성과 관련된 제어를 상속합니다. 물리적 액세스/사이트 보안 CA/RA 구성 관리, AWS 인프라 로그의 보안 및 AWS Private CA SOC 2 Type 2 보고서 범위 내에 있는 AWS 인프라의 취약성 평가(A.1 제어 환경, 섹션 C.1 서비스 약정, D.2 직원 사용자 액세스, D.3 논리적 보안, D.6 물리적 보안 및 환경 보호 D.7 변경 관리, D.8 데이터 무결성, 가용성, 및 중복성, 및 E.1 모니터링 활동).
5. Facility, Management, and Operational Controls (4.5.5)	Shared	이 섹션의 요구 사항을 충족하는 백업 및 보존 기간을 구성하는 것은 사용자의 책임입니다. AWS Private CA SOC 2 Type 2 보고서의 범위 내에 있는 로그의 가용성(구성 시)과 관련된 제어를 상속합니다(D.8 데이터 무결성, 가용성 및 중복성 참조).

CP/CPS 요구 사항	책임	추가 정보
5. Facility, Management, and Operational Controls (4.5.6)	N/A	AWS Private CA 는 키 전환을 지원하지 않습니다.
5. Facility, Management, and Operational Controls (4.5.7)	Shared	이 섹션의 요구 사항을 충족하는 사용 관련 인시던트 및 침해 처리 절차를 구현 AWS Private CA 하는 것은 사용자의 책임입니다. AWS Private CA SOC 2 Type 2 개인정보 보호 보고서의 범위에 속하는이 섹션의 요구 사항을 충족하는 데 도움이 되는 물리적 사이트 주택 및 인프라 운영과 관련된 인시던트, 침해 처리 절차, 비즈니스 연속성 및 재해 복구 절차를 상속합니다 (D.8 데이터 무결성, 가용성 및 중복성 및 D.10 개인정보 보호 참조).
5. Facility, Management, and Operational Controls (4.5.8)	You	CA 및 RA 아카이브 레코드의 관리인의 자격 증명을 포함하여 CA 또는 RA의 종료 및 해지 절차와 관련된 요구 사항을 문서화해야 합니다.
6. Technical Controls (4.6.1)	Shared	PKI의 키 생성 및 설치 요구 사항을 문서화하는 것은 사용자의 책임입니다. AWS Private CA 는 CA 키 생성용으로 인증된 FIPS 140-3 레벨 3의 암호화 모듈을 제공합니다.

CP/CPS 요구 사항	책임	추가 정보
6. Technical Controls (4.6.2)	Shared	사용자는 암호화 표준 요구 사항 및 다중 사용자 제어와 같은 프라이빗 키 보호 및 암호화 모듈 엔지니어링 제어를 문서화할 책임이 있습니다. AWS Private CA 는 CA 키 생성 및 HSMs에 대한 2자 물리적 액세스 제어용으로 인증된 FIPS 140-3 레벨 3 암호화 모듈을 제공합니다.
6. Technical Controls (4.6.3)	You	퍼블릭 키 보관 및 인증서 운영 기간과 같은 키 페어 관리의 다른 측면을 문서화하는 것은 사용자의 책임입니다.
6. Technical Controls (4.6.4)	N/A	AWS AWS Private CA HSMs 은 항상 온라인 상태이며 "활성화 데이터"라는 개념이 없습니다. **Note** CA 생성 및 인증서 발급 기능을 적절하게 제한하기 위해 Private CA에 대한 사용자 액세스 제어를 구현할 책임은 사용자에게 있습니다.

CP/CPS 요구 사항	책임	추가 정보
6. Technical Controls (4.6.5)	Shared	프라이빗 CA 사용에 대한 컴퓨터 보안 제어를 문서화하는 것은 사용자의 책임입니다. AWS 직원의 논리적 액세스, AWS 인프라의 네트워크 및 컴퓨터 보안 제어, AWS Private CA SOC 2 Type 2 보고서 범위 내에 있는 AWS 직원 계정의 암호 파라미터 제어와 관련된 제어를 상속합니다(D.2 직원 사용자 액세스, D.3 논리적 보안, D.6 물리적 보안 및 환경 보호 섹션 참조).
6. Technical Controls (4.6.6)	Shared	Private CA 사용과 관련된 보안 관리 제어를 문서화하는 것은 사용자의 책임입니다. AWS Private CA SOC 2 Type 2 보고서의 범위 내에 있는 AWS Private CA 서비스의 시스템 개발 제어와 관련된 제어를 상속합니다(섹션 D.7 변경 관리 참조).

CP/CPS 요구 사항	책임	추가 정보
6. Technical Controls (4.6.7)	Shared	PKI 환경에 해당하는 경우 Private CA 사용에 대한 네트워크 보안 제어를 문서화하는 것은 사용자의 책임입니다. AWS Private CA SOC 2 Type 2 보고서의 범위 내에 있는 AWS 인프라의 네트워크 보안 제어와 관련된 제어를 상속합니다(C.1 서비스 약정, D.3 논리적 보안 및 E.1 모니터링 활동 섹션 참조).
6. Technical Controls (4.6.8)	AWS Private CA	AWS Private CA 는 신뢰할 수 있는 시간 소스를 사용하여 CA 데이터의 타임스탬프를 지정합니다.
7. Certificate, CRL, and OCSP Profiles (All)	Shared	PKI 환경의 요구 사항을 충족하는 프로필 요구 사항 및 인증서 입력을 문서화하는 것은 사용자의 책임입니다. AWS Private CA 는 프로필 요구 사항을 충족하는 데 도움이 되는 프로필 템플릿을 제공합니다.

CP/CPS 요구 사항	책임	추가 정보
8. Compliance Audit and Other Assessment (All)	Shared	규정 준수 감사 및 기타 평가를 문서화하는 것은 사용자의 책임입니다. AWS Private CA 는 사용자와 감사자가 운영 및 규정 준수를 지원하기 위해 설정된 AWS 제어를 이해하는 데 도움이 되는 SOC 2 보고서를 제공합니다.
9. Other Business and Legal Matters	You	Private CA에 적용되는 일반적인 비즈니스 및 법적 문제를 문서화하는 것은 사용자의 책임입니다.

AWS Private CA 리소스 모니터링

모니터링은 AWS Private CA 및 다른 AWS 솔루션의 안정성, 가용성 및 성능을 유지하는 데 중요한 부분입니다.는 다음과 같은 모니터링 도구를 AWS 제공하여 문제 발생 시 모니터링하고 AWS Private CA 보고하며 적절한 경우 자동 조치를 취합니다.

- Amazon CloudWatch는 AWS 리소스와 AWS 에서 실행하는 애플리케이션을 실시간으로 모니터링합니다. 지표를 수집 및 추적하고, 사용자 지정 대시보드를 생성할 수 있으며, 지정된 지표가 지정한 임곗값에 도달하면 사용자에게 알리거나 조치를 취하도록 경보를 설정할 수 있습니다. 예를 들어 CloudWatch에서 Amazon EC2 인스턴스의 CPU 사용량 또는 기타 지표를 추적하고 필요할 때 자동으로 새 인스턴스를 시작할 수 있습니다. 자세한 내용은 Amazon CloudWatch 사용 설명서를 참조하세요.
- Amazon CloudWatch Logs로 Amazon EC2 인스턴스, CloudTrail, 기타 소스의 로그 파일을 모니터링, 저장 및 액세스할 수 있습니다. CloudWatch Logs는 로그 파일의 정보를 모니터링하고 특정 임계값에 도달하면 사용자에게 알릴 수 있습니다. 또한 매우 내구성이 뛰어난 스토리지에 로그 데이터를 저장할 수 있습니다. 자세한 내용은 Amazon CloudWatch Logs 사용 설명서를 참조하십시오.
- AWS CloudTrail은 직접 수행하거나 AWS 계정 를 대신하여 수행한 API 호출 및 관련 이벤트를 캡처하고 지정한 Amazon S3 버킷에 로그 파일을 전송합니다. 어떤 사용자 및 계정이 AWS를 호출했는지 어떤 소스 IP 주소에 호출이 이루어졌는지 언제 호출이 발생했는지 확인할 수 있습니다. 자세한 내용은 AWS CloudTrail 사용 설명서를 참조하십시오.
- EventBridge: 애플리케이션을 다양한 소스의 데이터와 쉽게 연결할 수 있는 서버리스 이벤트 버스 서비스입니다. EventBridge는 자체 애플리케이션, Software-as-a-Service(SaaS) 애플리케이션 및 AWS 서비스의 실시간 데이터 스트림을 제공하고 해당 데이터를 Lambda와 같은 대상으로 라우팅합니다. 이를 통해 서비스에서 발생하는 이벤트를 모니터링하고 이벤트 기반 아키텍처를 구축할 수 있습니다. 자세한 내용은 Amazon EventBridge 사용 설명서를 참조하세요.

다음 주제에서는에서 사용할 수 있는 AWS 클라우드 모니터링 도구에 대해 설명합니다 AWS Private CA.

AWS Private CA CloudWatch 지표

Amazon CloudWatch는 AWS 리소스에 대한 모니터링 서비스입니다. CloudWatch를 사용하여 지표를 수집 및 추적하고, 경보를 설정하고, AWS 리소스의 변경 사항에 자동으로 대응할 수 있습니다. CloudWatch 지표는 한 번 이상 게시됩니다.

AWS Private CA 는 다음과 같은 CloudWatch 지표를 지원합니다.

지표	설명
CRLGenerated	인증서 해지 목록(CRL) 이 생성되었습니다. 이 지표는 프라이빗 CA에만 적용됩니다.
MisconfiguredCRLBucket	CRL에 지정된 S3 버킷이 제대로 구성되지 않았습니다. 버킷 정책을 확인합니다. 이 지표는 프라이빗 CA에만 적용됩니다.
Time	발급 요청부터 발급 완료(또는 실패)까지의 시간(밀리초). 이 지표는 IssueCertificate 작업에만 적용됩니다.
Success	인증서가 성공적으로 발급되었습니다. 이 지표는 IssueCertificate 작업에만 적용됩니다.
Failure	작업이 실패했습니다. 이 지표는 IssueCertificate 작업에만 적용됩니다.

CloudWatch 지표에 대한 자세한 내용은 다음 주제를 참조하세요.

- Amazon CloudWatch 지표 사용
- Amazon CloudWatch 경보 생성

CloudWatch Events AWS Private CA 로 모니터링

Amazon CloudWatch Events를 사용하여 AWS 서비스를 자동화하고 애플리케이션 가용성 문제 또는 리소스 변경과 같은 시스템 이벤트에 자동으로 대응할 수 있습니다. AWS 서비스의 이벤트는 거의 실시간으로 CloudWatch Events로 전송됩니다. 관심이 있는 이벤트와 이벤트 발생 시 실행할 자동 작업을 표시하도록 간단한 규칙을 작성할 수 있습니다. CloudWatch 이벤트는 한 번 이상 게시됩니다. 자세한 내용은 이벤트에서 트리거되는 CloudWatch 이벤트 규칙 생성을 참조하십시오.

CloudWatch Events는 Amazon EventBridge를 사용하여 작업으로 바뀝니다. EventBridge를 사용하면 이벤트를 사용하여 AWS Lambda 함수, AWS Batch 작업, Amazon SNS 주제 등을 포함한 대상을 트리거할 수 있습니다. 자세한 내용은 Amazon EventBridge란 무엇입니까?를 참조하세요.

프라이빗 CA를 생성할 때 성공 또는 실패

이러한 이벤트는 CreateCertificateAuthority 작업에 의해 트리거됩니다.

Success

성공하면 작업은 새 CA의 ARN을 반환합니다.

```
{
   "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"
   }
}
```

실패

실패 시 작업은 CA에 대한 ARN을 반환합니다. ARN을 사용하여 DescribeCertificateAuthority를 호출하여 CA의 상태를 확인할 수 있습니다.

```
{
   "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"
   }
}
```

인증서 발급 시 성공 또는 실패

이러한 이벤트는 IssueCertificate 작업에 의해 트리거됩니다.

Success

성공하면 작업은 CA와 새 인증서의 ARN을 반환합니다.

```
{
   "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"
   }
}
```

실패

오류가 발생하면 작업은 인증서 ARN과 CA의 ARN을 반환합니다. 인증서 ARN을 사용하면 GetCertifier를 호출하여 실패 원인을 확인할 수 있습니다.

```
{
   "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"
   }
}
```

인증서 해지 시 성공

이 이벤트는 RevokeCertificate 작업에 의해 트리거됩니다.

해지가 실패하거나 인증서가 이미 해지된 경우에는 이벤트가 전송되지 않습니다.

Success

성공하면 작업은 CA 및 해지된 인증서의 ARN을 반환합니다.

```
{
   "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 생성 시 성공 또는 실패

이러한 이벤트는 RevokeCertificate 작업에 의해 트리거되며, 이로 인해 인증서 해지 목록(CRL)이 생성됩니다.

Success

성공하면 이 작업은 CRL과 연결된 CA의 ARN을 반환합니다.

```
{
   "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"
   }
}
```

실패 1 - 권한 오류로 인해 CRL이 Amazon S3에 저장되지 않았을 수 있음

이 오류가 발생하면 Amazon S3 버킷 권한을 확인합니다.

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

실패 2 - 내부 오류로 인해 CRL이 Amazon S3에 저장되지 않았을 수 있음

이 오류가 발생하면 작업을 다시 시도하십시오.

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

실패 3 - CRL 생성 AWS Private CA 실패

이 오류를 해결하려면 CloudWatch 지표를 확인하세요.

```
{
   "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 감사 보고서 생성 시 성공 또는 실패

이러한 이벤트는 CreateCertificateAuthorityAuditReport 작업에 의해 트리거됩니다.

Success

성공하면 작업은 CA의 ARN과 감사 보고서의 ID를 반환합니다.

```
{
   "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"
   }
}
```

실패

가 Amazon S3 버킷에 대한 PUT 권한이 AWS Private CA 없거나 버킷에서 암호화가 활성화되어 있거나 다른 이유로 감사 보고서가 실패할 수 있습니다.

```
{
   "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"
   }
}
```

를 사용하여 AWS Private Certificate Authority API 호출 로깅 AWS CloudTrail

AWS Private Certificate Authority 는 사용자 AWS CloudTrail, 역할 또는 서비스가 수행한 작업에 대한 레코드를 제공하는 AWS 서비스와 통합됩니다 AWS Private CA. CloudTrail은에 대한 API 호출 및 서명 작업을 이벤트 AWS Private CA 로 캡처합니다. 캡처되는 호출에는 AWS Private CA 콘솔로부터의 호출과 AWS Private CA API 작업에 대한 코드 호출이 포함됩니다. 추적을 생성하면 이벤트를 포함하여 CloudTrail 이벤트를 지속적으로 Amazon S3 버킷에 배포할 수 있습니다 AWS Private CA. 트레일을 구성하지 않은 경우에도 CloudTrail 콘솔의 이벤트 기록에서 최신 이벤트를 볼 수 있습니다. CloudTrail에서 수집한 정보를 사용하여 요청이 수행된 AWS Private CA IP 주소, 요청이 수행된 사람, 요청이 수행된 시간 및 추가 세부 정보를 확인할 수 있습니다.

CloudTrail에 대한 자세한 설명은 AWS CloudTrail 사용자 가이드를 참조하십시오.

AWS Private CA CloudTrail의 정보

CloudTrail은 계정을 생성할 AWS 계정 때에서 활성화됩니다. 활동이 발생하면 AWS Private CA해당 활동이 이벤트 기록의 다른 AWS 서비스 이벤트와 함께 CloudTrail 이벤트에 기록됩니다. 에서 최근 이벤트를 보고 검색하고 다운로드할 수 있습니다 AWS 계정. 자세한 내용은 CloudTrail 이벤트 기록을 사용하여 이벤트 보기를 참조하세요.

에 대한 이벤트를 AWS 계정포함하여에 이벤트를 지속적으로 기록하려면 추적을 AWS Private CA생성합니다. CloudTrail은 추적을 사용하여 Amazon S3 버킷으로 로그 파일을 전송할 수 있습니다. 콘솔에서 추적을 생성하면 기본적으로 모든 AWS 리전에 추적이 적용됩니다. 추적은 AWS 파티션의 모든 리전에서 이벤트를 로깅하고 지정한 Amazon S3 버킷으로 로그 파일을 전송합니다. 또한 CloudTrail 로그에서 수집된 이벤트 데이터를 추가로 분석하고 조치를 취하도록 다른 AWS 서비스를 구성할 수 있습니다. 자세한 내용은 다음 자료를 참조하세요.

- 추적 생성 개요
- CloudTrail 지원 서비스 및 통합
- CloudTrail에 대한 Amazon SNS 알림 구성
- 여러 리전에서 CloudTrail 로그 파일 받기 및 여러 계정에서 CloudTrail 로그 파일 받기

모든 AWS Private CA 작업은 CloudTrail에서 로깅되며 AWS Private CA API 참조에 문서화됩니다. 예를 들어 `ImportCACertificate`, `IssueCertificate`, `CreateAuditReport` 작업을 직접 호출하면 CloudTrail 로그 파일에 항목이 생성됩니다.

모든 이벤트 또는 로그 항목에는 요청을 생성했던 사용자에 대한 정보가 포함됩니다. 자격 증명을 이용하면 다음을 쉽게 판단할 수 있습니다.

- 요청이 루트 또는 AWS Identity and Access Management (IAM) 사용자 자격 증명으로 이루어졌는지 여부입니다.
- 역할 또는 페더레이션 사용자에 대한 임시 보안 인증을 사용하여 요청이 생성되었는지 여부.
- 요청이 다른 AWS 서비스에 의해 이루어졌는지 여부입니다.

자세한 내용은 CloudTrail userIdentity 요소를 참조하세요.

AWS Private CA 관리 이벤트

AWS Private CA 는 CloudTrail과 통합되어 사용자, 역할 또는 AWS 서비스가 수행한 API 작업을 기록합니다 AWS Private CA. CloudTrail을 사용하여 AWS Private CA API 요청을 실시간으로 모니터링하고 Amazon Simple Storage Service, Amazon CloudWatch Logs 및 Amazon CloudWatch Events에 로그를 저장할 수 있습니다.는 다음 작업 및 작업을 CloudTrail 로그 파일의 이벤트로 로깅하도록 AWS Private CA 지원합니다.

- CreateCertificateAuthority
- CreateCertificateAuthorityAuditReport
- CreatePermission
- DeleteCertificateAuthority
- DeletePermission
- DeletePolicy
- DescribeCertificateAuthority
- DescribeCertificateAuthorityReport

- GetCertificate
- GetCertificateAuthorityCertificate
- GetCertificateAuthorityCsr
- GetPolicy
- ImportCertificateAuthorityCertificate
- IssueCertificate
- ListCertificateAuthorities
- ListPermissions
- ListTags
- PutPolicy
- RestoreCertificateAuthority
- RevokeCertificate
- TagCertificateAuthority
- UntagCertificateAuthority
- UpdateCertificateAuthority
- `GenerateOCSPResponse` -가 OCSP 응답을 AWS Private CA 생성할 때 트리거됩니다.
- `SignCertificate` - 클라이언트가 IssueCertificate를 호출할 때 생성됩니다.
- `SignOCSPResponse` -가 OCSP 응답을 AWS Private CA 서명할 때 생성됩니다.
- `GenerateCRL` -가 인증서 해지 목록(CRL)을 AWS Private CA 생성할 때 생성됩니다.
- `SignCACSR` - AWS Private CA 가 인증 기관(CA) 인증서 서명 요청(CSR)에 서명할 때 생성됩니다.
- `SignCRL` -가 CRL에 AWS Private CA 서명할 때 생성됩니다.

예제 AWS Private CA 이벤트

추적이란 지정한 Amazon S3 버킷에 이벤트를 로그 파일로 입력할 수 있게 하는 구성입니다. CloudTrail 로그 파일에는 하나 이상의 로그 항목이 포함될 수 있습니다. 이벤트는 모든 소스로부터의 단일 요청을 나타내며 요청 작업, 작업 날짜와 시간, 요청 파라미터 등에 대한 정보가 들어 있습니다. CloudTrail 로그 파일은 퍼블릭 API 직접 호출의 주문 스택 트레이스가 아니므로 특정 순서로 표시되지 않습니다.

다음은 AWS Private CA CloudTrail 이벤트의 예입니다.

예제 1: 관리 이벤트, **IssueCertificate**

다음은 IssueCertificate 작업을 설명하는 CloudTrail 로그 항목을 보여 주는 예시입니다.

```
{
   "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"
   }
}
```

예제 2: 관리 이벤트, **ImportCertificateAuthorityCertificate**

다음은 ImportCertificateAuthorityCertificate 작업을 보여주는 CloudTrail 로그 항목이 나타낸 예시입니다.

```
{
   "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"
}
```

의 문제 해결 AWS Private Certificate Authority

AWS Private Certificate Authority을 사용하는 동안 문제가 발생할 경우 다음 주제를 참조하십시오.

주제

- AWS Private CA 인증서 해지 문제 해결
- AWS Private Certificate Authority 예외 메시지 문제 해결
- AWS Private CA Matter 준수 인증서 오류 문제 해결

AWS Private CA 인증서 해지 문제 해결

OCSP 응답 지연 시간

호출자가 리전별 에지 캐시 또는 발급 CA 리전과 지리적으로 멀리 떨어져 있는 경우 OCSP 응답 속도가 느려질 수 있습니다. 리전별 에지 캐시 가용성에 대한 자세한 내용은 글로벌 에지 네트워크를 참조하세요. 인증서를 사용할 리전과 가까운 리전에서 인증서를 발급하는 것이 좋습니다.

자체 서명된 인증서의 취소

자체 서명된 CA 인증서는 취소할 수 없습니다. 인증서를 기능적으로 취소하려면 CA를 삭제합니다.

AWS Private Certificate Authority 예외 메시지 문제 해결

AWS Private CA 명령은 여러 가지 이유로 실패할 수 있습니다. 각 예외와 예외를 해결하기 위한 권장 사항은 아래 표를 참조하십시오.

AWS Private CA 예외

에서 반환한 예외 AWS Private CA	설명	문제 해결
`AccessDeniedException`	해당 명령을 사용하는 데 필요한 권한을 사설 CA가 호출 계정에 위임되지 않았습니다.	의 권한 위임에 대한 자세한 내용은 섹션을 AWS Private CA 참조하세요ACM에 인증서 갱신 권한 할당.

에서 반환한 예외 AWS Private CA	설명	문제 해결
`InvalidArgsException`	잘못된 파라미터를 사용하여 인증서 생성 또는 갱신 요청이 수행되었습니다.	명령의 개별 설명서에서 입력 매개 파라미터가 유효한지 확인합니다. 새 인증서를 생성하는 경우, 요청된 서명 알고리즘을 CA의 키 유형과 함께 사용할 수 있는지 확인합니다.
`InvalidStateException`	연결된 사설 CA가 `ACTIVE` 상태가 아니므로 인증서를 갱신할 수 없습니다.	사설 CA 복원을 시도합니다. 사설 CA가 복원 기간을 벗어나면 CA를 복원할 수 없으며 인증서를 갱신할 수 없습니다.
`LimitExceededException`	각 CA(인증 기관)에는 발급이 가능한 인증서 할당량이 있습니다. 지정된 인증서와 연결된 사설 CA가 할당량에 도달했습니다. 자세한 내용은 AWS 일반 참조 가이드의 서비스 할당량을 참조하세요.	할당량 증가를 요청하려면 AWS Support 센터에 문의하세요.
`MalformedCSRException`	AWS Private CA 에 제출된 인증서 서명 요청(CSR)을 확인하거나 유효성 검사를 할 수 없습니다.	CSR이 올바르게 생성되고 구성되었는지 확인합니다.
`OtherException`	내부 오류로 인해 요청이 실패했습니다.	명령을 다시 실행해 봅니다. 문제가 지속되면 AWS Support 센터에 문의하세요.
`RequestFailedException`	AWS 환경의 네트워킹 문제로 인해 요청이 실패했습니다.	요청을 다시 시도하세요. 오류가 지속되면 Amazon VPC(VPC) 구성을 확인합니다.

에서 반환한 예외 AWS Private CA	설명	문제 해결
`ResourceNotFoundException`	인증서를 발급한 사설 CA가 삭제되어 더 이상 존재하지 않습니다.	다른 활성 CA에서 새 인증서를 요청합니다.
`ThrottlingException`	요청된 API 작업이 할당량을 초과하여 실패했습니다.	AWS Private CA에서 허용하는 것보다 많은 호출을 발행하고 있지 않은지 확인합니다. 할당량 초과가 아니라 일시적인 조건으로 인해 `ThrottlingException` 오류가 발생할 수도 있습니다. 할당량을 초과하여 호출을 하지 않았는데도 오류가 발생한 경우에는 요청을 다시 시도하십시오. 할당량에 따라 실행 중인 경우에는 할당량 증가를 요청할 수 있습니다. 자세한 내용은 AWS 일반 참조 가이드의 Service Quotas를 참조하세요.
`ValidationException`	요청의 입력 파라미터가 형식이 잘못되었거나, 루트 인증서의 유효 기간이 요청된 인증서의 유효 기간 이전에 끝납니다.	명령 입력 파라미터의 구문 요구 사항과 CA 루트 인증서의 유효 기간을 확인합니다. 유효 기간 변경에 대한 자세한 내용은 에서 프라이빗 CA 업데이트 AWS Private Certificate Authority 단원을 참조하십시오.

AWS Private CA Matter 준수 인증서 오류 문제 해결

Matter 연결 표준은 사물 인터넷(IoT) 장치의 보안 및 일관성을 개선하는 인증서 구성을 지정합니다. Matter 준수 루트 CA, 중간 CA 및 최종 엔터티 인증서를 만들기 위한 Java 샘플은 Matter 인증서를 구현 AWS Private CA 하는 데 사용에서 찾을 수 있습니다.

문제 해결을 지원하기 위해 Matter 개발자는 chip-cert라는 인증서 확인 도구를 제공합니다. 도구가 보고하는 오류는 다음 표에 수정 사항과 함께 나열되어 있습니다.

오류 코드	의미	문제 해결
0x000003 5	`BasicConstraints` , `KeyUsage`, 및 `Extension` `KeyUsage` 확장자는 중요로 표시되어야 합니다.	사용 사례에 맞는 올바른 템플릿을 선택해야 합니다.
0x000000 0	권한 키 식별자 확장이 있어야 합니다.	AWS Private CA 는 루트 인증서에 권한 키 식별자 확장을 설정하니다. CSR을 사용하여 Base64로 인코딩된 AuthorityKeyIdentifie성한 다음 이를 CustomExtension을 통해 전달해야 합니다. 자세은 노드 운영 인증서(NOC)에 대한 루트 CA를 활성화합니다. 및증 기관(PAA) 활성화 섹션을 참조하세요.
0x000000 E	인증서가 만료되었습니다.	사용하는 인증서가 만료되지 않았는지 확인하십시오.
0x000000 4	인증서 체인 유효성 검사에 실패했습니다.	API를 사용하여 적절하게 구성된 KeyUsage를 AWS Private CA 제공된 Java 예제를 사용하지 않고 Matter 준수 최종 엔터티 인증성하려고 하면이 오류가 발생할 수 있습니다. 기본적으로는 9비트 KeyUsage 확장 값을 AWS Private CA 생성트는 추가 바이트를 생성합니다. Matter는 형식 변환 중에 추가 B무시하므로 체인 유효성 검사에 실패합니다. 하지만 `APIPassth`템플릿의 CustomExtension을 사용하여 KeyUsage 값의 정확한수를 설정할 수 있습니다. 예제는 노드 운영 인증서(NOC) 생성 조하세요.

오류 코드	의미	문제 해결
		샘플 코드를 수정하거나 OpenSSL과 같은 대체 X.509 유틸리티 는 경우 체인 유효성 검사 오류를 방지하려면 수동 검증을 수행하 다. 변환에 손실이 없는지 확인하는 방법 1. openssl을 사용하여 인증서 (노드(최종 엔터티) 인증서에 유 인이 포함되어 있는지 확인합니다. 이 예제에서 `rcac.pem`는 CA 인증서, `icac.pem`는 중간 CA 인증서, `noc.pem`는 노드 입니다. ``` openssl verify -verbose -CAfile <(cat rcac.pem icac noc.pem ``` 2. chip-cert를 사용하여 PEM 형식의 노드 인증서를 TLV(태그, 형식으로 변환했다가 다시 되돌릴 수 있습니다. ``` ./chip-cert convert-cert noc.pem noc.chip -c ./chip-cert convert-cert noc.chip noc_converted.pem ``` 파일 `noc.pem` 및 `noc_converted.pem` 은 문자열 비교 도 확인한 것과 정확히 동일해야 합니다.

를 사용하여 Kubernetes 보호 AWS Private Certificate Authority

AWS Private Certificate Authority 를 사용하여 TLS 및 mTLS를 통한 보안 인증 및 암호화를 위한 인증서를 제공할 수 있습니다.는 인증서를 요청하고 AWS Private CA Kubernetes 보안 암호에 배포하며 인증서 갱신을 자동화하는 Kubernetes에 널리 채택된 cert-manager 추가 기능에 대한 오픈 소스 플러그인인 Kubernetes용 커넥터(`aws-privateca-issuer`)를 AWS Private CA 제공합니다. https://cert-manager.io/docs/

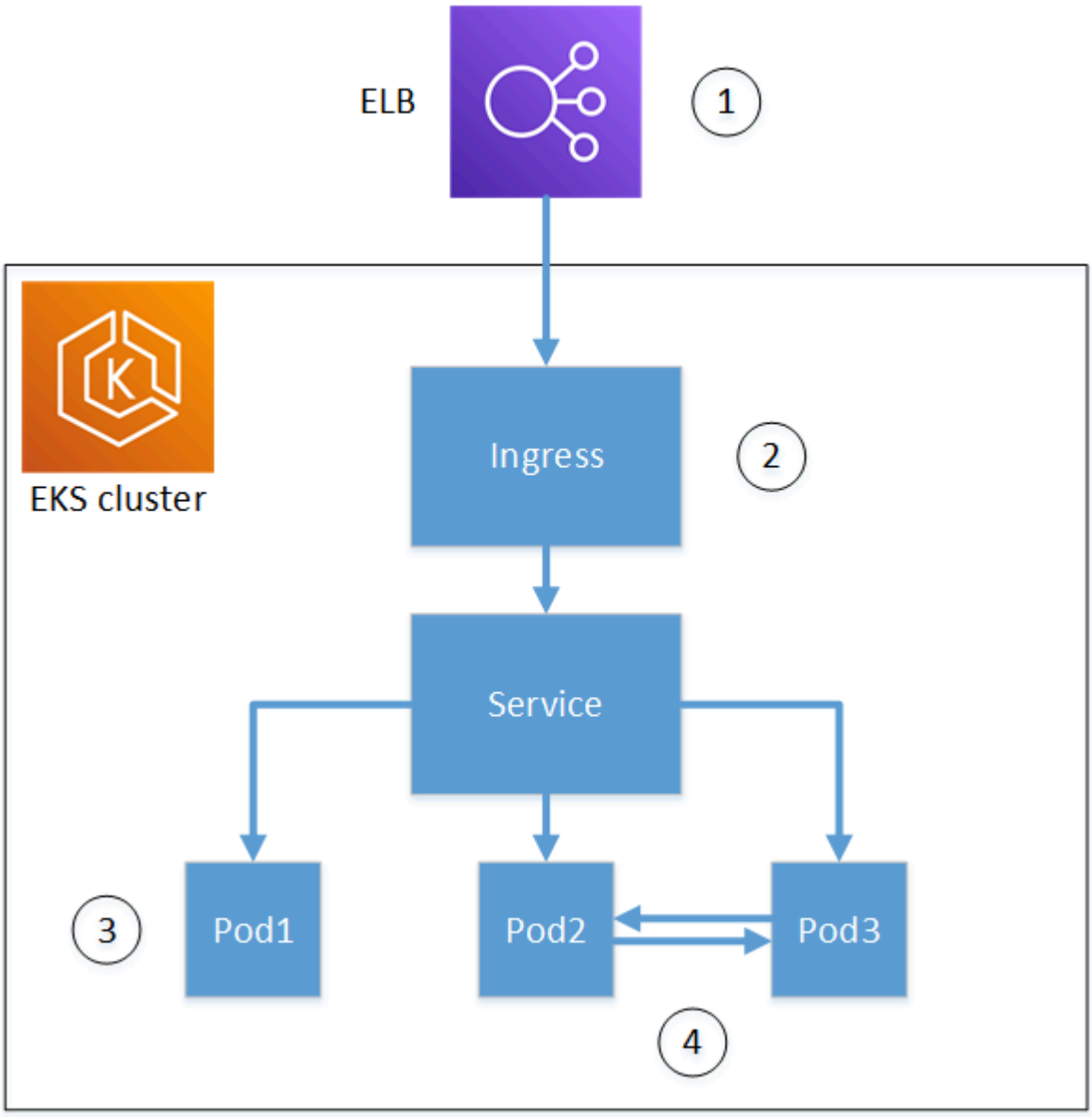
`aws-privateca-issuer` 플러그인을 사용하면를 통해 AWS Private CA 인증서를 발급할 수 있습니다`cert-manager`. 플러그인은 Amazon Elastic Kubernetes Service(Amazon EKS),의 자체 관리형 Kubernetes 클러스터 AWS또는 온프레미스 Kubernetes 클러스터에서 사용할 수 있습니다. 플러그인은 x86 아키텍처와 ARM 아키텍처 모두에서 작동합니다.

AWS Private CA 에는 내보낼 수 없는 HSM 지원 키가 있습니다. 액세스 제어 및 CA 작업 감사에 대한 규제 요구 사항이 있는 경우 AWS Private CA 를 사용하여 감사 가능성을 개선하고 규정 준수를 지원할 수 있습니다.

> **Note**
>
> Amazon EKS에서를 실행하는 경우 관리형 설치 환경을 위해 `cert-manager` 및 `aws-privateca-connector-for-kubernetes` 추가 기능을 사용하는 것이 좋습니다. 자세한 내용은 AWS 추가 기능을 참조하세요.

개념

다음 다이어그램은 Amazon EKS 클러스터에서 TLS를 사용하는 데 사용할 수 있는 몇 가지 옵션을 보여줍니다. 예제 클러스터는 로드 밸런서 뒤에 있습니다. 숫자는 TLS 보안 통신을 위한 가능한 엔드포인트를 식별합니다.



1. 로드 밸런서에서 종료

   Elastic Load Balancing(Elastic Load Balancing)은 AWS Certificate Manager 서비스와 통합됩니다. 로드 밸런`cert-manager`서에를 설치할 필요가 없습니다. 프라이빗 CA로 ACM을 프로비저닝하고, 프라이빗 CA로 인증서에 서명하고, Elastic Load Balancing 콘솔을 사용하여 인증서를 설치할 수 있습니다. AWS Private CA 인증서는 자동으로 갱신됩니다.

   또는AWS 로드 밸런서가 아닌에 프라이빗 인증서를 제공하여 TLS를 종료할 수 있습니다.

   이렇게 하면 원격 클라이언트와 로드 밸런서 간에 암호화된 통신이 제공됩니다. 로드 밸런서가 암호화되지 않은 상태로 Amazon EKS 클러스터로 전달된 후의 데이터입니다.

2. Kubernetes 수신 컨트롤러에서 종료

   수신 컨트롤러는 Amazon EKS 클러스터 내에 있으며 로드 밸런서 및 라우터 역할을 합니다. 수신 컨트롤러를 외부 통신을 위한 클러스터의 엔드포인트로 사용하려면 다음을 수행해야 합니다.

   - `cert-manager` 및 모두 설치 `aws-privateca-issuer`
   - 의 TLS 프라이빗 인증서로 컨트롤러를 프로비저닝합니다 AWS Private CA.

   로드 밸런서와 수신 컨트롤러 간의 통신은 암호화되고 데이터는 클러스터의 리소스로 암호화되지 않습니다.

3. 포드에서 종료

   각 포드는 스토리지 및 네트워크 리소스를 공유하는 하나 이상의 컨테이너 그룹입니다. `cert-manager` 및 `aws-privateca-issuer`를 모두 설치하고 프라이빗 CA로 클러스터를 프로비저닝하는 경우 Kubernetes는 필요에 따라 포드에 서명된 TLS 프라이빗 인증서를 설치할 수 있습니다. 기본적으로 포드에서 종료되는 TLS 연결은 클러스터의 다른 포드에서 사용할 수 없습니다.

4. 포드 간 보안 통신.

   인증서로 여러 포드를 프로비저닝하여 서로 통신할 수 있습니다. 가능한 시나리오는 다음과 같습니다.

   - Kubernetes를 사용한 프로비저닝은 자체 서명된 인증서를 생성했습니다. 이렇게 하면 포드 간의 통신이 보호되지만 자체 서명된 인증서는 HIPAA 또는 FIPS 요구 사항을 충족하지 않습니다.
   - 에서 서명한 인증서로 프로비저닝합니다 AWS Private CA. 이를 위해서는 `cert-manager` 및를 모두 설치해야 합니다`aws-privateca-issuer`. 그런 다음 Kubernetes는 필요에 따라 포드에 서명된 mTLS 인증서를 설치할 수 있습니다.

고려 사항

Kubernetes와 AWS Private Certificate Authority 함께를 사용할 때는 다음 사항을 고려해야 합니다.

cert-manager의 교차 계정 사용

CA에 대한 교차 계정 액세스 권한이 있는 관리자는 Kubernetes용 `cert-manager` 추가 기능을 사용하여 공유 CA를 사용하여 클러스터에 대한 인증서를 프로비저닝할 수 있습니다. 자세한 정보는 프라이빗 CA에 대한 교차 계정 액세스의 보안 모범 사례 섹션을 참조하세요.

교차 계정 시나리오에서는 특정 AWS Private CA 인증서 템플릿만 사용할 수 있습니다.

다음 표에는 cert-manager와 함께 사용하여 Kubernetes 클러스터를 프로비저닝할 수 있는 AWS Private CA 템플릿이 나열되어 있습니다.

Kubernetes에 지원되는 템플릿	교차 계정 사용을 위한 지원
BlankEndEntityCertificate_CSRPassthrough/V1 정의	아니요
CodeSigningCertificate/V1 정의	아니요
EndEntityCertificate/V1 정의	예
EndEntityClientAuthCertificate/V1 정의	예
EndEntityServerAuthCertificate/V1 정의	예
OCSPSigningCertificate/V1 정의	아니요

AWS Private CA Connector for Kubernetes를 시작합니다.

다음 주제에서는를 AWS Private CA 사용하여 Kubernetes 클러스터에서 통신을 보호하는 방법을 보여 줍니다. 또 다른 예는 GitHub의 Kubernetes에 대한 전송 중 암호화를 참조하세요.

프라이빗 인증 기관을 사용하여 Amazon EKS 클러스터와의 통신을 보호할 수 있습니다. 시작하기 전에 다음이 있는지 확인하십시오.

- 보안 정책으로 범위가 지정된 적절한 권한이 있는 AWS 계정입니다.

  Amazon EKS clusters

  JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "IAM",
      "Effect": "Allow",
      "Action": [
        "iam:CreateRole",
        "iam:AttachRolePolicy",
```

```
        "iam:GetRole"
      ],
      "Resource": "*"
    },
    {
      "Sid": "EKS",
      "Effect": "Allow",
      "Action": [
        "eks:CreateAddon",
        "eks:DescribeAddon",
        "eks:CreatePodIdentityAssociation",
        "eks:DescribeCluster"
      ],
      "Resource": "*"
    },
    {
      "Sid": "IAMPassRole",
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": "arn:aws:iam::*:role/CertManagerPrivateCARole"
    }
  ]
}
```

Other clusters

JSON

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

```
      "Effect": "Allow",
      "Action": [
        "rolesanywhere:CreateProfile"
      ],
      "Resource": "*"
    },
    {
      "Sid": "IAM",
      "Effect": "Allow",
      "Action": [
        "iam:CreateRole",
        "iam:AttachRolePolicy"
      ],
      "Resource": "*"
    },
    {
      "Sid": "IAMPassRole",
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": "arn:aws:iam::*:role/CertManagerPrivateCARole"
    }
  ]
}
```

- Kubernetes 클러스터. Amazon Elastic Kubernetes Service 클러스터를 생성하려면 Amazon EKS 빠른 시작 안내서를 참조하세요. 간소화를 위해 클러스터 이름을 포함할 환경 변수를 생성합니다.

```
export CLUSTER=aws-privateca-demo
```

- CA 및 Amazon EKS 클러스터가 AWS 리전 위치한 입니다. 간소화를 위해 리전을 보관할 환경 변수를 생성합니다.

```
export REGION=aws-region
```

- AWS Private CA 프라이빗 인증 기관의 Amazon 리소스 이름(ARN)입니다. 간소화를 위해 프라이빗 CA ARN을 보관할 환경 변수를 생성합니다.

```
export CA_ARN="arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID"
```

프라이빗 CA를 생성하려면에서 프라이빗 CA https://docs.aws.amazon.com/privateca/latest/userguide/create-CA.html생성을 참조하세요. AWS Private CA

- 다음 소프트웨어가 설치된 컴퓨터:
  - AWS CLI v2 구성됨
  - kubectl v1.13 이상
  - 비 Amazon EKS 클러스터의 경우 Helm v3

cert-manager 설치

프라이빗 CA를 사용하려면 인증서를 요청하고 배포하며 인증서 갱신을 자동화하는 cert-manager> 추가 기능을 설치해야 합니다. 또한 프라이빗 인증서를 발급할 수 있는 aws-private-ca-issuer 플러그인을 설치해야 합니다 AWS Private CA. 다음 단계에 따라 추가 기능 및 플러그인을 설치합니다.

Amazon EKS clusters

Amazon EKS 추가 기능으로 cert-manager를 설치합니다.

```
aws eks create-addon \
  --cluster-name $CLUSTER \
  --addon-name cert-manager \
  --region $REGION
```

Other clusters

Helmcert-manager을 사용하여를 설치합니다.

```
helm repo add jetstack https://charts.jetstack.io
helm repo update

helm install cert-manager jetstack/cert-manager \
  --namespace cert-manager \
  --create-namespace \
  --set crds.enabled=true
```

IAM 권한 구성

`aws-privateca-issuer` 플러그인에는가와 상호 작용하는 권한이 필요합니다 AWS Private CA. Amazon EKS 클러스터의 경우 포드 자격 증명을 사용합니다. 다른 클러스터의 경우를 사용합니다 AWS Identity and Access Management Roles Anywhere.

먼저 IAM 정책을 생성합니다. 정책은 `AWSPrivateCAConnectorForKubernetesPolicy` 관리형 정책을 사용합니다. 정책에 대한 자세한 내용은 AWS 관리형 정책 참조 가이드의 AWSPrivateCAConnectorForKubernetesPolicy를 참조하세요.

Amazon EKS clusters

1. 다음 신뢰 정책을 `trust-policy.json` 포함하는 라는 파일을 생성합니다.

   JSON

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

2. 다음 명령을 실행하여 IAM 역할을 생성합니다.

   ```
   ROLE_ARN=$(aws iam create-role \
     --role-name CertManagerPrivateCARole \
     --assume-role-policy-document file://trust-policy.json \
     --region $REGION \
     --output text \
     --query "Role.Arn")
   ```

```
aws iam attach-role-policy \
 --role-name CertManagerPrivateCARole \
 --policy-arn arn:aws:iam::aws:policy/AWSPrivateCAConnectorForKubernetesPolicy
```

Other clusters

1. 에 저장된 프라이빗 CA를 신뢰하는 트러스트 앵커를 생성합니다CA_ARN. 지침은 시작하기를 참조하세요 IAM Roles Anywhere. 환경 변수를 생성하여 트러스트 앵커 ARN을 저장합니다.

   ```
   export TRUST_ANCHOR_ARN=trustAnchorArn
   ```

2. 다음 신뢰 정책을 `trust-policy.json` 포함하는 라는 파일을 생성합니다.

   JSON

   ```
   {
       "Version":"2012-10-17",
       "Statement": [
           {
               "Sid": "TrustPolicyForSelfManagedOrOnPremiseClusters",
               "Effect": "Allow",
               "Principal": {
                   "Service": "rolesanywhere.amazonaws.com"
               },
               "Action": [
                   "sts:AssumeRole",
                   "sts:SetSourceIdentity",
                   "sts:TagSession"
               ],
               "Condition": {
                   "ArnEquals": {
                       "aws:SourceArn": [
                           "arn:aws:rolesanywhere:us-
   east-1:123456789012:trust-anchor/TRUST_ANCHOR_ARN"
                       ]
                   },
                   "StringEquals": {
                       "aws:PrincipalTag/x509Subject/CN": "aws-privateca-
   issuer"
                   }
               }
   ```

```
        }
    ]
}
```

3. 다음 명령을 실행하여 IAM 역할을 생성합니다.

```
ROLE_ARN=$(aws iam create-role \
  --role-name CertManagerPrivateCARole \
  --assume-role-policy-document file://trust-policy.json \
  --query "Role.Arn" \
  --region $REGION \
  --output text)

 aws iam attach-role-policy \
  --role-name CertManagerPrivateCARole \
  --region $REGION \
  --policy-arn arn:aws:iam::aws:policy/AWSPrivateCAConnectorForKubernetesPolicy
```

AWS Private CA 클러스터 발급자 설치 및 구성

`aws-privateca-connector-for-kubernetes` 추가 기능을 설치하려면 다음 명령을 사용합니다.

Amazon EKS clusters

추가 기능을 생성합니다.

```
aws eks create-addon --region $REGION \
  --cluster-name $CLUSTER \
  --addon-name aws-privateca-connector-for-kubernetes \
  --pod-identity-associations "[{
    \"serviceAccount\": \"aws-privateca-issuer\",
    \"roleArn\": \"$ROLE_ARN\"
  }]"
```

그런 다음 추가 기능이 활성화될 때까지 기다립니다.

```
aws eks describe-addon \
  --cluster-name $CLUSTER \
  --addon-name aws-privateca-connector-for-kubernetes \
  --region $REGION \
  --query 'addon.status'
```

Other clusters

1. IAM Roles Anywhere다음에서 프로필을 생성합니다.

```
PROFILE_ARN=$(aws rolesanywhere create-profile \
  --name "privateca-profile" \
  --role-arns "$ROLE_ARN" \
  --region "$REGION" \
  --query 'profile.profileArn' \
  --enabled \
  --output text)
```

2. Kubernetes용 커넥터와 함께 사용하고 AWS Private CA다음을 사용하여 인증 IAM Roles Anywhere 할 클라이언트 인증서를 생성합니다.

   a. 클라이언트 인증서에 대한 프라이빗 키를 생성합니다.

```
openssl genrsa -out client.key 2048
```

   b. 클라이언트 인증서에 대한 인증서 서명 요청(CSR)을 생성합니다.

```
openssl req -new \
  -key client.key \
  -out client.csr \
  -subj "/CN=aws-privateca-issuer"
```

   c. AWS Private CA다음에서 클라이언트 인증서를 발급합니다.

```
CERT_ARN=$(aws acm-pca issue-certificate \
  --signing-algorithm SHA256WITHRSA \
  --csr fileb://client.csr \
  --validity Value=1,Type=DAYS \
  --certificate-authority-arn "$CA_ARN" \
  --region "$REGION" \
  --query 'CertificateArn' \
  --output text)
```

   d. 클라이언트 인증서를 로컬에 저장합니다.

```
aws acm-pca get-certificate \
  --certificate-authority-arn $CA_ARN \
  --certificate-arn $CERT_ARN \
```

```
  --region $REGION \
  --query 'Certificate'
  --output text > pca-issuer-client-cert.pem
```

3. 클라이언트 인증서를 사용하여 클러스터에 AWS Private CA 발급자를 설치합니다.

   a. awspca Helm 리포지토리를 추가합니다.

   ```
   helm repo add awspca https://cert-manager.github.io/aws-privateca-issuer
   helm repo update
   ```

   b. 네임스페이스 생성:

   ```
   kubectl create namespace aws-privateca-issuer
   ```

   c. 이전에 생성한 인증서를 보안 암호에 넣습니다.

   ```
   kubectl create secret tls aws-privateca-credentials \
     -n aws-privateca-issuer \
     --cert=pca-issuer-client-cert.pem \
     --key=client.key
   ```

4. IAM Roles Anywhere다음을 사용하여 AWS Private CA 발급자를 설치합니다.

   a. 라는 파일을 생성values.yaml하여 IAM Roles Anywhere다음과 함께 사용할 AWS Private CA 발급자 플러그인을 구성합니다.

   ```
   cat > values.yaml <<EOF
   env:
     AWS_EC2_METADATA_SERVICE_ENDPOINT: "http://127.0.0.1:9911"

   extraContainers:
     - name: "rolesanywhere-credential-helper"
       image: "public.ecr.aws/rolesanywhere/credential-helper:latest"
       command: ["aws_signing_helper"]
       args:
         - "serve"
         - "--private-key"
         - "/etc/cert/tls.key"
         - "--certificate"
         - "/etc/cert/tls.crt"
         - "--role-arn"
         - "$ROLE_ARN"
   ```

```
      - "--profile-arn"
      - "$PROFILE_ARN"
      - "--trust-anchor-arn"
      - "$TRUST_ANCHOR_ARN"
    volumeMounts:
      - name: cert
        mountPath: /etc/cert/
        readOnly: true

volumes:
  - name: cert
    secret:
      secretName: aws-privateca-credentials
EOF
```

b. IAM Roles Anywhere다음을 사용하여 AWS Private CA 발급자를 설치합니다.

```
helm install aws-privateca-issuer awspca/aws-privateca-issuer \
  -n aws-privateca-issuer \
  -f values.yaml
```

발급자가 준비될 때까지 기다립니다. 다음 명령을 사용합니다.

```
kubectl wait --for=condition=ready pods --all -n aws-privateca-issuer --timeout=120s
```

그런 다음 설치를 확인하여 모든 포드가 READY 상태에 도달했는지 확인합니다.

```
kubectl -n aws-privateca-issuer get all
```

를 구성하려면 발급자의 구성을 `cluster-issuer.yaml`포함하는 라는 YAML 파일을 `aws-private-ca-cluster-issuer`생성합니다.

```
cat > cluster-issuer.yaml <<EOF
apiVersion: awspca.cert-manager.io/v1beta1
kind: AWSPCAClusterIssuer
metadata:
  name: aws-privateca-cluster-issuer
spec:
  arn: "$CA_ARN"
  region: "$REGION"
```

```
EOF
```

그런 다음 클러스터 구성을 적용합니다.

```
kubectl apply -f cluster-issuer.yaml
```

발급자의 상태를 확인합니다.

```
kubectl describe awspcaclusterissuer aws-privateca-cluster-issuer
```

다음과 유사한 응답이 나타납니다.

```
Status:
  Conditions:
    Last Transition Time:  2025-08-13T21:00:00Z
    Message:               AWS PCA Issuer is ready
    Reason:                Verified
    Status:                True
    Type:                  Ready
```

cert-manager를 사용하여 AWS Private CA 클라이언트 인증서 관리

Amazon EKS 클러스터를 사용하지 않는 경우에서 신뢰할 수 있는 인증서를 수동으로 부트스트랩한 후에서 관리하는 클라이언트 인증 인증서로 전환할 aws-privateca-issuer 수 있습니다cert-manager. 이렇게 하면가 클라이언트 인증 인증서를 자동으로 갱신cert-manager할 수 있습니다.

1. 라는 파일을 생성합니다pca-auth-cert.yaml.

   ```
   cat > pca-auth-cert.yaml <<EOF
   apiVersion: cert-manager.io/v1
   kind: Certificate
   metadata:
     name: aws-privateca-client-cert
     namespace: aws-privateca-issuer
   spec:
     secretName: aws-privateca-credentials
     duration: 168h
     renewBefore: 48h
     commonName: aws-privateca-issuer
     privateKey:
   ```

```
    algorithm: ECDSA
    size: 256
    rotationPolicy: Always
  usages:
    - client auth
  issuerRef:
    name: aws-privateca-cluster-issuer
    kind: AWSPCAClusterIssuer
    group: awspca.cert-manager.io
EOF
```

2. 새 관리형 클라이언트 인증 인증서를 생성합니다.

```
kubectl apply -f pca-auth-cert.yaml
```

3. 인증서가 생성되었는지 확인합니다.

```
kubectl get certificate aws-privateca-client-cert -n aws-privateca-issuer
```

다음과 유사한 응답이 나타납니다.

```
NAME                         READY   SECRET                        AGE
aws-privateca-client-cert    True    aws-privateca-credentials     19m
```

첫 번째 TLS 인증서 발급

이제 `cert-manager` 및가 설치`aws-privateca-issuer`되었으므로 인증서를 발급할 수 있습니다.

인증서 리소스가 `certificate.yaml` 포함된 라는 YAML 파일을 생성합니다.

```
cat > certificate.yaml <<EOF
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: example-certificate
  namespace: default
spec:
  secretName: example-certificate-tls
  issuerRef:
    name: aws-privateca-cluster-issuer
    kind: AWSPCAClusterIssuer
```

```
    group: awspca.cert-manager.io
  commonName: example.internal
  dnsNames:
    - example.internal
    - api.example.internal
  duration: 2160h # 90 days
  renewBefore: 360h # 15 days
  usages:
    - digital signature
    - key encipherment
    - server auth
EOF
```

다음 명령을 사용하여 인증서를 적용합니다.

```
kubectl apply -f certificate.yaml
```

그런 다음 다음 명령을 사용하여 인증서의 상태를 확인할 수 있습니다.

```
kubectl get certificate example-certificate
kubectl describe certificate example-certificate
```

다음과 비슷한 응답이 표시됩니다.

```
NAME                  READY   SECRET                     AGE
example-certificate   True    example-certificate-tls    30s
```

다음 명령을 사용하여 발급된 인증서를 검사할 수 있습니다.

```
kubectl get secret example-certificate-tls -o yaml
```

다음 명령을 사용하여 인증서를 디코딩하고 검사할 수도 있습니다.

```
kubectl get secret example-certificate-tls -o jsonpath='{.data.tls\.crt}' | base64 -d |
 openssl x509 -text -noout
```

예시

다음 예제에서는 AWS Private CA 를 Kubernetes 클러스터와 함께 사용하는 방법을 보여줍니다.

- 샘플: Kubernetes의 전송 중 암호화
- AWS Private CA 및 Amazon EKS를 사용하는 TLS 지원 Kubernetes 클러스터
- new AWS Load Balancer 컨트롤러를 사용하여 Amazon EKS에서 end-to-end TLS 암호화 설정

를 사용하여 Kubernetes 모니터링 AWS Private CA

Kubernetes 클러스터 프라이빗 CA를 모니터링하려면에 설명된 기술을 사용합니다AWS Private CA 리소스 모니터링. 다음을 사용하여 프라이빗 CA를 모니터링할 수 있습니다.

- AWS Private CA CloudWatch 지표
- CloudWatch Events AWS Private CA 로 모니터링
- 를 사용하여 AWS Private Certificate Authority API 호출 로깅 AWS CloudTrail

를 사용하여 Kubernetes 문제 해결 AWS Private CA

다음 절차에 `aws-private-ca-issuer` 따라에 대한 로그를 가져올 수 있습니다.

1. 포드의 이름을 가져옵니다.

```
kubectl get pods -A
```

2. 발급자 로그를 보려면 다음 명령을 사용합니다.

```
kubectl logs -n aws-privateca-issuer <pod-name> aws-privateca-issuer
```

3. IAM Roles Anywhere 로그를 보려면 다음 명령을 사용합니다.

```
kubectl logs -n aws-privateca-issuer <pod-name> rolesanywhere-credentials-helper
```

AWS Private CA 발급자의 상태를 확인하려면 다음 중 하나를 사용합니다.

발급자가 준비되었는지 확인하려면 다음 명령을 사용합니다.

```
kubectl get AWSPCAClusterIssuers -o json | jq '.items[].status
```

응답은 다음과 비슷해야 합니다.

```
{
  "conditions": [
    {
      "lastTransitionTime": "2024-07-03T13:56:37Z",
      "message": "Issuer verified",
      "reason": "Verified",
      "status": "True",
      "type": "Ready"
    }
  ]
}
```

발급자가 Ready 상태가 아닌 경우 message 필드는 발급자가 Ready 상태에 도달하지 못한 이유에 대한 정보를 제공합니다.

인증서가 준비되었는지 확인하려면 다음 명령을 사용합니다.

```
kubectl get certificates -o json | jq '.items[].status'
```

응답은 다음과 비슷해야 합니다.

```
{
  "conditions": [
    {
      "lastTransitionTime": "2024-07-03T13:58:13Z",
      "message": "Certificate is up to date and has not expired",
      "observedGeneration": 1,
      "reason": "Ready",
      "status": "True",
      "type": "Ready"
    }
  ],
  "notAfter": "2024-10-01T13:58:12Z",
  "notBefore": "2024-07-03T12:58:12Z",
  "renewalTime": "2024-09-16T13:58:12Z",
  "revision": 1
}
```

인증서가 Ready 상태가 아닌 경우 message 필드는 인증서가 Ready 상태에 도달하지 못한 이유에 대한 정보를 제공합니다.

AWS Private CA Active Directory용 커넥터

AWS Private CA 는에 필요한 인증서를 발급하고 관리할 수 있습니다 AWS Managed Microsoft AD. Active Directory용 AWS Private CA 커넥터(AD용 커넥터)를 사용하면 온프레미스 엔터프라이즈 또는 기타 타사 CA를 소유한 관리형 프라이빗 CA로 교체하여 AD에서 관리하는 사용자, 그룹 및 컴퓨터에 인증서를 등록할 수 있습니다.

와 함께 AD용 커넥터를 사용하여 AD 및 퍼블릭 키 인프라를 클라우드로 마이그레이션하여 온프레미스 인프라를 AWS Managed Microsoft AD 제거할 수 있습니다. 온프레미스 AD와 AWS Private CA 함께를 사용하려는 고객의 경우이 기능은 AWS Managed Microsoft AD 커넥터와도 통합됩니다.

주제

- AD 사용자를 위한 커넥터를 처음 사용하시나요?
- AD용 Connector 설정
- Active Directory용 AWS Private CA 커넥터 시작하기
- AWS Private CA Active Directory용 커넥터
- Amazon EventBridge를 사용하여 AD용 커넥터를 이벤트 기반 애플리케이션에 통합
- Active Directory용 AWS Private CA 커넥터 관련 문제 해결

AD 사용자를 위한 커넥터를 처음 사용하시나요?

AD용 커넥터를 처음 사용할 경우 먼저 다음 섹션을 읽을 것을 권장합니다.

- AWS Private CA란 무엇인가요?
- Directory Service란 무엇입니까?

AD용 액세스 커넥터

콘솔 AWS CLI, 및 APIs. AWS Private CA 콘솔, 콘솔 또는 검색 AWS Management Console 창에서 AD용 커넥터를 검색하여 Directory Service 콘솔의 커넥터에 액세스할 수 있습니다.

가격 책정

AD용 커넥터는 추가 비용 AWS Private CA 없이의 기능으로 제공됩니다. 프라이빗 인증 기관 및 해당 기관을 통해 발급한 인증서에 대한 비용만 지불하면 됩니다.

최신 AWS Private CA 요금 정보는 AWS Private Certificate Authority 요금을 참조하세요. AWS 요금 계산기를 사용하여 비용을 추정할 수도 있습니다.

AD용 Connector 설정

이 섹션의 단계는 AD용 커넥터를 사용하기 위한 사전 조건입니다. 이미 AWS 계정을 생성했다고 가정합니다. 이 페이지의 단계를 완료한 후 AD용 커넥터 생성을 시작할 수 있습니다.

1단계:를 사용하여 프라이빗 CA 생성 AWS Private CA

디렉터리 객체에 인증서를 발급하기 위한 사설 인증 기관(CA)을 설정합니다. 자세한 내용은 의 인증 기관 AWS Private CA 단원을 참조하십시오.

AD용 커넥터를 생성하려면 프라이빗 CA가 `Active` 상태여야 합니다. 프라이빗 CA의 보안 주체 이름에는 일반 이름이 포함되어야 합니다. 일반 이름이 없는 프라이빗 CA를 사용하여 커넥터를 만들려고 하면 커넥터 생성이 실패합니다.

2단계: Active Directory 설정

프라이빗 CA 외에도 Virtual Private Cloud(VPC)에 Active Directory가 필요합니다. AD용 커넥터는 Directory Service에서 제공하는 다음과 같은 디렉터리 유형을 지원합니다.

- AWS 관리형 Microsoft Active Directory:를 Directory Service 사용하면 Microsoft Active Directory(AD)를 관리형 서비스로 실행할 수 있습니다. AWS Directory Service for Microsoft Active Directory 라고도 하는 AWS Managed Microsoft AD는 Windows Server 2019로 구동됩니다. 를 사용하면 Microsoft Sharepoint AWS 클라우드및 사용자 지정 .Net 및 SQL Server 기반 애플리케이션을 포함하여에서 디렉터리 인식 워크로드를 실행할 AWS Managed Microsoft AD수 있습니다.
- Active Directory Connector: AD Connector는 클라우드에 정보를 캐싱하지 않고 디렉터리 요청을 온프레미스 Microsoft Active Directory로 리디렉션할 수 있는 디렉터리 게이트웨이입니다. AD Connector는 Amazon EC2에 호스팅된 도메인으로의 연결을 지원합니다.

(Active Directory 커넥터만 해당) 3단계: 서비스 계정에 권한 위임

> **Note**
>
> 사용 중인 경우 디렉터리 AWS Managed Microsoft AD 로 AD용 커넥터 서비스에 권한을 부여하면 추가 권한이 자동으로 위임됩니다. 이 사전 필수 단계를 건너뛸 수 있습니다.

Directory Service AD Connector를 사용하는 경우 서비스 계정에 추가 권한을 위임해야 합니다. 서비스 계정에 액세스 제어 목록(ACL)을 설정하여 다음 기능을 허용합니다.

- 서비스 보안 주체 이름(SPN)을 자체 추가 및 제거합니다.
- 다음 컨테이너에서 인증 기관을 생성하고 업데이트합니다.

```
#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
```

- NTAuthCertificate 인증 기관(CA) 객체를 생성하고 업데이트합니다. 참고: NTAuthCertificate CA 객체가 있는 경우 해당 객체에 대한 권한을 위임해야 합니다. 객체가 없는 경우 퍼블릭 키 서비스 컨테이너에 하위 객체를 생성할 권한을 위임해야 합니다.

```
#objects
CN=NTAuthCertificates,CN=Public Key Services,CN=Services,CN=Configuration
```

공식 Active Directory용 커넥터 리포지토리에서 사용할 수 있는 PowerShell 스크립트를 사용하여 Directory Service AD Connector 서비스 계정에 필요한 추가 권한을 위임할 수 있습니다.

이 스크립트는 NTAuthCertificates 인증 기관 객체를 생성합니다.

최신 버전의 스크립트 및 사용 세부 정보는 GitHub 리포지토리의 README를 참조하세요.

4단계: IAM 정책 생성

AD용 커넥터를 만들려면 커넥터 리소스를 만들고, 프라이빗 CA를 AD용 커넥터 서비스와 공유하고, 디렉터리를 통해 AD용 커넥터 서비스를 승인할 수 있는 IAM 정책이 필요합니다.

다음은 사용자 관리형 정책의 예제입니다.

JSON

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

```
            "Effect": "Allow",
            "Action": "pca-connector-ad:*",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "acm-pca:DescribeCertificateAuthority",
                "acm-pca:GetCertificate",
                "acm-pca:GetCertificateAuthorityCertificate",
                "acm-pca:ListCertificateAuthorities",
                "acm-pca:ListTags",
                "acm-pca:PutPolicy"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "acm-pca:IssueCertificate",
            "Resource": "*",
            "Condition": {
                "ArnLike": {
                    "acm-pca:TemplateArn": "arn:aws:acm-pca:::template/
BlankEndEntityCertificate_APIPassthrough/V*"
                },
                "ForAnyValue:StringEquals": {
                    "aws:CalledVia": "pca-connector-ad.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "ds:AuthorizeApplication",
                "ds:DescribeDirectories",
                "ds:ListTagsForResource",
                "ds:UnauthorizeApplication",
                "ds:UpdateAuthorizedApplication"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
```

```
                "ec2:CreateVpcEndpoint",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcEndpoints",
                "ec2:DescribeVpcs",
                "ec2:DeleteVpcEndpoints"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeTags",
                "ec2:DeleteTags",
                "ec2:CreateTags"
            ],
            "Resource": "arn:*:ec2:*:*:vpc-endpoint/*"
        }
    ]
}
```

AD용 커넥터에는 콘솔 및 명령줄 사용에 대한 추가 AWS RAM 권한이 필요합니다.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "ram:CreateResourceShare",
            "Resource": "*",
            "Condition": {
                "StringEqualsIfExists": {
                    "ram:Principal": "pca-connector-ad.amazonaws.com",
                    "ram:RequestedResourceType": "acm-pca:CertificateAuthority"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
```

```
                "ram:GetResourcePolicies",
                "ram:GetResourceShareAssociations",
                "ram:GetResourceShares",
                "ram:ListPrincipals",
                "ram:ListResources",
                "ram:ListResourceSharePermissions",
                "ram:ListResourceTypes"
            ],
            "Resource": "*"
        }
    ]
}
```

5단계: AD용 커넥터와 프라이빗 CA 공유

AWS Resource Access Manager 서비스 보안 주체 공유를 사용하여 커넥터 서비스와 프라이빗 CA를 공유해야 합니다.

AWS 콘솔에서 커넥터를 생성하면 리소스 공유가 자동으로 생성됩니다.

를 사용하여 리소스 공유를 생성할 때 create-resource-share 명령을 AWS CLI AWS RAM 사용합니다.

다음 명령은 리소스 공유를 생성합니다.

```
$  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
```

CreateConnector를 호출하는 서비스 보안 주체는 PCA에 대한 인증서 발급 권한을 가집니다. AD용 커넥터를 사용하는 서비스 보안 주체가 AWS Private CA 리소스에 대한 일반적인 액세스 권한을 갖지 못하도록 하려면 `CalledVia`를 사용하여 해당 사용 권한을 제한합니다.

6단계: 디렉터리 등록 생성

커넥터가 디렉터리와 통신할 수 있도록 디렉터리로 AD용 커넥터 서비스를 승인합니다. AD용 커넥터 서비스를 승인하려면 디렉터리 등록을 생성해야 합니다. 디렉터리 등록 생성에 대한 자세한 내용은 디렉터리 등록 관리 섹션을 참조하세요.

7단계: 보안 그룹 구성

VPC와 AD 커넥터용 커넥터 간의 통신은를 통해 이루어 AWS PrivateLink지므로 VPC에서 포트 443 TCP를 여는 인바운드 규칙이 있는 보안 그룹(들)이 필요합니다. 커넥터를 생성하는 경우 이 보안 그룹을 입력하라는 메시지가 표시됩니다. 소스를 사용자 지정으로 지정하고 VPC의 CIDR 블록을 선택할 수 있습니다. 이를 더 제한하도록 선택할 수 있습니다(예: IP, CIDR, 보안 그룹 ID).

8단계: 디렉터리 객체에 대한 네트워크 액세스 구성

디렉터리 객체는 다음 도메인에서 온라인 인증서 상태 프로토콜(OCSP) 및 인증서 해지 목록(CRLs)을 검증하기 위해 퍼블릭 인터넷 액세스가 필요합니다.

```
*.windowsupdate.com
*.amazontrust.com
```

최소 필수 액세스 규칙:

- OCSP 및 CRL 통신에 필요합니다.

  ```
  TCP 80: (HTTP) to 0.0.0.0/0
  ```

- AD용 커넥터에 필요합니다.

  ```
  TCP 443: (HTTPS) to 0.0.0.0/0
  ```

- Active Directory에 필요:

  ```
  TCP 88: (Kerberos) to Domain Controller IP range
  TCP/UDP 389/636: (LDAP/LDAPS) to Domain Controller IP range, depending on Domain
   Controller configuration
  TCP/UDP 53: (DNS) to 0.0.0.0/0
  ```

디바이스에 퍼블릭 인터넷 액세스 권한이 없는 경우 오류 코드와 함께 인증서 발급이 간헐적으로 실패합니다. `WS_E_OPERATION_TIMED_OUT`.

> **Note**
>
> Amazon EC2 인스턴스에 대한 보안 그룹을 구성하는 경우 7단계에서 동일한 그룹일 필요는 없습니다.

Active Directory용 AWS Private CA 커넥터 시작하기

Active Directory용 AWS Private CA 커넥터를 사용하면 인증 및 암호화를 위해 프라이빗 CA에서 Active Directory 객체로 인증서를 발급할 수 있습니다. 커넥터를 만들면 AWS Private Certificate Authority 가 디렉터리 객체가 인증서를 요청할 수 있도록 VPC에 엔드포인트를 생성합니다.

인증서를 발급하려면 커넥터와 커넥터용 AD 호환 템플릿을 만들어야 합니다. 템플릿을 생성하는 경우 AD 그룹에 대한 등록 권한을 설정할 수 있습니다.

주제

- 시작하기 전 준비 사항
- 1단계: 커넥터 생성
- 2단계: Microsoft Active Directory 정책 구성
- 3단계: 템플릿 생성
- 4단계: Microsoft 그룹 권한 구성

시작하기 전 준비 사항

다음 자습서에서는 AD용 커넥터 및 커넥터 템플릿을 생성하는 프로세스를 안내합니다. 이 자습서를 따르려면 먼저 섹션에 나열된 사전 조건을 충족해야 합니다.

1단계: 커넥터 생성

커넥터를 생성하려면 섹션을 참조하세요Active Directory용 커넥터 생성.

2단계: Microsoft Active Directory 정책 구성

AD용 커넥터는 고객의 그룹 정책 객체(GPO) 구성을 보거나 관리할 수 없습니다. GPO는 AD 요청을 고객의 AWS Private CA 또는 다른 인증 또는 인증서 판매 서버로 라우팅하는 것을 제어합니다. 잘못된 GPO 구성으로 인해 요청이 잘못 라우팅될 수 있습니다. AD용 커넥터를 구성하고 테스트하는 것은 고객의 몫입니다.

그룹 정책은 커넥터와 연결되며 단일 AD에 대해 여러 커넥터를 생성하도록 선택할 수 있습니다. 그룹 정책 구성이 다른 경우 각 커넥터에 대한 액세스 제어를 관리하는 것은 사용자의 몫입니다.

데이터 플레인 호출의 보안은 Kerberos 및 VPC 구성에 따라 달라집니다. VPC에 대한 액세스 권한이 있는 사람은 누구나 해당 AD에 인증되어 있는 한 데이터 플레인 호출을 할 수 있습니다. 이는 AWSauth 경계 외부에 존재하며 권한 부여 및 인증 관리는 고객 여러분에게 달려 있습니다.

를 Directory Service enable-ca-enrollment-policy 사용하는 경우 명령을 AWS Managed Microsoft AD 사용하여 AWS Managed Microsoft AD 인스턴스의 도메인 컨트롤러에서 GPOs를 구성합니다.

다음 명령은 도메인 컨트롤러 등록을 활성화합니다.

```
$  aws ds enable-ca-enrollment-policy \
    --pca-connector-arn MyPcaConnectorAdArn \
    --directory-id MyDirectoryId
```

AD 커넥터를 사용하는 경우 다음 단계에 따라 커넥터를 생성할 때 생성된 URI를 가리키는 GPO를 생성합니다. 콘솔 또는 명령줄에서 AD용 커넥터를 사용하려면 이 단계가 필요합니다.

GPO를 구성합니다.

1. DC에서 서버 관리자를 엽니다.
2. 도구로 이동하여 콘솔 오른쪽 상단에 있는 그룹 정책 관리를 선택합니다.
3. 포리스트 > 도메인으로 이동합니다. 도메인 이름을 선택하고 도메인을 마우스 오른쪽 버튼으로 클릭합니다. 이 도메인에서 GPO를 만들고 여기에 연결...을 선택한 다음 이름에 PCA GPO를 입력합니다.
4. 이제 새로 만든 GPO가 도메인 이름 아래에 표시됩니다.
5. PCA GPO를 선택하고 편집을 선택합니다. 이 링크이며 변경 내용이 전역으로 전파된다는 경고 메시지가 표시된 대화 상자가 열리면 계속하려면 메시지를 확인합니다. 그룹 정책 관리 편집기가 열려야 합니다.
6. 그룹 정책 관리 편집기에서 컴퓨터 구성 > 정책 > Windows 설정 > 보안 설정 > 공개 키 정책(폴더 선택)으로 이동합니다.

7. 객체 유형으로 이동하여 인증서 서비스 클라이언트 - 인증서 등록 정책을 선택합니다.
8. 옵션에서 구성 모델을 활성화로 변경합니다.
9. Active Directory 등록 정책이 선택되어 있고 활성화되었는지 확인합니다. 추가를 선택합니다.
10. 인증서 등록 정책 서버 창이 열려야 합니다.
11. 등록 서버 정책 URI 입력 필드에 커넥터를 만들 때 생성된 인증서 등록 정책 서버 엔드포인트를 입력합니다.
12. 인증 유형은 Windows 통합으로 그대로 둡니다.
13. 검증을 선택합니다. 검증에 성공한 후 추가를 선택합니다. 대화 상자가 닫힙니다.
14. 인증서 서비스 클라이언트 - 인증서 등록 정책으로 돌아가서 새로 만든 커넥터 옆의 확인란을 선택하여 커넥터가 기본 등록 정책인지 확인합니다.
15. Active Directory 등록 정책을 선택하고 제거를 선택합니다.
16. 확인 대화 상자에서 예를 선택하여 LDAP 기반 인증을 삭제합니다.
17. 인증서 서비스 클라이언트 > 인증서 등록 정책 창에서 적용 및 확인을 선택하고 창을 닫습니다.
18. 공개 키 정책 폴더로 이동하여 인증서 서비스 클라이언트 - 자동 등록을 선택합니다.
19. 구성 모델 옵션을 활성화로 변경합니다.
20. 만료된 인증서 갱신과 인증서 업데이트가 모두 선택되어 있는지 확인합니다. 다른 설정은 현재 값 그대로 둡니다.
21. 적용을 선택한 다음 확인을 선택하고 대화 상자를 닫습니다.

다음으로 사용자 구성을 위한 퍼블릭 키 정책을 구성합니다. 사용자 구성 > 정책 > Windows 설정 > 보안 설정 > 퍼블릭 키 정책으로 이동합니다. 6단계부터 21단계까지 설명된 절차에 따라 사용자 구성을 위한 공개 키 정책을 구성합니다.

GPOs 및 퍼블릭 키 정책 구성을 완료하면 도메인의 객체가 AD용 AWS Private CA 커넥터에서 인증서를 요청하고에서 발급한 인증서를 가져옵니다 AWS Private CA.

3단계: 템플릿 생성

템플릿을 생성하려면 섹션을 참조하세요커넥터 템플릿 생성.

4단계: Microsoft 그룹 권한 구성

Microsoft 그룹 권한을 구성하려면 섹션을 참조하세요AD 템플릿용 커넥터 액세스 제어 항목 관리.

AWS Private CA Active Directory용 커넥터

이 섹션의 절차에서는 Active Directory(AD) 커넥터를 생성하고, 템플릿을 구성하고, AWS Private CA 및 Active Directory와 통합하는 방법을 설명합니다. AD용 AWS Private CA 커넥터 콘솔에서 또는의 AD용 커넥터 섹션을 사용하거나 AD용 커넥터 API를 AWS CLI AWS Private CA 사용하여 이러한 작업을 수행할 수 있습니다.

> **Note**
>
> AD용 AWS Private CA 커넥터는와 긴밀하게 통합되어 있지만 AWS Private CA두 서비스에는 별도의 APIs. 자세한 내용은 AWS Private Certificate Authority API 참조 및 Active DirectoryI용 AWS Private CA 커넥터 API 참조를 참조하세요.

Active Directory용 커넥터 생성

다음 절차에 따라 콘솔, 명령줄 또는 API for Connector for Active Directory를 사용하여 AWS Private CA 커넥터를 생성합니다.

Console

콘솔을 사용하여 커넥터를 생성하려면

AWS 계정에 로그인하고에서 Active Directory용 AWS Private CA 커넥터 콘솔을 엽니다**https://console.aws.amazon.com/pca-connector-ad/home**.

1. 최초 서비스 랜딩 페이지 또는 Active Directory용 커넥터 페이지에서 커넥터 생성을 선택합니다.
2. Active Directory용 프라이빗 CA 커넥터 만들기 페이지에서 Active Directory 섹션에 정보를 제공합니다.
   - Active Directory 유형 선택에서 사용 가능한 두 가지 유형 중 하나를 선택합니다.
     - AWS Directory Service for Microsoft Active Directory -에서 관리하는 Active Directory를 지정합니다 Directory Service.
     - AWS AD 커넥터를 포함한 온프레미스 Active Directory - AD 커넥터를 사용하여 온프레미스에서 호스팅하는 Active Directory에 액세스합니다.
   - 디렉터리 선택 아래의 목록에서 디렉터리를 선택합니다.

또는 디렉터리 생성을 선택하여 Directory Service 콘솔을 새 창에서 열 수도 있습니다. 새 디렉터리 생성을 마치면 Active Directory용 AWS Private CA 커넥터 콘솔로 돌아가 디렉터리 목록을 새로 고칩니다. 새 디렉터리를 선택할 수 있어야 합니다.

> **Note**
>
> 디렉터리를 생성할 때는 AD용 커넥터가 Directory Service 콘솔에서 제공하는 다음 디렉터리 유형만 지원한다는 점에 유의하십시오.
>
> - AWS 관리형 Microsoft AD
> - AD Connector

- VPC 엔드포인트의 보안 그룹 선택에서 목록에서 보안 그룹을 선택합니다.

  또는 보안 그룹 생성을 선택할 수 있습니다. 그러면 Amazon EC2 콘솔이 새 창에 보안 그룹 생성 페이지로 열립니다. 보안 그룹 생성을 마치면 Active Directory용 AWS Private CA 커넥터 콘솔로 돌아가 보안 그룹 목록을 새로 고칩니다. 새 보안 그룹을 선택할 수 있어야 합니다.

3. IP 주소 유형 섹션에서 다음 옵션 중에서 선택합니다.

   - IPv4 - 서비스에 대한 IPv4 연결을 활성화합니다. 디렉터리를 호스팅하는 모든 서브넷에 IPv4 주소 범위가 있는 경우에만이 옵션을 선택합니다.
   - 듀얼 스택 - 서비스에 대한 IPv4 및 IPv6 연결을 모두 활성화합니다. 디렉터리를 호스팅하는 모든 서브넷에 IPv4 및 IPv6 주소 범위가 모두 있는 경우에만이 옵션을 선택합니다.

4. 프라이빗 인증 기관 섹션의 목록에서 프라이빗 CA를 선택합니다.

   또는 프라이빗 CA 생성을 선택하면 콘솔이 새 창에서 AWS Private CA 프라이빗 인증 기관 페이지로 열립니다. CA 생성을 마치면 Active Directory용 AWS Private CA 커넥터 콘솔로 돌아가서 CA 목록을 새로 고칩니다. 새 CA를 선택할 수 있어야 합니다.

5. 태그 - 옵션 창에서 AD 리소스의 메타데이터를 적용하고 제거할 수 있습니다. 태그는 키-값 문자열 페어로, 키는 리소스별로 고유해야 하고 값은 선택사항입니다. 이 창에는 테이블에 있는 리소스의 기존 태그가 모두 표시됩니다. 다음 작업이 지원됩니다.

   - 태그 관리를 선택하여 태그 관리 페이지를 엽니다.
   - 새 태그를 추가하려면 태그 생성을 선택합니다. 키 필드를 채우고 선택적으로 값 필드를 입력합니다. 태그를 적용하려면 변경 사항 저장을 선택합니다.
   - 태그 옆에 있는 제거 버튼을 선택하여 삭제 대상으로 표시하고 변경 사항 저장을 선택하여 확인합니다.

6. 필요한 정보를 제공하고 선택 사항을 검토한 후 커넥터 생성을 선택합니다. 그러면 Active Directory용 커넥터 세부 정보 페이지가 열리고 커넥터가 만들어지는 동안 진행 상황을 볼 수 있습니다.

   커넥터 생성 프로세스가 완료되면 커넥터에 서비스 보안 주체 이름을 할당합니다.

API

API를 사용하여 커넥터를 생성하려면

API를 사용하여 Active Directory용 커넥터를 생성하려면 Active Directory용 AWS Private CA 커넥터 API의 CreateConnector 작업을 사용합니다.

CLI

를 사용하여 커넥터를 생성하려면 AWS CLI

CLI를 사용하여 Active Directory용 커넥터를 생성하려면의 Active Directory용 AWS Private CA 커넥터 섹션에서 create-connector 명령을 사용합니다 AWS CLI.

커넥터 템플릿 생성

템플릿은 인증서가 발급된 후 어떻게 보여야 하는지와 클라이언트가 인증서를 어떻게 처리해야 하는지에 대한 구성 목록입니다. 다음 절차에서는 템플릿을 생성하는 방법을 설명합니다.

Console

콘솔을 사용하여 템플릿을 생성하려면

1. AWS 계정에 로그인하고에서 Active Directory용 AWS Private CA 커넥터 콘솔을 엽니다**https://console.aws.amazon.com/pca-connector-ad/home**.
2. Active Directory용 커넥터 목록에서 커넥터를 선택한 다음 세부 정보 보기를 선택합니다.
3. 커넥터의 세부 정보 페이지에서 템플릿 섹션을 찾은 다음 템플릿 생성을 선택합니다.
4. 템플릿 생성 페이지의 템플릿 생성 방법 섹션에서 방법 옵션 중 하나를 선택합니다.

   - 사전 정의된 템플릿에서 시작(기본값) - AD 애플리케이션용으로 사전 정의된 템플릿 목록에서 선택합니다.
     - 코드 서명
     - 컴퓨터

- 도메인 컨트롤러 인증
- EFS 복구 에이전트
- 등록 에이전트
- 등록 에이전트(컴퓨터)
- IPSec
- Kerberos 인증
- RAS 및 IAS 서버
- 스마트카드 로그온
- 신뢰 목록 서명
- 사용자 서명
- 워크스테이션 인증

- 생성한 기존 템플릿에서 시작 - 이전에 생성한 사용자 지정 템플릿 목록에서 선택합니다.
- 빈 템플릿에서 시작 - 이 옵션을 선택하면 완전히 새로운 템플릿을 만들기 시작할 수 있습니다.

5. 인증서 설정 섹션에서 이 템플릿을 기반으로 하는 인증서에 대한 다음 설정을 정의합니다.

- 인증서 유형 - 사용자 인증서를 만들지 컴퓨터 인증서를 만들지 여부를 지정합니다.
- 자동 등록 - 이 템플릿을 기반으로 인증서 자동 등록을 활성화할지 여부를 선택합니다.
- 유효 기간 - 인증서 유효 기간을 시간, 일, 주, 월 또는 년의 정수 값으로 지정합니다. 최소값은 2시간입니다.
- 갱신 기간 - 인증서 갱신 기간을 시간, 일, 주, 월 또는 년의 정수 값으로 지정합니다. 갱신 기간은 유효 기간의 75% 를 넘지 않아야 합니다.
- 보안 주체 이름 - Active Directory에 포함된 정보를 기반으로 보안 주체 이름에 포함할 옵션을 하나 이상 선택합니다.

> **Note**
>
> 보안 주체 이름 또는 보안 주체 대체 이름 옵션을 하나 이상 지정해야 합니다.

- 일반 이름
- 일반 이름으로서의 DNS
- 디렉터리 경로

- 이메일

- 보안 주체 대체 이름 - Active Directory에 포함된 정보를 기반으로 보안 주체 대체 이름에 포함할 옵션을 하나 이상 선택합니다.

> **Note**
>
> 보안 주체 이름 또는 보안 주체 대체 이름 옵션을 하나 이상 지정해야 합니다.

  - 디렉터리 GUID
  - DNS 이름
  - 도메인 DNS
  - 이메일
  - 서비스 보안 주체 이름(SPN)
  - 사용자 보안 주체 이름(UPN)

6. 인증서 요청 처리 및 등록 옵션 섹션에서 다음 옵션 중 하나를 선택하여 템플릿을 기반으로 인증서의 용도를 지정합니다.

   - 서명
   - 암호화(Encryption)
   - 서명 및 암호화
   - 서명 및 스마트카드 로그온

   다음으로, 다음 기능 중 활성화할 기능을 선택합니다. 옵션은 인증서 용도에 따라 다릅니다.

   - 유효하지 않은 인증서 삭제(보관하지 않음)
   - 대칭 알고리즘 포함
   - 내보낼 수 있는 프라이빗 키

   마지막으로 인증서 등록 옵션을 선택합니다. 옵션은 인증서 용도에 따라 다릅니다.

   - 사용자 입력이 필요하지 않음
   - 등록 시 사용자에게 메시지 표시
   - 등록 중에 사용자에게 메시지를 표시하고 사용자 입력을 요구함

7. 애플리케이션 정책 섹션에서 적용되는 모든 애플리케이션 정책을 선택합니다. 사용 가능한 정책은 여러 페이지에 걸쳐 나열되어 있습니다. 이전 설정 때문에 일부 정책이 미리 선택되어 있을 수 있습니다.
8. 사용자 지정 애플리케이션 정책 섹션에서 템플릿에 사용자 지정 OID를 추가하고 애플리케이션 정책 확장이 중요한지 여부를 지정할 수 있습니다.
9. 암호화 설정 섹션에서 이 템플릿을 기반으로 하는 인증서에 대한 다음 범주의 암호화 설정을 선택합니다.
10. 그룹 및 권한 섹션에서 템플릿, 기존 그룹 및 등록 권한을 보거나, 새 그룹 및 권한 추가 버튼을 선택하여 새 그룹 및 권한을 추가할 수 있습니다. 버튼을 누르면 다음 정보가 필요한 양식이 열립니다.

    - 표시 이름
    - 보안 식별자(SID)
    - 등록(허용 | 거부 | 설정 안 함 옵션 포함)
    - 자동 등록(허용 | 거부 | 설정 안 함 옵션 포함)
11. 대체 템플릿 섹션에서 현재 템플릿이 AD에서 만든 하나 이상의 템플릿을 대체한다는 사실을 Active Directory에 알릴 수 있습니다. 대체할 Active Directory에서 템플릿 추가를 선택하고 대체 템플릿의 일반 이름을 지정하여 대체 템플릿을 적용합니다.
12. 태그 - 옵션 창에서 AD 리소스의 메타데이터를 적용하고 제거할 수 있습니다. 태그는 키-값 문자열 페어로, 키는 리소스별로 고유해야 하고 값은 선택사항입니다. 이 창에는 테이블에 있는 리소스의 기존 태그가 모두 표시됩니다. 다음 작업이 지원됩니다.

    - 태그 관리를 선택하여 태그 관리 페이지를 엽니다.
    - 새 태그를 추가하려면 태그 생성을 선택합니다. 키 필드를 채우고 선택적으로 값 필드를 입력합니다. 태그를 적용하려면 변경 사항 저장을 선택합니다.
    - 태그 옆에 있는 제거 버튼을 선택하여 삭제 대상으로 표시하고 변경 사항 저장을 선택하여 확인합니다.
13. 필요한 정보를 제공하고 선택 사항을 검토한 후 템플릿 생성을 선택합니다. 그러면 새 템플릿의 설정을 검토하고, 템플릿을 편집 또는 삭제하고, 그룹 및 권한을 관리하고, 대체된 템플릿을 관리하고, 태그를 관리하고, 인증서 보유자의 자동 재등록을 설정할 수 있는 템플릿 세부 정보가 열립니다.

API

API를 사용하여 커넥터 템플릿을 생성하려면

Active Directory용 AWS Private CA 커넥터 API에서 CreateTemplate 작업을 사용합니다.

CLI

를 사용하여 커넥터 템플릿을 생성하려면 AWS CLI

의 Active Directory용 AWS Private CA 커넥터 섹션에서 create-template 명령을 사용합니다 AWS CLI.

Active Directory용 템플릿 업데이트

다음 절차에 따라 콘솔, 명령줄 또는 API for AWS Private CA Connector for Active Directory를 사용하여 템플릿을 업데이트합니다.

Console

콘솔을 사용하여 템플릿을 업데이트하려면

AWS 계정에 로그인하고에서 Active Directory용 AWS Private CA 커넥터 콘솔을 엽니다**https://console.aws.amazon.com/pca-connector-ad/home**.

1. Active Directory용 커넥터 목록에서 업데이트하려는 템플릿의 커넥터를 선택합니다. 커넥터 템플릿을 보고 수정하려면 편집을 선택합니다.
2. 커넥터의 템플릿 세부 정보 페이지에서 편집을 선택합니다. 프롬프트에 따라 업데이트합니다. 영역 편집이 완료되면 저장을 선택하여 변경 사항을 저장합니다.

API

API를 사용하여 템플릿을 업데이트하려면

API를 사용하여 Active Directory의 템플릿을 업데이트하려면 Active Directory용 AWS Private CA 커넥터 API의 UpdateTemplate 작업을 사용합니다.

CLI

를 사용하여 템플릿을 업데이트하려면 AWS CLI

CLI를 사용하여 Active Directory용 커넥터를 업데이트하려면의 Active Directory용 AWS Private CA 커넥터 섹션에서 update-template 명령을 사용합니다 AWS CLI.

Active Directory용 커넥터가 템플릿 변경 사항을 전파하는 방법

AWS Private CA 는 클라이언트가 8시간마다 정책 캐시를 새로 고칠 때 정책에 템플릿을 적용합니다. 여기에는 템플릿 그룹 액세스 제어 항목에 대한 변경 사항이 포함됩니다. 클라이언트가 캐시를 새로 고치면 커넥터에서 사용 가능한 템플릿을 쿼리합니다. 자동 등록 새로 고침의 경우 클라이언트는 다음 조건 중 하나 또는 둘 다와 일치하는 인증서를 발급합니다.

- 인증서가 갱신 기간 내에 있습니다.
- 클라이언트 디바이스에 인증서가 없습니다.

수동 새로 고침의 경우 클라이언트가 커넥터를 쿼리하므로 템플릿을 발급으로 설정해야 합니다.

디버깅하는 경우 정책 캐시를 수동으로 지워 템플릿 변경 사항을 즉시 확인할 수 있습니다. 이렇게 하려면 클라이언트에서 다음 Powershell 명령을 실행합니다.

```
certutil -f -user -policyserver * -policycache delete
```

Active Directory용 커넥터 나열

Active Directory용 AWS Private CA 커넥터 콘솔 또는를 사용하여 소유한 커넥터를 나열 AWS CLI 할 수 있습니다.

Console

콘솔을 사용하여 커넥터를 나열하는 방법

1. AWS 계정에 로그인하고에서 Active Directory용 AWS Private CA 커넥터 콘솔을 엽니다https://console.aws.amazon.com/pca-connector-ad/home.
2. Active Directory용 커넥터 목록에 있는 정보를 검토합니다. 오른쪽 상단의 페이지 번호를 사용하여 여러 페이지의 커넥터를 탐색할 수 있습니다. 각 커넥터는 기본적으로 다음과 같은 정보 열을 표시하는 행을 차지합니다.

- 커넥터 ID - 커넥터의 고유 ID입니다.
- 디렉터리 이름 - 커넥터와 관련된 Active Directory 리소스입니다.
- 커넥터 상태 - 커넥터 상태입니다. 가능한 값은 생성 | 활성 | 삭제 | 실패입니다.
- 서비스 보안 주체 이름 상태 - 커넥터와 연결된 서비스 보안 주체 이름(SPN)의 상태입니다. 가능한 값은 생성 | 활성 | 삭제 | 실패입니다.

- 디렉토리 등록 상태 - 보조 디렉터리의 등록 상태. 가능한 값은 생성 | 활성 | 삭제 | 실패입니다.
- 생성 시간 - 커넥터 생성 시점의 타임스탬프.

콘솔 상단 오른쪽의 기어 모양 아이콘을 선택하면 페이지 크기 설정을 사용하여 페이지에 표시되는 커넥터 수를 사용자 지정할 수 있습니다.

API

API를 사용하여 커넥터를 나열하는 방법

Active Directory용 AWS Private CA 커넥터 API에서 ListConnectors 작업을 사용합니다.

CLI

를 사용하여 커넥터를 나열하려면 AWS CLI

list-connectors 명령을 사용하여 커넥터를 나열합니다.

커넥터 템플릿 나열

Active Directory용 AWS Private CA 커넥터 콘솔 또는 AWS CLI 를 사용하여 소유한 커넥터의 템플릿을 나열할 수 있습니다. 커넥터 템플릿은 AWS Private CA BlankEndEntityCertificate_APIPassthrough/V1 템플릿을 기반으로 합니다.

Console

콘솔을 사용하여 리스너를 업데이트하는 방법

1. AWS 계정에 로그인하고에서 Active Directory용 AWS Private CA 커넥터 콘솔을 엽니다**https://console.aws.amazon.com/pca-connector-ad/home**.
2. Active Directory용 커넥터 목록에서 커넥터를 선택한 다음 세부 정보 보기를 선택합니다.
3. 커넥터 세부 정보 페이지에서 템플릿 섹션의 정보를 검토합니다. 오른쪽 상단의 페이지 번호를 사용하여 여러 페이지의 템플릿을 탐색할 수 있습니다. 각 템플릿은 다음 정보 열을 표시하는 행을 차지합니다.

- 템플릿 이름 - 사람이 읽을 수 있는 템플릿의 이름입니다.
- 템플릿 상태 - 템플릿의 상태입니다. 가능한 값은 다음과 같습니다. 활성 | 삭제.
- 템플릿 ID - 템플릿의 고유 식별자입니다.

API

API를 사용하여 커넥터를 나열하는 방법

Active Directory용 AWS Private CA 커넥터 API의 ListTemplates 작업을 사용하여 지정된 커넥터의 템플릿을 나열합니다.

CLI

를 사용하여 커넥터를 나열하려면 AWS CLI

list-templates 명령을 사용하여 지정된 커넥터의 템플릿을 나열합니다.

커넥터 세부 정보 보기

다음 절차에 따라 콘솔, 명령줄 또는 API for AWS Private CA Connector for Active Directory에서 커넥터의 구성 세부 정보를 봅니다.

Console

콘솔을 사용하여 커넥터에 대한 세부 정보를 보려면

1. AWS 계정에 로그인하고에서 Active Directory용 AWS Private CA 커넥터 콘솔을 엽니다**https://console.aws.amazon.com/pca-connector-ad/home**.
2. Active Directory용 커넥터 목록에서 커넥터를 선택한 다음 세부 정보 보기를 선택합니다.
3. 커넥터 세부 정보 페이지에서 커넥터 세부 정보 창의 정보를 검토합니다. 이 정보에는 다음이 포함됩니다.

   - 커넥터 ID
   - 커넥터 상태
   - 추가 상태 세부 정보
   - 커넥터 ARN
   - 인증서 등록 정책 서버 엔드포인트
   - 디렉터리 이름
   - 디렉터리 ID
   - AWS Private CA 제목
   - AWS Private CA 상태
   - IP 주소 유형

- VPC 엔드포인트 및 보안 그룹

4. 템플릿 창에서 커넥터와 연결된 템플릿을 만들거나 관리할 수 있습니다.
5. 서비스 사용자 이름(SPN) 창에서 커넥터와 관련된 서비스 보안 주체 이름을 볼 수 있습니다.
6. 디렉터리 등록 창에서 커넥터와 관련된 디렉터리 등록을 보거나 변경할 수 있습니다.
7. 태그 - 옵션 창에서 커넥터와 관련된 태그를 만들거나 관리할 수 있습니다.

API

API를 사용하여 커넥터를 나열하는 방법

Active Directory용 AWS Private CA 커넥터 API에서 GetConnector 작업을 사용합니다.

CLI

를 사용하여 커넥터를 나열하려면 AWS CLI

의 Active Directory용 AWS Private CA 커넥터 섹션에서 get-connector 명령을 사용합니다 AWS CLI.

커넥터 템플릿 세부 정보 보기

다음 절차에 따라 콘솔, 명령줄 또는 API for Connector for Active Directory를 사용하여 AWS Private CA 커넥터 템플릿의 구성 세부 정보를 봅니다.

Console

콘솔을 사용하여 커넥터 템플릿의 세부 정보를 보려면

1. AWS 계정에 로그인하고에서 Active Directory용 AWS Private CA 커넥터 콘솔을 엽니다**https://console.aws.amazon.com/pca-connector-ad/home**.
2. Active Directory용 커넥터 목록에서 커넥터를 선택한 다음 세부 정보 보기를 선택합니다.
3. 커넥터 세부 정보 페이지에서 템플릿 섹션의 정보를 검토하고 검사하려는 템플릿을 선택합니다. 그런 다음 세부 정보 보기를 선택합니다.
4. 세부 정보 페이지의 템플릿 세부 정보 창에는 템플릿에 대한 다음 정보가 표시됩니다.

- 템플릿 이름
- 템플릿 ID

- 템플릿 상태
- 템플릿 스키마 버전
- 템플릿 버전
- 템플릿 ARN
- 인증서 유형
- 자동 등록이 켜져 있습니다
- 유효 기간
- 갱신 기간
- 보안 주체 이름 요구 사항
- 보안 주체 대체 이름 요구 사항
- 인증서 요청 및 등록 설정
- 암호화 제공자 카테고리
- 주요 알고리즘
- 최소 키 크기(비트)
- 해시 알고리즘
- 암호화 제공업체
- 키 사용 확장 설정

이 창에서 편집, 삭제 및 작업 버튼을 사용하여 다음 작업을 수행할 수도 있습니다.

- Edit
- 삭제
- 그룹 및 권한 관리 - 자세한 내용은 그룹 및 권한 구성을 참조하세요.
- 대체된 템플릿 관리 - 자세한 내용은 검토 및 생성을 참조하세요.
- 태그 관리 – 자세한 내용은 AD 리소스용 태그 지정 커넥터 섹션을 참조하세요.
- 모든 인증서 보유자 재등록 - 이 설정을 사용하면 템플릿의 주 버전을 자동으로 늘릴 수 있습니다. 템플릿으로 등록할 수 있는 Active Directory 그룹의 모든 구성원은 해당 템플릿을 사용하여 발급된 새 인증서를 받게 됩니다. 자세한 내용은 업데이트 템플릿 API를 참조하세요.

5. 하단 창에는 템플릿 구성을 변경할 수 있는 탭 행이 표시됩니다.

- 그룹 및 권한 - Active Directory 그룹이 이 템플릿을 사용하여 인증서를 등록할 수 있는 권한을 보고 관리합니다. 자세한 내용은 그룹 및 권한 구성을 참조하세요

- 애플리케이션 정책 - 템플릿 애플리케이션 정책을 보고 관리합니다. 자세한 내용은 애플리케이션 할당 정책을 참조하세요.
- 대체된 템플릿 - 대체된 템플릿을 보고 관리합니다. 자세한 내용은 검토 및 생성을 참조하세요.
- 태그 선택 사항 - 이 템플릿에서 태그 지정을 보고 관리합니다. 자세한 내용은 AD 리소스용 태그 지정 커넥터 단원을 참조하십시오.

API

API를 사용하여 커넥터를 나열하는 방법

Active Directory용 AWS Private CA 커넥터 API에서 GetTemplate 작업을 사용합니다.

CLI

를 사용하여 커넥터를 나열하려면 AWS CLI

의 Active Directory용 AWS Private CA 커넥터 섹션에서 get-template 명령을 사용합니다 AWS CLI.

디렉터리 등록 관리

Console

콘솔을 사용하여 디렉터리 등록을 관리하려면

커넥터의 디렉터리 등록은 Active Directory용 AWS Private CA 커넥터 콘솔의 최상위 수준에서 관리할 수 있습니다. 이 항목에서는 사용 가능한 관리 옵션에 대해 설명합니다.

1. AWS 계정에 로그인하고에서 Active Directory용 AWS Private CA 커넥터 콘솔을 엽니다**https://console.aws.amazon.com/pca-connector-ad/home** .
2. 왼쪽 탐색 영역에서 디렉터리 등록을 선택합니다.
3. 디렉터리 등록 페이지에는 다음 필드가 포함된 등록된 디렉터리 테이블이 표시됩니다.

   - 디렉터리 ID - 디렉터리의 고유 ID
   - 디렉터리 이름 - 디렉터리 도메인 사이트 이름
   - 디렉터리 유형
   - 등록됨 - 등록 상태입니다. 지원되는 값은 생성 | 활성 | 삭제 | 실패입니다.

- 디렉터리 상태 - 디렉터리 상태입니다.

사용자는 등록 디렉토리를 사용하여 새 등록을 생성할 수 있습니다.

4. 나열된 등록 중 하나를 선택하여 관리할 수 있습니다. 이렇게 하면 등록 세부 정보 보기 및 디렉터리 등록 취소 버튼이 활성화됩니다. 등록 세부 정보 보기 버튼을 클릭하면 등록에 대한 세부 정보 페이지가 열립니다.
5. 디렉터리 등록 세부 정보 창에 다음 정보가 표시됩니다.

   - 디렉터리 도메인 사이트 이름
   - 디렉터리 ID - 디렉터리의 고유 ID입니다. 링크를 선택하면 AWS Directory Service 콘솔로 이동합니다.
   - 디렉터리 유형
   - 상태 - 디렉터리 상태
   - 디렉터리 등록 ARN - 디렉터리 등록의 Amazon 리소스 이름
   - 추가 상태 정보
6. 커넥터 및 서비스 보안 주체 이름(SPN) 창에서 커넥터의 SPN을 관리할 수 있습니다. 자세한 내용은 커넥터 세부 정보 보기를 참조하세요.
7. 태그 - 옵션 창에서 AD 리소스의 메타데이터를 적용하고 제거할 수 있습니다. 태그는 키-값 문자열 페어로, 키는 리소스별로 고유해야 하고 값은 선택사항입니다. 이 창에는 테이블에 있는 리소스의 기존 태그가 모두 표시됩니다. 다음 작업이 지원됩니다.

   - 태그 관리를 선택하여 태그 관리 페이지를 엽니다.
   - 새 태그를 추가하려면 태그 생성을 선택합니다. 키 필드를 채우고 선택적으로 값 필드를 입력합니다. 태그를 적용하려면 변경 사항 저장을 선택합니다.
   - 태그 옆에 있는 제거 버튼을 선택하여 삭제 대상으로 표시하고 변경 사항 저장을 선택하여 확인합니다.

API

API를 사용하여 디렉터리 등록을 관리하려면

생성: Active Directory용 AWS Private CA 커넥터 API의 CreateDirectoryRegistration 작업입니다.

검색: Active Directory용 AWS Private CA 커넥터 API의 GetDirectoryRegistration 작업.

목록: Active Directory용 AWS Private CA 커넥터 API의 ListDirectoryRegistrations 작업입니다.

삭제: Active Directory용 AWS Private CA 커넥터 API의 DeleteDirectoryRegistration 작업.

CLI

CLI를 사용하여 디렉터리 등록을 관리하려면

생성:의 Active Directory용 AWS Private CA 커넥터 섹션에서 create-directory-registration 명령을 사용합니다 AWS CLI.

검색:의 Active Directory용 AWS Private CA 커넥터 섹션에서 get-directory-registration 명령입니다 AWS CLI.

목록:의 Active Directory용 AWS Private CA 커넥터 섹션에 있는 list-directory-registrations 명령입니다 AWS CLI.

삭제:의 Active Directory용 AWS Private CA 커넥터 섹션에서 delete-directory-registration 명령입니다 AWS CLI.

AD 템플릿용 커넥터 액세스 제어 항목 관리

액세스 제어 항목은 특정 AD용 커넥터 템플릿에 대한 인증서를 등록할 수 있거나 등록할 수 없는 Active Directory 그룹을 제어합니다. AD용 커넥터에서 그룹 및 권한을 생성하거나 관리할 수 있는 경우 Active Directory에서 그룹 객체의 보안 식별자(SID)를 제공해야 합니다. 다음 PowerShell 명령을 사용하여 SID를 가져올 수 있습니다. SIDs 대한 자세한 내용은 Microsoft Directory Domain Services 설명서의 보안 식별자 작동 방식을 참조하세요.

```
$ Get-ADGroup -Identity "my_active_directory_group_name"
```

다음 절차에서는 AD용 커넥터 템플릿 액세스 그룹 항목을 생성하고 관리하는 방법을 보여줍니다.

Console

콘솔을 사용하여 템플릿 그룹 권한을 관리하려면

템플릿의 세부 정보 페이지에서 기존 템플릿에 대한 그룹 및 권한을 관리할 수 있습니다. 자세한 내용은 커넥터 템플릿 세부 정보 보기를 참조하세요.

특정 템플릿의 인증서를 등록할 수 있는 그룹과 등록할 수 없는 그룹에 대한 권한을 설정합니다. 그룹의 보안 식별자(SID)를 제공합니다. 그런 다음 그룹에 대한 등록 및 자동 등록 권한을 설정합니다. 자동 등록의 경우 등록과 자동 등록을 모두 “허용”으로 설정해야 합니다.

API

API를 사용하여 템플릿 그룹 권한을 관리하려면

생성: Active Directory용 AWS Private CA 커넥터 API의 CreateTemplateGroupAccessControlEntry 작업입니다.

업데이트: Active Directory용 AWS Private CA 커넥터 API의 UpdateTemplateGroupAccessControlEntry 작업.

검색: Active Directory용 AWS Private CA 커넥터 API의 GetTemplateGroupAccessControlEntry 작업입니다.

목록: Active Directory용 AWS Private CA 커넥터 API의 ListTemplateGroupAccessControlEntries 작업입니다.

삭제: Active Directory용 AWS Private CA 커넥터 API의 DeleteTemplateGroupAccessControlEntry 작업입니다.

CLI

CLI를 사용하여 템플릿 그룹 권한을 관리하려면

생성: AWS CLI의 Active Directory용 AWS Private CA 커넥터 섹션에 있는 create-template-group-access-control-entry 명령.

업데이트: AWS CLI의 Active Directory용 AWS Private CA 커넥터 섹션에 있는 update-template-group-access-control-entry 명령.

검색:의 Active Directory용 AWS Private CA 커넥터 섹션에서 get-template-group-access-control-entry 명령을 검색합니다 AWS CLI.

목록: AWS CLI의 Active Directory용 AWS Private CA 커넥터 섹션에 있는 list-template-group-access-control-entries 명령.

삭제: AWS CLI의 Active Directory용 AWS Private CA 커넥터 섹션에 있는 delete-template-group-access-control-entries 명령.

서비스 보안 주체 이름 구성

커넥터의 서비스 보안 주체 이름을 구성하는 방법을 알아봅니다.

Console

콘솔을 사용하여 서비스 보안 주체 이름 관리를 관리하려면

기존 AD 커넥터의 서비스 보안 주체 이름(SPN)은 커넥터의 세부 정보 페이지에서 관리할 수 있습니다. 자세한 내용은 디렉터리 등록 관리의 커넥터 세부 정보 보기를 참조하세요.

API

API를 사용하여 서비스 보안 주체 이름을 관리하려면

생성: Active Directory용 AWS Private CA 커넥터 API의 CreateServicePrincipalName 작업입니다.

검색: Active Directory용 AWS Private CA 커넥터 API의 GetServicePrincipalName 작업입니다.

목록: Active Directory용 AWS Private CA 커넥터 API의 ListServicePrincipalNames 작업입니다.

삭제: Active Directory용 AWS Private CA 커넥터 API의 DeleteServicePrincipalName 작업입니다.

CLI

CLI를 사용하여 서비스 보안 주체 이름을 관리하려면

생성:의 Active Directory용 AWS Private CA 커넥터 섹션에서 create-service-principal-name 명령을 생성합니다 AWS CLI.

검색:의 Active Directory용 AWS Private CA 커넥터 섹션에서 get-service-principal-name 명령을 검색합니다 AWS CLI.

목록:의 Active Directory용 AWS Private CA 커넥터 섹션에 있는 list-service-principal-names 명령입니다 AWS CLI.

삭제:의 Active Directory용 AWS Private CA 커넥터 섹션에서 delete-service-principal-name 명령입니다 AWS CLI.

AD 리소스용 태그 지정 커넥터

커넥터, 템플릿 및 디렉토리 등록에 태그를 적용할 수 있습니다. 태그를 지정하면 리소스에 메타데이터가 추가되어 구성 및 관리에 도움이 될 수 있습니다.

Console

콘솔을 사용하여 리소스 태그 지정을 관리하려면

기존 리소스의 태깅은 해당 리소스의 세부정보 페이지에서 관리됩니다. 자세한 내용은 다음 절차를 참조하세요.

- 커넥터 템플릿 세부 정보 보기
- 디렉터리 등록 관리

API

API를 사용하여 리소스 태그 지정을 관리하려면

태그: Active Directory용 AWS Private CA 커넥터 API의 TagResource 작업입니다.

Active Directory용 AWS Private CA 커넥터 API의 태그 나열: ListTagsForResource 작업.

Untag: Active Directory용 AWS Private CA 커넥터 API의 UntagResource 작업입니다.

중요 - 태그를 사용하여 기밀 데이터가 포함된 객체를 라벨링하는 것이 허용됩니다. 하지만 태그 자체에는 개인 식별 정보((PII), 민감한 정보 또는 기밀 정보가 포함되어서는 안 됩니다.

CLI

CLI를 사용하여 리소스 태그 지정을 관리하려면

의 Active Directory용 AWS Private CA 커넥터 섹션에서 태그: tag-resource 명령 AWS CLI.

의 Active Directory용 AWS Private CA 커넥터 섹션에서 태그 나열: list-tags-for-resource 명령 AWS CLI.

Untag:의 Active Directory용 AWS Private CA 커넥터 섹션에서 untag-resource 명령입니다 AWS CLI.

Amazon EventBridge를 사용하여 AD용 커넥터를 이벤트 기반 애플리케이션에 통합

AD용 커넥터에서 발생하는 이벤트를 사용하여 애플리케이션 구성 요소 간에 통신하고 다운스트림 프로세스를 시작하는 이벤트 기반 애플리케이션(EDAs)에 AD용 커넥터를 통합할 수 있습니다.

예를 들어 계정에서 다음 AD용 커넥터 이벤트가 발생할 때 다른 AWS 서비스 또는 사용자 지정 구성 요소를 호출할 수 있습니다.

- 인증서가 생성되거나 생성이 실패할 때 생성됩니다.
- 인증서가 등록되었거나 등록에 실패합니다.

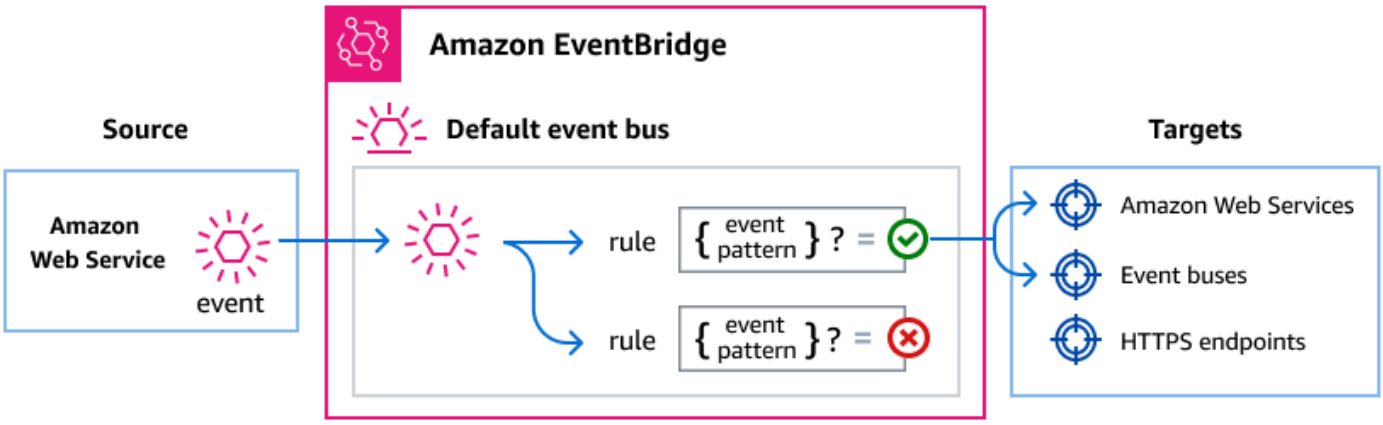
이렇게 하려면 Amazon EventBridge를 사용하여 AD용 커넥터에서 다른 소프트웨어 구성 요소로 이벤트를 라우팅합니다. Amazon EventBridge는 이벤트를 사용하여 애플리케이션 구성 요소를 함께 연결하는 서버리스 서비스이므로 추가 코드 및 작업 없이 AD용 커넥터와 같은 AWS 서비스를 이벤트 기반 아키텍처에 더 쉽게 통합할 수 있습니다.

EventBridge가 AD 이벤트용 커넥터를 라우팅하는 방법

다음은 AD 이벤트용 커넥터에서 EventBridge가 작동하는 방법입니다.

많은 AWS 서비스와 마찬가지로 AD용 커넥터는 이벤트를 생성하고 EventBridge 기본 이벤트 버스로 전송합니다. 이벤트 버스는 이벤트를 수신하여 지정한 목적지 또는 대상으로 라우팅하는 라우터입니다. 대상에는 다른 AWS 서비스, 사용자 지정 애플리케이션 및 SaaS 파트너 애플리케이션이 포함될 수 있습니다.

EventBridge는 사용자가 이벤트 버스에서 생성한 규칙에 따라 이벤트를 라우팅합니다. 각 규칙마다 필터 또는 이벤트 패턴을 지정하여 원하는 이벤트만 선택할 수 있습니다. 이벤트가 이벤트 버스로 전송될 때마다 EventBridge는 이벤트를 각 규칙과 비교합니다. 이벤트가 규칙과 일치하면 EventBridge는 이벤트를 지정된 대상으로 라우팅합니다.



AD 이벤트용 커넥터

EventBridge로 전송된 AD 이벤트용 커넥터 목록은 EventBridge 이벤트 참조의 AD용 커넥터 주제를 참조하세요.

이벤트 구조

AWS 서비스의 모든 이벤트에는 두 가지 유형의 데이터가 포함됩니다.

- 이벤트의 소스인 AWS 서비스, 이벤트가 생성된 시간, 이벤트가 발생한 계정 및 리전 등 이벤트에 대한 메타데이터를 포함하는 공통 필드 집합. 이러한 일반 필드의 정의는 Amazon EventBridge 이벤트 참조의 이벤트 구조를 참조하세요.
- 해당 특정 서비스 이벤트와 관련된 데이터를 포함한 `detail` 필드.

AD 이벤트용 커넥터와 일치하는 이벤트 패턴 생성

이벤트 패턴은 이벤트가 포함해야 하는 데이터를 지정하는 필터입니다.

각 이벤트 패턴은 다음을 포함하는 JSON 객체입니다.

- 이벤트를 전송하는 서비스를 식별하는 `source` 속성입니다. AD 이벤트용 커넥터의 경우 소스는 입니다`aws.pca-connector-ad`.
- (선택 사항): 일치해야 하는 이벤트 이름의 배열을 포함하는 `detail-type` 속성.
- (선택 사항): 일치시킬 다른 이벤트 데이터를 포함하는 `detail` 속성입니다.

예를 들어 다음 이벤트 패턴은 AD용 커넥터에서 모든 인증서 정책 등록 성공 이벤트를 선택합니다.

```
{
  "source": ["aws.pca-connector-ad"],
  "detail-type": ["Certificate Policy Enrollment Succeeded"]
}
```

이벤트 작성에 대한 자세한 내용은 EventBridge 사용 설명서의 이벤트 패턴을 참조하세요.

EventBridge에서 이벤트 수신

AD 인증서용 커넥터를 규칙의 대상으로 지정할 수 있습니다. 이를 통해 AD용 커넥터는 다른 AWS 서비스, 사용자 지정 애플리케이션 및 SaaS 파트너를 비롯한 다양한 소스에서 이벤트를 수신할 수 있습니다. 자세한 내용은 EventBridge 사용 설명서의 이벤트에 반응하는 규칙 생성을 참조하세요.

대상으로 지정할 수 있는 AWS 서비스의 전체 목록은 EventBridge 이벤트 참조의 대상 유형을 참조하세요.

Active Directory용 AWS Private CA 커넥터 관련 문제 해결

여기의 정보를 사용하여 AD용 AWS Private Certificate Authority 커넥터를 진단하고 수정할 수 있습니다.

주제

- AD용 커넥터 오류 코드 문제 해결
- AD 커넥터 생성 실패를 위한 커넥터 문제 해결
- AD SPN용 커넥터 생성 실패 문제 해결
- AD용 커넥터 템플릿 업데이트 문제 해결

AD용 커넥터 오류 코드 문제 해결

AD용 커넥터는 여러 가지 이유로 오류 메시지를 보냅니다. 각 오류에 대한 자세한 내용과 오류 해결에 대한 권장 사항은 다음 표를 참조하세요. Amazon EventBridge 스케줄러 이벤트(이벤트 소스: `aws.pca-connector-ad`)를 구독하거나 Windows에서 수동 등록을 사용하여 이러한 오류를 수신할 수 있습니다.

오류 코드	근본 원인:	문제 해결
0x8FFFA000	Kerberos 인증이 실패했습니다.	디렉터리에 연결할 수 있고 클라이언트가 사용자 또는 컴퓨터인지 확인합니다. 자동 등록을 사용하는 경우 리소스 서비스 보안 주체를 수정합니다 AWS . Active Directory UI를 사용하여 인증서를 받는 경우에는 `gpupdate /force` 을 실행합니다.
0x8FFFA001	SOAP 메시지에는 작업 헤더가 포함되어야 합니다.	작업 헤더 추가.
0x8FFFA002	커넥터는 연결된 프라이빗 CA에 액세스할 수 없습니다.	프라이빗 CA와 AD 서비스용 커넥터 간에 공유할 AWS Resource Access

오류 코드	근본 원인:	문제 해결
		Manager(RAM)를 생성하여 프라이빗 CA를 커넥터와 공유합니다.
0x8FFFA003	이 커넥터의 프라이빗 CA가 활성 상태가 아닙니다.	프라이빗 CA를 활성 상태로 이동합니다. 프라이빗 CA가 보류 중인 인증서 상태인 경우 CA 인증서를 설치합니다.
0x8FFFA004	이 커넥터의 프라이빗 CA가 없습니다.	인증 기관이 삭제됨 상태인 경우 인증 기관을 활성 상태로 이동합니다. 프라이빗 CA가 영구적으로 삭제된 경우 다른 CA로 새 커넥터를 생성합니다.
0x8FFFA005	템플릿에서 인증서 보안 주체 또는 보안 주체 대체 이름의 `directoryGuid` 속성을 지정했지만 요청자의 AD 객체에서 속성을 찾을 수 없습니다.	Active Directory가 디렉터리에 대한 `directoryGuid` 를 생성하지 않았습니다. Active Directory에서 문제를 해결합니다.
0x8FFFA006	템플릿에서 인증서 보안 주체 또는 보안 주체 대체 이름의 `dnsHostName` 속성을 지정했지만 요청자의 AD 객체에서 속성을 찾을 수 없습니다.	AD 객체에 `dnsHostName` 속성을 추가합니다.
0x8FFFA007	템플릿에서 인증서 제목 또는 보안 주체 대체 이름에 포함되도록 이메일 속성을 지정했지만 요청자의 AD 객체에서 속성을 찾을 수 없습니다.	AD 객체에 이메일 속성을 추가합니다

오류 코드	근본 원인:	문제 해결
0x8FFFA008	SOAP 메시지에는 http://schemas.microsoft.com/windows/pki/2009/01/enrollmentpolicy/IPolicy/GetPolicies 또는 http://schemas.microsoft.com/windows/pki/2009/01/enrollment/RST/wstep 중 하나의 작업 헤더가 있어야 합니다.	지정된 값 중 하나를 사용하도록 작업 헤더를 업데이트합니다.
0x8FFFA009	바이너리 보안 토큰이 http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd#base64binary 에서 인코딩되어 있어야 합니다.	바이너리 보안 토큰 유형을 업데이트합니다.
0x8FFFA00A	바이너리 보안 토큰이 잘못되었습니다.	CSR이 올바르게 생성되었는지 확인합니다.
0x8FFFA00B	바이너리 보안 토큰의 값 유형은 http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd#PKCS7 또는 http://schemas.microsoft.com/windows/pki/2009/01/enrollment#PKCS10 중 하나여야 합니다.	바이너리 보안 토큰 값 유형을 유효한 값으로 업데이트합니다.

오류 코드	근본 원인:	문제 해결
0x8FFFA00C	바이너리 보안 토큰에 잘못된 CMS가 포함되어 있습니다.	Base64는 유효하지만 암호화 메시지 구문(CMS)은 유효하지 않습니다. CMS 구문을 검토합니다.
0x8FFFA00D	바이너리 보안 토큰에 잘못된 CSR이 포함되어 있습니다.	CSR이 올바르게 생성되었는지 확인합니다.
0x8FFFA00E	프라이빗 CA가 특정 템플릿을 사용하여 인증서를 발급하지 못했습니다.	에서 검증 예외를 검토합니다 AWS Private CA. Amazon EventBridge 또는에서 검증 예외를 볼 수 있습니다 AWS CloudTrail.
0x8FFFA00F	SOAP 메시지에는 `http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue` 의 요청 유형은 있어야 합니다.	요청 유형을 `http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue` 로 설정합니다.
0x8FFFA010	SOAP 메시지에는 커넥터의 `CertificateEnrollmentPolicyServerEndpoint` 필드 또는 XCEP 응답의 URI 필드의 헤더가 있어야 합니다.	요청 보안 토큰의 헤더를 XCEP 응답의 `CertificateEnrollmentPolicyServerEndpoint` 필드 또는 URI 필드로 설정합니다.
0x8FFFA011	SOAP 메시지에는 작업 헤더가 하나만 있어야 합니다.	요청 보안 토큰의 SOAP 메시지 헤더를 검토하고 헤더를 올바르게 설정합니다.
0x8FFFA012	SOAP 메시지에는 하나의 `messageId` 헤더가 있어야 합니다.	요청 보안 토큰의 SOAP 메시지 헤더를 검토하고 헤더를 올바르게 설정합니다.

오류 코드	근본 원인:	문제 해결
0x8FFFA013	SOAP 메시지의 헤더에는 하나만 포함되어야 합니다.	요청 보안 토큰의 SOAP 메시지 헤더를 검토하고 헤더를 올바르게 설정합니다.
0x8FFFA014	요청자는 요청된 템플릿에 액세스할 수 없습니다.	액세스 제어 항목을 생성하여 요청자 그룹이 요청된 템플릿을 사용하여 등록할 수 있도록 허용합니다.
0x8FFFA015	`CertificateTemplateInformation` 또는 `CertificateTemplateName` 확장자가 BinarySecurityToken에 있어야 합니다.	CSR에 보안 확장을 추가합니다.
0x8FFFA016	해당 커넥터에 대해 요청한 템플릿을 찾을 수 없습니다.	템플릿은 각 커넥터의 하위 리소스입니다. `createTemplate` 를 사용하여 커넥터용 템플릿을 생성합니다.
0x8FFFA017	요청 제한 때문에 요청이 거부되었습니다.	요청 속도를 늦춥니다.
0x8FFFA018	SOAP 메시지에는 `to` 헤더가 포함되어야 합니다.	SOAP 메시지의 헤더를 검토합니다.
0x8FFFA019	헤더가 인식되지 않아 SOAP 메시지를 처리할 수 없습니다.	SOAP 메시지의 헤더를 검토합니다.
0x8FFFA01A	템플릿에서 인증서 보안 주체 또는 보안 주체 대체 이름에 포함되도록 UPN 속성을 지정했지만 요청자의 AD 객체에서 속성을 찾을 수 없습니다.	Active Directory 객체에 UPN을 추가합니다.

AD 커넥터 생성 실패를 위한 커넥터 문제 해결

AD 커넥터 생성용 커넥터는 다양한 이유로 실패할 수 있습니다. 커넥터 생성에 실패하면 API 응답에서 실패 이유를 받게 됩니다. 콘솔을 사용하는 경우 커넥터 세부 정보 페이지의 커넥터 세부 정보 컨테이너 내 추가 상태 세부 정보 필드에 실패 이유가 표시됩니다. 다음 표에서는 실패 이유와 권장 해결 단계를 설명합니다.

실패 상태	설명	문제 해결
CA_CERTIFICATE_REGISTRATION_FAILED	AD용 커넥터가 CA 인증서를 디렉터리로 가져올 수 없습니다.	사전 조건 페이지를 검토하고 서비스 계정에 올바른 권한이 있는지 확인합니다. 서비스 계정에 올바른 권한을 위임한 후 실패한 커넥터를 삭제하고 새 커넥터를 생성합니다. 권한 위임에 대한 자세한 내용은 AWS Directory Service 관리 안내서의 서비스 계정에 권한 위임을 참조하세요.
DIRECTORY_ACCESS_DENIED	AD용 커넥터가 디렉터리에 액세스할 수 없습니다.	디렉터리에 대한 AD용 커넥터 액세스 권한을 부여해야 합니다. 4단계: IAM 정책 생성 섹션을 검토하여 AWS 계정과 연결된 IAM 정책을 통해 디렉터리에 액세스하고 설명할 수 있는지 확인합니다. AWS 역할에 올바른 권한을 부여한 후 실패한 커넥터를 삭제하고 새 커넥터를 생성합니다. AD 커넥터와 함께 AWS Directory Service AD용 커넥터를 사용하는 경우 AD 커넥터 서비스 계정의 암호가 만료되지 않고 유효한지 확인합니

실패 상태	설명	문제 해결
		다. AD Connector 서비스 계정에 대한 자세한 내용은 AD Connector 관리 안내서의 AD Connector 시작하기를 참조하세요.
INTERNAL_FAILURE	AD용 커넥터에 내부 장애가 발생했습니다.	나중에 다시 시도해 주세요. 실패한 커넥터를 삭제하고 새 커넥터를 생성합니다.
INSUFFICIENT_FREE_ADDRESSES	VPC 서브넷에는 사용 가능한 프라이빗 IP 주소가 하나 이상 있어야 합니다.	서브넷에 사용 가능한 프라이빗 IP 주소가 있는지 확인합니다. 실패한 커넥터를 삭제하고 새 커넥터를 생성합니다.
INVALID_SUBNET_IP_PROTOCOL	디렉터리와 연결된 서브넷이 지정된 IP 주소 유형을 지원하지 않기 때문에 AD용 커넥터가 VPC에서 엔드포인트를 생성할 수 없습니다.	디렉터리를 호스팅하는 VPC 및 서브넷이 선택한 IP 주소 유형을 지원하는지 확인합니다. 자세한 내용은 IP 주소 유형을 참조하세요. 실패한 커넥터를 삭제하고 지원되는 IP 주소 유형으로 새 커넥터를 생성합니다.

실패 상태	설명	문제 해결
PRIVATECA_ACCESS_DENIED	AD용 커넥터가 프라이빗 CA에 액세스할 수 없습니다.	사전 조건 페이지를 검토하고 커넥터를 생성할 권한이 있는지 확인합니다. 자세한 내용은 4단계: IAM 정책 생성을 참조하세요. AWS CLI 또는 API를 통해 커넥터를 생성하는 경우 사전 조건 페이지를 검토하고를 사용하여 AD용 커넥터와 프라이빗 CA를 공유했는지 확인합니다 AWS Resource Access Manager. IAM 권한 및 AWS RAM 리소스 공유를 확인하고 수정한 후 실패한 커넥터를 삭제하고 새 커넥터를 생성합니다.
PRIVATECA_RESOURCE_NOT_FOUND	AD용 커넥터가 지정된 프라이빗 CA를 찾을 수 없습니다.	올바른 프라이빗 CA Amazon 리소스 이름(ARN)을 지정한 다음 실패한 커넥터를 삭제하고 의도한 프라이빗 CA ARN을 사용하여 새 커넥터를 생성해야 합니다.
SECURITY_GROUP_NOT_IN_VPC	디렉터리를 호스팅하는 VPC에 보안 그룹이 없습니다.	디렉터리를 호스팅하는 VPC에 있는 보안 그룹을 사용합니다. 자세한 내용은 7단계: 보안 그룹 구성 단원을 참조하십시오. 실패한 커넥터를 삭제하고 VPC에 있는 보안 그룹을 사용하여 새 커넥터를 생성합니다.

실패 상태	설명	문제 해결
VPC_ACCESS_DENIED	AD용 커넥터는 디렉터리를 호스팅하는 Amazon VPC에 액세스할 수 없습니다.	IAM 권한을 확인합니다. 실패한 커넥터를 삭제하고 새 커넥터를 생성합니다. 액세스 권한이 포함된 IAM 정책의 예는 섹션을 참조하세요. 4단계: IAM 정책 생성
VPC_ENDPOINT_LIMIT_EXCEEDED	AD용 커넥터는 Amazon VPC에서 엔드포인트를 생성할 수 없습니다. 계정에 대해 생성할 수 있는 VPC 엔드포인트 한도에 도달했습니다.	Amazon VPC 엔드포인트를 삭제하거나 한도 증가를 요청합니다. 두 단계 중 하나를 완료했으면 실패한 커넥터를 삭제하고 새 커넥터를 생성합니다. 할당량에 대한 자세한 내용은 Amazon Virtual Private Cloud Service 할당량을 참조하세요.
VPC_RESOURCE_NOT_FOUND	AD용 커넥터가 지정된 VPC를 찾을 수 없습니다.	올바른 VPC를 지정했고 VPC가 존재하는지 확인합니다. 그런 다음 실패한 커넥터를 삭제하고 올바른 VPC ID를 사용하여 새 커넥터를 생성합니다.

AD SPN용 커넥터 생성 실패 문제 해결

서비스 보안 주체 이름(SPN) 생성은 다양한 이유로 실패할 수 있습니다. SPN 생성에 실패하면 API 응답에 실패 이유가 표시됩니다. 콘솔을 사용하는 경우 서비스 보안 주체 이름(SPN) 컨테이너의 추가 상태 세부 정보 필드 아래에 있는 커넥터 세부 정보 페이지에 실패 이유가 표시됩니다. 다음 표에서는 실패 이유와 권장 해결 단계를 설명합니다.

실패 상태	설명	문제 해결
DIRECTORY_ACCESS_DENIED	AD용 커넥터는 디렉터리에 액세스할 수 없습니다.	

실패 상태	설명	문제 해결
		AD용 커넥터에 디렉터리에 대한 액세스 권한을 부여합니다. 디렉터리 액세스 권한을 부여하는 권한이 포함된 IAM 정책의 예는 섹션을 참조하세요4단계: IAM 정책 생성.
DIRECTORY_NOT_REACHABLE	AD용 커넥터는 디렉터리에 액세스할 수 없습니다.	와 디렉터리 간의 네트워크를 확인하고 SPN을 다시 생성 AWS 해 보십시오.
DIRECTORY_RESOURCE_NOT_FOUND	AD용 커넥터가 지정된 디렉터리를 찾을 수 없습니다.	올바른 디렉터리 ID를 지정한 다음 실패한 커넥터를 삭제하고 원하는 디렉터리 ID를 사용하여 새 커넥터를 생성해야 합니다.
INTERNAL_FAILURE	AD용 커넥터에 내부 장애가 발생했습니다.	나중에 다시 시도해 주세요.
SPN_EXISTS_ON_DIFFERENT_AD_OBJECT	서비스 보안 주체 이름(SPN)이 다른 Active Directory 객체에 있습니다.	Active Directory 객체에서 SPN을 삭제하고 SPN을 다시 생성해 봅니다.
SPN_LIMIT_EXCEEDED	AD용 커넥터는 디렉터리당 SPN 한도에 도달했기 때문에 SPNs 생성할 수 없습니다. 디렉터리당 최대 SPNs 수는 10개입니다.	계정에서 하나 이상의 SPNs 삭제하고 SPN을 다시 생성해 보세요.

AD용 커넥터 템플릿 업데이트 문제 해결

템플릿 또는 그룹 액세스 제어 항목을 변경했지만 변경 사항이 표시되지 않는 경우, 이는 클라이언트가 정책 캐시를 새로 고칠 때 8시간마다 정책 캐싱이 정책에 템플릿을 AWS Private CA 적용하기 때문일

수 있습니다. 클라이언트는 캐시를 새로 고치면 커넥터에서 사용 가능한 템플릿을 쿼리합니다. 자동 등록 새로 고침의 경우 클라이언트는 다음 조건 중 하나 또는 둘 다와 일치하는 인증서를 발급합니다.

- 인증서가 갱신 기간 내에 있습니다.
- 인증서가 클라이언트 디바이스에 없습니다.

수동 새로 고침의 경우 클라이언트가 커넥터를 쿼리하므로 템플릿을 발급으로 설정해야 합니다.

디버깅하는 경우 정책 캐시를 수동으로 지워 템플릿 변경 사항을 즉시 볼 수 있습니다. 이렇게 하려면 클라이언트에서 다음 Powershell 명령을 실행합니다.

```
certutil -f -user -policyserver * -policycache delete
```

AWS Private CA SCEP용 커넥터

SCEP(Simple Certificate Enrollment Protocol)용 커넥터는 SCEP 지원 모바일 디바이스 및 네트워킹 장비에 AWS Private Certificate Authority 연결됩니다. SCEP용 커넥터를 사용하면 AWS Private CA 를 사용하여 인증서를 발급하고 SCEP 디바이스를 등록할 수 있습니다. SCEP용 커넥터는 널리 사용되는 모바일 디바이스 관리(MDM) 시스템과 함께 사용할 수 있으며 SCEP를 지원하는 클라이언트 또는 엔드포인트와 함께 작동하도록 설계되었습니다.

주제

- 특성
- SCEP용 커넥터 시작하기
- 관련 서비스
- SCEP용 액세스 커넥터
- 가격 책정
- SCEP 개념용 커넥터
- SCEP 고려 사항 및 제한 사항에 대한 커넥터 이해
- SCEP용 커넥터 설정
- SCEP용 커넥터 시작하기
- SCEP용 커넥터에 대한 MDM 시스템 구성
- SCEP용 커넥터의 보안
- SCEP용 모니터 커넥터
- SCEP 문제에 대한 AWS Private Certificate Authority 커넥터 문제 해결

특성

SCEP 프로토콜 지원 - SCEP는 인증 기관(CA)에서 디지털 자격 증명 인증서를 가져와 모바일 디바이스 및 네트워킹 기어에 배포하기 위해 널리 채택된 프로토콜입니다. SCEP용 커넥터를 사용하면 SCEP를 사용하여 엔드포인트를 등록할 수 있습니다.

모바일 디바이스 등록 - Microsoft Intune 및 Jamf Pro를 비롯한 널리 사용되는 MDM 시스템에서 SCEP용 커넥터를 사용할 수 있습니다.

대규모 인증서 발급 - 커넥터의 SCEP 엔드포인트를 통해 인증서를 요청하도록 SCEP 지원 디바이스를 구성한 후 클라이언트가 자동으로 인증서를 요청할 수 있습니다 AWS Private CA.

SCEP용 커넥터 시작하기

시작하려면 커넥터를 생성하고 커넥터에 사용할 프라이빗 CA를 지정하는 데 도움이 되는 SCEP용 커넥터 관리 콘솔에서 안내 마법사를 시작합니다. 이 단계를 완료한 후 SCEP용 커넥터는 MDM 시스템 또는 네트워킹 장비에 입력할 수 있는 엔드포인트 및 기타 구성 파라미터를 제공합니다. MDM 시스템 또는 네트워킹 장비를 구성한 후 클라이언트는 자동으로 인증서를 요청합니다 AWS Private CA. SCEP용 커넥터를 시작하는 방법에 대한 자세한 내용은 섹션을 참조하세요SCEP용 커넥터 시작하기.

관련 서비스

SCEP용 커넥터는 다음 AWS 서비스와 관련이 있습니다.

- AWS Private Certificate Authority - 자체 프라이빗 CA 운영에 대한 선결제 투자 및 지속적인 유지 관리 비용 없이 가용성이 높은 프라이빗 CA 서비스를 AWS Private CA 제공합니다.
- AWS Private CA Active Directory용 커넥터 - AD용 커넥터는 Active Directory(AD)를에 연결합니다 AWS Private CA. 커넥터는에서 AD에서 관리하는 사용자 및 시스템으로의 인증서 교환 AWS Private CA 을 브로커합니다.

SCEP용 액세스 커넥터

다음 인터페이스 중 하나를 사용하여 SCEP 커넥터용 커넥터를 생성, 액세스 및 관리할 수 있습니다.

- AWS Management Console - SCEP용 커넥터에 액세스하는 데 사용할 수 있는 웹 인터페이스를 제공합니다. SCEP 관리 콘솔용 커넥터를 참조하세요.
- AWS Command Line Interface - SCEP용 커넥터 등 다양한 AWS 서비스에 대한 명령을 제공합니다. AWS CLI 는 Windows, macOS 및 Linux에서 지원됩니다. 자세한 내용은 AWS Command Line Interface 단원을 참조하십시오.
- AWS SDKs - 언어별 APIs 제공하고 서명 계산, 요청 재시도 처리, 오류 처리와 같은 많은 연결 세부 정보를 처리합니다. 자세한 내용은 AWS Command Line Interface 단원을 참조하십시오.
- SCEP API용 커넥터 - HTTPS 요청을 사용하여 호출하는 하위 수준 API 작업을 제공합니다. SCEP용 커넥터 API를 사용하는 것이 서비스에 액세스하는 가장 직접적인 방법입니다. 그러나 SCEP용 커넥터 API를 사용하려면 애플리케이션에서 요청에 서명할 해시 생성, 오류 처리와 같은 하위 수준의 세부 정보를 처리해야 합니다. 자세한 내용은 SCEP용 커넥터 API 참조를 참조하세요.

가격 책정

SCEP용 커넥터는 추가 비용 AWS Private CA 없이의 기능으로 제공됩니다. 커넥터를 생성하고 업데이트하는 데 사용되는 AWS Private Certificate Authority 작업 및 인증서에 대해서만 비용을 지불합니다.

최신 AWS Private CA 요금 정보는 AWS Private Certificate Authority 요금을 참조하세요. AWS 요금 계산기를 사용하여 비용을 추정할 수도 있습니다.

SCEP 개념용 커넥터

SCEP용 커넥터는의 추가 기능입니다 AWS Private Certificate Authority.

다음은 SCEP용 커넥터의 주요 개념입니다.

인증서 서명 요청(CSR)

디지털 인증서 발급을 위해 CA에 제공되는 필수 정보입니다. 이 정보에는 퍼블릭 키와 자격 증명이 포함됩니다.

챌린지 암호

SCEP 프로토콜은 CA에서 인증서를 발급하기 전에 챌린지 암호를 사용하여 요청을 인증합니다. SCEP용 커넥터는 커넥터 유형에 따라 SCEP 챌린지 암호를 처리합니다. 자세한 내용은 SCEP용 커넥터에 대한 MDM 시스템 구성 단원을 참조하십시오.

인증서 해지

인증서 취소는 만료 날짜 이전에 발급된 인증서를 취소하는 프로세스입니다. API AWS Command Line Interface, AWS SDK 또는에서 RevokeCertificate를 호출하여 커넥터와 연결된 프라이빗 CA 인증서를 취소할 수 있습니다 AWS CloudFormation.

SCEP용 커넥터

SCEP용 커넥터는 SCEP 지원 디바이스 AWS Private CA 에 연결됩니다.

모바일 디바이스 관리

모바일 디바이스 관리(MDM)를 사용하면 IT 관리자가 스마트폰, 태블릿 및 기타 엔드포인트 또는 디바이스에 대한 정책을 제어, 보호 및 적용할 수 있습니다. 많은 MDM 시스템은 SCEP 기반 인증서 등록을 위한 내장 통합을 제공합니다.

SCEP

SCEP는 인증서를 자동으로 배포하는 표준화된 프로토콜(RFC 8894)입니다. 프로토콜은 디바이스가 CA에서 인증서를 요청할 수 있는 엔드포인트를 제공합니다. SCEP는 챌린지 암호를 사용하여 디바이스에 대한 인증서 발급을 승인합니다. SCEP는 일반적으로 모바일 디바이스 관리(MDM) 시스템 및 네트워킹 장비에 적용됩니다. MDM 솔루션을 사용하면 IT 관리자가 스마트폰, 태블릿 및 Apple 워크스테이션과 같은 기타 엔터티에 대한 정책을 제어, 보호 및 적용할 수 있습니다. 대부분의 MDM 솔루션은 Microsoft Intune, Apple MDM, Jamf Pro와 같은 SCEP를 지원합니다. 라우터, 로드 밸런서, Wi-Fi 허브, VPN 디바이스 및 방화벽과 같은 대부분의 네트워킹 장비가 자동 인증서 등록에 SCEP를 사용합니다.

SCEP 프로필

SCEP 프로파일에는 인증서 프로파일을 정의하는 데 사용되는 구성 파라미터가 포함되어 있습니다. 여기에는 인증서 유효 기간, 키 크기, SCEP 구성 이름, 챌린지 암호, 실패한 시도 횟수 및 재시도 간격, 인증서 발급과 관련된 기타 정보가 포함됩니다. MDM 시스템 및 인증서 관리 플랫폼은 일반적으로 인증을 위해 인증서를 요청할 SCEP 프로파일을 클라이언트로 전송합니다.

SCEP 고려 사항 및 제한 사항에 대한 커넥터 이해

SCEP용 커넥터 사용 시 다음 고려 사항 및 제한 사항에 유의하세요.

고려 사항

CA 작동 모드

범용 작동 모드를 사용하는 프라이빗 CAs에서만 SCEP용 커넥터를 사용할 수 있습니다. SCEP용 커넥터는 기본적으로 유효 기간이 1년인 인증서를 발급합니다. 수명이 짧은 인증서 모드를 사용하는 프라이빗 CA는 유효 기간이 7일 이상인 인증서 발급을 지원하지 않습니다. 작동 모드에 대한 자세한 내용은 섹션을 참조하세요AWS Private CA CA 모드 이해.

챌린지 암호

- 챌린지 암호를 매우 신중하게 배포하고 신뢰할 수 있는 개인 및 클라이언트와만 공유합니다. 단일 챌린지 암호를 사용하여 보안 위험이 있는 모든 주체 및 SANs과 함께 모든 인증서를 발급할 수 있습니다.
- 범용 커넥터를 사용하는 경우 챌린지 암호를 자주 수동으로 교체하는 것이 좋습니다.

RFC 8894 준수

SCEP용 커넥터는 HTTP 엔드포인트 대신 HTTPS 엔드포인트를 제공하여 RFC 8894 프로토콜과 다릅니다.

CSRs

- SCEP용 커넥터로 전송되는 인증서 서명 요청(CSR)에 확장 키 사용(EKU) 확장이 포함되지 않은 경우 EKU 값을 로 설정합니다`clientAuthentication`. 자세한 내용은 4.2.1.12을 참조하세요. RFC 5280의 확장 키 사용.
- CSRs에서 `ValidityPeriod` 및 `ValidityPeriodUnits` 사용자 지정 속성을 지원합니다. CSR에가 포함되지 않은 경우 유효 기간이 1년인 인증서를 발급`ValidityPeriod`합니다. MDM 시스템에서 이러한 속성을 설정하지 못할 수 있습니다. 하지만 설정할 수 있는 경우 이를 지원합니다. 이러한 속성에 대한 자세한 내용은 szENROLLMENT_NAME_VALUE_PAIR를 참조하세요.

엔드포인트 공유

커넥터의 엔드포인트를 신뢰할 수 있는 당사자에게만 배포합니다. 고유한 정규화된 도메인 이름과 경로를 찾을 수 있는 사람은 누구나 CA 인증서를 검색할 수 있으므로 엔드포인트를 보안 암호로 취급합니다.

제한 사항

SCEP용 커넥터에는 다음 제한 사항이 적용됩니다.

동적 챌린지 암호

범용 커넥터를 통해서만 정적 챌린지 암호를 생성할 수 있습니다. 범용 커넥터에서 동적 암호를 사용하려면 커넥터의 정적 암호를 사용하는 고유한 교체 메커니즘을 구축해야 합니다. SCEP for Microsoft Intune 커넥터 유형은 Microsoft Intune을 사용하여 관리하는 동적 암호에 대한 지원을 제공합니다.

HTTP

SCEP용 커넥터는 HTTPS만 지원하고 HTTP 호출에 대한 리디렉션을 생성합니다. 시스템이 HTTP에 의존하는 경우 SCEP용 커넥터가 제공하는 HTTP 리디렉션을 수용할 수 있는지 확인합니다.

공유 프라이빗 CAs

소유자인 프라이빗 CAs에서만 SCEP용 커넥터를 사용할 수 있습니다.

SCEP용 커넥터 설정

이 섹션의 절차는 SCEP용 커넥터를 시작하는 데 도움이 됩니다. 이미 AWS 계정을 생성했다고 가정합니다. 이 페이지의 단계를 완료한 후 SCEP용 커넥터 생성을 진행할 수 있습니다.

주제

- 1단계: AWS Identity and Access Management 정책 생성
- 2단계: 프라이빗 CA 생성
- 3단계:를 사용하여 리소스 공유 생성 AWS Resource Access Manager

1단계: AWS Identity and Access Management 정책 생성

SCEP용 커넥터를 생성하려면 커넥터에 필요한 리소스를 생성 및 관리하고 사용자를 대신하여 인증서를 발급할 수 있는 기능을 SCEP용 커넥터에 부여하는 IAM 정책을 생성해야 합니다. IAM에 대한 자세한 내용은 IAM 사용 설명서의 IAM이란 무엇입니까?를 참조하세요.

다음 예제는 SCEP용 커넥터에 사용할 수 있는 고객 관리형 정책입니다.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "pca-connector-scep:*",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "acm-pca:DescribeCertificateAuthority",
                "acm-pca:GetCertificate",
                "acm-pca:GetCertificateAuthorityCertificate",
                "acm-pca:ListCertificateAuthorities",
                "acm-pca:ListTags",
                "acm-pca:PutPolicy"
            ],
            "Resource": "*"
```

```
        },
        {
            "Effect": "Allow",
            "Action": "acm-pca:IssueCertificate",
            "Resource": "*",
            "Condition": {
                "ArnLike": {
                    "acm-pca:TemplateArn": "arn:aws:acm-pca:::template/
BlankEndEntityCertificate_APICSRPassthrough/V*"
                },
                "ForAnyValue:StringEquals": {
                    "aws:CalledVia": "pca-connector-scep.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "ram:CreateResourceShare",
                "ram:GetResourcePolicies",
                "ram:GetResourceShareAssociations",
                "ram:GetResourceShares",
                "ram:ListPrincipals",
                "ram:ListResources",
                "ram:ListResourceSharePermissions",
                "ram:ListResourceTypes"
            ],
            "Resource": "*"
        }
    ]
}
```

2단계: 프라이빗 CA 생성

SCEP용 커넥터를 사용하려면의 프라이빗 CA를 커넥터 AWS Private Certificate Authority 에 연결해야 합니다. SCEP 프로토콜에 있는 고유한 보안 취약성으로 인해 커넥터에만 있는 프라이빗 CA를 사용하는 것이 좋습니다.

프라이빗 CA는 다음 요구 사항을 충족해야 합니다.

- 활성 상태여야 하며 범용 작동 모드를 사용해야 합니다.
- 프라이빗 CA를 소유해야 합니다. 교차 계정 공유를 통해 공유된 프라이빗 CA는 사용할 수 없습니다.

SCEP용 커넥터와 함께 사용하도록 프라이빗 CA를 구성할 때 다음 고려 사항에 유의하세요.

- DNS 이름 제약 조건 - SCEP 디바이스에 대해 발급된 인증서에서 허용되거나 금지된 도메인을 제어하는 방법으로 DNS 이름 제약 조건을 사용하는 것이 좋습니다. 자세한 내용은 에서 DNS 이름 제약 조건을 적용하는 방법을 참조하세요 AWS Private Certificate Authority.
- 해지 - 프라이빗 CA에서 OCSP 또는 CRLs 활성화하여 해지를 허용합니다. 자세한 내용은 AWS Private CA 인증서 해지 방법 계획 단원을 참조하십시오.
- PII - CA 인증서에 개인 식별 정보(PII) 또는 기타 기밀 또는 민감한 정보를 추가하지 않는 것이 좋습니다. 보안 악용이 발생할 경우 민감한 정보의 노출을 제한하는 데 도움이 됩니다.
- 루트 인증서를 트러스트 스토어에 저장 - GetCertificateAuthorityCertificate의 인증서와 반환 값을 확인할 수 있도록 루트 CA 인증서를 디바이스 트러스트 스토어에 저장합니다. 트러스트 스토어와 관련된 트러스트 스토어에 대한 자세한 내용은 섹션을 AWS Private CA참조하세요루트 CA.

프라이빗 CA를 생성하는 방법에 대한 자세한 내용은 섹션을 참조하세요에서 프라이빗 CA 생성 AWS Private CA.

3단계:를 사용하여 리소스 공유 생성 AWS Resource Access Manager

SCEP API용 커넥터, AWS Command Line Interface AWS SDK 또는 SCEP용 커넥터 API를 사용하여 프로그래밍 방식으로 SCEP용 커넥터를 사용하는 경우 AWS Resource Access Manager 서비스 보안 주체 공유를 사용하여 SCEP용 커넥터와 프라이빗 CA를 공유해야 합니다. 이렇게 하면 SCEP용 커넥터에 프라이빗 CA에 대한 공유 액세스 권한이 부여됩니다. AWS 콘솔에서 커넥터를 생성하면 리소스 공유가 자동으로 생성됩니다. 리소스 공유에 대한 자세한 내용은 AWS RAM 사용 설명서의 리소스 공유 생성을 참조하세요.

를 사용하여 리소스 공유를 생성하려면 명령을 사용할 AWS CLI AWS RAM create-resource-share 수 있습니다. 다음 명령은 리소스 공유를 생성합니다. 공유하려는 프라이빗 CA의 ARN을 *resource-arns* 값으로 지정합니다.

```
$  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
```

를 호출하는 서비스 보안 주체는 프라이빗 CA에 대한 `CreateConnector` 인증서 발급 권한이 있습니다. SCEP용 커넥터를 사용하는 서비스 보안 주체가 AWS Private CA 리소스에 대한 일반적인 액세스 권한을 갖지 못하도록 하려면를 사용하여 권한을 제한합니다`CalledVia`.

SCEP용 커넥터 시작하기

SCEP용 AWS Private Certificate Authority 커넥터를 사용하면 사설 CA에서 SCEP 지원 디바이스 및 모바일 디바이스 관리(MDM) 시스템으로 인증서를 발급할 수 있습니다. 커넥터를 생성할 때는 인증서를 요청할 수 있는 SCEP URL을 AWS Private Certificate Authority 생성하고 MDM 시스템에 통합하는 데 사용할 수 있는 정보도 제공합니다.

인증서를 발급하려면 AWS Private Certificate Authority 프라이빗 CA를 생성하고 커넥터를 생성한 다음 SCEP 지원 MDM 시스템 및 디바이스를 구성하여 커넥터에서 인증서를 요청해야 합니다.

주제

- 시작하기 전 준비 사항
- 1단계: 커넥터 생성
- 2단계: MDM 시스템에 커넥터 세부 정보 복사

시작하기 전 준비 사항

다음 자습서에서는 SCEP용 커넥터를 생성하는 프로세스를 안내합니다.

이 자습서를 따르려면 프라이빗 CA와 SCEP 지원 디바이스가 필요합니다. 또한 먼저 SCEP용 커넥터 설정 섹션에 나열된 사전 조건을 충족해야 합니다.

다음 절차에서는 AWS 콘솔을 사용하여 커넥터를 생성하는 방법을 안내합니다.

작업

- 1단계: 커넥터 생성
- 2단계: MDM 시스템에 커넥터 세부 정보 복사

1단계: 커넥터 생성

범용 커넥터 또는 Microsoft Intune용 SCEP용 커넥터를 생성합니다. 범용 커넥터는 SCEP 지원 엔드포인트와 함께 사용하도록 설계되었으며 SCEP 챌린지 암호를 관리합니다. SCEP for Microsoft Intune용 커넥터는 Microsoft Intune과 함께 사용되며 Microsoft Intune을 사용하여 챌린지 암호를 관리합니다.

General-purpose

범용 커넥터를 생성하려면

AWS 계정에 로그인하고에서 SCEP용 커넥터 콘솔을 엽니다https://console.aws.amazon.com/pca-connector-scep/home .

1. 커넥터 생성을 선택합니다.
2. 커넥터 생성 페이지에서 필요에 따라 이름 태그 필드에 친숙한 이름을 커넥터에 지정합니다. 커넥터 목록에 이름이 표시됩니다. 원하는 경우 태그 추가를 선택하여 커넥터에 태그를 더 추가할 수 있습니다. 태그는 AWS 리소스에 할당하는 레이블입니다. 각 태그는 키와 값(선택사항)으로 구성됩니다. 태그를 사용하여 리소스를 검색 및 필터링하거나 AWS 비용을 추적할 수 있습니다.
3. 커넥터 유형에서 범용을 선택합니다.
4. 프라이빗 CA에서이 커넥터와 함께 사용할 프라이빗 CA를 선택합니다. 또는 프라이빗 CA 생성을 선택하여 새 CA를 생성합니다. SCEP 프로토콜의 고유한 취약성으로 인해이 커넥터 전용 프라이빗 CA를 사용하는 것이 좋습니다. 새 CA를 생성한 경우 생성이 완료되면 SCEP용 커넥터 콘솔로 AWS Private CA돌아가 프라이빗 CAs 목록을 새로 고칩니다. 새 프라이빗 CA를 선택할 수 있어야 합니다.
5. 챌린지 암호에서 챌린지 암호 자동 생성을 선택합니다. 이 커넥터를 생성할 때 정적 챌린지 암호를 생성합니다.
6. 연결에서 퍼블릭을 선택하여 퍼블릭 인터넷을 통해 액세스할 수 있는 커넥터를 생성합니다. 또는 프라이빗을 선택하고 VPC 엔드포인트를 지정하여 특정 VPC 엔드포인트를 통해서만이 커넥터에 액세스할 수 있도록 제한합니다.
7. 커넥터 생성을 선택합니다.

Microsoft Intune

Microsoft Intune용 SCEP용 커넥터를 생성하려면

AWS 계정에 로그인하고에서 SCEP용 커넥터 콘솔을 엽니다https://console.aws.amazon.com/pca-connector-scep/home .

1. 커넥터 생성을 선택합니다.
2. 커넥터 생성 페이지에서 필요에 따라 이름 태그 필드에 친숙한 이름을 커넥터에 지정합니다. 커넥터 목록에 이름이 표시됩니다. 원하는 경우 태그 추가를 선택하여 커넥터에 태그를 더 추가할 수 있습니다. 태그는 AWS 리소스에 할당하는 레이블입니다. 각 태그는 키와 값(선택사

항)으로 구성됩니다. 태그를 사용하여 리소스를 검색 및 필터링하거나 AWS 비용을 추적할 수 있습니다.

3. 커넥터 유형에서 Microsoft Intune을 선택합니다.

   a. 애플리케이션(클라이언트) ID에 Microsoft Entra ID 앱 등록의 애플리케이션(클라이언트) ID를 입력합니다. SCEP용 커넥터와 함께 Microsoft Intune을 사용하는 방법에 대한 자세한 내용은 섹션을 참조하세요SCEP용 커넥터에 대한 MDM 시스템 구성.

   b. 디렉터리(테넌트) ID 또는 기본 도메인에 Microsoft Entra ID 앱 등록의 디렉터리(테넌트) ID 또는 기본 도메인을 입력합니다.

4. 프라이빗 CA에서이 커넥터에 사용할 프라이빗 CA를 선택합니다. 또는 프라이빗 CA 생성을 선택하여 새 CA를 생성합니다. SCEP 프로토콜의 고유한 취약성으로 인해이 커넥터 전용 프라이빗 CA를 사용하는 것이 좋습니다. 새 CA를 생성한 경우 생성이 완료되면 SCEP용 커넥터 콘솔로 AWS Private CA돌아가 프라이빗 CAs 목록을 새로 고칩니다. 새 프라이빗 CA를 선택할 수 있어야 합니다.

5. 연결에서 퍼블릭을 선택하여 퍼블릭 인터넷을 통해 액세스할 수 있는 커넥터를 생성합니다. 또는 프라이빗을 선택하고 VPC 엔드포인트를 지정하여 특정 VPC 엔드포인트를 통해서만이 커넥터에 액세스할 수 있도록 제한합니다.

6. 커넥터 생성을 선택합니다.

2단계: MDM 시스템에 커넥터 세부 정보 복사

커넥터를 생성한 후 커넥터에서 MDM 시스템으로 다음 세부 정보를 복사해야 합니다. 콘솔을 사용하여 커넥터의 세부 정보를 보려면 SCEP용 커넥터 콘솔 페이지의 목록에서 커넥터를 선택합니다.

- SCEP URL - SCEP 클라이언트가 인증서를 요청할 커넥터의 엔드포인트입니다. 이 엔드포인트는 신뢰할 수 있는 엔터티에만 제공해야 합니다.
- (범용) 챌린지 암호 - 챌린지 암호에서 이전 절차에서 자동으로 생성한 암호를 선택한 다음 암호 보기를 선택하여 암호를 확인합니다. 추가 암호를 생성하려면 암호 생성을 선택합니다. 암호를 신중하게 배포하고 신뢰할 수 있는 개인과 클라이언트에게만 배포하도록 주의하십시오. 단일 챌린지 암호를 사용하여 모든 주체 및 SANs과 함께 모든 인증서를 발급할 수 있으므로 주의해야 합니다.
- (Microsoft Intune) Open ID 값 - Microsoft Intune과 통합하는 경우 Open ID 발급자, Open ID 주체 및 Open ID 대상을 Microsoft Entra 앱 등록의 OpenID Connect(OIDC) 자격 증명에 복사해야 합니다. 자세한 내용은 SCEP용 커넥터에 대한 MDM 시스템 구성 단원을 참조하십시오.

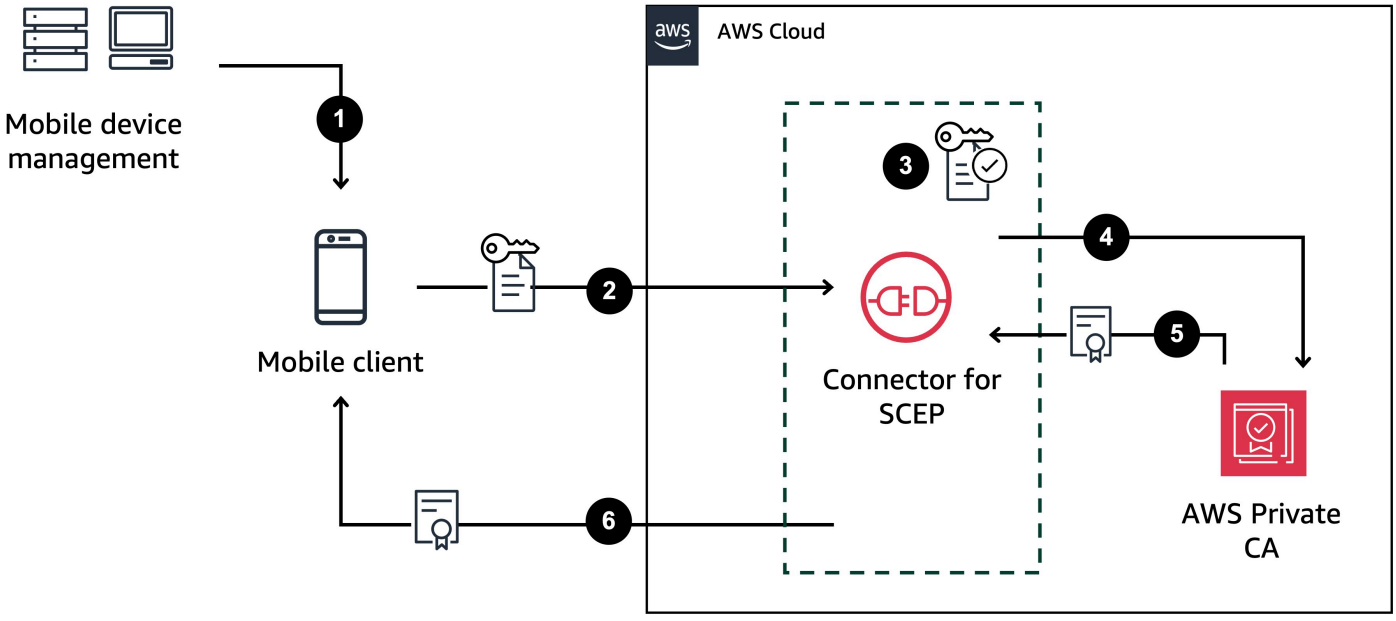
SCEP용 커넥터에 대한 MDM 시스템 구성

단순 인증서 등록 프로토콜(SCEP)은 인증서 등록 및 갱신에 사용되는 표준 프로토콜입니다. SCEP용 커넥터는에서 SCEP 클라이언트 AWS Private Certificate Authority 로 인증서를 자동으로 발급하는 RFC 8894 기반 SCEP 서버입니다. 커넥터를 생성할 때 SCEP용 커넥터는 SCEP 클라이언트가 인증서를 요청할 수 있는 HTTPS 엔드포인트를 제공합니다. 클라이언트는 서비스에 대한 인증서 서명 요청(CSR)의 일부로 포함된 챌린지 암호를 사용하여 인증합니다. Connector for SCEP를 Microsoft Intune, Omnissa Workspace ONE, Jamf Pro 등 널리 사용되는 모바일 디바이스 관리(MDM) 시스템과 함께 사용하여 모바일 디바이스를 등록할 수 있습니다. SCEP를 지원하는 모든 클라이언트 또는 엔드포인트에서 작동하도록 설계되었습니다.

SCEP용 커넥터는 범용 커넥터와 Microsoft Intune용 SCEP용 커넥터라는 두 가지 유형의 커넥터를 제공합니다. 다음 섹션에서는 작동 방식과 이를 사용하도록 MDM 시스템을 구성하는 방법을 설명합니다.

범용 커넥터

범용 커넥터는 전용 커넥터가 있는 Microsoft Intune을 제외하고 SCEP를 지원하는 모바일 디바이스 엔드포인트와 함께 작동하도록 설계되었습니다. Jamf Pro 또는 Omnissa Workspace ONE과 같은 범용 커넥터를 사용하면 SCEP 챌린지 암호를 관리할 수 있습니다. 다음 다이어그램은 모바일 디바이스 관리(MDM) 시스템을 예로 사용하지만 다른 SCEP 지원 시스템 또는 디바이스에도 동일한 기능이 적용됩니다.



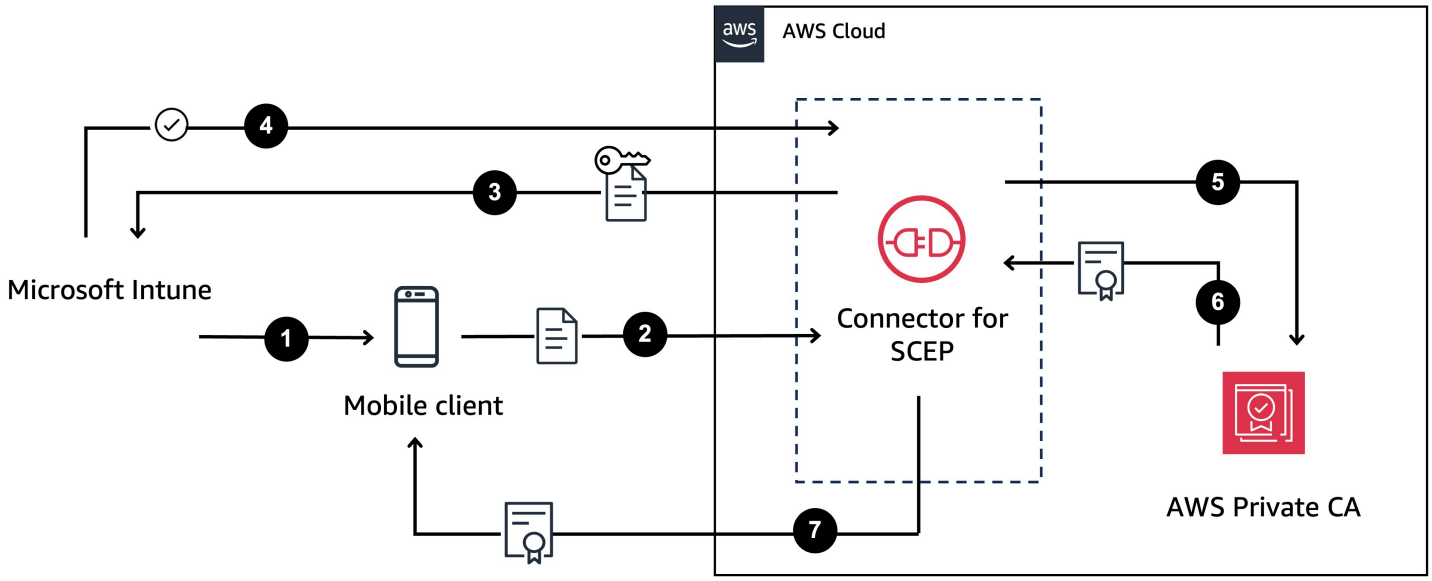
1. MDM 시스템(또는 기타 디바이스 또는 시스템)은 SCEP 프로파일을 모바일 클라이언트로 전송합니다. SCEP 프로파일에는 인증서 유효 기간, 챌린지 암호 및 인증서 발급과 관련된 기타 정보와 같이 인증서 프로파일을 정의하는 구성 파라미터가 포함되어 있습니다.

2. 모바일 클라이언트는 인증서를 요청하고 챌린지 암호가 포함된 인증서 서명 요청(CSR)도 전송합니다.
3. SCEP용 커넥터는 챌린지 암호를 검증합니다. 유효한 경우 서비스는 모바일 클라이언트를 AWS Private CA 대신하여에 인증서를 요청합니다.
4. AWS Private CA 는 인증서를 발급하고 SCEP용 커넥터로 전송합니다.
5. SCEP용 커넥터는 발급된 인증서를 모바일 클라이언트로 전송합니다.

AWS Private CA Microsoft Intune용 SCEP용 커넥터

AWS Private CA SCEP for Microsoft Intune용 커넥터는 Microsoft Intune과 함께 사용하도록 설계되었습니다. SCEP for Microsoft Intune 커넥터 유형을 사용하면 Microsoft Intune을 사용하여 SCEP 챌린지 암호를 관리할 수 있습니다. Microsoft Intune에서 SCEP용 커넥터 사용에 대한 자세한 내용은 섹션을 참조하세요SCEP용 커넥터용 Microsoft Intune 구성.

Microsoft Intune에서 SCEP용 커넥터를 사용하려면 Microsoft Intune API를 사용하여 특정 기능을 활성화하고 유효한 Microsoft Intune 라이선스를 보유해야 합니다. Microsoft Intune® 앱 보호 정책도 검토해야 합니다.



1. Microsoft Intune은 SCEP 프로파일을 모바일 클라이언트로 전송합니다. 프로필에는 모바일 클라이언트가 CSR에 배치하는 암호화된 챌린지 암호가 포함되어 있습니다.
2. 모바일 클라이언트는 인증서를 요청하고 CSR을 SCEP용 커넥터로 전송합니다.
3. SCEP용 커넥터는 권한 부여를 위해 CSR을 Microsoft Intune에 전송합니다.

4. Microsoft Intune은 CSR에서 챌린지 암호를 해독합니다. 유효한 경우 Microsoft Intune은 SCEP용 커넥터에 승인을 전송하여 모바일 클라이언트에 인증서를 발급합니다.
5. SCEP용 커넥터는 모바일 클라이언트를 AWS Private CA 대신하여에서 인증서를 요청합니다.
6. AWS Private CA 는 인증서를 발급하고 SCEP용 커넥터로 전송합니다.
7. SCEP용 커넥터는 발급된 인증서를 모바일 클라이언트로 전송합니다.

주제

- SCEP용 커넥터용 Jamf Pro 구성
- SCEP용 커넥터용 Microsoft Intune 구성
- SCEP용 커넥터용 Omnissa Workspace ONE 구성

SCEP용 커넥터용 Jamf Pro 구성

Jamf Pro 모바일 디바이스 관리(MDM) 시스템에서를 외부 인증 기관(CA) AWS Private CA 으로 사용할 수 있습니다. 이 가이드에서는 범용 커넥터를 생성한 후 Jamf Pro를 구성하는 방법에 대한 지침을 제공합니다.

SCEP용 커넥터용 Jamf Pro 구성

이 가이드에서는 SCEP용 커넥터와 함께 사용하도록 Jamf Pro를 구성하는 방법에 대한 지침을 제공합니다. Jamf Pro 및 SCEP용 커넥터를 성공적으로 구성하면 관리형 디바이스에 인증서를 발급 AWS Private CA 할 수 있습니다.

Jamf Pro 요구 사항

Jamf Pro 구현은 다음 요구 사항을 충족해야 합니다.

- Jamf Pro에서 인증서 기반 인증 활성화 설정을 활성화해야 합니다. 이 설정에 대한 세부 정보는 Jamf Pro 설명서의 Jamf Pro 보안 설정 페이지에서 확인할 수 있습니다.

1단계: (선택 사항 - 권장) 사설 CA의 지문 가져오기

지문은 다른 시스템 또는 애플리케이션과 신뢰를 구축할 때 CA의 ID를 확인하는 데 사용할 수 있는 프라이빗 CA의 고유 식별자입니다. 인증 기관(CA) 지문을 통합하면 관리형 디바이스가 연결 중인 CA를 인증하고 예상 CA에서만 인증서를 요청할 수 있습니다. Jamf Pro에서 CA 지문을 사용하는 것이 좋습니다.

프라이빗 CA에 대한 지문을 생성하려면

1. AWS Private CA 콘솔에서 또는 GetCertificateAuthorityCertificate를 사용하여 프라이빗 CA 인증서를 가져옵니다. `ca.pem` 파일로 저장합니다.
2. OpenSSL 명령줄 유틸리티를 설치합니다.
3. OpenSSL에서 다음 명령을 실행하여 지문을 생성합니다.

```
openssl x509 -in ca.pem -sha256 -fingerprint
```

2단계: Jamf Pro에서 외부 CA AWS Private CA 로 구성

SCEP용 커넥터를 생성한 후에는 Jamf Pro에서를 외부 인증 기관(CA) AWS Private CA 으로 설정해야 합니다. 를 글로벌 외부 CA AWS Private CA 로 설정할 수 있습니다. 또는 Jamf Pro 구성 프로파일을 사용하여 조직의 디바이스 하위 집합에 인증서 발급과 같은 다양한 사용 사례에 AWS Private CA 대해와 다른 인증서를 발급할 수 있습니다. Jamf Pro 구성 프로파일 구현에 대한 지침은이 문서의 범위를 벗어납니다.

Jamf Pro에서를 외부 인증 기관(CA) AWS Private CA 으로 구성하려면

1. Jamf Pro 콘솔에서 설정 > 글로벌 > PKI 인증서로 이동하여 PKI 인증서 설정 페이지로 이동합니다.
2. 관리 인증서 템플릿 탭을 선택합니다.
3. 외부 CA를 선택합니다.
4. 편집을 선택합니다.
5. (선택 사항) 구성 프로필에 대해 Jamf Pro를 SCEP 프록시로 활성화를 선택합니다. Jamf Pro 구성 프로파일을 사용하여 특정 사용 사례에 맞는 다양한 인증서를 발급할 수 있습니다. Jamf Pro에서 구성 프로파일을 사용하는 방법에 대한 지침은 Jamf Pro 설명서의 구성 프로파일에 대한 SCEP 프록시로 Jamf Pro 활성화를 참조하세요.
6. 컴퓨터 및 모바일 디바이스 등록에 SCEP 지원 외부 CA 사용을 선택합니다.
7. (선택 사항) 컴퓨터 및 모바일 디바이스 등록을 위해 Jamf Pro를 SCEP 프록시로 사용을 선택합니다. 프로파일 설치에 실패하는 경우 섹션을 참조하세요프로파일 설치 실패 문제 해결.
8. 커넥터 세부 정보에서 SCEP용 커넥터 SCEP URL을 복사하여 Jamf Pro의 URL 필드에 붙여넣습니다. 커넥터의 세부 정보를 보려면 SCEP용 커넥터 목록에서 커넥터를 선택합니다. 또는 GetConnector를 호출하여 URL을 가져오고 응답에서 `Endpoint` 값을 복사할 수 있습니다.

9. (선택 사항) 이름 필드에 인스턴스의 이름을 입력합니다. 예를 들어 이름을 로 지정할 수 있습니다AWS Private CA.
10. 챌린지 유형으로 정적을 선택합니다.
11. 커넥터에서 챌린지 암호를 복사하여 챌린지 필드에 붙여 넣습니다. 커넥터에는 여러 개의 챌린지 암호가 있을 수 있습니다. 커넥터의 챌린지 암호를 보려면 AWS 콘솔에서 커넥터의 세부 정보 페이지로 이동하여 암호 보기 버튼을 선택합니다. 또는 GetChallengePassword(GetChallengePassword)를 호출하여 커넥터의 챌린지 암호를 가져오고 응답에서 `Password` 값을 복사할 수 있습니다. 챌린지 암호 사용에 대한 자세한 내용은 섹션을 참조하세요SCEP 고려 사항 및 제한 사항에 대한 커넥터 이해.
12. 챌린지 암호를 챌린지 확인 필드에 붙여 넣습니다.
13. 키 크기를 선택합니다. 키 크기는 2048 이상인 것이 좋습니다.
14. (선택 사항) 디지털 서명으로 사용을 선택합니다. 디바이스에 Wi-Fi 및 VPN과 같은 리소스에 대한 보안 액세스 권한을 부여하려면 인증 목적으로이 옵션을 선택합니다.
15. (선택 사항) 키 암호화에 사용을 선택합니다.
16. (선택 사항 - 권장) 지문 필드에 16진수 문자열을 입력합니다. 관리형 디바이스가 CA를 확인하고 CA에서만 인증서를 요청할 수 있도록 CA 지문을 추가하는 것이 좋습니다. 프라이빗 CA에 대한 지문을 생성하는 방법에 대한 지침은 섹션을 참조하세요1단계: (선택 사항 - 권장) 사설 CA의 지문 가져오기.
17. 저장을 선택합니다.

3단계: 구성 프로필 서명 인증서 설정

Jamf Pro를 SCEP용 커넥터와 함께 사용하려면 커넥터와 연결된 프라이빗 CA에 대한 서명 및 CA 인증서를 제공해야 합니다. 두 인증서가 모두 포함된 프로필 서명 인증서 키 스토어를 Jamf Pro에 업로드하여이 작업을 수행할 수 있습니다.

다음은 인증서 키 스토어를 생성하여 Jamf Pro에 업로드하는 단계입니다.

- 내부 프로세스를 사용하여 인증서 서명 요청(CSR)을 생성합니다.
- 커넥터와 연결된 프라이빗 CA에서 서명한 CSR을 가져옵니다.
- 프로필 서명 인증서와 CA 인증서를 모두 포함하는 프로필 서명 인증서 키 스토어를 생성합니다.
- 인증서 키 스토어를 Jamf Pro에 업로드합니다.

다음 단계에 따라 디바이스가 프라이빗 CA에서 서명한 구성 프로파일을 검증하고 인증하여 Jamf Pro에서 SCEP용 커넥터를 사용할 수 있도록 할 수 있습니다.

1. 다음 예제에서는 OpenSSL 및를 사용하지 AWS Certificate Manager만 원하는 방법을 사용하여 인증서 서명 요청을 생성할 수 있습니다.

   AWS Certificate Manager console

   ACM 콘솔을 사용하여 프로필 서명 인증서를 생성하려면

   1. ACM을 사용하여 프라이빗 PKI 인증서를 요청합니다. 다음을 포함합니다.

      - 유형 - MDM 시스템의 SCEP 인증 기관 역할을 하는 것과 동일한 프라이빗 CA 유형을 사용합니다.
      - 인증 기관 세부 정보 섹션에서 인증 기관 메뉴를 선택하고 Jamf Pro의 CA 역할을 하는 프라이빗 CA를 선택합니다.
      - 도메인 이름 - 인증서에 포함할 도메인 이름을 제공합니다. 와 같은 정규화된 도메인 이름(FQDN) `www.example.com`또는와 같은 베어 또는 정점 도메인 이름`example.com`(제외)을 사용할 수 있습니다`www.`.

   2. ACM을 사용하여 이전 단계에서 생성한 프라이빗 인증서를 내보냅니다. 인증서, 인증서 체인 및 암호화된 키에 대한 파일 내보내기를 선택합니다. 암호는 다음 단계에서 필요하므로 준비해 둡니다.

   3. 터미널에서 내보낸 파일이 포함된 폴더에서 다음 명령을 실행하여 이전 단계에서 생성한 암호로 인코딩된 `output.p12` 파일에 PKCS#12 번들을 씁니다.

      ```
      openssl pkcs12 -export \
        -in "Exported Certificate.txt" \
        -certfile "Certificate Chain.txt" \
        -inkey "Exported Certificate Private Key.txt" \
        -name example \
        -out output.p12 \
        -passin pass:your-passphrase \
        -passout pass:your-passphrase
      ```

   AWS Certificate Manager CLI

   ACM CLI를 사용하여 프로필 서명 인증서를 생성하려면

   - 다음 명령은 ACM에서 인증서를 생성한 다음 파일을 PKCS#12 번들로 내보내는 방법을 보여줍니다.

```
PCA=<Enter your Private CA ARN>

CERTIFICATE=$(aws acm request-certificate \
    --certificate-authority-arn $PCA \
    --domain-name <any valid domain name, such as test.name> \
    | jq -r '.CertificateArn')

while [[ $(aws acm describe-certificate \
  --certificate-arn $CERTIFICATE \
  | jq -r '.Certificate.Status') != "ISSUED" ]] do sleep 1; done

aws acm export-certificate \
  --certificate-arn $CERTIFICATE \
  --passphrase password | jq -r '.Certificate' > Certificate.pem
aws acm export-certificate \
  --certificate-arn $CERTIFICATE \
  --passphrase password | jq -r '.CertificateChain' > CertificateChain.pem
aws acm export-certificate \
  --certificate-arn $CERTIFICATE \
  --passphrase password | jq -r '.PrivateKey' > PrivateKey.pem

openssl pkcs12 -export \
  -in "Certificate.pem" \
  -certfile "CertificateChain.pem" \
  -inkey "PrivateKey.pem" \
  -name example \
  -out output.p12 \
  -passin pass:passphrase \
  -passout pass:passphrase
```

OpenSSL CLI

OpenSSL CLI를 사용하여 프로필 서명 인증서를 생성하려면

1. OpenSSL을 사용하여 다음 명령을 실행하여 프라이빗 키를 생성합니다.

```
openssl genrsa -out local.key 2048
```

2. 인증서 서명 요청(CSR) 생성:

```
openssl req -new -key local.key -sha512 -out local.csr -
subj "/CN=MySigningCertificate/O=MyOrganization" -addext
 keyUsage=critical,digitalSignature,nonRepudiation
```

3. 를 사용하여 이전 단계에서 생성한 CSR을 사용하여 서명 인증서를 AWS CLI발급합니다. 다음 명령을 실행하고 응답에 인증서 ARN을 기록해 둡니다.

```
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. 다음 명령을 실행하여 서명 인증서를 가져옵니다. 이전 단계의 인증서 ARN을 지정합니다.

```
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. 다음 명령을 실행하여 CA 인증서를 가져옵니다.

```
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. OpenSSL을 사용하여 서명 인증서 키 스토어를 p12 형식으로 출력합니다. 4단계와 5단계에서 생성한 CRT 파일을 사용합니다.

```
openssl pkcs12 -export -in local.crt -inkey local.key -certfile ca.crt -name
 "CA Chain" -out local.p12
```

7. 메시지가 표시되면 내보내기 암호를 입력합니다. 이 암호는 Jamf Pro에 제공할 키 스토어 암호입니다.

2. Jamf Pro에서 관리 인증서 템플릿으로 이동하여 외부 CA 창으로 이동합니다.
3. 외부 CA 창 하단에서 서명 및 CA 인증서 변경을 선택합니다.
4. 화면의 지침에 따라 외부 CA에 대한 서명 및 CA 인증서를 업로드합니다.

4단계: (선택 사항) 사용자 시작 등록 중 인증서 설치

클라이언트 디바이스와 프라이빗 CA 간에 신뢰를 설정하려면 디바이스가 Jamf Pro에서 발급한 인증서를 신뢰하는지 확인해야 합니다. 등록 프로세스 중에 인증서를 요청할 때 Jamf Pro의 사용자 시작 등

록 설정을 사용하여 클라이언트 디바이스에 AWS Private CA의 CA 인증서를 자동으로 설치할 수 있습니다.

프로파일 설치 실패 문제 해결

컴퓨터 및 모바일 디바이스 등록을 위한 SCEP 프록시로 Jamf Pro 사용을 활성화한 후 프로파일 설치에 실패하는 경우 디바이스 로그를 참조하고 다음을 시도하세요.

디바이스 로그 오류 메시지	완화
`Profile installation failed. Unable to obtain certificate from SCEP server at "<your-jamf-endpoint>.jamfcloud.com". <MDM-SCEP:15001>`	등록을 시도하는 동안이 오류 메시지가 표시되면 등록을 다시 시도하세요. 등록에 성공하려면 몇 번의 시도가 필요할 수 있습니다.
`Profile installation failed. Unable to obtain certificate from SCEP server at "<your-jamf-endpoint>.jamfcloud.com". <MDM-SCEP:14006>`	챌린지 암호가 잘못 구성되었을 수 있습니다. Jamf Pro의 챌린지 암호가 커넥터의 챌린지 암호와 일치하는지 확인합니다.

SCEP용 커넥터용 Microsoft Intune 구성

Microsoft Intune 모바일 디바이스 관리(MDM) 시스템에서를 외부 인증 기관(CA) AWS Private CA 으로 사용할 수 있습니다. 이 가이드에서는 SCEP for Microsoft Intune용 커넥터를 생성한 후 Microsoft Intune을 구성하는 방법에 대한 지침을 제공합니다.

사전 조건

Microsoft Intune용 SCEP용 커넥터를 생성하기 전에 다음 사전 조건을 완료해야 합니다.

- Entra ID를 생성합니다.
- Microsoft Intune 테넌트를 생성합니다.
- Microsoft Entra ID에서 앱 등록을 생성합니다. 앱 등록에 대한 애플리케이션 수준 권한을 관리하는 방법에 대한 자세한 내용은 Microsoft Entra 설명서의 Microsoft Entra ID에서 앱의 요청된 권한 업데이트를 참조하세요. 앱 등록에는 다음 권한이 있어야 합니다.

  - Intune에서 scep_challenge_provider를 설정합니다.
  - Microsoft Graph의 경우 Application.Read.All 및 User.Read를 설정합니다.
- 앱 등록 관리자 동의에서 애플리케이션을 부여해야 합니다. 자세한 내용은 Microsoft Entra 설명서의 애플리케이션에 테넌트 전체 관리자 동의 부여를 참조하세요.

> **Tip**
>
> 앱 등록을 생성할 때 애플리케이션(클라이언트) ID와 디렉터리(테넌트) ID 또는 기본 도메인을 기록해 둡니다. Microsoft Intune용 SCEP용 커넥터를 생성할 때 이러한 값을 입력합니다. 이러한 값을 가져오는 방법에 대한 자세한 내용은 Microsoft Entra 설명서의 리소스에 액세스할 수 있는 Microsoft Entra 애플리케이션 및 서비스 보안 주체 생성을 참조하세요.

1단계: Microsoft Entra ID 애플리케이션을 사용할 수 있는 AWS Private CA 권한 부여

SCEP for Microsoft Intune용 커넥터를 생성한 후에는 SCEP용 커넥터가 Microsoft Intune과 통신할 수 있도록 Microsoft 앱 등록에 따라 페더레이션 자격 증명을 생성해야 합니다.

Microsoft Intune에서 외부 CA AWS Private CA 로를 구성하려면

1. Microsoft Entra ID 콘솔에서 앱 등록으로 이동합니다.
2. SCEP용 커넥터와 함께 사용하도록 생성한 애플리케이션을 선택합니다. 클릭하는 애플리케이션의 애플리케이션(클라이언트) ID는 커넥터를 생성할 때 지정한 ID와 일치해야 합니다.
3. 관리형 드롭다운 메뉴에서 인증서 및 보안 암호를 선택합니다.
4. 페더레이션 자격 증명 탭을 선택합니다.
5. 자격 증명 추가를 선택합니다.
6. 페더레이션 자격 증명 시나리오 드롭다운 메뉴에서 기타 발급자를 선택합니다.
7. SCEP for Microsoft Intune용 커넥터 세부 정보에서 OpenID 발급자 값을 복사하여 발급자 필드에 붙여 넣습니다. 커넥터의 세부 정보를 보려면 AWS 콘솔의 SCEP용 커넥터 목록에서 커넥터를 선택합니다. 또는 GetConnector를 호출하여 URL을 가져온 다음 응답에서 `Issuer` 값을 복사할 수 있습니다.
8. 유형에서 명시적 주체 식별자를 선택합니다.
9. 커넥터에서 OpenID 제목 값을 복사하여 값 필드에 붙여 넣습니다. AWS 콘솔의 커넥터 세부 정보 페이지에서 OpenID 발급자 값을 볼 수 있습니다. 또는 GetConnector를 호출하여 URL을 가져온 다음 응답에서 `Audience` 값을 복사할 수 있습니다.

10. (선택 사항) 이름 필드에 인스턴스의 이름을 입력합니다. 예를 들어 이름을 로 지정할 수 있습니다AWS Private CA.
11. (선택 사항) 설명 필드에 설명을 입력합니다.
12. SCEP for Microsoft Intune용 커넥터 세부 정보에서 OpenID 대상 값을 복사하여 대상 필드에 붙여 넣습니다. 커넥터의 세부 정보를 보려면 AWS 콘솔의 SCEP용 커넥터 목록에서 커넥터를 선택합니다. 또는 GetConnector를 호출하여 URL을 가져온 다음 응답에서 `Subject` 값을 복사할 수 있습니다.
13. 추가를 선택합니다.

2단계: Microsoft Intune 구성 프로필 설정

Microsoft Intune AWS Private CA 을 호출할 수 있는 권한을 부여한 후에는 Microsoft Intune을 사용하여 디바이스가 인증서 발급을 위해 SCEP용 커넥터에 연결하도록 지시하는 Microsoft Intune 구성 프로파일을 생성해야 합니다.

1. 신뢰할 수 있는 인증서 구성 프로필을 생성합니다. 신뢰를 구축하려면 SCEP용 커넥터와 함께 사용 중인 체인의 루트 CA 인증서를 Microsoft Intune에 업로드해야 합니다. 신뢰할 수 있는 인증서 구성 프로필을 생성하는 방법에 대한 자세한 내용은 Microsoft Intune 설명서의 Microsoft Intune에 대한 신뢰할 수 있는 루트 인증서 프로필을 참조하세요.
2. 새 인증서가 필요할 때 디바이스가 커넥터를 가리키도록 SCEP 인증서 구성 프로파일을 생성합니다. 구성 프로필의 프로필 유형은 SCEP 인증서여야 합니다. 구성 프로필의 루트 인증서의 경우 이전 단계에서 생성한 신뢰할 수 있는 인증서를 사용해야 합니다.

   SCEP 서버 URLs 경우 커넥터 세부 정보에서 SCEP URL을 복사하여 SCEP 서버 URLs 필드에 붙여 넣습니다. 커넥터의 세부 정보를 보려면 SCEP용 커넥터 목록에서 커넥터를 선택합니다. 또는 ListConnectors를 호출하여 URL을 가져온 다음 응답에서 `Endpoint` 값을 복사할 수 있습니다. Microsoft Intune에서 구성 프로파일을 생성하는 방법에 대한 지침은 Microsoft Intune 설명서의 Microsoft Intune에서 SCEP 인증서 프로파일 생성 및 할당을 참조하세요.

   > **Note**
   >
   > 비 Mac OS 및 iOS 디바이스의 경우 구성 프로필에서 유효 기간을 설정하지 않으면 SCEP용 커넥터는 유효 기간이 1년인 인증서를 발급합니다. 구성 프로필에서 확장 키 사용(EKU) 값을 설정하지 않으면 SCEP용 커넥터는 로 설정된 EKU로 인증서를 발급합니다`Client Authentication (Object Identifier: 1.3.6.1.5.5.7.3.2)`. macOS 또는 iOS 디바이스의 경우 Microsoft Intune은 구성 프로필의 `ExtendedKeyUsage` 또는 `Validity` 파라미터를 준수하지 않습니다. 이러한 디바

이스의 경우 SCEP용 커넥터는 클라이언트 인증을 통해 이러한 디바이스에 1년 유효 기간이 있는 인증서를 발급합니다.

3단계: SCEP용 커넥터에 대한 연결 확인

SCEP용 커넥터 엔드포인트를 가리키는 Microsoft Intune 구성 프로파일을 생성한 후 등록된 디바이스가 인증서를 요청할 수 있는지 확인합니다. 확인하려면 정책 할당 실패가 없는지 확인합니다. 확인하려면 Intune 포털에서 디바이스 > 디바이스 관리 > 구성으로 이동하여 구성 정책 할당 실패 아래에 나열된 항목이 없는지 확인합니다. 있는 경우 이전 절차의 정보로 설정을 확인합니다. 설정이 올바르고 여전히 장애가 있는 경우 모바일 디바이스에서 사용 가능한 데이터 수집을 참조하세요.

디바이스 등록에 대한 자세한 내용은 Microsoft Intune 설명서의 디바이스 등록이란 무엇입니까?를 참조하세요.

SCEP용 커넥터용 Omnissa Workspace ONE 구성

를 Omnissa Workspace ONE UEM(통합 엔드포인트 관리) 시스템에서 외부 인증 기관(CA) AWS Private CA 으로 사용할 수 있습니다. 이 가이드에서는 SCEP 커넥터를 생성한 후 Omnissa Workspace ONE을 구성하는 방법에 대한 지침을 제공합니다 AWS.

사전 조건

Omnissa Workspace ONE용 SCEP 커넥터를 생성하기 전에 다음 사전 조건을 완료해야 합니다.

- AWS 콘솔에서 프라이빗 CA를 생성합니다. 자세한 내용은 에서 프라이빗 CA 생성 AWS Private CA 단원을 참조하십시오.
- 범용 SCEP 커넥터를 생성합니다. 자세한 내용은 커넥터 생성을 참조하세요.
- 조직 그룹 ID가 있는 활성 Omnissa Workspace ONE 환경 관리자 계정이 있어야 합니다.
- Apple 디바이스를 등록하는 경우 MDM용 Apple 푸시 알림 서비스(APNs)를 구성합니다. 자세한 내용은 Omnissa 설명서의 APNs 인증서를 참조하세요.

1단계: Omnissa Workspace ONE에서 인증 기관 및 템플릿 정의

AWS 콘솔에서 프라이빗 CA 및 SCEP 커넥터를 생성한 후 Omnissa Workspace ONE에서 인증 기관 및 템플릿을 정의합니다.

인증 기관 AWS Private CA 으로 추가

1. 시스템 메뉴에서 엔터프라이즈 통합을 선택한 다음 인증 기관을 선택합니다.
2. + ADD를 선택하고 다음 정보를 제공합니다.
   - 이름: AWS-Private-CA.
   - 설명: 디바이스 인증서 발급에 AWS Private CA 대한 설명입니다.
   - 권한 유형: 일반 SCEP를 선택합니다.
   - SCEP URL: SCEP URL을 입력합니다 AWS Private CA.
   - 챌린지 유형: 정적을 선택합니다.
   - 정적 챌린지: AWS 콘솔의 SCEP용 커넥터 구성에서 SCEP 정적 챌린지 암호를 입력합니다.
   - 재시도 제한 시간 및 최대 재시도 횟수 값을 입력합니다.
3. 구성을 저장합니다.

인증서 템플릿 생성

1. 시스템 메뉴에서 엔터프라이즈 통합을 선택하고 인증 기관을 선택한 다음 템플릿을 선택합니다.
2. 템플릿 추가를 선택하고 다음 정보를 제공합니다.
   - 템플릿 이름: Device-Cert-Template.
   - 인증 기관: AWS-Private-CA를 선택합니다.
   - 제목 이름: 사용자 지정 가능한 필드입니다. 속성 목록에서 변수 값을 선택할 수 있습니다. 예: CN={DeviceReportedName}, O={DevicePlatform}, OU={CustomAttribute1}
   - 프라이빗 키 길이: 2048비트.
   - 프라이빗 키 유형: 필요에 따라 서명 및 암호화를 선택합니다.
   - 자동 갱신: 활성화됨/비활성화됨(필요에 따라 다름).
3. 템플릿의 이름을 (으)로 지정합니다.

2단계: Omnissa Workspace ONE UEM 프로파일 구성 설정

디바이스를 SCEP용 커넥터로 전달하여 인증서를 발급하도록 Omnissa Workspace ONE UEM에서 프로파일을 생성합니다.

인증서 배포를 위한 SCEP 디바이스 프로파일 생성

1. 리소스 메뉴에서 프로필 및 기준을 선택한 다음 프로필을 선택합니다.
2. 추가를 선택한 다음 프로필 추가를 선택합니다.
3. 디바이스 플랫폼(Android, iOS, macOS, Windows)을 선택합니다.
4. 관리 유형과 컨텍스트를 적절하게 설정합니다.
5. 이름: Device-Cert-Profile을 설정합니다.
6. SCEP 페이로드로 스크롤합니다.
7. SCEP를 선택한 다음 +추가를 선택합니다.
8. 다음 구성을 사용합니다.

   - SCEP:
     - 자격 증명 소스에서 정의된 인증 기관(기본값)을 선택합니다.
     - 인증 기관에서 AWS-Private-CA를 선택합니다.
     - 인증서 템플릿에서 1단계에 정의된 Device-Cert-Template을 선택합니다.
9. 다음을 선택하고 할당 섹션의 목록에서 적절한 스마트 그룹(디바이스의 할당 그룹)을 선택합니다.
10. 할당 유형을 자동으로 선택하여 자동 갱신을 활성화합니다.
11. 프로필을 저장하고 게시합니다.

> **Note**
>
> 자세한 내용은 Omnissa 설명서의 SCEP를 참조하세요.

3단계: Omnissa Workspace ONE에 디바이스 등록

스마트 그룹 생성 또는 확인

1. 그룹 및 설정에서 그룹을 선택한 다음 할당 그룹을 선택합니다.
2. POC-Devices 스마트 그룹을 생성하거나 편집합니다.

   - 이름: POC-Devices.
   - 디바이스 유형: 모두 또는 특정 플랫폼(예: Android 또는 iOS)을 선택합니다.
   - 기준: UserGroup, 플랫폼 및 OS, OEM 및 모델을 사용하여 대상 디바이스를 그룹화하는 기준을 지정합니다.

- 소유권: 개인 또는 회사 디바이스의 경우 모두를 선택합니다.

3. 대상 디바이스를 저장하고 미리 보기 탭에 나타나는지 확인합니다.

수동 디바이스 등록

Android

- Google Play에서 Workspace ONE Intelligent Hub 앱을 다운로드합니다.
- 앱을 열고 등록 URL을 입력하거나 QR 코드를 스캔합니다.
- 로그인하고 프롬프트에 따라 MDM 관리형 디바이스로 등록합니다.

iOS/macOS

- 디바이스에서 Safari를 열고 등록 URL(예: https://<WorkspaceONEUEMHostname>/enroll)로 이동합니다.
- 사용자 자격 증명으로 로그인합니다.
- App Store에서 Workspace ONE Intelligent Hub 앱을 다운로드하여 설치합니다.
- 프롬프트에 따라 설정 > 일반 > VPN 및 디바이스 관리 > 프로필 > 설치에서 MDM 프로필을 설치합니다.

Windows

- Workspace ONE 서버 또는 Microsoft Store에서 Workspace ONE Intelligent Hub를 다운로드합니다.
- 등록 URL 및 자격 증명을 사용하여 Hub를 통해 등록합니다.

디바이스 > 목록 보기 > 추가 작업 > 스마트 그룹에 할당에서 등록된 디바이스를 POC-디바이스 스마트 그룹에 할당합니다.

자세한 내용은 Omnissa 설명서의 자동 디바이스 등록을 참조하세요.

등록 확인

1. Omnissa Workspace ONE UEM 콘솔에서 디바이스로 이동한 다음 목록 보기로 이동합니다.
2. 등록된 디바이스가 등록됨 상태로 표시되는지 확인합니다.
3. 디바이스 세부 정보의 그룹 탭에서 디바이스가 POC-Devices 스마트 그룹에 있는지 확인합니다.

4단계: 인증서 발급

인증서 발급 트리거

1. 디바이스 목록 보기에서 등록된 디바이스를 선택합니다.
2. 쿼리 버튼에서를 선택하여 체크인 메시지를 표시합니다.
3. Device-Cert-Profile은를 통해 인증서를 발급해야 합니다 AWS Private CA.

인증서 설치 확인

Android

설정을 선택한 다음 보안을 선택하고 신뢰할 수 있는 자격 증명을 선택한 다음 사용자를 선택하여 인증서를 확인합니다.

iOS

설정으로 이동한 다음 일반, VPN 및 디바이스 관리, 구성 프로필을 차례로 선택합니다. AWS-Private-CA의 인증서가 있는지 확인합니다.

macOS

키체인 액세스를 연 다음 시스템 키체인을 열고 인증서를 확인합니다.

Windows

certmgr.msc, Personal, Certificates를 차례로 열어 인증서를 확인합니다.

문제 해결

SCEP 오류(예: "22013 - SCEP 서버가 잘못된 응답을 반환함")

- Workspace ONE에서 SCEP URL과 정적 챌린지 암호가 일치하는지 확인합니다 AWS Private CA.
- SCEP 엔드포인트 연결 테스트: curl <SCEP_URL>.
- AWS CloudTrail 로그에 AWS Private CA 오류(`IssueCertificate`예: 실패)가 있는지 확인합니다.

APNs(iOS/macOS)

- APNs가 유효하고 올바른 조직 그룹에 할당되었는지 확인합니다.
- APNs 연결 테스트: telnet gateway.push.apple.com 2195.

프로파일 설치 실패

- 디바이스가 올바른 스마트 그룹(디바이스, 목록 보기, 그룹)에 있는지 확인합니다.
- 프로파일 동기화 강제 적용: 추가 작업, 전송, 프로파일 목록.

로그

- Android: Logcat 또는 Workspace ONE 로그를 사용합니다.
- iOS/macOS: 로그 표시 --predicate 'process == 'mdmclient'' --last 1h(Xcode/Apple Configurator 사용).
- Windows: 이벤트 뷰어, 애플리케이션 및 서비스 로그, Microsoft-Windows-DeviceManagement.
- Workspace ONE UEM: 모니터링, 보고서 및 분석, 이벤트, 디바이스 이벤트.

의 SCEP 모니터링용 커넥터에 대한 자세한 내용은 SCEP용 커넥터 모니터링을 AWS참조하세요.

보안 고려 사항

- SCEP URLs과 보안 암호를 안전하게 저장합니다. 자세한 내용은 AWS Secrets Manager 서비스를 참조하세요.
- 스마트 그룹 기준을 대상 디바이스로만 제한합니다.
- Apple 푸시 알림(APNs) 인증서를 정기적으로 갱신합니다(1년간 유효).
- 개념 증명 프로젝트의 짧은 인증서 유효 기간을 설정하여 위험을 최소화합니다.
- 개인용 디바이스의 경우 정리에서 모든 프로필과 인증서를 제거해야 합니다.

SCEP 커넥터를 사용하여 Omnissa Workspace ONE UEM 및 CA 통합을 구성하는 방법에 대한 자세한 내용은 Omnissa Workspace ONE의 SCEP 설명서를 참조하세요.

SCEP용 커넥터의 보안

의 클라우드 보안 AWS 이 최우선 순위입니다. AWS 고객은 보안에 가장 민감한 조직의 요구 사항을 충족하도록 구축된 데이터 센터 및 네트워크 아키텍처의 이점을 누릴 수 있습니다.

보안은 AWS 와 사용자 간의 공동 책임입니다. 공동 책임 모델은 이를 클라우드의 보안과 클라우드 내 보안으로 설명합니다.

- 클라우드 보안 - AWS 는 클라우드에서 AWS 서비스를 실행하는 인프라를 보호할 책임이 있습니다 AWS .

- 클라우드의 보안 - 사용자의 책임은 사용하는 AWS 서비스에 따라 결정됩니다. 또한 귀하는 데이터의 민감도, 회사 요구 사항, 관련 법률 및 규정을 비롯한 기타 요소에 대해서도 책임이 있습니다.

이 설명서는 SCEP용 커넥터를 사용할 때 공동 책임 모델을 적용하는 방법을 이해하는 데 도움이 됩니다. 다음 주제에서는 보안 및 규정 준수 목표에 맞게 SCEP용 커넥터를 구성하는 방법을 보여줍니다.

주제

- SCEP VPC 엔드포인트용 커넥터(AWS PrivateLink)

SCEP VPC 엔드포인트용 커넥터(AWS PrivateLink)

인터페이스 VPC 엔드포인트를 구성하여 VPC와 SCEP용 커넥터 간에 프라이빗 연결을 생성할 수 있습니다. 인터페이스 엔드포인트는 SCEP API 작업용 커넥터에 비공개AWS PrivateLink로 액세스하는 기술로 구동됩니다.는 Amazon 네트워크를 통해 VPC와 SCEP용 커넥터 간의 모든 네트워크 트래픽을 AWS PrivateLink 라우팅하여 개방형 인터넷에 노출되지 않도록 합니다. 각 VPC 엔드포인트는 하나 이상의 탄력적 네트워크 인터페이스 및 VPC 서브넷의 프라이빗 IP 주소로 표현됩니다.

인터페이스 VPC 엔드포인트는 인터넷 게이트웨이, NAT 디바이스, VPN Direct Connect 연결 또는 연결 없이 VPC를 SCEP용 커넥터에 직접 연결합니다. VPC의 인스턴스는 SCEP용 커넥터 API와 통신하는 데 퍼블릭 IP 주소가 필요하지 않습니다.

VPC를 통해 SCEP용 커넥터를 사용하려면 VPC 내에 있는 인스턴스에서 연결해야 합니다. 또는 AWS Virtual Private Network (Site-to-Site VPN) 또는를 사용하여 프라이빗 네트워크를 VPC에 연결할 수 있습니다 Direct Connect. 에 대한 자세한 내용은 Amazon VPC 사용 설명서의 VPN 연결을 Site-to-Site VPN참조하세요. https://docs.aws.amazon.com/vpc/latest/userguide/vpn-connections.html Direct Connect에 대한 자세한 내용은 Direct Connect 사용 설명서의 연결 생성을 참조하세요.

SCEP용 커넥터는를 사용할 필요가 AWS PrivateLink없지만 추가 보안 계층으로 사용하는 것이 좋습니다. AWS PrivateLink 및 VPC 엔드포인트에 대한 자세한 내용은를 통한 서비스 액세스를 참조하세요 AWS PrivateLink.

SCEP VPC 엔드포인트용 커넥터에 대한 고려 사항

SCEP용 커넥터에 대한 인터페이스 VPC 엔드포인트를 설정하기 전에 다음 고려 사항에 유의하세요.

- SCEP용 커넥터는 일부 가용 영역에서 VPC 엔드포인트를 지원하지 않을 수 있습니다. VPC 엔드포인트를 생성할 때는 먼저 관리 콘솔에서 지원을 확인하세요. 지원되지 않는 가용 영역은 “이 가용 영역에서 서비스가 지원되지 않음”으로 표시됩니다.

- VPC 엔드포인트는 교차 리전 요청을 지원하지 않습니다. 커넥터를 생성하는 리전과 동일한 리전에서 엔드포인트를 생성해야 합니다.
- VPC 엔드포인트는 Amazon Route 53을 통해 Amazon이 제공하는 DNS만 지원합니다. 자신의 DNS를 사용하는 경우에는 조건적인 DNS 전송을 사용할 수 있습니다. 자세한 내용은 Amazon VPC 사용 설명서의 DHCP 옵션 세트를 참조하세요.
- VPC 엔드포인트에 연결된 보안 그룹은 VPC의 프라이빗 서브넷에서 443 포트로 들어오는 연결을 허용해야 합니다.

SCEP용 커넥터에 대한 VPC 엔드포인트 생성

https://console.aws.amazon.com/vpc/ 또는에서 VPC 콘솔을 사용하여 SCEP용 커넥터 서비스에 대한 VPC 엔드포인트를 생성할 수 있습니다 AWS Command Line Interface. 자세한 내용은 Amazon VPC 사용 설명서의 인터페이스 엔드포인트 생성 절차를 참조하세요. SCEP용 커넥터는 VPC 내의 모든 API 작업에 대한 호출을 지원합니다.

엔드포인트를 생성할 때 `com.amazonaws.region.pca-connector-scep`를 서비스 이름으로 지정합니다.

엔드포인트에 대해 프라이빗 DNS 호스트 이름을 활성화한 경우 이제 SCEP 엔드포인트용 기본 커넥터가 VPC 엔드포인트로 확인됩니다. 기본 서비스 엔드포인트의 전체 목록은 서비스 엔드포인트 및 할당량을 참조하십시오.

프라이빗 DNS 호스트 이름을 활성화하지 않은 경우, Amazon VPC는 다음 형식으로 사용할 수 있는 DNS 엔드포인트를 제공합니다.

```
vpc-endpoint-id.pca-connector-scep.region.vpce.amazonaws.com
```

자세한 내용은 Amazon VPC 사용 설명서의 VPC 엔드포인트(AWS PrivateLink)를 참조하세요.

SCEP용 커넥터에 대한 VPC 엔드포인트 정책 생성

SCEP용 커넥터에 대한 Amazon VPC 엔드포인트 정책을 생성하여 다음을 지정할 수 있습니다.

- 작업을 수행할 수 있는 보안 주체.
- 수행할 수 있는 작업
- 작업을 수행할 수 있는 리소스

자세한 내용은 Amazon VPC 가이드의 VPC 엔드포인트를 사용하여 서비스에 대한 액세스 제어를 참조하세요.

예 - SCEP 작업용 커넥터에 대한 VPC 엔드포인트 정책

엔드포인트에 연결되면 다음 정책은 모든 보안 주체에 대해 지정된 커넥터 리소스의 나열된 SCEP 작업용 커넥터에 대한 액세스 권한을 부여합니다.

```
{
   "Statement": [
      {
         "Principal": "*",
         "Effect": "Allow",
         "Action": [
            "pca-connector-scep:GetConnector",
            "pca-connector-scep:ListConnectors"
         ],
         "Resource": "arn:aws:pca-connector-scep:region:account:connector/connector-id"
      }
   ]
}
```

SCEP 등록 작업용 커넥터에 대한 VPC 엔드포인트 생성

SCEP용 커넥터는 `GetCACaps` 및와 같은 등록 작업을 위한 별도의 VPC 엔드포인트 서비스를 제공합니다`PKIOperation`.

등록 엔드포인트를 생성할 때 `com.amazonaws.region.pca-connector-scep.enroll`를 서비스 이름으로 지정합니다.

커넥터를 생성할 때 선택적으로를 지정`VpcEndpointId`하여 특정 VPC 엔드포인트를 통해서만 커넥터에 액세스할 수 있도록 제한할 수 있습니다.

프라이빗 DNS 호스트 이름을 활성화하지 않은 경우, Amazon VPC는 다음 형식으로 사용할 수 있는 DNS 엔드포인트를 제공합니다.

```
vpc-endpoint-id.enroll.pca-connector-scep.region.vpce.amazonaws.com
```

> **Note**
>
> 커넥터에 연결하려면 VPC 엔드포인트 DNS 이름이 아닌 커넥터 세부 정보에 포함된 엔드포인트 URL을 사용해야 합니다. 그러나 커넥터 엔드포인트 URL의 DNS 이름 부분을 AZ별 DNS 이름과 같은 유효한 VPC 엔드포인트 DNS 이름으로 바꿀 수 있습니다.
> 예를 들어 AZ 특정 DNS 이름을 사용하려면
>
> ```
> https://vpc-endpoint-id.enroll.pca-connector-scep.region.vpce.amazonaws.com/account-id-connector-id/UUID
> ```
>
> with
>
> ```
> https://vpc-endpoint-id-availability-zone.enroll.pca-connector-scep.region.vpce.amazonaws.com/account-id-connector-id/UUID
> ```

예 - SCEP 등록 작업용 커넥터에 대한 VPC 엔드포인트 정책

VPC 엔드포인트 정책을 연결하여 등록 작업에 대한 액세스를 제어할 수 있습니다. 엔드포인트에 연결되면 다음 정책은 모든 보안 주체에게 `GetCACaps` 및 `PKIOperation` 작업에 대한 액세스 권한을 부여합니다. 스탠자의 리소스는 커넥터입니다.

SCEP 등록 작업용 커넥터는 SigV4로 인증되지 않습니다. 따라서 IAM 보안 주체와 연결되지 않고 VPC 엔드포인트 정책에서 익명으로 간주됩니다. 따라서 VPC 엔드포인트 정책은 이러한 작업에 대한 모든 보안 주체를 허용해야 합니다.

```
{
   "Statement": [
      {
         "Principal": "*",
         "Effect": "Allow",
         "Action": [
            "pca-connector-scep:GetCACaps",
            "pca-connector-scep:GetCACert",
            "pca-connector-scep:PKIOperation"
         ],
         "Resource": [
            arn:aws:pca-connector-scep:us-east-1:111122223333:connector/11223344-1234-1122-2233-112233445566
         ]
```

```
        }
    ]
}
```

SCEP용 모니터 커넥터

모니터링은 SCEP용 커넥터 및 기타 AWS 솔루션의 안정성, 가용성 및 성능을 유지하는 데 중요한 부분입니다.는 SCEP용 커넥터를 모니터링하고, 이상이 있을 때 보고하고, 필요한 경우 자동 조치를 취할 수 있도록 다음 모니터링 도구를 AWS 제공합니다.

- AWS CloudTrail은 직접 수행하거나 AWS 계정 을 대신하여 수행한 API 직접 호출 및 관련 이벤트를 캡처하고 지정한 Amazon S3 버킷에 로그 파일을 전송합니다. AWS APIs라고 하는 사용자 및 계정, 호출이 수행된 소스 IP 주소, 호출이 발생한 시기를 식별할 수 있습니다.

  CloudTrail 데이터 이벤트를 모니터링하는 경우 로그에는 클라이언트 디바이스의 모든 최근 요청 목록이 포함됩니다. 데이터 이벤트는 IP 주소, 수행된 작업 유형, 작업이 `failed` 상태를 초래하는 경우 오류 코드 및 세부 메시지와 같은 클라이언트 디바이스 정보를 식별하는 것과 함께 제공됩니다. 자세한 내용은 AWS CloudTrail 사용 설명서를 참조하십시오.
- EventBridge: 애플리케이션을 다양한 소스의 데이터와 쉽게 연결할 수 있는 서버리스 이벤트 버스 서비스입니다. EventBridge는 자체 애플리케이션, Software-as-a-Service(SaaS) 애플리케이션 및 AWS 서비스의 실시간 데이터 스트림을 제공하고 해당 데이터를 Lambda 및 CloudWatch Logs와 같은 대상으로 라우팅합니다. 이를 통해 서비스에서 발생하는 이벤트를 모니터링하고 이벤트 기반 아키텍처를 구축할 수 있습니다. 자세한 내용은 Amazon EventBridge 사용 설명서를 참조하세요.

주제

- EventBridge를 사용하여 SCEP용 커넥터 자동화
- 를 사용한 SCEP API 호출용 로그 커넥터 AWS CloudTrail

EventBridge를 사용하여 SCEP용 커넥터 자동화

Amazon EventBridge를 사용하여 AWS 서비스를 자동화하고 애플리케이션 가용성 문제 또는 리소스 변경과 같은 시스템 이벤트에 자동으로 대응할 수 있습니다. AWS 서비스의 이벤트는 거의 실시간으로 EventBridge로 전송됩니다. 관심이 있는 이벤트와 이벤트 발생 시 실행할 자동 작업을 표시하도록 간단한 규칙을 작성할 수 있습니다. EventBridge는 한 번 이상 게시됩니다. 자세한 내용은 EventBridge에서 이벤트에 반응하는 규칙 생성을 참조하세요.

CloudWatch Events는 EventBridge를 사용하여 작업으로 전환됩니다. EventBridge를 사용하면 이벤트를 사용하여 대상을 트리거할 수 있습니다. 자세한 내용은 Amazon EventBridge란 무엇입니까?를 참조하세요.

SCEP 이벤트 유형용 커넥터

인증서 발급 성공

SCEP용 커넥터는 `PkiOperationPost` 요청에 대한 응답으로 인증서를 발급할 때 EventBridge에 `Certificate Issuance Succeeded` 이벤트를 전송합니다.

다음은 이벤트에 대한 예제 데이터입니다.

```
{
   "version": "0",
   "id": "event_ID",
   "detail-type": "Certificate Issuance Succeeded",
   "source": "aws.pca-connector-scep",
   "account": "account",
   "time": "2024-09-12T19:14:56Z",
   "region": "region",
   "resources":[
       "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
       "arn:aws:pca-connector-scep:us-
east-1:111122223333:connector/11223344-1234-1122-2233-112233445566"
   ],
   "detail": {
       "result": "success",
       "requestType": "PkiOperationPost",
       "certificateArn": "arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID"
   }
}
```

인증서 발급 실패

`PkiOperationPost` 요청에 대한 응답으로 인증서를 발급할 수 없는 경우 SCEP용 커넥터는 EventBridge에 `Certificate Issuance Failed` 이벤트를 전송합니다.

다음은 이벤트에 대한 예제 데이터입니다.

```
{
```

```
   "version": "0",
   "id": "event_ID",
   "detail-type": "Certificate Issuance Failed",
   "source": "aws.pca-connector-scep",
   "account": "account",
   "time": "2024-09-12T19:14:56Z",
   "region": "region",
   "resources":[
       "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
       "arn:aws:pca-connector-scep:us-
east-1:111122223333:connector/11223344-1234-1122-2233-112233445566"
   ],
   "detail": {
       "result": "failure",
       "requestType": "PkiOperationPost",
       "reason": "The certificate authority is not active."
   }
}
```

인증 기관 인증서 검색 성공

SCEP용 커넥터는 `GetCACert` 요청을 수신하고 커넥터의 프라이빗 CA 인증서를 성공적으로 검색할 때 EventBridge로 `Certificate Authority Certificate Retrieval Succeeded` 이벤트를 전송합니다.

다음은 이벤트에 대한 예제 데이터입니다.

```
{
   "version": "0",
   "id": "event_ID",
   "detail-type": "Certificate Authority Certificate Retrieval Succeeded",
   "source": "aws.pca-connector-scep",
   "account": "account",
   "time": "2024-09-12T19:14:56Z",
   "region": "region",
   "resources":[
       "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
       "arn:aws:pca-connector-scep:us-
east-1:111122223333:connector/11223344-1234-1122-2233-112233445566"
   ],
   "detail": {
```

```
        "result": "success",
        "requestType": "GetCACert"
    }
}
```

인증 기관 인증서 검색 실패

SCEP용 커넥터는 GetCACert 요청을 수신하고 커넥터의 프라이빗 CA 인증서를 검색할 수 없는 경우 EventBridge로 Certificate Authority Certificate Retrieval Failed 이벤트를 전송합니다. 이벤트에는 실패 이유가 포함됩니다.

다음은 이벤트에 대한 예제 데이터입니다.

```
{
    "version": "0",
    "id": "event_ID",
    "detail-type": "Certificate Authority Certificate Retrieval Failed",
    "source": "aws.pca-connector-scep",
    "account": "account",
    "time": "2024-09-12T19:14:56Z",
    "region": "region",
    "resources":[
        "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
        "arn:aws:pca-connector-scep:us-
east-1:111122223333:connector/11223344-1234-1122-2233-112233445566"
    ],
    "detail": {
        "result": "failure",
        "requestType": "GetCACert",
        "reason": "The certificate authority certificate validity must be at least one
 year from today."
    }
}
```

인증 기관 인증서 검색 성공

SCEP용 커넥터는 GetCACert 요청을 수신하고 커넥터의 프라이빗 CA 인증서를 성공적으로 검색할 때 EventBridge로 Certificate Authority Certificate Retrieval Succeeded 이벤트를 전송합니다.

다음은 이벤트에 대한 예제 데이터입니다.

```
{
   "version": "0",
   "id": "event_ID",
   "detail-type": "Certificate Authority Certificate Retrieval Succeeded",
   "source": "aws.pca-connector-scep",
   "account": "account",
   "time": "2024-09-12T19:14:56Z",
   "region": "region",
   "resources":[
       "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
       "arn:aws:pca-connector-scep:us-
east-1:111122223333:connector/11223344-1234-1122-2233-112233445566"
   ],
   "detail": {
       "result": "success",
       "requestType": "GetCACert"
   }
}
```

인증 기관 기능 검색 성공

SCEP용 커넥터는 SCEP `GetCACaps` 요청을 수신하고 CA의 기능을 성공적으로 검색할 때 EventBridge로 `Certificate Authority Capabilities Retrieval Succeeded` 이벤트를 전송합니다.

다음은 이벤트에 대한 예제 데이터입니다.

인증 기관 기능 검색 실패

SCEP용 커넥터는 SCEP `GetCACaps` 요청을 수신하고 CA의 기능을 검색할 수 없을 때 EventBridge로 `Certificate Authority Capabilities Retrieval Failed` 이벤트를 전송합니다. 이벤트에 실패 이유를 포함합니다.

다음은 이벤트에 대한 예제 데이터입니다.

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

```
     "arn:aws:pca-connector-scep:us-
east-1:111122223333:connector11223344-1234-1122-2233-112233445566"
     ],
 "detailType":"Certificate Authority Capabilities Retrieval Failed",
 "detail": {
     "result":"failure",
     "requestType":"GetCACaps",
     "reason":"The request was denied due to request throttling."
 },
 "source":"aws.pca-connector-scep","accountId":"111122223333"
 }
```

지원되지 않는 작업이 호출됨

지원되지 않는 작업이 호출됨

커넥터 엔드포인트로 전송된 작업이 지원되지 않거나 알 수 없는 경우 SCEP용 커넥터는 EventBridge로 `Unsupported Operation Invoked` 이벤트를 전송합니다.

```
{
   "version": "0",
   "id": "event_ID",
   "detail-type": "Unsupported Operation Invoked",
   "source": "aws.pca-connector-scep",
   "account": "account",
   "time": "2024-09-12T19:14:56Z",
   "region": "region",
   "resources":[
      "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
      "arn:aws:pca-connector-scep:us-
east-1:111122223333:connector/11223344-1234-1122-2233-112233445566"
   ],
   "detail": {}
}
```

EventBridge 규칙 생성

EventBridge에서는 CloudTrail에서 기록한 이벤트에 응답하는 규칙을 생성할 수 있습니다. SCEP용 커넥터에서 로깅한 모든 이벤트를 포함하는 규칙을 생성하려면 소스를 로 설정합니다`aws.pca-connector-scep`. 규칙에 대한 자세한 내용은 Amazon EventBridge에서 규칙 생성을 참조하세요.

를 사용한 SCEP API 호출용 로그 커넥터 AWS CloudTrail

SCEP(Simple Certificate Enrollment Protocol)용 커넥터는 사용자 AWS CloudTrail, 역할, 클라이언트 또는 서비스가 수행한 작업에 대한 레코드를 제공하는 AWS 서비스와 통합됩니다. CloudTrail은 SCEP용 커넥터에 대한 모든 API 호출을 이벤트로 캡처합니다. 캡처되는 호출에는 SCEP용 커넥터 콘솔의 호출과 SCEP용 커넥터 API 작업에 대한 코드 호출이 포함됩니다. 추적을 생성하면 SCEP용 커넥터 이벤트를 포함한 CloudTrail 이벤트를 지속적으로 Amazon S3 버킷에 배포할 수 있습니다. 추적을 구성하지 않은 경우에도 이벤트 기록에서 CloudTrail 콘솔의 최신 이벤트를 볼 수 있습니다. CloudTrail에서 수집한 정보를 사용하여 SCEP용 커넥터에 수행된 요청, 요청이 수행된 IP 주소, 요청을 수행한 사람, 요청이 수행된 시간 및 추가 세부 정보를 확인할 수 있습니다.

CloudTrail에 대한 자세한 내용은 AWS CloudTrail 사용 설명서를 참조하세요.

CloudTrail의 SCEP 정보용 커넥터

CloudTrail은 계정을 생성할 AWS 계정 때에서 활성화됩니다. SCEP용 커넥터에서 활동이 발생하면 해당 활동이 이벤트 기록의 다른 AWS 서비스 이벤트와 함께 CloudTrail 이벤트에 기록됩니다. 에서 최근 이벤트를 보고 검색하고 다운로드할 수 있습니다 AWS 계정. 자세한 내용은 CloudTrail 이벤트 기록을 사용하여 이벤트 보기를 참조하세요.

SCEP용 커넥터 이벤트를 AWS 계정포함하여에 이벤트를 지속적으로 기록하려면 추적을 생성합니다. CloudTrail은 추적을 사용하여 Amazon S3 버킷으로 로그 파일을 전송할 수 있습니다. 콘솔에서 트레일을 생성하면 기본적으로 모든 AWS 리전에 트레일이 적용됩니다. 추적은 AWS 파티션의 모든 리전에서 이벤트를 로깅하고 지정한 Amazon S3 버킷으로 로그 파일을 전송합니다. 또한 CloudTrail 로그에서 수집된 이벤트 데이터를 추가로 분석하고 조치를 취하도록 다른 AWS 서비스를 구성할 수 있습니다. 자세한 내용은 다음 자료를 참조하세요.

- 추적 생성 개요
- CloudTrail 지원 서비스 및 통합
- CloudTrail에 대한 Amazon SNS 알림 구성
- 여러 리전에서 CloudTrail 로그 파일 받기 및 여러 계정에서 CloudTrail 로그 파일 받기

모든 SCEP용 커넥터 작업은 CloudTrail에서 로깅되며 SCEP용 커넥터 API 참조에 문서화됩니다. 예를 들어 `CreateConnector`, `GetConnector`, `CreateChallenge` 작업을 직접 호출하면 CloudTrail 로그 파일에 항목이 생성됩니다.

모든 이벤트 또는 로그 항목에는 요청을 생성했던 사용자에 관한 정보가 포함됩니다. ID 정보를 이용하면 다음을 쉽게 판단할 수 있습니다.

- 요청이 루트 또는 AWS Identity and Access Management (IAM) 사용자 자격 증명으로 이루어졌는지 여부입니다.
- 역할 또는 페더레이션 사용자의 임시 자격 증명을 사용하여 요청이 생성되었는지 여부.
- 요청이 다른 AWS 서비스에서 이루어졌는지 여부입니다.
- SCEP 클라이언트 디바이스에서 요청했는지 여부입니다.

자세한 내용은 CloudTrail userIdentity 요소를 참조하세요.

SCEP 관리 이벤트용 커넥터

SCEP용 커넥터는 CloudTrail과 통합되어 SCEP용 커넥터에서 사용자, 역할 또는 AWS 서비스가 수행한 API 작업을 기록합니다. CloudTrail을 사용하여 SCEP API 요청에 대한 커넥터를 실시간으로 모니터링하고 Amazon Simple Storage Service, Amazon CloudWatch Logs 및 Amazon CloudWatch Events에 로그를 저장할 수 있습니다. SCEP용 커넥터는 다음 작업을 CloudTrail 로그 파일에 이벤트로 로깅하는 것을 지원합니다.

- CreateChallenge
- CreateConnector
- GetConnector
- GetChallengeMetadata
- GetChallengePassword
- DeleteConnector
- DeleteChallenge

CloudTrail의 SCEP 데이터 이벤트용 커넥터

데이터 이벤트는 클라이언트가 커넥터 엔드포인트에 SCEP `GetCACaps` 메시지를 보낼 때 리소스에서 또는 리소스에서 수행되는 리소스 작업에 대한 정보를 제공합니다. 이를 데이터 플레인 작업이라고도 합니다. 데이터 이벤트는 흔히 대량 활동입니다. 기본적으로 CloudTrail은 데이터 이벤트를 로깅하지 않으며 CloudTrail 이벤트 기록은 이를 기록하지 않습니다.

데이터 이벤트에는 추가 요금이 적용됩니다. CloudTrail 요금에 관한 자세한 내용은 AWS CloudTrail 요금을 참조하세요.

CloudTrail 콘솔 AWS CLI또는 CloudTrail API 작업을 사용하여 `AWS::PCAConnectorSCEP::Connector` 리소스 유형에 대한 데이터 이벤트를 로깅할 수 있습

니다. 데이터 이벤트를 로깅하는 방법에 관한 자세한 내용은 AWS CloudTrail 사용 설명서의 AWS Management Console을 사용한 데이터 이벤트 로깅 및 AWS Command Line Interface를 사용한 이벤트 데이터 로깅을 참조하세요.

다음 표에는 데이터 이벤트를 로깅할 수 있는 SCEP용 커넥터 리소스 유형이 나열되어 있습니다. 데이터 이벤트 유형(콘솔) 열에는 CloudTrail 콘솔의 데이터 이벤트 유형 목록에서 선택할 값이 표시됩니다. resources.type 값 열에는 AWS CLI 또는 CloudTrail APIs를 사용하여 고급 이벤트 선택기를 구성할 때 지정하는 `resources.type` 값이 표시됩니다. CloudTrail에 로깅되는 데이터 API 열에는 리소스 유형에 대해 CloudTrail에 로깅된 API 직접 호출이 표시됩니다.

데이터 이벤트 유형(콘솔)	resources.type 값	CloudTrail에 로깅되는 데이터 API
지원	`AWS::PCAConnectorSCEP::Connector`	• `PKIOperationGet` - PKCSReq 메시지가 포함된 HTTP GET SCEP 요청이 커넥터의 데이터플레인 엔드포인트에 수행되고 해당 메시지의 작업이 로 설정된 경우 생성됩니다`PKIOperation` . • `PKIOperationPost` - PKCSReq 메시지가 포함된 HTTP POST SCEP 요청이 커넥터의 데이터플레인 엔드포인트에 수행되고 해당 메시지의 작업이 로 설정된 경우 생성됩니다`PKIOperation` . • `GetCACaps` - `GetCACaps` 메시지가 포함된 SCEP 요청이 커넥터의 데이터플레인 엔드포인트에 이루어진 경우 생성됩니다. • `GetCACert` - `GetCACert` 메시지가 포함된 SCEP 요

데이터 이벤트 유형(콘솔)	resources.type 값	CloudTrail에 로깅되는 데이터 API
		청이 커넥터의 데이터 영역 엔드포인트에 이루어진 경우 생성됩니다.

eventName, readOnly 및 resources.ARN 필드를 필터링하여 중요한 이벤트만 로깅하도록 고급 이벤트 선택기를 구성할 수 있습니다. 다음 예제는 특정 함수에 대한 이벤트만 로깅하는 데이터 이벤트 구성의 JSON 뷰입니다. 이러한 필드에 대한 자세한 내용은 AWS CloudTrail API 참조의 AdvancedFieldSelector 섹션을 참조하세요.

```
[
  {
    "name": "connector-scep-events",
    "fieldSelectors": [
      {
        "field": "eventCategory",
        "equals": [
          "Data"
        ]
      },
      {
        "field": "resources.type",
        "equals": [
          "AWS::PCAConnectorSCEP::Connector"
        ]
      },
      {
        "field": "resources.ARN",
        "equals": [
          "arn:aws:pca-connector-scep:US West (N.
 California):111122223333:connector/11223344-1122-2233-3344-cae95a00d2a7"
        ]
      }
    ]
  }
]
```

예제 항목

트레일이란 지정한 S3 버킷에 이벤트를 로그 파일로 입력할 수 있게 하는 구성입니다. CloudTrail 로그 파일에는 하나 이상의 로그 항목이 포함될 수 있습니다. 이벤트는 모든 소스로부터의 단일 요청을 나타내며 요청 작업, 작업 날짜와 시간, 요청 파라미터 등에 대한 정보가 들어 있습니다. CloudTrail 로그 파일은 퍼블릭 API 직접 호출의 주문 스택 트레이스가 아니므로 특정 순서로 표시되지 않습니다.

예제 1: 관리 이벤트, **`CreateConnector`**

다음은 `CreateConnector` 작업을 설명하는 CloudTrail 로그 항목을 보여 주는 예시입니다.

```
{
        "eventVersion": "1.09",
        "userIdentity": {
          "type": "AssumedRole",
          "principalId": "AABB1122CCDD4455HHJJ1:11cc33nn2a97724dc48a89071111111111",
          "arn": "arn:aws:sts::111122223333:assumed-role/Admin",
          "accountId": "111122223333",
          "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
          "sessionContext": {
              "sessionIssuer": {
                  "type": "Role",
                  "principalId": "AABB1122CCDD4455HHJJ1",
                  "arn": "arn:aws:iam::111122223333:role/Admin",
                  "accountId": "111122223333",
                  "userName": "my-user-name"
              },
              "attributes": {
                  "creationDate": "2024-08-16T17:46:41Z",
                  "mfaAuthenticated": "false"
              }
          }
        },
        "eventTime": "2024-08-16T17:48:07Z",
        "eventSource": "pca-connector-scep.amazonaws.com",
        "eventName": "CreateConnector",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "10.0.0.0",
        "userAgent": "Python/3.11.8 Darwin/22.6.0 exe/x86_64",
        "requestParameters": {
          "ClientToken": "11223344-2222-3333-4444-666555444555",
          "CertificateAuthorityArn": "arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/a1b2c3d4-5678-90ab-cdef-EXAMPLE22222"
```

```
        },
        "responseElements": {
          "ConnectorArn": "arn:aws:pca-connector-scep:us-east-1:111122223333:connector/
a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
        },
        "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
        "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
        "readOnly": false,
        "eventType": "AwsApiCall",
        "managementEvent": true,
        "recipientAccountId": "111122223333",
        "eventCategory": "Management"
        }
```

예제 2: 관리 이벤트, **`CreateChallenge`**

다음은 `CreateChallenge` 작업을 설명하는 CloudTrail 로그 항목을 보여 주는 예시입니다.

```
{
        "eventVersion": "1.09",
        "userIdentity": {
          "type": "AssumedRole",
          "principalId": "AABB1122CCDD4455HHJJ1:11cc33nn2a97724dc48a89071111111111",
          "arn": "arn:aws:sts::111122223333:assumed-role/Admin",
          "accountId": "111122223333",
          "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
          "sessionContext": {
              "sessionIssuer": {
                  "type": "Role",
                  "principalId": "AABB1122CCDD4455HHJJ1",
                  "arn": "arn:aws:iam::111122223333:role/Admin",
                  "accountId": "111122223333",
                  "userName": "user-name"
              },
              "attributes": {
                  "creationDate": "2024-08-16T17:46:41Z",
                  "mfaAuthenticated": "false"
              }
          }
        },
        "eventTime": "2024-08-16T17:47:52Z",
        "eventSource": "pca-connector-scep.amazonaws.com",
        "eventName": "CreateChallenge",
        "awsRegion": "us-east-1",
```

```
        "sourceIPAddress": "10.0.0.0",
        "userAgent": "Python/3.11.8 Darwin/22.6.0 exe/x86_64",
        "requestParameters": {
          "ConnectorArn": "arn:aws:pca-connector-scep:us-east-1:111122223333:connector/
a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
          "ClientToken": "11223344-2222-3333-4444-666555444555"
        },
        "responseElements": {
          "Challenge": {
              "Arn": "arn:aws:pca-connector-scep:us-
east-1:111122223333:connector/9cac40bc-acba-412e-9a24-f255ef2fe79a/a1b2c3d4-5678-90ab-
cdef-EXAMPLE22222",
              "ConnectorArn": "arn:aws:pca-connector-scep:us-
east-1:111122223333:connector/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
              "CreatedAt": 1723830472.942,
              "Password": "***",
              "UpdatedAt": 1723830472.942
          }
        },
        "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
        "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
        "readOnly": false,
        "eventType": "AwsApiCall",
        "managementEvent": true,
        "recipientAccountId": "111122223333",
        "eventCategory": "Management"
        }
```


예제 3: 관리 이벤트, **GetChallengePassword**

다음은 GetChallengePassword 작업을 설명하는 CloudTrail 로그 항목을 보여 주는 예시입니다.


```
{
        "eventVersion": "1.09",
        "userIdentity": {
          "type": "AssumedRole",
          "principalId": "AABB1122CCDD4455HHJJ1:11cc33nn2a97724dc48a89071111111111",
          "arn": "arn:aws:sts::111122223333:assumed-role/Admin",
          "accountId": "111122223333",
          "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
          "sessionContext": {
              "sessionIssuer": {
                  "type": "Role",
                  "principalId": "AABB1122CCDD4455HHJJ1",
```

```
                "arn": "arn:aws:iam::111122223333:role/Admin",
                "accountId": "905418114790",
                "userName": "111122223333"
            },
            "attributes": {
                "creationDate": "2024-08-16T17:55:01Z",
                "mfaAuthenticated": "false"
            }
          },
          "invokedBy": "signin.amazonaws.com"
        },
        "eventTime": "2024-08-16T17:55:54Z",
        "eventSource": "pca-connector-scep.amazonaws.com",
        "eventName": "GetChallengePassword",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "10.0.0.0",
        "userAgent": "Python/3.11.8 Darwin/22.6.0 exe/x86_64",
        "requestParameters": {
          "ChallengeArn": "arn:aws:pca-connector-scep:us-east-1:111122223333:challenge/
a1b2c3d4-5678-90ab-cdef-EXAMPLE33333"
        },
        "responseElements": null,
        "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
        "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
        "readOnly": true,
        "eventType": "AwsApiCall",
        "managementEvent": true,
        "recipientAccountId": "111122223333",
        "eventCategory": "Management"
        }
```


예제 4: 데이터 이벤트, **`PkiOperationPost`**

다음 예제에서는 실패한 PkiOperationPost 호출을 보여주는 CloudTrail 로그 항목을 보여줍니다. 로그에는 오류 코드와 오류 메시지와 실패에 대한 설명이 포함되어 있습니다.


```
{
        "eventVersion": "1.10",
        "userIdentity": {
          "type": "FederatedUser",
          "principalId": "111122223333",
          "accountId": "111122223333"
        },
```

```
        "eventTime": "2024-08-16T17:40:09Z",
        "eventSource": "pca-connector-scep.amazonaws.com",
        "eventName": "PkiOperationPost",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "10.0.0.0",
        "userAgent": "Python/3.11.8 Darwin/22.6.0 exe/x86_64",
        "errorCode": "BadRequestException",
        "errorMessage": "The certificate authority is not in a valid state for issuing
 certificates (Service: AcmPca, Status Code: 400, Request ID: a1b2c3d4-5678-90ab-cdef-
EXAMPLE55555)",
        "requestParameters": null,
        "responseElements": null,
        "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
        "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
        "readOnly": false,
        "resources": [
          {
              "accountId": "111122223333",
              "type": "AWS::PCAConnectorSCEP::Connector",
              "ARN": "arn:aws:pca-connector-scep:us-east-1:111122223333:connector/
a1b2c3d4-5678-90ab-cdef-EXAMPLE33333"
          }
        ],
        "eventType": "AwsApiCall",
        "managementEvent": false,
        "recipientAccountId": "905418114790",
        "eventCategory": "Data",
        "tlsDetails": {
          "clientProvidedHostHeader": "111122223333-a1b2c3d4-5678-90ab-cdef-
EXAMPLE33333.enroll.pca-connector-scep.us-east-1.api.aws"
        }
        }
```


SCEP 문제에 대한 AWS Private Certificate Authority 커넥터 문제 해결

SCEP용 커넥터 구현과 관련된 문제를 해결해야 할 수 있습니다. 이 장에서는 서비스에서 전송한 HTTP 및 클라이언트 오류에 대한 자세한 정보를 제공합니다.

주제

- SCEP용 커넥터의 HTTP 오류 문제 해결

- SCEP 클라이언트 오류에 대한 커넥터 문제 해결

SCEP용 커넥터의 HTTP 오류 문제 해결

클라이언트가 SCEP용 커넥터 데이터 영역 API 작업을 트리거하고 오류가 발생하면 SCEP용 커넥터는 오류에 대한 정보와 함께 HTTP 응답 코드를 요청 클라이언트에 전송합니다.

클라이언트에 직접 제공되는 서비스 응답 외에도 SCEP용 모니터 커넥터 섹션에 설명된 모니터링 도구를 사용하여 HTTP 오류를 초래하는 오류를 보고 디버깅할 수 있습니다.

다음은 서비스에서 SCEP 클라이언트에 반환하는 오류 메시지, 잠재적 원인 및 문제 해결을 위해 취할 수 있는 단계입니다.

HTTP 400 잘못된 요청

HTTP 400 응답 코드는 요청에 누락되거나 유효하지 않은 데이터와 같은 명백한 클라이언트 오류로 인해 SCEP용 커넥터가 요청을 처리할 수 없음을 의미합니다. SCEP 프로토콜별 오류로 인해 오류가 발생하는 경우 SCEP용 커넥터에는 메시지에 SCEP 응답이 바이너리로 포함됩니다. SCEP APIs용 커넥터는 다음과 같은 이유로 400개의 응답을 반환할 수 있습니다.

응답 헤더 (x-amzn-ErrorType)	오류 메시지(x-amzn-ErrorMessage)	근본 원인:	문제 해결	SCEP 응답을 포함하나요?
LimitExceededException	인증 기관 발급 한도를 초과했습니다.	커넥터와 연결된 사설 인증 기관(CA)이 발급할 수 있는 인증서 수에 대한 할당량을 초과했습니다.	SCEP 커넥터는 수명 동안 하나의 프라이빗 CA에만 연결할 수 있습니다. 프라이빗 CA의 한도를 모두 사용한 경우 새 커넥터를 생성하거나 할당량 증가를 요청합니다. 프라이빗 CA 할당량에 대한 자세한 내용은 AWS Private Certifica	아니요

응답 헤더(x-amzn-ErrorType)	오류 메시지(x-amzn-ErrorMessage)	근본 원인:	문제 해결	SCEP 응답을 포함하나요?
			te Authority 할당량을 참조하세요.	
ValidationException	요청에 base64가 포함되어야 합니다.	본문이 유효한 Base64가 아니므로 SCEP용 커넥터가 HTTP GET 요청을 처리할 수 없습니다.	가능하면 HTTP GET 메시지 대신 HTTP POST 메시지를 사용하도록 클라이언트를 구성합니다. HTTP GET을 사용해야 하는 경우 메시지는 Base64 형식을 사용해야 합니다. 클라이언트가 이러한 요구 사항과 호환되지 않는 경우 AWS Support에 문의하여 지원을 받으세요.	아니요
ValidationException	인증 기관이 활성 상태가 아닙니다.	커넥터와 연결된 프라이빗 CA가 비활성 상태입니다.	프라이빗 CA를 다시 활성화합니다. 자세한 내용은 에서 프라이빗 CA 업데이트 AWS Private Certificate Authority 단원을 참조하세요.	아니요

응답 헤더(x-amzn-ErrorType)	오류 메시지(x-amzn-ErrorMessage)	근본 원인:	문제 해결	SCEP 응답을 포함하나요?
ValidationException	인증 기관 인증서 유효 기간은 오늘부터 1년 이상이어야 합니다.	범용 커넥터와 연결된 프라이빗 CA의 유효 기간은 오늘부터 1년 이어야 합니다.	오늘부터 1년 이상의 유효 기간이 있는 인증서를 다시 발급합니다. 인증서 관리에 대한 자세한 내용은 섹션을 참조하세요프라이빗 CA 수명 주기 관리.	아니요
ValidationException	요청에 포함된 인증서가 만료되었습니다.	각 트랜잭션에서 클라이언트 디바이스가 생성한 임시 인증서가 서비스 수신 시 만료되었습니다.	클라이언트 디바이스에 시간 설정이 제대로 구성되지 않아 날짜가 실시간보다 뒤쳐진 인증서를 생성하고 있을 가능성이 높습니다. 이 문제를 해결할 수 없는 경우 AWS Support에 문의하여 도움을 받으세요.	아니요
ValidationException	요청에 잘못된 암호화 메시지 구문이 포함되어 있습니다.	서비스가 SCEP 요청 메시지를 디코딩할 수 없습니다.	SCEP 메시지가 SCEP RFC 8894에 정의된 암호화 메시지 구문을 준수하는지 확인합니다. 이 문제를 해결할 수 없는 경우 AWS Support에 문의하여 도움을 받으세요.	아니요

응답 헤더(x-amzn-ErrorType)	오류 메시지(x-amzn-ErrorMessage)	근본 원인:	문제 해결	SCEP 응답을 포함하나요?
ValidationException	커넥터가 활성 상태가 아닙니다.	커넥터의 상태가 활성 상태가 아닙니다.	콘솔 또는 API의 상태 필드에서 커넥터의 상태를 찾을 수 있습니다. 커넥터의 상태는 생성, 활성, 삭제 또는 실패일 수 있습니다. 상태가 생성 중이면 나중에 요청을 시도하세요. 상태가 실패하면 상태 이유를 보고 문제를 해결한 다음 새 커넥터를 생성합니다.	아니요
ValidationException	요청에 유효한 인증서가 포함되어 있어야 합니다.	클라이언트의 요청 메시지에 포함된 임시 인증서가 누락되었거나 유효하지 않습니다.	SCEP 호환 클라이언트는 자체 인증을 위해 자체 서명된 인증서를 제공해야 합니다. 클라이언트가 필요한 자체 서명 인증서를 제공할 수 없는 경우 AWS Support에 문의하여 지원을 받으세요.	아니요

응답 헤더(x-amzn-ErrorType)	오류 메시지(x-amzn-ErrorMessage)	근본 원인:	문제 해결	SCEP 응답을 포함하나요?
ValidationException	요청 URI가 잘못되었습니다.	요청의 URI 경로 또는 쿼리가 유효하지 않아 SCEP용 커넥터가 요청을 구문 분석할 수 없습니다.	관리자는 일반적으로 모바일 디바이스 관리(MDM) 시스템을 통해 관리되는 클라이언트 디바이스의 구성 설정을 확인해야 합니다. 자세한 내용은 2단계: MDM 시스템에 커넥터 세부 정보 복사 단원을 참조하십시오.	아니요
ValidationException	요청에 정확히 하나의 호스트 헤더가 필요합니다.	클라이언트가 요청에 유효한 HTTP 호스트 헤더를 제공하지 않았습니다.이 헤더는 요청을 처리하는 데 필요합니다.	HTTP 호스트 헤더는 서로 다른 커넥터로 들어오는 요청을 구분하는 데 필요합니다. 클라이언트가 필요한 HTTP 호스트 헤더를 제공할 수 없는 경우 AWS Support에 문의하여 지원을 받으세요.	아니요
ValidationException	요청을 디코딩할 수 없습니다. 유효한 SCEP 요청을 전송하십시오.	서비스가 클라이언트가 보낸 암호화 메시지 구문(CMS) 요청을 디코딩하고 처리할 수 없습니다.	클라이언트가 SCEP 구현에 문제가 있는 경우 응답의 요청 ID(`x-amzn-requestid`)를 기록하고에 문의하세요AWS Support.	아니요

응답 헤더(x-amzn-ErrorType)	오류 메시지(x-amzn-ErrorMessage)	근본 원인:	문제 해결	SCEP 응답을 포함하나요?
ValidationException	요청에서 파생된 값으로 응답을 인코딩할 수 없습니다. 유효한 SCEP 요청을 전송하십시오.	서비스가 SCEP 응답을 인코딩할 수 없습니다.	이 문제는 일반적으로 서비스가 제공된 요청자 인증서를 사용하여 SCEP 응답 메시지를 올바르게 인코딩할 수 없는 경우에 발생합니다. 예를 들어 요청자 인증서에 SCEP용 커넥터가 지원하지 않는 타원 곡선 디지털 서명 알고리즘(ECDSA) 키가 있는 경우이 문제가 발생할 수 있습니다. 이 문제가 발생하면 먼저 RSA를 사용하도록 MDM 또는 SCEP 클라이언트를 구성합니다. 그래도 문제를 해결할 수 없는 경우 응답의 요청 ID(`x-amzn-requestid`)를 기록하고 AWS Support에 문의하여 도움을 받으세요.	아니요

응답 헤더 (x-amzn-ErrorType)	오류 메시지(x-amzn-ErrorMessage)	근본 원인:	문제 해결	SCEP 응답을 포함하나요?
ValidationException	지원되지 않는 알고리즘: <OID>	요청이 지원되지 않는 암호화 알고리즘에 의해 서명되거나 암호화되었습니다.	당사의 서비스는 특정 오래된 암호화 알고리즘과 취약한 암호화 알고리즘을 지원하지 않습니다. 이 정보는 `GetCACaps` 요청을 통해 클라이언트에 전달됩니다. 그러나 일부 클라이언트는이 방법을 사용하여 지원되는 알고리즘을 확인하지 않을 수 있습니다. 클라이언트가 서비스에서 지원하는 암호화 알고리즘과 호환되지 않는 것으로 보이는 경우 AWS Support에 문의하여 지원을 받으세요.	아니요

응답 헤더(x-amzn-ErrorType)	오류 메시지(x-amzn-ErrorMessage)	근본 원인:	문제 해결	SCEP 응답을 포함하나요?
ValidationException	지원되지 않는 PkiOperation messageType입니다.	요청 메시지에 잘못된 `PkiOperation` 메시지 유형이 포함되어 있어 서비스에서 처리할 수 없습니다.	당사의 서비스는 RFC 8894에 정의된 SCEP 프로토콜 메시지 유형의 하위 집합만 지원합니다. 특히 CertRep, PKCSReq, GetCert, GetCRL 및 CertPoll 메시지 유형을 인식하고 처리합니다. GetCACaps 메서드를 통해 지원되는 메시지 유형을 클라이언트에 전달합니다. 안타깝게도 일부 클라이언트는 이 방법을 사용하지 않을 수 있으며 서비스의 기능을 준수하지 않을 수 있습니다. 클라이언트가 서비스에서 지원하는 SCEP 메시지 유형과 호환되지 않는 것으로 보이는 경우에 문의하세요AWS Support.	아니요

응답 헤더 (x-amzn-ErrorType)	오류 메시지(x-amzn-ErrorMessage)	근본 원인:	문제 해결	SCEP 응답을 포함하나요?
BadRequestException	챌린지 암호가 잘못되었습니다.	클라이언트가 제공한 챌린지 암호가 연락한 서비스 엔드포인트 및 연결된 커넥터에 유효하지 않았습니다. 챌린지 암호는 권한이 있는 클라이언트만 서비스에 액세스할 수 있도록 SCEP 프로토콜에 정의된 필수 보안 조치입니다.	클라이언트가 요청에 올바른 챌린지 암호를 제공하고 있는지 확인합니다. 콘솔의 커넥터 세부 정보 또는 GetChallengePassword API를 통해 찾을 수 있습니다. 자세한 내용은 2단계: MDM 시스템에 커넥터 세부 정보 복사 단원을 참조하십시오.	예
BadRequestException	인증서 서명 요청에 정확히 하나의 챌린지 암호가 필요합니다.	클라이언트는 요청에 0개 또는 여러 개의 챌린지 암호를 제공했습니다.	클라이언트가 요청에 하나의 챌린지 암호를 제공하고 있는지 확인합니다. 콘솔의 커넥터 세부 정보 또는 GetChallengePassword API를 통해 챌린지 암호를 찾을 수 있습니다. 자세한 내용은 2단계: MDM 시스템에 커넥터 세부 정보 복사 단원을 참조하십시오.	예

응답 헤더(x-amzn-ErrorType)	오류 메시지(x-amzn-ErrorMessage)	근본 원인:	문제 해결	SCEP 응답을 포함하나요?
BadRequestException	커넥터는 Azure에 액세스할 수 없습니다.	Microsoft Intune용 커넥터는 Microsoft Intune을 통해 클라이언트 요청을 승인합니다. 이를 위해서는 SCEP용 커넥터가 Azure 리소스에 액세스할 수 있는 권한을 부여해야 합니다.	에 자세히 설명된 권한을 구성합니다1단계: Microsoft Entra ID 애플리케이션을 사용할 수 있는 AWS Private CA 권한 부여.	예
BadRequestException	Azure 애플리케이션은 <action>을 수행할 수 있는 액세스 권한이 없습니다.	Microsoft Intune용 커넥터는 Microsoft Intune을 통해 클라이언트 요청을 승인합니다. 이를 위해서는 SCEP용 커넥터가 Azure 리소스에 액세스할 수 있는 권한을 부여해야 합니다.	에 자세히 설명된 권한을 구성합니다1단계: Microsoft Entra ID 애플리케이션을 사용할 수 있는 AWS Private CA 권한 부여.	예
BadRequestException	Azure 애플리케이션을 찾을 수 없습니다.	Microsoft Intune용 커넥터는 Microsoft Intune을 통해 클라이언트 요청을 승인합니다. 이 오류는 Microsoft Entra ID에 앱 등록이 없거나 커넥터의 Intune 세부 정보가 잘못 구성되었음을 나타냅니다.	SCEP용 커넥터용 Microsoft Intune 구성 주제의 지침을 따릅니다.	예

응답 헤더(x-amzn-ErrorType)	오류 메시지(x-amzn-ErrorMessage)	근본 원인:	문제 해결	SCEP 응답을 포함하나요?
BadRequestException	Intune 인증서 서명 요청 검증에 실패했습니다. 이유: <이유>	Microsoft Intune용 커넥터는 Microsoft Intune을 통해 클라이언트 요청을 승인합니다. 이 오류 메시지는 Intune 검증 프로세스가 실패했으며 해당 Intune 오류 코드가 제공되었음을 나타냅니다.	SCEP용 커넥터용 Microsoft Intune 구성 주제의 지침을 따릅니다. 문제가 지속되면 Microsoft Support에 문의하십시오.	예

응답 헤더 (x-amzn-ErrorType)	오류 메시지(x-amzn-ErrorMessage)	근본 원인:	문제 해결	SCEP 응답을 포함하나요?
BadRequestException	지원되지 않는 PkiOperation messageType: <message type>.	요청 메시지에 잘못된 메시지 유형이 포함되어 있어 서비스에서 처리할 수 없습니다.	당사의 서비스는 RFC 8894에 정의된 SCEP 프로토콜 메시지 유형의 하위 집합만 지원합니다. 특히 CertRep, PKCSReq, GetCert, GetCRL 및 CertPoll 메시지 유형을 인식하고 처리합니다. GetCACaps 메서드를 통해 지원되는 메시지 유형을 클라이언트에 전달합니다. 안타깝게도 일부 클라이언트는 이 방법을 사용하지 않을 수 있으며 서비스의 기능을 준수하지 않을 수 있습니다. 클라이언트가 서비스에서 지원하는 SCEP 메시지 유형과 호환되지 않는 것으로 보이는 경우에 문의하세요AWS Support.	예

응답 헤더 (x-amzn-ErrorType)	오류 메시지(x-amzn-ErrorMessage)	근본 원인:	문제 해결	SCEP 응답을 포함하나요?
BadRequestException	키 알고리즘 또는 길이는 지원되지 않습니다.	이 서비스는 인증서 서명 요청에 포함된 제공된 퍼블릭 키를 지원하지 않습니다.	서비스는 최대 16,384비트의 표준 RSA 키와 최대 521비트의 ECDSA 키만 지원합니다. 클라이언트가 현재 지원되지 않는 알고리즘을 사용해야 하는 경우 AWS Support에 문의하여 도움을 받으세요.	예

HTTP 401 무단

401 승인되지 않음 응답 상태 코드는 요청된 리소스에 대한 유효한 인증 자격 증명이 없기 때문에 클라이언트 요청이 완료되지 않았음을 나타냅니다.

응답 헤더 (x-amzn-ErrorType)	오류 메시지(x-amzn-ErrorMessage)	근본 원인:	문제 해결	SCEP 응답을 포함하나요?
AccessDeniedException	커넥터는 인증 기관에 액세스할 수 없습니다.	SCEP용 커넥터는 커넥터의 연결된 프라이빗 CA에 액세스할 수 없습니다.	를 사용하여 SCEP용 커넥터와 프라이빗 CA를 공유합니다 AWS Resource Access Manager.	아니요
AccountDoesNotExis	AWS 계정이 존재하지 않습니다.	SCEP 리소스용 커넥터가 더 이상 존재하지 않습니다.	대상 리소스를 소유한 계정이 삭제되었습니다. 실수로이 작업을	아니요

응답 헤더(x-amzn-ErrorType)	오류 메시지(x-amzn-ErrorMessage)	근본 원인:	문제 해결	SCEP 응답을 포함하나요?
tException			수행한 경우 해지 후 90일 AWS Support 이내에에 문의하세요.	

HTTP 404를 찾을 수 없음

HTTP 404 응답 코드는 일반적으로 찾고 있던 리소스를 찾을 수 없음을 의미합니다.

응답 헤더(x-amzn-ErrorType	오류 메시지(x-amzn-ErrorMessage)	근본 원인:	문제 해결	SCEP 응답을 포함하나요?
ResourceNotFoundException	인증 기관이 존재하지 않습니다.	커넥터의 연결된 프라이빗 CA가 삭제되었습니다.	실수로 삭제된 경우 프라이빗 인증 기관(CA)을 복원할 수 있는 유예 기간이 있습니다. 자세한 내용은 프라이빗 CA 복원 단원을 참조하십시오.	아니요
ResourceNotFoundException	엔드포인트가 <URL>인 커넥터가 존재하지 않습니다.	클라이언트 디바이스가 기존 커넥터에 속하지 않는 URL에 연결을 시도했습니다.	클라이언트가 커넥터에 올바른 엔드포인트를 제공하는지 확인합니다. 커넥터의를 보려면 GetConnector API를 `Endpoint`호출하거나 콘솔의 커넥터 세부 정보 페이지에서 확인합니다.	아니요

HTTP 409 충돌

HTTP 409 충돌 응답은 요청이 시작된 이후 커넥터와 연결된 프라이빗 CA가 변경되었음을 알립니다.

응답 헤더(x-amzn-ErrorType)	오류 메시지(x-amzn-ErrorMessage)	근본 원인:	문제 해결	SCEP 응답을 포함하나요?
ConflictException	요청이 시작된 이후 커넥터가 변경되었습니다.	커넥터와 연결된 프라이빗 CA가 업데이트되어 SCEP를 통해 클라이언트 디바이스와 통신하는 데 사용되는 커넥터의 내부 인증서가 교체됩니다. 이 인증서 교체로 인해 새 인증서가 배포 중이므로 업데이트 기간 동안 일시적인 문제가 발생할 수 있습니다. 그러나 이 오류는 적시에 자동으로 해결되어야 합니다.	몇 분 후에 요청을 다시 시도하세요. 문제가 해결되지 않으면 AWS Support에 문의하여 도움을 받으세요.	아니요

HTTP 429 요청 과다

SCEP용 커넥터에는 리전별로 계정 수준 할당량이 있습니다. 커넥터에 대한 요청 한도를 초과하면 HTTP 429 오류와 함께 요청이 거부됩니다. 할당량을 늘려야 하는 경우 AWS Private Certificate Authority 엔드포인트 및 할당량을 참조하세요.

응답 헤더(x-amzn-ErrorType)	오류 메시지(x-amzn-ErrorMessage)	근본 원인:	문제 해결	SCEP 응답을 포함하나요?
ThrottlingException	요청 스로틀링으로 인해 요청이 거부되었습니다.	이 커넥터에 너무 많은 요청이 실행되어 일부 요청이 거부됩니다. 이 인증서 교체로 인해 새 인증서가 배포 중이므로 업데이트 기간 동안 일시적인 문제가 발생할 수 있습니다. 그러나이 오류는 적시에 자동으로 해결되어야 합니다.	커넥터에 대한 요청 한도를 초과하면 요청이 거부됩니다. 할당량을 늘려야 하는 경우 SCEP 엔드포인트 및 할당량용 커넥터를 참조하세요.	아니요

SCEP 클라이언트 오류에 대한 커넥터 문제 해결

다음 지침을 사용하여 SCEP용 커넥터와 관련된 클라이언트 오류를 해결합니다.

메시지 예제	근본 원인:	Solution
ECDSA 키는 지원되지 않습니다.	커넥터는 RSA 대신 ECDSA 키를 사용하는 프라이빗 CA에 연결됩니다. 이 서비스는 ECDSA 키를 지원하지만 일부 클라이언트 디바이스는이 알고리즘과 호환되지 않을 수 있습니다.	ECDSA 대신 RSA로 암호화된 프라이빗 CA를 사용하는 것이 좋습니다. RSA를 사용하는 프라이빗 CA를 생성하는 경우 새 커넥터도 생성해야 합니다. 커넥터는 수명을 통해 하나의 프라이빗 CA에만 연결할 수 있습니다.
암호화 또는 서명 인증서가 없음	RFC 8894에 따르면 SCEP 서비스는 중간 CA 인증서를 클라이언트에 반환합니다. 이러한 인증서는 클라이언트	호환되는 클라이언트를 사용할 수 없는 경우 AWS Support에 문의하여 지원을 받으세요.

메시지 예제	근본 원인:	Solution
	가 SCEP 프로토콜의 일부로 암호화 및 서명 검증 작업을 수행하는 데 사용됩니다. SCEP용 커넥터는 암호화 및 서명 검증 모두에 동일한 인증서를 사용하며, 이는 일반적인 접근 방식입니다. 그러나 일부 클라이언트는 대신 두 개의 개별 인증서를 가질 것으로 예상할 수 있습니다.	

AWS Private CA 서비스 할당량

AWS Private CA 는 허용된 수의 인증서 및 인증 기관에 할당량을 할당합니다. API 작업에 대한 요청 요금에는 quotas. AWS Private CA quotas가 적용됩니다. 계정 AWS 및 리전에 따라 다릅니다.

AWS Private CA 는 API 작업에 따라 다른 속도로 API 요청을 제한합니다. 제한은 요청이 초당 요청 수에 대한 작업의 할당량을 초과하기 때문에가 유효한 요청을 AWS Private CA 거부한다는 의미입니다. 요청이 제한되면는 ThrottlingException 오류를 AWS Private CA 반환합니다.는 APIs에 대한 최소 요청 속도를 보장하지 AWS Private CA 않습니다.

조정할 수 있는 할당량을 확인하려면의 AWS Private CA 할당량 표를 참조하세요AWS 일반 참조.

AWS Service Quotas을 사용하여 현재 할당량을 확인하고 할당량 증가를 요청할 수 있습니다.

AWS Private CA 할당량의 up-to-date 목록을 보려면

1. AWS 계정에 로그인합니다.
2. https://console.aws.amazon.com/servicequotas/에서 Service Quotas 콘솔을 엽니다.
3. 서비스 목록에서 AWS Certificate Manager 프라이빗 인증 기관(ACM PCA)을 선택합니다. 서비스 할당량 목록의 각 할당량에는 현재 적용된 할당량 값, 기본 할당량 값, 할당량 조정 가능 여부가 표시됩니다. 할당량 이름을 선택하면 할당량에 대한 자세한 정보를 볼 수 있습니다.

할당량 증가 요청

1. 서비스 할당량 목록에서 조정 가능한 할당량을 위한 라디오 버튼을 선택합니다.
2. 할당량 증가 요청 버튼을 선택합니다.
3. 할당량 증가 요청 양식을 작성하여 제출합니다.

AWS Private CA 는와 통합됩니다 AWS Certificate Manager. ACM 콘솔 AWS CLI또는 ACM API를 사용하여 기존 프라이빗 CA에서 프라이빗 인증서를 요청할 수 있습니다. ACM에서 관리하는 이러한 프라이빗 PKI 인증서에는 PCA 할당량과 ACM이 공개 및 가져온 인증서에 적용하는 할당량이 모두 적용됩니다. ACM 요구 사항에 대한 자세한 내용은 AWS Certificate Manager 사용 설명서의 프라이빗 인증서 및 할당량 요청을 참조하세요. https://docs.aws.amazon.com/acm/latest/userguide/acm-limits.html

문서 기록

다음 표에서는 2018년 1월 이후 이 설명서의 중요한 변경 사항을 설명합니다. Amazon은 여기 나와 있는 주요 변경 사항 외에도 설명과 예제를 업데이트하고 고객이 제공한 피드백을 반영하도록 설명서를 자주 업데이트하고 있습니다. 중요한 변경 사항에 대해 알림을 받으려면 오른쪽 상단 모서리의 링크를 사용하여 RSS 피드를 구독합니다.

변경 사항	설명	날짜
문서 업데이트	새로운 시작 절차, 예제, 모니터링 및 문제 해결 주제로 에서 보안 Kubernetes AWS Private Certificate Authority를 업데이트했습니다.	2025년 10월 1일
듀얼 스택 지원	AWS Private Certificate Authority 는 듀얼 스택을 지원합니다.	2025년 6월 23일
이제 AD용 커넥터에 대한 하위 도메인 지원을 정식으로 사용할 수 있습니다.	이제 하위 도메인을 사용하여 AD용 커넥터를 설정할 수 있습니다.	2025년 6월 2일
새로운 관리형 정책: AWSPrivateCAConnectoForKubernetesPolicy	Kubernetes용 AWS Private CA 커넥터와 함께 사용할 수 있도록 도입된 새로운 관리형 정책입니다.	2025년 5월 19일
업데이트된 정책 AWSPrivateCAPrivilegedUser 및 AWSPrivateCAUser 관리형 정책	AWSPrivateCAUser 및 ArnLike에서 StringLike 로 대체되었습니다AWSPrivateCAPrivilegedUser . 와일드카드를 포함하도록 템플릿 ARN을 arn:aws:acm-pca:::template 로 업데이트했습	2025년 1월 22일

	니다`arn:aws:acm-pca:*:*:template` .	
이제 SCEP용 커넥터를 정식으로 사용할 수 있습니다.	이제 SCEP용 커넥터를 정식 버전으로 사용할 수 있습니다.	2024년 9월 16일
새로운 문제 해결 주제	Active Directory용 커넥터 템플릿 업데이트와 관련된 문제를 해결하는 데 도움이 되는 새 주제가 추가되었습니다.	2024년 7월 31일
AD 템플릿용 커넥터 업데이트 방법 추가	AD용 커넥터 템플릿을 업데이트하는 방법과 AWS Private CA 에서 이러한 업데이트를 전파하는 방법을 설명하는 절차가 추가되었습니다.	2024년 7월 31일
이제 SCEP for Omnissa Workspace ONE용 커넥터를 정식 버전으로 사용할 수 있습니다.	이제 SCEP for Omnissa Workspace ONE용 커넥터를 정식 버전으로 사용할 수 있습니다.	2024년 7월 23일
감사 보고서에 대한 제약 조건 추가	AWS Private CA 는 감사 보고서에 사용되는 버킷에서 Amazon S3 객체 잠금 사용을 지원하지 않습니다.	2024년 7월 3일
이제 중국 리전에서 SM2 지원	AWS Private CA 이제는 중국 리전에서만 SM2 서명 알고리즘을 지원합니다.	2024년 6월 27일
AWS Private CA 는 이제 SCEP용 커넥터(미리 보기)를 지원합니다.	SCEP용 커넥터를 사용하여 SCEP 지원 클라이언트 및 디바이스 AWS Private CA 에 연결합니다.	2024년 6월 11일
새로운 커넥터 문제 해결 지침	커넥터 문제 해결 및 SPN 생성 실패에 대한 새 단원이 추가되었습니다.	2024년 4월 4일

Matter에 대한 CDP 확장 추가	Matter에 대한 인증서 해지 목록 배포 지점(CDP) 확장에 대한 지원을 추가합니다.	2024년 1월 25일
AWS Private CA mDL에 대한 API 지원	모바일 운전면허증(mDL)에 대한 ISO/IEC 표준을 준수하는 인증서를 생성하기 위한 API 지원이 추가되었습니다.	2024년 1월 16일
AWS Private CA Active Directory용 커넥터	사용 설명서, API 및 CLI는 AD용 커넥터를 지원합니다. 자세한 내용은 AD용 커넥터 설명서를 참조하세요.	2023년 8월 24일
새 서비스 이름과 일치하도록 보안 정책 이름 변경	표준 권한을 지정하는 AWS 관리형 IAM 정책의 새 이름 채택 AWS Private CA. 자세한 내용은 AWS 관리형 정책 섹션을 참조하세요.	2023년 2월 13일
AWS 관리형 정책에 대한 변경 트래커 추가	표준 권한을 지정하는 AWS 관리형 IAM 정책의 변경 사항을 추적하기 위해 설명서가 추가되었습니다 AWS Private CA. 자세한 내용은 AWS 관리형 정책에 대한 업데이트 AWS Private CA를 참조하세요.	2022년 11월 11일
수명이 짧은 인증서를 발급하는 CAs 대한 API 및 CLI 지원	CA 사용 모드가 도입되면 범용 인증서 또는 단기 전용 인증서를 발급하도록 CA를 구성할 수 있습니다. 자세한 내용은 인증 기관 모드를 참조하세요.	2022년 10월 24일

서비스 리브랜딩 및 콘솔 업데이트	서비스의 이름이 AWS Private Certificate Authority (AWS Private CA)로 변경됩니다. AWS Private CA 콘솔은 전체 설명서로 연결되는 통합 도움말 패널을 포함하여 사용성을 개선했습니다.	2022년 9월 27일
Matter 준수 인증서 지원	세 가지 새로운 인증서 템플릿은 Matter 준수 CA 및 최종 엔터티 인증서에 대한 지원을 추가합니다. 자세한 내용은 인증서 템플릿 이해를 참조하세요.	2022년 7월 20일
새로운 리전 지원	아시아 태평양(자카르타)의 엔드포인트가 추가되었습니다. AWS Private CA 엔드포인트의 전체 목록은 ACM Private Certificate Authority 엔드포인트 및 할당량을 참조하세요.	2022년 5월 4일
사용자 지정 특성 및 확장 지원	사용자 지정 CA 및 인증서를 구성하려면 CustomAttribute 객체를 사용하고 사용자 지정 인증서를 구성하려면 CustomExtension 객체를 사용합니다.	2022년 3월 16일
관리형 OCSP 지원	OCSP를 포함한 해지 옵션에 대한 인증서 해지 방법 설정을 참조하세요.	2021년 8월 18일
CRL용 S3 퍼블릭 액세스 차단 기능 지원	S3 퍼블릭 액세스 차단 기능 활성화를 참조하세요.	2021년 5월 27일
신규 및 업데이트된 Java 구현 예제	ACM 프라이빗 CA API 사용(Java 예제)을 참조하세요.	2020년 9월 9일

새로운 리전 지원	아프리카(케이프타운) 및 유럽(밀라노)에 대한 엔드포인트가 추가되었습니다. AWS Private CA 엔드포인트의 전체 목록은 AWS Certificate Manager 프라이빗 인증 기관 엔드포인트 및 할당량을 참조하세요.	2020년 8월 27일
교차 계정 프라이빗 CA 액세스 지원	AWS Certificate Manager 사용자는 자신이 소유하지 않은 프라이빗 CAs 사용하여 인증서를 발급할 수 있는 권한이 부여될 수 있습니다. 자세한 내용은 프라이빗 CA에 대한 교차 계정 액세스를 참조하세요.	2020년 8월 17일
VPC 엔드포인트(PrivateLink) 지원	네트워크 보안을 강화하기 위해 VPC 엔드포인트(AWS PrivateLink) 사용에 대한 지원이 추가되었습니다. 자세한 내용은 ACM Private CA VPC 엔드포인트(AWS PrivateLink)를 참조하세요.	2020년 3월 26일
전용 보안 섹션 추가	에 대한 보안 설명서 AWS 가 전용 보안 섹션으로 통합되었습니다. 보안에 대한 자세한 내용은 AWS Certificate Manager 프라이빗 인증 기관의 보안을 참조하세요.	2020년 3월 26일
템플릿 ARN이 감사 보고서에 추가되었습니다.	자세한 내용은 사설 CA에 대한 감사 보고서 생성을 참조하십시오.	2020년 3월 6일

CloudFormation 지원	에 대한 지원이 추가되었습니다 CloudFormation. 자세한 내용은 CloudFormation 사용 설명서의 ACMPCA 리소스 유형 참조를 참조하세요.	2020년 1월 22일
CloudWatch 이벤트 통합	CA 생성, 인증서 발급 및 CRL 생성을 포함한 비동기식 이벤트에 있어 CloudWatch 이벤트와 통합되었습니다. 자세한 내용은 CloudWatch 이벤트 사용을 참조하십시오.	2019년 12월 23일
FIPS 엔드포인트	FIPS 엔드포인트에 for AWS GovCloud(미국 동부) 및 AWS GovCloud(미국 서부)가 추가되었습니다. AWS Private CA 엔드포인트의 전체 목록은 AWS Certificate Manager 프라이빗 인증 기관 엔드포인트 및 할당량을 참조하세요.	2019년 12월 13일
태그 기반 권한	새 API인 `TagResource` , `UntagResource` 및 `ListTagsForResource` 를 사용한 태그 기반 권한이 지원됩니다. 태그 기반 제어에 대한 일반적인 내용은 IAM 리소스 태그를 사용하여 IAM 사용자 및 역할에 대한 액세스 제어를 참조하십시오.	2019년 11월 5일

이름 제약 조건 적용	가져온 CA 인증서에 보안 주체 이름 제약 조건을 적용하기 위한 지원이 추가되었습니다. 자세한 내용은 사설 CA에 이름 제약 조건 적용을 참조하십시오.	2019년 10월 28일
새 인증서 템플릿	를 사용한 코드 서명용 템플릿을 포함하여 새 인증서 템플릿이 추가되었습니다 AWS Signer. 자세한 내용은 템플릿 사용을 참조하십시오.	2019년 10월 1일
CA 계획	AWS Private CA를 사용한 PKI 계획에 대한 새로운 섹션이 추가되었습니다 . 자세한 내용은 ACM 사설 CA 배포 계획을 참조하십시오.	2019년 9월 30일
리전 지원 추가	AWS 아시아 태평양(홍콩) 리전에 대한 리전 지원이 추가되었습니다. 지원되는 리전의 전체 목록은 AWS Certificate Manager 프라이빗 인증 기관 엔드포인트 및 할당량을 참조하세요.	2019년 7월 24일
프라이빗 CA 계층 구조에 대한 전체 지원 추가	루트 CA 생성 및 호스팅이 지원되기 때문에 외부 상위 요소가 필요하지 않습니다.	2019년 6월 20일

리전 지원 추가	AWS GovCloud(미국 서부 및 미국 동부) 리전에 대한 리전 지원이 추가되었습니다. 지원되는 리전의 전체 목록은 AWS Certificate Manager 프라이빗 인증 기관 엔드포인트 및 할당량을 참조하세요.	2019년 5월 8일
리전 지원 추가	AWS 아시아 태평양(뭄바이 및 서울), 미국 서부(캘리포니아 북부) 및 EU(파리 및 스톡홀름) 리전에 대한 리전 지원이 추가되었습니다. 지원되는 리전의 전체 목록은 AWS Certificate Manager 프라이빗 인증 기관 엔드포인트 및 할당량을 참조하세요.	2019년 4월 4일
인증서 갱신 워크플로 테스트	이제 고객은 ACM 관리형 갱신 워크플로의 구성을 수동으로 테스트할 수 있습니다. 자세한 내용은 ACM의 관리형 갱신 구성 테스트를 참조하세요.	2019년 3월 14일
리전 지원 추가	AWS EU(런던) 리전에 대한 리전 지원이 추가되었습니다. 지원되는 리전의 전체 목록은 AWS Certificate Manager 프라이빗 인증 기관 엔드포인트 및 할당량을 참조하세요.	2018년 8월 1일
삭제된 CA 복원	프라이빗 CA 복원을 통해 고객은 인증 기관(CA)가 삭제된 후 최대 30일 동안 복원할 수 있습니다. 자세한 내용은 사설 CA 복원을 참조하십시오.	2018년 6월 20일

이전 업데이트

다음 표에서는 2018년 6월 AWS Private Certificate Authority 이전의 설명서 릴리스 기록을 설명합니다.

변경	설명	Date
새 안내서	이 릴리스는 AWS Private Certificate Authority을 도입했습니다.	2018년 4월 4일

기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.