aws

사용자 가이드

AWS AppConfig

AWS AppConfig: 사용자 가이드

Table of Contents

AWS AppConfig란 무엇인가요? 1
AWS AppConfig 사용 사례 2
AWS AppConfig 사용의 이점 2
AWS AppConfig 작동 방식 3
AWS AppConfig 시작하기 5
SDK 5
AWS AppConfig 요금 6
AWS AppConfig 할당량 6
설 AWS AppConfig정 7
가입하세요. AWS 계정 7
관리 사용자 생성 7
프로그래밍 방식 액세스 권한 부여 8
(선택 사항) 경보를 기반으로 롤백 권한을 구성합니다. CloudWatch 9
1단계: 경보를 기반으로 롤백을 위한 권한 정책 생성 CloudWatch 10
2단계: 경보 기반 롤백을 위한 IAM 역할 생성 CloudWatch 11
3단계: 트러스트 관계 추가 12
[생성 중] 13
구성의 예 14
구성 프로필 IAM 역할에 대한 정보 16
네임스페이스 생성 18
AWS AppConfig 애플리케이션 생성 (콘솔) 19
AWS AppConfig 응용 프로그램 만들기 (명령줄) 19
환경 생성 21
AWS AppConfig 환경 만들기 (콘솔) 21
AWS AppConfig 환경 만들기 (명령줄) 22
AWS AppConfig에서 구성 프로필 생성 24
유효성 검사기에 대한 정보 25
기능 플래그 구성 프로필 생성 28
자유 형식 구성 프로필 생성 42
구성 데이터의 기타 소스 54
AWS Secrets Manager 54
배포 55
배포 전략 작업 55
미리 정의된 배포 전략 58

배치 전략 생성 59
구성 배포 63
구성 배포 (콘솔) 64
구성 배포 (명령줄) 65
CodePipeline과 배포 통합 68
통합의 작동 방식 69
검색 70
AWS AppConfig 데이터 플레인 서비스 정보 71
간소화된 검색 방법 72
AWS AppConfig 에이전트 Lambda 확장 프로그램을 사용하여 구성 데이터 검색 72
Amazon EC2 인스턴스에서 구성 데이터 검색 126
Amazon ECS 및 Amazon EKS 에서 구성 데이터 검색 139
추가 검색 기능 151
AWS AppConfig 에이전트 로컬 개발 161
API를 직접 호출하여 구성 검색 163
구성 예제 검색 164
워크플로우 확장 166
AWS AppConfig 확장 프로그램 정보 166
1단계: 확장으로 수행할 작업 결정 167
2단계: 확장 실행 시기 결정 168
3단계: 확장 연결 생성 169
4단계: 구성 배포 및 확장 액션이 수행되었는지 확인 169
AWS 작성 확장 작업 169
아마존 CloudWatch 에비빌리티 확장 프로그램 사용하기 170
AWS AppConfig deployment events to Amazon EventBridge 확장 작업 171
AWS AppConfig deployment events to Amazon SNS 확장 작업 173
AWS AppConfig deployment events to Amazon SQS 확장 작업 176
Jira 확장 작업 178
안내: 사용자 지정 확장 만들기 AWS AppConfig 183
사용자 지정 확장을 위한 Lambda 함수 생성 AWS AppConfig 184
사용자 지정 확장에 대한 권한 구성 AWS AppConfig 189
사용자 지정 AWS AppConfig 확장 생성 191
사용자 지정 AWS AppConfig 확장 프로그램의 확장 프로그램 연결 만들기 194
사용자 지정 확장을 호출하는 작업 수행 AWS AppConfig 195
Jira와의 확장 통합 195
코드 샘플 196

호스팅된 구성 저장소에 저장된 자유 형식 구성 생성 또는 업데이트 196
Secrets Manager에 저장된 시크릿에 대한 구성 프로파일 생성 198
구성 프로필 배포 200
AWS AppConfig에이전트를 사용하여 자유 형식 구성 프로필 읽기 204
AWS AppConfig에이전트를 사용하여 특정 기능 플래그 읽기 206
GetLatestConfig API 작업을 사용하여 자유 형식 구성 프로필 읽기 208
환경 정리하기 212
보안 218
최소 권한 액세스 구현 218
AWS AppConfig의 저장된 데이터 암호화 219
AWS PrivateLink 223
고려 사항 223
인터페이스 엔드포인트 생성 224
엔드포인트 정책을 생성 224
Secrets Manager 키 교체 225
AWS AppConfig에서 배포한 Secrets Manager 암호의 자동 교체 설정 225
모니터링 228
CloudTrail 로그 228
AWS AppConfig에 대한 정보 CloudTrail 228
AWS AppConfig의 데이터 이벤트 CloudTrail 229
AWS AppConfig의 관리 이벤트 CloudTrail 231
AWS AppConfig 로그 파일 항목 이해 231
AWS AppConfig데이터 플레인 호출에 대한 로깅 지표 232
CloudWatch지표에 대한 경보 생성 235
사용 설명서 기록 236
AWS 용어집 252
..... ccliii

AWS AppConfig란 무엇인가요?

AWS AppConfig 기능 플래그와 동적 구성을 사용하면 소프트웨어 빌더가 전체 코드를 배포하지 않고도 프로덕션 환경에서 애플리케이션 동작을 빠르고 안전하게 조정할 수 있습니다. AWS AppConfig는 소프트웨어 릴리스 빈도를 높이고, 애플리케이션 복원력을 개선하며, 새로운 문제를 더 빠르게 해결하는 데 도움이 됩니다. 기능 플래그를 사용하면 새로운 기능을 모든 사용자에게 완전히 배포하기 전에 새로운 기능을 점진적으로 사용자에게 릴리스하고 이러한 변경의 영향을 측정할 수 있습니다. 운영 플래그와 동적 구성을 사용하여 차단 목록, 허용 목록, 조절 제한, 로깅 상세도를 업데이트하고 기타 운영 조정을 수행하여 프로덕션 환경의 문제에 신속하게 대응할 수 있습니다.

> **Note**
>
> AWS AppConfig는 AWS Systems Manager의 기능입니다.

효율성을 개선하고 변경 사항을 더 빠르게 릴리스하세요

새로운 기능과 함께 기능 플래그를 사용하면 프로덕션 환경에 변경 사항을 릴리스하는 프로세스가 빨라집니다. 릴리스 전에 복잡한 병합이 필요한 수명이 긴 개발 브랜치에 의존하는 대신 기능 플래그를 사용하면 트렁크 기반 개발을 사용하여 소프트웨어를 작성할 수 있습니다. 기능 플래그를 사용하면 사용자가 볼 수 없는 CI/CD 파이프라인에서 시험판 코드를 안전하게 롤아웃할 수 있습니다. 변경 사항을 릴리스할 준비가 되면 새 코드를 배포하지 않고도 기능 플래그를 업데이트할 수 있습니다. 출시가 완료된 후에도 코드 배포를 롤백할 필요 없이 플래그가 블록 스위치 역할을 하여 새로운 기능을 비활성화할 수 있습니다.

내장된 안전 기능으로 의도하지 않은 변경이나 장애를 방지하세요

AWS AppConfig는 애플리케이션 장애를 일으킬 수 있는 기능 플래그를 활성화하거나 구성 데이터를 업데이트하지 않도록 하는 데 도움이 되는 다음과 같은 안전 기능을 제공합니다.

- 유효성 검사기: 프로덕션 환경에 변경 사항을 배포하기 전에 구성 데이터가 구문상 및 의미상 올바른지 확인합니다.
- 배포 전략: 배포 전략을 사용하면 몇 분 또는 몇 시간에 걸쳐 프로덕션 환경에 변경 사항을 천천히 릴리스할 수 있습니다.
- 모니터링 및 자동 롤백: AWS AppConfig는 Amazon CloudWatch와 통합되어 애플리케이션의 변경 사항을 모니터링합니다. 잘못된 구성 변경으로 인해 애플리케이션이 비정상적으로 작동하고 이 변경으로 인해 CloudWatch에서 경보가 트리거되는 경우, AWS AppConfig는 변경 사항을 자동으로 롤백하여 애플리케이션 사용자에게 미치는 영향을 최소화합니다.

안전하고 확장 가능한 기능 플래그 배포

AWS AppConfig는 AWS Identity and Access Management (IAM)과 통합되어 서비스에 대한 세분화된 역할 기반 액세스를 제공합니다. AWS AppConfig는 또한 암호화를 위해 AWS Key Management Service (AWS KMS), 감사를 위해 AWS CloudTrail와 통합됩니다. 외부 고객에게 릴리스되기 전에는 서비스를 대규모로 사용하는 내부 고객을 대상으로 모든 AWS AppConfig 안전 제어를 처음 개발하고 검증했습니다.

AWS AppConfig 사용 사례

애플리케이션 구성 콘텐츠가 애플리케이션마다 크게 다를 수 있음에도 불구하고 AWS AppConfig는 광범위한 고객 요구를 포괄하는 다음과 같은 사용 사례를 지원합니다.

- 기능 플래그 및 토글 — 통제된 환경에서 고객에게 새로운 기능을 안전하게 릴리스하십시오. 문제가 발생할 경우 변경 사항을 즉시 롤백할 수 있습니다.
- 애플리케이션 조정 — 애플리케이션 변경 사항을 신중하게 도입하면서 프로덕션 환경의 사용자를 대상으로 변경 사항이 미치는 영향을 테스트합니다.
- 허용 목록 또는 차단 목록 — 새 코드를 배포하지 않고도 프리미엄 기능에 대한 액세스를 제어하거나 특정 사용자를 즉시 차단할 수 있습니다.
- 중앙 집중식 구성 스토리지 — 모든 워크로드에서 구성 데이터를 체계적이고 일관되게 유지합니다. AWS AppConfig를 사용하여 AWS AppConfig 호스팅된 구성 스토어, AWS Secrets Manager, Systems Manager 파라미터 스토어 또는 Amazon S3에 저장된 구성 데이터를 배포할 수 있습니다.

AWS AppConfig 사용의 이점

AWS AppConfig는 조직에 다음과 같은 이점을 제공합니다.

- 고객의 예상치 못한 가동 중지 시간을 줄이세요

  AWS AppConfig는 구성의 유효성을 검사하는 규칙을 생성할 수 있도록 함으로써 애플리케이션 가동 중지 시간을 줄입니다. 유효하지 않은 구성은 배포할 수 없습니다. AWS AppConfig에서는 구성의 유효성을 검사하기 위한 다음 두 가지 옵션을 제공합니다.

  - 구문 유효성 검사의 경우 JSON 스키마를 사용할 수 있습니다. AWS AppConfig는 JSON 스키마를 사용하여 구성의 유효성을 검사하여 구성 변경이 애플리케이션 요구 사항을 준수하는지 확인합니다.
  - 시맨틱 검증의 경우 AWS AppConfig는 사용자가 소유한 AWS Lambda 함수를 호출하여 구성 내에서 데이터를 검증할 수 있습니다.

- 대상 집합에 변경 사항을 신속하게 배포

  AWS AppConfig는 중앙 위치에서 구성 변경 사항을 배포하여 대규모로 애플리케이션 관리를 단순화합니다. AWS AppConfig는 AWS AppConfig 호스팅된 구성 저장소, Systems Manager 파라미터 스토어, 파라미터, Systems Manager(SSM) 문서 및 Amazon S3에 저장된 구성을 지원합니다. EC2 인스턴스, AWS Lambda, 컨테이너, 모바일 애플리케이션 또는 IoT 디바이스에서 호스팅되는 애플리케이션에서 AWS AppConfig를 사용할 수 있습니다.

  대상은 Systems Manager SSM 에이전트, 또는 다른 Systems Manager 기능에서 필요한 IAM 인스턴스 프로필을 사용하여 구성될 필요가 없습니다. 즉, AWS AppConfig는 비관리형 인스턴스와 함께 작동합니다.
- 중단 없이 애플리케이션 업데이트

  AWS AppConfig는 중량(heavy-weight) 빌드 프로세스 없이, 또는 대상을 서비스 중단하지 않고 런타임 시 대상에 구성 변경 사항을 배포합니다.
- 애플리케이션 전반의 변경 사항 배포 제어

  대상에 구성 변경 사항을 배포할 때 AWS AppConfig를 사용하면 배포 전략을 사용하여 위험을 최소화할 수 있습니다. 배포 전략을 사용하면 구성 변경 사항을 플릿에 천천히 롤아웃할 수 있습니다. 배포 중에 문제가 발생하는 경우 대부분의 호스트에 적용되기 전에 구성 변경 내용을 롤백할 수 있습니다.

AWS AppConfig 작동 방식

이 섹션은 AWS AppConfig 작동 방식 및 시작 방법에 대한 개괄적인 설명을 제공합니다.

1. 클라우드에서 관리하고자 하는 코드의 구성 값을 식별하십시오.

   AWS AppConfig 아티팩트를 만들기 전에 AWS AppConfig를 이용해 코드에서 동적으로 관리할 구성 데이터를 식별하는 것이 좋습니다. 좋은 예로는 기능 플래그 또는 토글, 허용 및 차단 목록, 로깅 상세 정보, 서비스 제한, 제한 규칙 등이 있습니다.

   구성 데이터가 이미 클라우드에 있는 경우 AWS AppConfig 검증, 배포 및 확장 기능을 활용하여 구성 데이터 관리를 더욱 간소화할 수 있습니다.
2. 애플리케이션 네임스페이스 생성

   네임스페이스를 만들려면 애플리케이션이라는 AWS AppConfig 아티팩트를 만들어야 합니다. 애플리케이션은 단순히 폴더와 같은 조직적 구성입니다.

3. 환경 생성

   각 AWS AppConfig 애플리케이션에 대해 하나 이상의 환경을 정의합니다. 환경은 `Beta` 또는 `Production` 환경의 애플리케이션, AWS Lambda 함수, 또는 컨테이너와 같은 대상의 논리적 그룹입니다. 애플리케이션 하위 구성 요소(예: `Web`, `Mobile` 및 `Back-end`)에 대한 환경을 정의할 수도 있습니다.

   각 환경에 대해 Amazon CloudWatch 경보를 구성할 수 있습니다. 시스템은 구성 배포 중에 경보를 모니터링합니다. 경보가 트리거되면 시스템이 구성을 롤백합니다.

4. 구성 프로필 생성

   구성 프로필에는 무엇보다도 저장된 위치에서 AWS AppConfig가 구성 데이터를 찾을 수 있는 URI와 프로필 유형이 포함됩니다. AWS AppConfig는 기능 플래그와 자유 형식 구성이라는 두 가지 구성 프로필 유형을 지원합니다. 기능 플래그 구성 프로필은 해당 데이터를 AWS AppConfig 호스팅된 구성 저장소에 저장하며 URI는 간단하게 `hosted`입니다. 자유 형식 구성 프로필의 경우에 에서 자유 형식 구성 프로필 생성 AWS AppConfig에 설명된 대로 AWS AppConfig 호스팅된 구성 저장소 또는 AWS AppConfig와 통합되는 모든 AWS 서비스에 데이터를 저장할 수 있습니다.

   구성 프로파일에는 구성 데이터가 구문상 및 의미상 올바른지 확인하기 위해 선택적 유효성 검사기가 포함될 수도 있습니다. AWS AppConfig에서는 배포를 시작할 때 유효성 검사기를 사용하여 검사를 수행합니다. 오류가 발견되면 배포는 이전 구성 데이터로 롤백합니다.

5. 구성 데이터 배포

   새로 배포를 생성할 때 다음을 지정할 수 있습니다.

   - 애플리케이션 ID
   - 구성 프로필 ID
   - 구성 버전
   - 구성 데이터를 배포할 환경 ID
   - 변경 사항을 얼마나 빨리 적용할지를 정의하는 배포 전략 ID

   StartDeployment API 작업을 호출하면 AWS AppConfig는 다음 작업을 수행합니다.

   1. 구성 프로필의 위치 URI를 사용하여 기본 데이터 저장소에서 구성 데이터를 검색합니다.
   2. 구성 프로필을 생성할 때 지정한 유효성 검사기를 사용하여 구성 데이터가 구문상 및 의미상 올바른지 확인합니다.
   3. 애플리케이션에서 검색할 수 있도록 데이터 사본을 캐시합니다. 이 캐시된 복사본을 배포된 데이터라고 합니다.

6. 구성을 검색합니다

   AWS AppConfig 에이전트를 로컬 호스트로 구성하고 에이전트가 구성 업데이트를 위해 AWS AppConfig를 폴링하도록 할 수 있습니다. 에이전트는 StartConfigurationSession 및 GetLatestConfiguration API 작업을 호출하고 구성 데이터를 로컬에 캐시합니다. 데이터를 검색하기 위해 애플리케이션은 로컬 호스트 서버에 HTTP 호출을 수행합니다. AWS AppConfig 에이전트는 간소화된 검색 방법에 설명된 대로 여러 사용 사례를 지원합니다.

   사용 사례에서 AWS AppConfig 에이전트가 지원되지 않는 경우 StartConfigurationSession 및 GetLatestConfiguration API 작업을 직접 호출하여 구성 업데이트를 위해 AWS AppConfig를 폴링하도록 애플리케이션을 구성할 수 있습니다.

AWS AppConfig 시작하기

다음 리소스는 AWS AppConfig로 직접 작업하는 데 도움이 될 수 있습니다.

Amazon Web Services 유튜브 채널에서 더 많은 AWS 동영상을 보세요.

다음 블로그는 AWS AppConfig와 그 기능을 자세히 알아보는 데 도움이 됩니다.

- AWS AppConfig 기능 플래그 사용
- AWS AppConfig 기능 플래그 및 구성 데이터 검증을 위한 모범 사례

SDK

AWS AppConfig 언어별 SDK에 대한 자세한 내용은 다음 리소스를 참조하세요.

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

AWS AppConfig 요금

AWS AppConfig 요금은 구성 데이터 및 기능 플래그 검색을 기준으로 사용한 만큼만 지불하는 방식입니다. 비용을 최적화하려면 AWS AppConfig 에이전트를 사용하는 것이 좋습니다. 자세한 내용은 AWS Systems Manager 요금을 참조하세요.

AWS AppConfig 할당량

다른 Systems Manager 할당량과 함께 AWS AppConfig 엔드포인트 및 서비스 할당량에 대한 정보는 Amazon Web Services 일반 참조에 있습니다.

> **Note**
>
> AWS AppConfig 구성을 저장하는 서비스의 할당량에 대한 자세한 내용은 구성 저장소 할당량 및 제한 섹션을 참조하십시오.

설 AWS AppConfig정

아직 등록하지 않았다면 관리 사용자를 AWS 계정 등록하고 생성하세요.

가입하세요. AWS 계정

계정이 없는 경우 다음 단계를 완료하여 계정을 만드세요. AWS 계정

가입하려면 AWS 계정

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

   등록 절차 중에는 전화를 받고 키패드로 인증 코드를 입력하는 과정이 있습니다.

   에 AWS 계정가입하면 AWS 계정 루트 사용자a가 생성됩니다. 루트 사용자에게는 계정의 모든 AWS 서비스 및 리소스 액세스 권한이 있습니다. 보안 모범 사례는 관리 사용자에게 관리자 액세스 권한을 할당하고, 오직 루트 사용자만 루트 사용자 액세스 권한이 필요한 태스크를 수행하는 것입니다.

AWS 가입 절차가 완료된 후 확인 이메일을 보냅니다. 언제든지 https://aws.amazon.com/으로 가서 내 계정(My Account)을 선택하여 현재 계정 활동을 보고 계정을 관리할 수 있습니다.

관리 사용자 생성

등록한 AWS 계정후에는 일상적인 작업에 루트 사용자를 사용하지 않도록 관리 사용자를 보호하고 AWS IAM Identity Center활성화하고 생성하십시오 AWS 계정 루트 사용자.

보안을 유지하세요. AWS 계정 루트 사용자

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

   루트 사용자를 사용하여 로그인하는 데 도움이 필요하면 AWS 로그인 User Guide의 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 참조하십시오.

관리 사용자 생성

1. IAM Identity Center를 활성화합니다.

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

   를 ID 소스로 사용하는 방법에 대한 자습서는 사용 설명서의 기본값으로 IAM Identity Center 디렉터리사용자 액세스 구성을 참조하십시오. IAM Identity Center 디렉터리 AWS IAM Identity Center

관리 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 참조하십시오.AWS 로그인

프로그래밍 방식 액세스 권한 부여

사용자가 AWS 외부 사용자와 상호 작용하려는 경우 프로그래밍 방식의 액세스가 필요합니다. AWS Management Console프로그래밍 방식의 액세스 권한을 부여하는 방법은 액세스하는 사용자 유형에 따라 다릅니다. AWS

사용자에게 프로그래밍 방식 액세스 권한을 부여하려면 다음 옵션 중 하나를 선택합니다.

프로그래밍 방식 액세스가 필요한 사용자는 누구인가요?	To	액세스 권한을 부여하는 사용자
작업 인력 ID (IAM Identity Center가 관리하는 사용자)	임시 자격 증명을 사용하여 AWS CLI, AWS SDK 또는 API에 대한 프로그래밍 요청에 서명할 수 있습니다. AWS	사용하고자 하는 인터페이스에 대한 지침을 따릅니다. • AWS CLI 에 대한 내용은 사용 설명서의 AWS CLI사용을

프로그래밍 방식 액세스가 필요한 사용자는 누구인가요?	To	액세스 권한을 부여하는 사용자
		AWS IAM Identity Center위한 구성을 참조하십시오.AWS Command Line Interface • AWS SDK, 도구 및 AWS API의 경우 AWS SDK 및 도구 참조 안내서의 IAM ID 센터 인증을 참조하십시오.
IAM	임시 자격 증명을 사용하여 AWS CLI, AWS SDK 또는 API에 대한 프로그래밍 방식 요청에 서명할 수 있습니다. AWS	IAM 사용 설명서의 AWS 리소스와 함께 임시 자격 증명 사용의 지침을 따르십시오.
IAM	(권장되지 않음) 장기 자격 증명을 사용하여 AWS CLI, AWS SDK 또는 API에 대한 프로그래밍 요청에 서명할 수 있습니다. AWS	사용하고자 하는 인터페이스에 대한 지침을 따릅니다. • 에 대한 내용은 사용 설명서의 IAM 사용자 자격 증명을 사용한 인증을 참조하십시오. AWS CLIAWS Command Line Interface • AWS SDK 및 도구의 경우 SDK 및 도구 참조 안내서의 장기 자격 증명을 사용한 인증을 참조하십시오.AWS • AWS API의 경우 IAM 사용 설명서의 IAM 사용자의 액세스 키 관리를 참조하십시오.

(선택 사항) 경보를 기반으로 롤백 권한을 구성합니다. CloudWatch

하나 이상의 Amazon CloudWatch 경보에 대한 응답으로 이전 버전의 구성으로 AWS AppConfig 롤백하도록 구성할 수 있습니다. CloudWatch 경보에 응답하도록 배포를 구성할 때는 AWS Identity and

Access Management (IAM) 역할을 지정합니다. AWS AppConfig 경보를 CloudWatch 모니터링하려면 이 역할이 필요합니다.

> **Note**
>
> IAM 역할은 현재 계정에 속해야 합니다. 기본적으로 현재 계정이 소유한 알람만 AWS AppConfig 모니터링할 수 있습니다. 다른 계정의 지표에 대한 응답으로 배포를 롤백하도록 AWS AppConfig 구성하려면 계정 간 경보를 구성해야 합니다. 자세한 내용은 Amazon CloudWatch 사용 설명서의 교차 계정 지역 CloudWatch 콘솔을 참조하십시오.

다음 절차를 사용하여 경보를 기반으로 AWS AppConfig 롤백할 수 있는 IAM 역할을 생성하십시오. CloudWatch 이 섹션에는 다음 절차가 포함됩니다.

1. 1단계: 경보를 기반으로 롤백을 위한 권한 정책 생성 CloudWatch
2. 2단계: 경보 기반 롤백을 위한 IAM 역할 생성 CloudWatch
3. 3단계: 트러스트 관계 추가

1단계: 경보를 기반으로 롤백을 위한 권한 정책 생성 CloudWatch

다음 절차를 사용하여 API 작업을 호출할 AWS AppConfig 권한을 부여하는 IAM 정책을 생성합니다. `DescribeAlarms`

경보 기반 롤백을 위한 IAM 권한 정책을 만들려면 CloudWatch

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 탐색 창에서 정책을 선택한 후 정책 생성을 선택합니다.
3. 정책 생성 페이지에서 JSON 탭을 선택합니다.
4. JSON 탭에 있는 기본 내용을 다음 권한 정책으로 바꾼 후 다음: 태그를 선택합니다.

> **Note**
>
> CloudWatch 복합 경보에 대한 정보를 반환하려면 아래 그림과 같이 DescribeAlarmsAPI 작업에 * 권한을 할당해야 합니다. `DescribeAlarms`범위가 더 좁으면 복합 경보에 대한 정보를 반환할 수 없습니다.

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

5. 이 역할에 대한 태그를 입력하고 다음: 검토를 선택합니다.
6. 검토 페이지에서 이름 필드에 **SSMCloudWatchAlarmDiscoveryPolicy**를 입력합니다.
7. 정책 생성(Create policy)을 선택합니다. 시스템에서 정책 페이지로 돌아갑니다.

2단계: 경보 기반 롤백을 위한 IAM 역할 생성 CloudWatch

다음 절차를 사용하여 IAM 역할을 생성하고 이전 절차에서 생성한 정책을 할당합니다.

경보 기반 롤백을 위한 IAM 역할 생성하기 CloudWatch

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 탐색 창에서 역할을 선택한 후 역할 생성을 선택합니다.
3. 신뢰할 수 있는 유형의 엔터티 선택 아래에서 AWS 서비스를 선택합니다.
4. 이 역할을 사용할 서비스 선택에서 사용자를 대신해 EC2: AWS 서비스 호출을 위한 EC2 인스턴스 허용을 선택한 후 다음: 권한을 선택합니다.
5. 연결된 권한 정책 페이지에서 SSM을 검색합니다. CloudWatchAlarmDiscoveryPolicy
6. 정책을 선택한 후 다음: 태그를 선택합니다.
7. 이 역할에 대한 태그를 입력하고 다음: 검토를 선택합니다.
8. 역할 생성 페이지에서 역할 이름 필드에 **SSMCloudWatchAlarmDiscoveryRole**을 입력한 다음 역할 생성을 선택합니다.
9. 역할 페이지에서 방금 생성한 역할을 선택합니다. 요약 페이지가 열립니다.

3단계: 트러스트 관계 추가

다음 절차에 따라 생성한 역할이 AWS AppConfig를 신뢰하도록 구성합니다.

신뢰 관계를 추가하려면 AWS AppConfig

1. 방금 생성한 역할에 대한 요약 페이지에서 신뢰 관계 탭을 선택한 후 신뢰 관계 편집을 선택합니다.
2. 다음 예와 같이 "appconfig.amazonaws.com"만 포함하도록 정책을 편집합니다.

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

3. 신뢰 정책 업데이트를 선택합니다.

기능 플래그 및 자유 형식 구성 데이터 생성 AWS AppConfig

이 섹션의 항목은 에서 다음 작업을 완료하는 데 도움이 됩니다. AWS AppConfig이러한 작업을 수행하면 구성 데이터를 배포하는 데 중요한 아티팩트가 만들어집니다.

1. 애플리케이션 네임스페이스 생성

   응용 프로그램 네임스페이스를 만들려면 응용 프로그램이라는 AWS AppConfig 아티팩트를 만듭니다. 애플리케이션은 단순히 폴더와 같은 조직적 구성입니다.

2. 환경 생성

   각 AWS AppConfig 애플리케이션에 대해 하나 이상의 환경을 정의합니다. 환경은 `Beta` 또는 `Production` 환경의 애플리케이션과 같은 AWS AppConfig 대상의 논리적 배포 그룹입니다. 애플리케이션 하위 구성 요소(예: `AWS Lambda functions`, `Containers`, `Web`, `Mobile` 및 `Back-end`)에 대한 환경을 정의할 수도 있습니다.

   문제가 되는 구성 변경을 자동으로 롤백하도록 각 환경에 대해 Amazon CloudWatch 경보를 구성할 수 있습니다. 시스템은 구성 배포 중에 경보를 모니터링합니다. 경보가 트리거되면 시스템이 구성을 롤백합니다.

3. 구성 프로필 생성

   구성 프로필에는 무엇보다도 저장된 위치에서 구성 데이터를 찾을 수 있는 URI와 프로필 유형이 포함됩니다. AWS AppConfig AWS AppConfig 기능 플래그와 자유 형식 구성이라는 두 가지 구성 프로파일 유형을 지원합니다. 기능 플래그 구성 프로파일은 해당 데이터를 AWS AppConfig 호스팅된 구성 저장소에 저장하며 URI는 간단합니다. `hosted` 자유 형식 구성 프로파일의 경우 에 설명된 대로 AWS AppConfig 호스팅된 구성 저장소나 통합되는 다른 Systems Manager 기능 또는 AWS 서비스에 데이터를 저장할 수 있습니다. AWS AppConfig에서 자유 형식 구성 프로필 생성 AWS AppConfig

   구성 프로파일에는 구성 데이터가 구문 및 의미상 정확한지 확인하는 선택적 유효성 검사기도 포함될 수 있습니다. AWS AppConfig 배포를 시작할 때 유효성 검사기를 사용하여 검사를 수행합니다. 오류가 발견되면 구성의 대상을 변경하기 전에 배포가 중지됩니다.

   > **Note**
   >
   > Amazon Simple Storage Service (Amazon S3) 에 비밀을 AWS Secrets Manager 저장하거나 데이터를 관리해야 하는 특별한 요구가 없는 한, 대부분의 기능과 향상된 기능을 제공하는 호스팅된 구성 저장소에 구성 데이터를 호스팅하는 것이 좋습니다. AWS AppConfig

주제

- 구성의 예
- 구성 프로필 IAM 역할에 대한 정보
- AWS AppConfig에서 애플리케이션을 위한 네임스페이스 생성
- AWS AppConfig에서 애플리케이션을 위한 환경을 만듭니다
- AWS AppConfig에서 구성 프로필 생성
- 구성 데이터의 기타 소스

구성의 예

의 AWS Systems Manager기능을 사용하여 AWS AppConfig애플리케이션 구성을 생성, 관리하고 신속하게 배포할 수 있습니다. 구성은 애플리케이션의 동작에 영향을 미치는 설정의 모음입니다. 여기 몇 가지 예가 있습니다.

기능 플래그 구성

다음 기능 플래그 구성은 리전별로 모바일 결제 및 기본 결제를 활성화 또는 비활성화합니다.

JSON

```
{
  "allow_mobile_payments": {
    "enabled": false
  },
  "default_payments_per_region": {
    "enabled": true
  }
}
```

YAML

```
---
allow_mobile_payments:
  enabled: false
default_payments_per_region:
  enabled: true
```

운영 구성

다음과 같은 자유 형식 구성은 애플리케이션이 요청을 처리하는 방식을 제한합니다.

JSON

```
{
  "throttle-limits": {
    "enabled": "true",
    "throttles": [
      {
        "simultaneous_connections": 12
      },
      {
        "tps_maximum": 5000
      }
    ],
    "limit-background-tasks": [
      true
    ]
  }
}
```

YAML

```
---
throttle-limits:
  enabled: 'true'
  throttles:
  - simultaneous_connections: 12
  - tps_maximum: 5000
  limit-background-tasks:
  - true
```

액세스 제어 목록 구성

다음 액세스 제어 목록 자유 형식 구성은 애플리케이션에 액세스할 수 있는 사용자 또는 그룹을 지정합니다.

JSON

```
{
  "allow-list": {
    "enabled": "true",
```

```
    "cohorts": [
      {
        "internal_employees": true
      },
      {
        "beta_group": false
      },
      {
        "recent_new_customers": false
      },
      {
        "user_name": "Jane_Doe"
      },
      {
        "user_name": "John_Doe"
      }
    ]
  }
}
```

YAML

```
---
allow-list:
  enabled: 'true'
  cohorts:
  - internal_employees: true
  - beta_group: false
  - recent_new_customers: false
  - user_name: Jane_Doe
  - user_name: Ashok_Kumar
```

구성 프로필 IAM 역할에 대한 정보

를 사용하여 AWS AppConfig구성 데이터에 대한 액세스를 제공하는 IAM 역할을 생성할 수 있습니다. 또는 직접 IAM 역할을 생성할 수 있습니다. 를 사용하여 AWS AppConfig역할을 생성하는 경우 시스템은 역할을 생성하고 선택한 구성 소스 유형에 따라 다음 권한 정책 중 하나를 지정합니다.

구성 소스는 Secrets Manager 암호

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
             ],
            "Resource": [
                "arn:aws:secretsmanager:AWS ##:account_ID:secret:secret_name-a1b2c3"
            ]
        }
    ]
}
```

구성 소스가 Parameter Store 파라미터임

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ssm:GetParameter"
            ],
            "Resource": [
                "arn:aws:ssm:AWS ##:account_ID:parameter/parameter_name"
            ]
        }
    ]
    }
```

구성 소스가 SSM 문서임

```
{

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

```
                "arn:aws:ssm:AWS ##:account_ID:document/document_name"
            ]
        }
    ]
}
```

를 사용하여 AWS AppConfig역할을 생성하는 경우 시스템은 해당 역할에 대해 다음과 같은 신뢰 관계도 생성합니다.

```
{

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

AWS AppConfig에서 애플리케이션을 위한 네임스페이스 생성

이 섹션의 절차는 응용 프로그램이라는 AWS AppConfig 아티팩트를 만드는 데 도움이 됩니다. 애플리케이션은 단순히 애플리케이션의 네임스페이스를 식별하는 폴더와 같은 조직적 구성입니다. 이 조직 구성은 실행 코드의 일부 단위와 관계가 있습니다. 예를 들어, 사용자가 설치한 모바일 응용 프로그램의 구성 데이터를 구성하고 관리하기 MyMobileApp 위해 호출되는 응용 프로그램을 만들 수 있습니다. 기능 플래그 또는 자유 형식 구성 데이터를 배포하고 검색하는 AWS AppConfig 데 사용할 수 있으려면 먼저 이러한 아티팩트를 만들어야 합니다.

> **Note**
>
> 를 AWS CloudFormation 사용하여 응용 프로그램, 환경, 구성 프로필, 배포, 배포 전략 및 호스팅된 구성 버전을 비롯한 AWS AppConfig 아티팩트를 만들 수 있습니다. 자세한 내용은AWS CloudFormation 사용 설명서의 AWS AppConfig 리소스 유형 참조를 참조하십시오.

주제

- AWS AppConfig 애플리케이션 생성 (콘솔)
- AWS AppConfig 응용 프로그램 만들기 (명령줄)

AWS AppConfig 애플리케이션 생성 (콘솔)

AWS Systems Manager 콘솔을 사용하여 AWS AppConfig 애플리케이션을 생성하려면 다음 절차를 따르십시오.

애플리케이션을 생성하려면

1. https://console.aws.amazon.com/systems-manager/appconfig/ 에서 AWS Systems Manager 콘솔을 엽니다.
2. 애플리케이션 페이지에서 애플리케이션 생성을 선택합니다.
3. 이름에 애플리케이션의 이름을 입력합니다.
4. 설명에 애플리케이션에 대한 설명을 입력합니다.
5. (선택 사항) 확장 섹션의 목록에서 확장 프로그램을 선택합니다. 자세한 정보는 AWS AppConfig 확장 프로그램 정보을 참조하세요.
6. (선택 사항) 태그 섹션에 키와 선택적 값을 입력합니다. 하나의 리소스에 대해 최대 50개의 태그를 지정할 수 있습니다.
7. 애플리케이션 생성을 선택합니다.

AWS AppConfig 응용 프로그램을 만든 다음 환경 탭을 표시합니다. AWS AppConfig에서 애플리케이션을 위한 환경을 만듭니다로 이동합니다.

AWS AppConfig 응용 프로그램 만들기 (명령줄)

다음 절차는 AWS CLI (Linux 또는 Windows에서) 를 사용하거나 AWS AppConfig 응용 프로그램을 만드는 AWS Tools for PowerShell 방법을 설명합니다.

단계별로 애플리케이션을 생성하려면

1. 를 엽니다 AWS CLI.
2. 다음 명령을 실행하여 애플리케이션을 생성합니다.

   Linux

```
aws appconfig create-application \
```

```
  --name A_name_for_the_application \
  --description A_description_of_the_application \
  --tags User_defined_key_value_pair_metadata_for_the_application
```

Windows

```
aws appconfig create-application ^
  --name A_name_for_the_application ^
  --description A_description_of_the_application ^
  --tags User_defined_key_value_pair_metadata_for_the_application
```

PowerShell

```
New-APPCApplication `
  -Name Name_for_the_application `
  -Description Description_of_the_application `
  -Tag Hashtable_type_user_defined_key_value_pair_metadata_for_the_application
```

시스템은 다음과 같은 정보를 반환합니다.

Linux

```
{
   "Id": "Application ID",
   "Name": "Application name",
   "Description": "Description of the application"
}
```

Windows

```
{
   "Id": "Application ID",
   "Name": "Application name",
   "Description": "Description of the application"
}
```

PowerShell

```
ContentLength    : Runtime of the command
Description      : Description of the application
```

```
HttpStatusCode   : HTTP Status of the runtime
Id               : Application ID
Name             : Application name
ResponseMetadata : Runtime Metadata
```

AWS AppConfig에서 애플리케이션을 위한 환경을 만듭니다

각 AWS AppConfig 애플리케이션에 대해 하나 이상의 환경을 정의합니다. 환경은 또는 환경의 애플리케이션, AWS Lambda 함수 `Beta` 또는 `Production` 컨테이너와 같은 AppConfig 대상의 논리적 배포 그룹입니다. 애플리케이션 하위 구성 요소(예: `Web`, `Mobile` 및 `Back-end`)에 대한 환경을 정의할 수도 있습니다. 각 환경에 맞게 Amazon CloudWatch 경보를 구성할 수 있습니다. 시스템은 구성 배포 중에 경보를 모니터링합니다. 경보가 트리거되면 시스템이 구성을 롤백합니다.

시작하기 전

경보에 대한 응답으로 구성을 롤백할 수 있게 AWS AppConfig 하려면 CloudWatch 경보에 응답할 수 있는 권한을 가진 AWS Identity and Access Management (IAM) 역할을 구성해야 합니다. AWS AppConfig CloudWatch 다음 절차에서 이 역할을 선택합니다. 자세한 정보는 (선택 사항) 경보를 기반으로 롤백 권한을 구성합니다. CloudWatch 을 참조하세요.

주제

- AWS AppConfig 환경 만들기 (콘솔)
- AWS AppConfig 환경 만들기 (명령줄)

AWS AppConfig 환경 만들기 (콘솔)

AWS Systems Manager 콘솔을 사용하여 AWS AppConfig 환경을 만들려면 다음 절차를 따르십시오.

환경을 생성하려면

1. https://console.aws.amazon.com/systems-manager/appconfig/ 에서 AWS Systems Manager 콘솔을 엽니다.
2. 애플리케이션 탭에서 애플리케이션 이름을 선택하여 세부 정보 페이지를 엽니다.
3. 환경 탭을 선택한 다음 환경 만들기를 선택합니다.
4. 이름에 환경의 이름을 입력합니다.
5. 설명에 환경에 대한 설명을 입력합니다.

6. (선택 사항) 모니터 섹션에서 IAM 역할 필드를 선택한 다음 경보가 트리거되는 경우 구성을 롤백할 권한이 있는 IAM 역할을 선택합니다.
7. CloudWatch 경보 목록에서 모니터링할 하나 이상의 경보를 선택합니다. AWS AppConfig 이러한 경보 중 하나가 경보 상태가 되면 구성 배포를 롤백합니다.
8. (선택 사항) 확장 연결 섹션의 목록에서 확장을 선택합니다. 자세한 정보는 AWS AppConfig 확장 프로그램 정보을 참조하세요.
9. (선택 사항) 태그 섹션에 키와 선택적 값을 입력합니다. 하나의 리소스에 대해 최대 50개의 태그를 지정할 수 있습니다.
10. 환경 생성을 선택합니다.

AWS AppConfig 환경을 만든 다음 환경 세부 정보 페이지를 표시합니다. AWS AppConfig에서 구성 프로필 생성로 이동합니다.

AWS AppConfig 환경 만들기 (명령줄)

다음 절차는 AWS CLI (Linux 또는 Windows에서) 를 사용하거나 AWS AppConfig 환경을 만드는 AWS Tools for PowerShell 방법을 설명합니다.

단계별로 환경을 생성하려면

1. 를 엽니다 AWS CLI.
2. 다음 명령을 실행해 환경을 생성합니다.

   Linux

   ```
   aws appconfig create-environment \
     --application-id The_application_ID \
     --name A_name_for_the_environment \
     --description A_description_of_the_environment \
     --monitors
    "AlarmArn=ARN_of_the_Amazon_CloudWatch_alarm,AlarmArnRole=ARN_of_the_IAM
    role_for_AWS AppConfig_to_monitor_AlarmArn" \
     --tags User_defined_key_value_pair_metadata_of_the_environment
   ```

   Windows

   ```
   aws appconfig create-environment ^
     --application-id The_application_ID ^
     --name A_name_for_the_environment ^
   ```

```
  --description A_description_of_the_environment ^
  --monitors
 "AlarmArn=ARN_of_the_Amazon_CloudWatch_alarm,AlarmArnRole=ARN_of_the_IAM
 role_for_AWS AppConfig_to_monitor_AlarmArn" ^
  --tags User_defined_key_value_pair_metadata_of_the_environment
```

PowerShell

```
New-APPCEnvironment `
  -Name Name_for_the_environment `
  -ApplicationId The_application_ID
  -Description Description_of_the_environment `
  -Monitors
 @{"AlarmArn=ARN_of_the_Amazon_CloudWatch_alarm,AlarmArnRole=ARN_of_the_IAM
 role_for_AWS AppConfig_to_monitor_AlarmArn"} `
  -Tag Hashtable_type_user_defined_key_value_pair_metadata_of_the_environment
```

시스템은 다음과 같은 정보를 반환합니다.

Linux

```
{
   "ApplicationId": "The application ID",
   "Id": "The_environment ID",
   "Name": "Name of the environment",
   "State": "The state of the environment",
   "Description": "Description of the environment",

   "Monitors": [
      {
         "AlarmArn": "ARN of the Amazon CloudWatch alarm",
         "AlarmRoleArn": "ARN of the IAM role for AppConfig to monitor AlarmArn"
      }
   ]
}
```

Windows

```
{
   "ApplicationId": "The application ID",
   "Id": "The environment ID",
```

```
   "Name": "Name of the environment",
   "State": "The state of the environment"
   "Description": "Description of the environment",

   "Monitors": [
      {
         "AlarmArn": "ARN of the Amazon CloudWatch alarm",
         "AlarmRoleArn": "ARN of the IAM role for AppConfig to monitor AlarmArn"
      }
   ]
}
```

PowerShell

```
ApplicationId     : The application ID
ContentLength     : Runtime of the command
Description       : Description of the environment
HttpStatusCode    : HTTP Status of the runtime
Id                : The environment ID
Monitors          : {ARN of the Amazon CloudWatch alarm, ARN of the IAM role for
 AppConfig to monitor AlarmArn}
Name              : Name of the environment
Response Metadata : Runtime Metadata
State             : State of the environment
```

AWS AppConfig에서 구성 프로필 생성로 이동합니다.

AWS AppConfig에서 구성 프로필 생성

구성 프로필에는 무엇보다도 저장된 위치에서 구성 데이터를 찾을 수 있는 URI와 구성 유형이 포함됩니다. AWS AppConfig AWS AppConfig 기능 플래그와 자유 형식 구성이라는 두 가지 유형의 구성 프로필을 지원합니다. 기능 플래그 구성은 AWS AppConfig 호스팅된 구성 저장소에 데이터를 저장하며 URI는 간단합니다. `hosted` 자유 형식 구성은 AWS AppConfig 호스팅된 구성 저장소, 다양한 Systems Manager 기능 또는 다음과 통합되는 AWS 서비스에 데이터를 저장할 수 있습니다. AWS AppConfig자세한 정보는 에서 자유 형식 구성 프로필 생성 AWS AppConfig을 참조하세요.

구성 프로파일에는 구성 데이터가 구문 및 의미상 정확한지 확인하는 선택적 유효성 검사기도 포함될 수 있습니다. AWS AppConfig 배포를 시작할 때 유효성 검사기를 사용하여 검사를 수행합니다. 오류가 발견되면 구성의 대상을 변경하기 전에 배포가 중지됩니다.

> **Note**
>
> 가능한 경우 대부분의 기능과 향상된 기능을 제공하는 AWS AppConfig 호스팅된 구성 저장소에서 구성 데이터를 호스팅하는 것이 좋습니다.

주제

- 유효성 검사기에 대한 정보
- 에서 기능 플래그 구성 프로필 생성 AWS AppConfig
- 에서 자유 형식 구성 프로필 생성 AWS AppConfig

유효성 검사기에 대한 정보

구성 프로필을 생성할 때 최대 두 개의 유효성 검사기를 지정할 수 있습니다. 유효성 검사기는 구성 데이터가 구문론적, 의미론적으로 올바른지 확인합니다. 유효성 검사기를 사용하려는 경우 구성 프로필을 만들기 전에 유효성 검사기를 만들어야 합니다. AWS AppConfig 다음과 같은 유형의 유효성 검사기를 지원합니다.

- AWS Lambda 함수: 기능 플래그 및 자유 형식 구성을 지원합니다.
- JSON 스키마: 자유 형식 구성에 지원됩니다. (JSON 스키마를 기준으로 기능 플래그를AWS AppConfig 자동으로 검증합니다.)

주제

- AWS Lambda 함수 검사기
- JSON 스키마 유효성 검사기

AWS Lambda 함수 검사기

Lambda 함수 유효성 검사기는 아래와 같은 이벤트 스키마로 구성되어야 합니다. AWS AppConfig 는 이 스키마를 사용하여 Lambda 함수를 간접적으로 호출합니다. 내용은 base64로 인코딩된 문자열이고 URI는 문자열입니다.

```
{
    "applicationId": "The application ID of the configuration profile being
 validated",
```

```
    "configurationProfileId": "The ID of the configuration profile being validated",
    "configurationVersion": "The version of the configuration profile being validated",
    "content": "Base64EncodedByteString",
    "uri": "The configuration uri"
}
```

AWS AppConfig `X-Amz-Function-Error`Lambda 헤더가 응답에 설정되었는지 확인합니다. 함수에서 예외가 발생하는 경우 Lambda는 이 헤더를 설정합니다. `X-Amz-Function-Error`에 대한 자세한 내용은AWS Lambda 개발자 안내서의 AWS Lambda의 오류 처리 및 자동 재시도를 참조하십시오.

다음은 성공적인 유효성 검사를 위한 Lambda 응답 코드의 간단한 예입니다.

```
import json

def handler(event, context):
     #Add your validation logic here
     print("We passed!")
```

다음은 실패한 유효성 검사를 위한 Lambda 응답 코드의 간단한 예입니다.

```
def handler(event, context):
     #Add your validation logic here
     raise Exception("Failure!")
```

다음은 구성 파라미터가 소수인 경우에만 유효성을 검사하는 또 다른 예입니다.

```
function isPrime(value) {
    if (value < 2) {
        return false;
    }

    for (i = 2; i < value; i++) {
        if (value % i === 0) {
            return false;
        }
    }

    return true;
}

exports.handler = async function(event, context) {
```

```
    console.log('EVENT: ' + JSON.stringify(event, null, 2));
    const input = parseInt(Buffer.from(event.content, 'base64').toString('ascii'));
    const prime = isPrime(input);
    console.log('RESULT: ' + input + (prime ? ' is' : ' is not') + ' prime');
    if (!prime) {
        throw input + "is not prime";
    }
}
```

AWS AppConfig 및 API 작업을 호출할 때 검증 `StartDeployment` Lambda를 호출합니다. `ValidateConfigurationActivity` Lambda를 간접적으로 호출할 `appconfig.amazonaws.com` 권한을 제공해야 합니다. 자세한 내용은 서비스에 대한 함수 액세스 권한 부여를 참조하십시오. AWS AWS AppConfig 시작 지연 시간을 포함하여 Lambda 검증 실행 시간을 15초로 제한합니다.

JSON 스키마 유효성 검사기

SSM 문서에서 구성을 생성하는 경우 해당 구성에 대한 JSON 스키마를 지정하거나 생성해야 합니다. 템플릿은 각 애플리케이션 구성 설정에 대해 허용 가능한 속성을 정의하는 JSON 스키마입니다. 이 JSON 스키마는 규칙 세트와 비슷하게 새로운 구성 설정이나 업데이트된 구성 설정이 애플리케이션에서 요구하는 모범 사례를 따르는지 여부를 확인하는 역할을 합니다. 의 예는 다음과 같습니다.

```
    {
      "$schema": "http://json-schema.org/draft-04/schema#",
      "title": "$id$",
      "description": "BasicFeatureToggle-1",
      "type": "object",
      "additionalProperties": false,
      "patternProperties": {
          "[^\\s]+$": {
              "type": "boolean"
          }
      },
      "minProperties": 1
    }
```

SSM 문서에서 구성을 생성하면 시스템은 구성이 스키마 요구 사항을 준수하는지 자동으로 확인합니다. 준수하지 않으면 AWS AppConfig 는 유효성 검사 오류를 반환합니다.

> **Important**
>
> JSON 스키마 검사기에 대한 다음 중요 정보를 유념하십시오.

- SSM 문서에 저장된 구성 데이터는 시스템에 구성을 추가하기 전에 연결된 JSON 스키마에 대해 유효성을 검사해야 합니다. SSM 파라미터에는 검증 방법이 필요하지 않지만 를 사용하여 새 SSM 파라미터 구성이나 업데이트된 SSM 파라미터 구성에 대한 검증 검사를 생성하는 것이 좋습니다. AWS Lambda
- SSM 문서의 구성은 `ApplicationConfiguration` 문서 유형을 사용합니다. 해당 JSON 스키마는 `ApplicationConfigurationSchema` 문서 유형을 사용합니다.
- AWS AppConfig 인라인 스키마에 대한 JSON 스키마 버전 4.X를 지원합니다. 애플리케이션 구성에 다른 버전의 JSON 스키마가 필요한 경우 Lambda 유효성 검사기를 생성해야 합니다.

에서 기능 플래그 구성 프로필 생성 AWS AppConfig

기능 플래그를 사용하여 애플리케이션 내 기능을 활성화 또는 비활성화하거나 플래그 속성을 사용하여 애플리케이션 기능의 다양한 특성을 구성할 수 있습니다. AWS AppConfig 플래그와 플래그 속성에 대한 데이터 및 메타데이터가 포함된 기능 플래그 형식으로 기능 플래그 구성을 AWS AppConfig 호스팅된 구성 저장소에 저장합니다. AWS AppConfig 호스팅된 구성 저장소에 대한 자세한 내용은 AWS AppConfig 호스팅된 구성 저장소에 대한 정보 섹션을 참조하세요.

주제

- 기능 플래그 및 기능 플래그 구성 프로필 생성(콘솔)
- 기능 플래그 및 기능 플래그 구성 프로필 생성(명령줄)
- AWS.AppConfig.FeatureFlags에 대한 유형 참조

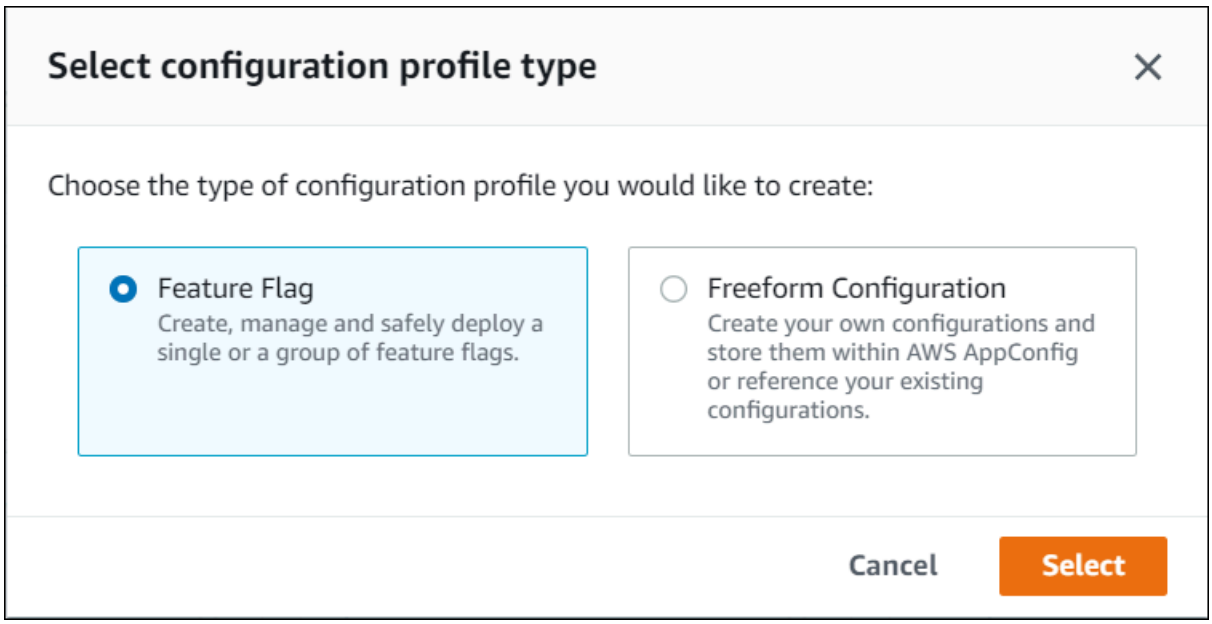
기능 플래그 및 기능 플래그 구성 프로필 생성(콘솔)

다음 절차에 따라 AWS AppConfig 콘솔을 사용하여 AWS AppConfig 기능 플래그 구성 프로필과 기능 플래그 구성을 만들 수 있습니다.

구성 프로파일을 생성하려면

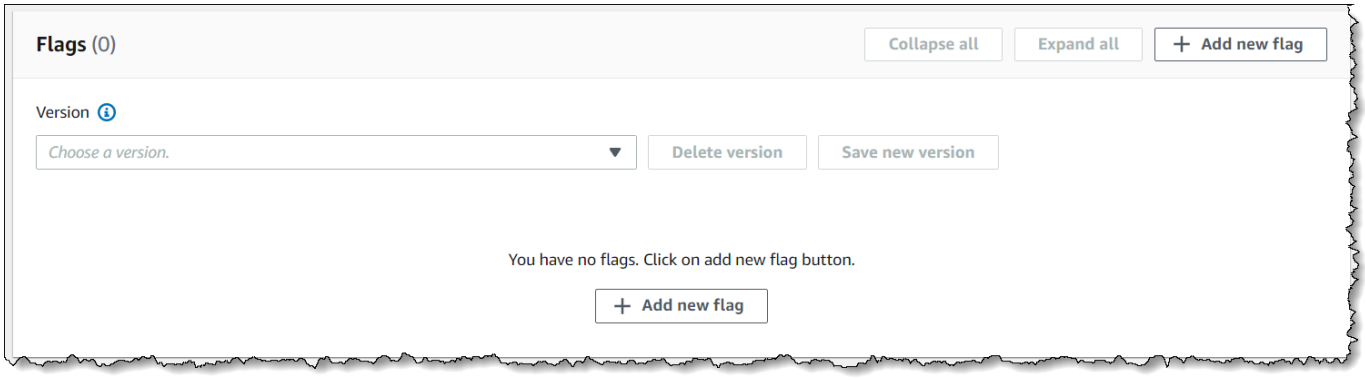
1. https://console.aws.amazon.com/systems-manager/appconfig/ 에서 AWS Systems Manager 콘솔을 엽니다.
2. 응용 프로그램 탭에서 구성 만들기에서 만든 응용 프로그램을 선택한 다음 AWS AppConfig 구성 프로필 및 기능 플래그 탭을 선택합니다.

3. 생성을 선택하세요.
4. 기능 플래그를 선택합니다.

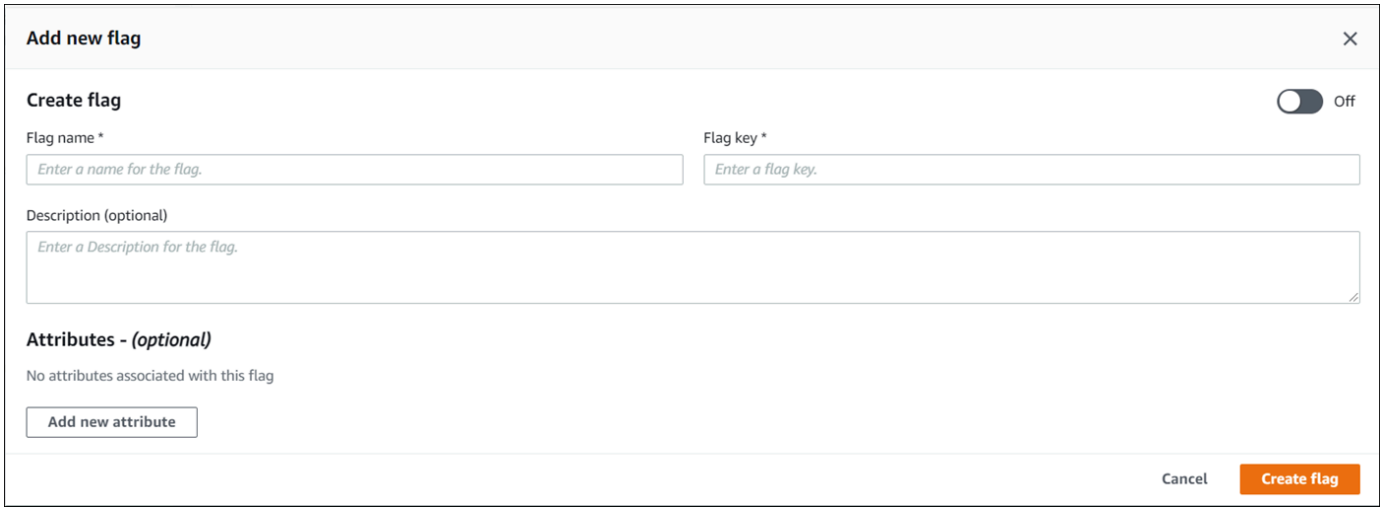


기능 플래그를 생성하려면

1. 생성한 구성에서 새 플래그 추가를 선택합니다.



2. 플래그 이름 및 설명(선택 사항)을 제공합니다. 플래그 키는 입력한 이름의 공백을 밑줄로 대체하여 자동으로 채워집니다. 다른 값이나 형식을 원하는 경우 플래그 키를 편집할 수 있습니다. 플래그를 생성한 후에는 플래그 이름은 편집할 수 있지만 플래그 키는 편집할 수 없습니다.



3. 토글 버튼을 사용하여 기능 플래그를 활성화할지 비활성화할지 지정합니다.
4. (선택 사항) 기능 플래그에 속성 및 속성 제약 조건을 추가합니다. 속성을 사용하면 플래그 내에 추가 값을 제공할 수 있습니다. 선택적으로 지정된 제약 조건에 대해 속성 값을 검증할 수 있습니다. 제약 조건은 예상치 못한 값이 애플리케이션에 배포되지 않도록 합니다.

   AWS AppConfig 기능 플래그는 다음 유형의 속성과 해당 제약 조건을 지원합니다.

유형	Constraint	설명
문자열	정규식	문자열의 Regex 패턴
	Enum	문자열에 사용할 수 있는 값 목록
숫자	최소	속성에 대한 최소 숫자 값
	Maximum	속성에 대한 최대 숫자 값
부울	None	None
문자열 배열	정규식	배열 요소의 정규식 패턴
	Enum	배열 요소에 사용할 수 있는 값 목록
숫자 배열	최소	배열 요소의 최소 숫자 값

유형	Constraint	설명
	Maximum	배열 요소의 최대 숫자 값

> **Note**
>
> 다음 정보를 참고하세요.
>
> - 속성 이름에서, “활성화”라는 단어는 예약어입니다. “활성화”라는 기능 플래그 속성은 생성할 수 없습니다. 다른 예약어는 없습니다.
> - 기능 플래그의 속성은 해당 플래그가 활성화된 경우에만 `GetLatestConfiguration` 응답에 포함됩니다.
> - 필수 값을 선택하여 속성 값이 필요한지 여부를 지정합니다.

5. 새 버전 저장을 선택합니다.

AWS AppConfig에서 기능 플래그 및 구성 데이터 배포 로 이동합니다.

기능 플래그 및 기능 플래그 구성 프로필 생성(명령줄)

다음 절차는 AWS Command Line Interface (Linux 또는 Windows) 또는 Windows용 도구를 사용하여 AWS AppConfig 기능 플래그 구성 프로필을 만드는 PowerShell 방법을 설명합니다. 원하는 경우 AWS CloudShell 을 사용하여 아래 나열된 명령을 실행할 수 있습니다. 자세한 내용은 AWS CloudShell사용 설명서의 AWS CloudShell 이란 무엇입니까? 섹션을 참조하십시오.

기능 플래그 구성을 단계별로 생성하려면

1. 를 엽니다 AWS CLI.
2. 유형을 `AWS.AppConfig.FeatureFlags`로 지정하여 기능 플래그 구성 프로필을 생성합니다. 구성 프로필은 위치 URI에 대해 `hosted`을 사용해야 합니다.

   Linux

```
aws appconfig create-configuration-profile \
  --application-id The_application_ID \
  --name A_name_for_the_configuration_profile \
  --location-uri hosted \
```

```
  --type AWS.AppConfig.FeatureFlags
```

Windows

```
aws appconfig create-configuration-profile ^
  --application-id The_application_ID ^
  --name A_name_for_the_configuration_profile ^
  --location-uri hosted ^
  --type AWS.AppConfig.FeatureFlags
```

PowerShell

```
New-APPCConfigurationProfile `
  -Name A_name_for_the_configuration_profile `
  -ApplicationId The_application_ID `
  -LocationUri hosted `
  -Type AWS.AppConfig.FeatureFlags
```

3. 기능 플래그 구성 데이터를 생성하십시오. 데이터는 JSON 형식이어야 하며 `AWS.AppConfig.FeatureFlags` JSON 스키마를 준수해야 합니다. 스키마에 대한 자세한 내용은 AWS.AppConfig.FeatureFlags에 대한 유형 참조 섹션을 참조하십시오.
4. `CreateHostedConfigurationVersion` API를 사용하여 기능 플래그 구성 데이터를 AWS AppConfig에 저장합니다.

   Linux

   ```
   aws appconfig create-hosted-configuration-version \
     --application-id The_application_ID \
     --configuration-profile-id The_configuration_profile_id \
     --content-type "application/json" \
     --content file://path/to/feature_flag_configuration_data \
     file_name_for_system_to_store_configuration_data
   ```

   Windows

   ```
   aws appconfig create-hosted-configuration-version ^
     --application-id The_application_ID ^
     --configuration-profile-id The_configuration_profile_id ^
     --content-type "application/json" ^
   ```

```
  --content file://path/to/feature_flag_configuration_data ^
  file_name_for_system_to_store_configuration_data
```

PowerShell

```
New-APPCHostedConfigurationVersion `
  -ApplicationId The_application_ID `
  -ConfigurationProfileId The_configuration_profile_id `
  -ContentType "application/json" `
  -Content file://path/to/feature_flag_configuration_data `
  file_name_for_system_to_store_configuration_data
```

다음은 Linux 샘플 명령입니다.

```
aws appconfig create-hosted-configuration-version \
  --application-id 1a2b3cTestApp \
  --configuration-profile-id 4d5e6fTestConfigProfile \
  --content-type "application/json" \
  --content Base64Content
```

content 파라미터는 다음과 같은 base64 인코딩된 데이터를 사용합니다.

```
{
  "flags": {
    "flagkey": {
      "name": "WinterSpecialBanner"
    }
  },
  "values": {
    "flagkey": {
      "enabled": true
    }
  },
  "version": "1"
}
```

시스템은 다음과 같은 정보를 반환합니다.

Linux

```
{
   "ApplicationId"          : "1a2b3cTestApp",
   "ConfigurationProfileId" : "4d5e6fTestConfigProfile",
   "VersionNumber"          : "1",
   "ContentType"            : "application/json"
}
```

Windows

```
{
   "ApplicationId"          : "1a2b3cTestApp",
   "ConfigurationProfileId" : "4d5e6fTestConfigProfile",
   "VersionNumber"          : "1",
   "ContentType"            : "application/json"
}
```

PowerShell

```
ApplicationId          : 1a2b3cTestApp
ConfigurationProfileId : 4d5e6fTestConfigProfile
VersionNumber          : 1
ContentType            : application/json
```

`service_returned_content_file`에는 AWS AppConfig 생성된 일부 메타데이터가 포함된 구성 데이터가 들어 있습니다.

> **Note**
>
> 호스팅된 구성 버전을 생성할 때 데이터가 `AWS.AppConfig.FeatureFlags` JSON 스키마를 AWS AppConfig 준수하는지 확인합니다. AWS AppConfig 또한 데이터의 각 기능 플래그 속성이 해당 속성에 대해 정의한 제약 조건을 충족하는지 확인합니다.

AWS.AppConfig.FeatureFlags에 대한 유형 참조

AWS.AppConfig.FeatureFlags JSON 스키마를 참조로 사용하여 기능 플래그 구성 데이터를 생성합니다.

```
{
      "$schema": "http://json-schema.org/draft-07/schema#",
      "definitions": {
        "flagSetDefinition": {
          "type": "object",
          "properties": {
            "version": {
              "$ref": "#/definitions/flagSchemaVersions"
            },
            "flags": {
              "$ref": "#/definitions/flagDefinitions"
            },
            "values": {
              "$ref": "#/definitions/flagValues"
            }
          },
          "required": ["version", "flags"],
          "additionalProperties": false
        },
        "flagDefinitions": {
          "type": "object",
          "patternProperties": {
            "^[a-z][a-zA-Z\\d-]{0,63}$": {
              "$ref": "#/definitions/flagDefinition"
            }
          },
          "maxProperties": 100,
          "additionalProperties": false
        },
        "flagDefinition": {
          "type": "object",
          "properties": {
            "name": {
              "$ref": "#/definitions/customerDefinedName"
            },
            "description": {
              "$ref": "#/definitions/customerDefinedDescription"
            },
```

```
            "_createdAt": {
              "type": "string"
            },
            "_updatedAt": {
              "type": "string"
            },
            "_deprecation": {
              "type": "object",
              "properties": {
                "status": {
                  "type": "string",
                  "enum": ["planned"]
                }
              },
             "additionalProperties": false
            },
            "attributes": {
              "$ref": "#/definitions/attributeDefinitions"
            }
          },
          "additionalProperties": false
        },
        "attributeDefinitions": {
          "type": "object",
          "patternProperties": {
            "^[a-z][a-zA-Z\\d-_]{0,63}$": {
              "$ref": "#/definitions/attributeDefinition"
            }
          },
          "maxProperties": 25,
          "additionalProperties": false
        },
        "attributeDefinition": {
          "type": "object",
          "properties": {
            "description": {
              "$ref": "#/definitions/customerDefinedDescription"
            },
            "constraints": {
              "oneOf": [
                { "$ref": "#/definitions/numberConstraints" },
                { "$ref": "#/definitions/stringConstraints" },
                { "$ref": "#/definitions/arrayConstraints" },
                { "$ref": "#/definitions/boolConstraints" }
```

```
            ]
          }
        },
        "additionalProperties": false
      },
      "flagValues": {
        "type": "object",
        "patternProperties": {
          "^[a-z][a-zA-Z\\d-_]{0,63}$": {
            "$ref": "#/definitions/flagValue"
          }
        },
        "maxProperties": 100,
        "additionalProperties": false
      },
      "flagValue": {
        "type": "object",
        "properties": {
          "enabled": {
            "type": "boolean"
          },
          "_createdAt": {
            "type": "string"
          },
          "_updatedAt": {
            "type": "string"
          }
        },
        "patternProperties": {
          "^[a-z][a-zA-Z\\d-_]{0,63}$": {
            "$ref": "#/definitions/attributeValue",
            "maxProperties": 25
          }
        },
        "required": ["enabled"],
        "additionalProperties": false
      },
      "attributeValue": {
        "oneOf": [
          { "type": "string", "maxLength": 1024 },
          { "type": "number" },
          { "type": "boolean" },
          {
            "type": "array",
```

```
            "oneOf": [
              {
                "items": {
                  "type": "string",
                  "maxLength": 1024
                }
              },
              {
                "items": {
                  "type": "number"
                }
              }
            ]
          }
        ],
        "additionalProperties": false
      },
      "stringConstraints": {
        "type": "object",
        "properties": {
          "type": {
            "type": "string",
            "enum": ["string"]
    },
          "required": {
            "type": "boolean"
          },
          "pattern": {
            "type": "string",
            "maxLength": 1024
          },
          "enum": {
            "type": "array",
            "maxLength": 100,
            "items": {
              "oneOf": [
                {
                  "type": "string",
                  "maxLength": 1024
                },
                {
                  "type": "integer"
                }
              ]
```

```
            }
          }
        },
        "required": ["type"],
        "not": {
          "required": ["pattern", "enum"]
        },
        "additionalProperties": false
      },
      "numberConstraints": {
        "type": "object",
        "properties": {
          "type": {
            "type": "string",
            "enum": ["number"]
    },
          "required": {
            "type": "boolean"
          },
          "minimum": {
            "type": "integer"
          },
          "maximum": {
            "type": "integer"
          }
        },
        "required": ["type"],
        "additionalProperties": false
      },
      "arrayConstraints": {
        "type": "object",
        "properties": {
          "type": {
            "type": "string",
            "enum": ["array"]
    },
          "required": {
            "type": "boolean"
          },
          "elements": {
            "$ref": "#/definitions/elementConstraints"
    }
        },
        "required": ["type"],
```

```
          "additionalProperties": false
        },
        "boolConstraints": {
          "type": "object",
          "properties": {
            "type": {
              "type": "string",
              "enum": ["boolean"]
      },
            "required": {
              "type": "boolean"
            }
          },
          "required": ["type"],
          "additionalProperties": false
        },
        "elementConstraints": {
          "oneOf": [
            { "$ref": "#/definitions/numberConstraints" },
            { "$ref": "#/definitions/stringConstraints" }
          ]
        },
        "customerDefinedName": {
          "type": "string",
          "pattern": "^[^\\n]{1,64}$"
        },
        "customerDefinedDescription": {
          "type": "string",
          "maxLength": 1024
        },
        "flagSchemaVersions": {
          "type": "string",
          "enum": ["1"]
        }
      },
      "type": "object",
      "$ref": "#/definitions/flagSetDefinition",
      "additionalProperties": false
    }
```

> **Important**
>
> 기능 플래그 구성 데이터를 검색하려면 애플리케이션이 `GetLatestConfiguration` API를 호출해야 합니다. 더 이상 사용되지 않는 `GetConfiguration` 호출로는 기능 플래그 구성 데이터를 검색할 수 없습니다. 자세한 내용은 AWS AppConfig API GetLatestConfiguration참조를 참조하십시오.

애플리케이션이 새로 배포된 구성을 GetLatestConfiguration호출하고 수신하면 기능 플래그와 속성을 정의하는 정보가 제거됩니다. 단순화된 JSON에는 지정한 각 플래그 키와 일치하는 키 맵이 포함되어 있습니다. 단순화된 JSON에는 `enabled` 속성의 매핑된 `true` 또는 `false` 값도 포함됩니다. 플래그가 `enabled`을 `true`로 설정하면 플래그의 모든 속성도 표시됩니다. 다음 JSON 스키마는 JSON 출력 형식을 설명합니다.

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "patternProperties": {
    "^[a-z][a-zA-Z\\d-_]{0,63}$": {
      "$ref": "#/definitions/attributeValuesMap"
    }
  },
  "maxProperties": 100,
  "additionalProperties": false,
  "definitions": {
    "attributeValuesMap": {
      "type": "object",
      "properties": {
        "enabled": {
          "type": "boolean"
        }
      },
      "required": ["enabled"],
      "patternProperties": {
        "^[a-z][a-zA-Z\\d-_]{0,63}$": {
          "$ref": "#/definitions/attributeValue"
        }
      },
      "maxProperties": 25,
      "additionalProperties": false
    },
```

```
    "attributeValue": {
      "oneOf": [
        { "type": "string","maxLength": 1024 },
        { "type": "number" },
        { "type": "boolean" },
        {
          "type": "array",
          "oneOf": [
            {
              "items": {
                "oneOf": [
                  {
                    "type": "string",
                    "maxLength": 1024
                  }
                ]
              }
            },
            {
              "items": {
                "oneOf": [
                  {
                    "type": "number"
                  }
                ]
              }
            }
          ]
        }
      ],
      "additionalProperties": false
    }
  }
}
```

에서 자유 형식 구성 프로필 생성 AWS AppConfig

구성 프로필에는 무엇보다도 저장된 위치에서 구성 데이터를 찾을 수 있는 URI와 프로필 유형이 포함됩니다. AWS AppConfig AWS AppConfig 기능 플래그와 자유 형식 구성이라는 두 가지 구성 프로파일 유형을 지원합니다. 기능 플래그 구성 프로파일은 해당 데이터를 AWS AppConfig 호스팅된 구성 저장소에 저장하며 URI는 간단합니다. `hosted` 자유 형식 구성 프로파일의 경우 AWS AppConfig 호스팅된

구성 저장소 또는 다음 AWS 서비스 및 Systems Manager 기능 중 하나에 데이터를 저장할 수 있습니다.

위치	지원되는 파일 형식
AWS AppConfig 호스팅된 구성 저장소	를 사용하여 추가한 경우 YAML, JSON 및 텍스트 AWS Management Console AWS AppConfig CreateHostedConfigurationVersionAPI 작업을 사용하여 추가한 경우 모든 파일 유형
Amazon Simple Storage Service(S3)	모두
AWS CodePipeline	파이프라인(서비스가 정의함)
AWS Secrets Manager	보안 암호(서비스가 정의함)
AWS Systems Manager 파라미터 스토어	표준 및 보안 문자열 파라미터(파라미터 저장소에서 정의)
AWS Systems Manager 문서 저장소 (SSM 문서)	YAML, JSON, 텍스트

구성 프로필에는 구성 데이터가 구문 및 의미상 정확한지 확인하는 선택적 유효성 검사기도 포함될 수 있습니다. AWS AppConfig 배포를 시작할 때 유효성 검사기를 사용하여 검사를 수행합니다. 오류가 발견되면 구성의 대상을 변경하기 전에 배포가 중지됩니다.

> **Note**
>
> 가능한 경우 대부분의 기능과 향상된 기능을 제공하는 AWS AppConfig 호스팅된 구성 저장소에서 구성 데이터를 호스팅하는 것이 좋습니다.

AWS AppConfig 호스팅된 구성 저장소 또는 SSM 문서에 저장된 자유 형식 구성의 경우 구성 프로파일을 생성할 때 Systems Manager 콘솔을 사용하여 자유 형식 구성을 생성할 수 있습니다. 이 프로세스는 이 주제의 뒷부분에서 설명합니다.

파라미터 스토어, Secrets Manager 또는 Amazon S3에 저장된 자유 형식 구성의 경우 먼저 파라미터, 암호 또는 객체를 생성하여 관련 구성 저장소에 저장해야 합니다. 파라미터 데이터를 저장한 후 이 주제의 절차를 사용하여 구성 프로필을 생성할 수 있습니다.

주제

- 구성 저장소 할당량 및 제한
- AWS AppConfig 호스팅된 구성 저장소에 대한 정보
- Amazon S3에 저장되는 구성 정보
- 자유 형식 구성 및 구성 프로필 생성

구성 저장소 할당량 및 제한

에서 지원하는 구성 AWS AppConfig 저장소에는 다음과 같은 할당량 및 제한이 있습니다.

	AWS AppConfig 호스팅된 구성 저장소	Amazon S3	Systems Manager Parameter Store	AWS Secrets Manager	Systems Manager Document 저장소	AWS CodePipeline
구성 크기 제한	기본값은 2MB, 최대 4MB	2MB S3가 아닌 에 의해 AWS AppConfig 시행됨	4KB(프리 티어)/8KB(고급 파라미터)	64KB	64KB	2MB 에 의해 AWS AppConfig 시행됨, 아니요 CodePipeline
리소스 스토리지 제한	1GB	무제한	10,000개 파라미터(프리 티어)/100,000개 파라미터(고급 파라미터)	500,000	500개 문서	애플리케이션당 구성 프로필 수로 제한됩니다.(애플리케이션당 프로필 100개)

	AWS AppConfig 호스팅된 구성 저장소	Amazon S3	Systems Manager Parameter Store	AWS Secrets Manager	Systems Manager Document 저장소	AWS CodePipeline
서버 측 암호화	예	SSE-S3, SSE-KMS	예	예	아니요	예
AWS CloudFormation 지원	예	데이터 생성 또는 업데이트용 아님	예	예	아니요	예
요금	무료	Amazon S3 요금을 참조하십시오.	AWS Systems Manager 요금 참조	AWS Secrets Manager 요금 참조	무료	AWS CodePipeline 요금 참조

AWS AppConfig 호스팅된 구성 저장소에 대한 정보

AWS AppConfig 내부 또는 호스팅된 구성 저장소가 포함됩니다. 구성은 2MB 이하여야 합니다. AWS AppConfig 호스팅된 구성 저장소는 다른 구성 저장소 옵션에 비해 다음과 같은 이점을 제공합니다.

- Amazon Simple Storage Service(S3) 또는 파라미터 스토어와 같은 다른 서비스를 설정하고 구성할 필요가 없습니다.
- 구성 저장소를 사용하기 위해 구성 AWS Identity and Access Management (IAM) 권한을 사용할 필요는 없습니다.
- 구성을 YAML, JSON 또는 텍스트 문서로 저장할 수 있습니다.
- 저장소는 사용이 무료입니다.
- 구성을 생성한 후 구성 프로필을 생성할 때 저장소에 추가할 수 있습니다.

Amazon S3에 저장되는 구성 정보

구성을 Amazon Simple Storage Service(S3) 버킷에 저장할 수 있습니다. 구성 프로필을 만들 때 URI를 버킷의 단일 S3 객체에 지정합니다. 또한 객체를 가져올 AWS AppConfig 권한을 부여하는 (IAM) 역

할의 Amazon 리소스 이름 AWS Identity and Access Management (ARN) 을 지정합니다. Amazon S3 객체에 대한 구성 프로필을 만들기 전에 다음 제한 사항에 유의하십시오.

제한	Details
용량	S3 객체로 저장된 구성의 용량은 최대 1MB입니다.
객체 암호화	구성 프로필은 SSE-S3 및 SSE-KMS로 암호화된 객체를 대상으로 할 수 있습니다.
스토리지 클래스	AWS AppConfig `STANDARD`,, `INTELLIGENT_TIERING` `REDUCED_REDUNDANCY` `STANDARD_IA` , 및 같은 S3 스토리지 클래스를 지원합니다. `ONEZONE_IA` 모든 S3 Glacier 클래스(`GLACIER` 및 `DEEP_ARCHIVE`)는 지원되지 않습니다.
버전 관리	AWS AppConfig S3 객체가 버전 관리를 사용해야 합니다.

Amazon S3 객체로 저장된 구성에 대한 권한 구성

S3 객체로 저장된 구성에 대한 구성 프로필을 생성할 때는 객체를 가져올 AWS AppConfig 권한을 부여하는 IAM 역할에 ARN을 지정해야 합니다. 이 역할에는 다음 권한이 포함되어야 합니다.

S3 객체에 액세스할 수 있는 권한

- s3: GetObject
- s3: GetObjectVersion

S3 버킷을 나열할 수 있는 권한

s3: ListAllMyBuckets

객체가 저장된 S3 버킷에 액세스할 수 있는 권한

- s3: GetBucketLocation
- s3: GetBucketVersioning

- s3: ListBucket
- s3: ListBucketVersions

다음 절차를 완료하여 S3 객체에 구성을 저장할 수 있는 역할을 생성하십시오. AWS AppConfig

S3 객체 액세스를 위한 IAM 정책 생성

다음 절차를 사용하여 S3 객체에 구성을 저장할 수 있는 IAM 정책을 생성합니다. AWS AppConfig

S3 객체 액세스를 위한 IAM 정책을 생성하려면

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 탐색 창에서 정책을 선택한 후 정책 생성을 선택합니다.
3. 정책 생성 페이지에서 JSON 탭을 선택합니다.
4. 다음 샘플 정책을 S3 버킷 및 구성 객체에 대한 정보로 업데이트합니다. 그런 다음 JSON 탭의 텍스트 필드에 정책을 붙여 넣습니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:GetObjectVersion"
      ],
      "Resource": "arn:aws:s3:::my-bucket/my-configurations/my-configuration.json"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetBucketLocation",
        "s3:GetBucketVersioning",
        "s3:ListBucketVersions",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::my-bucket"
      ]
    },
    {
```

```
      "Effect": "Allow",
      "Action": "s3:ListAllMyBuckets",
      "Resource": "*"
    }
  ]
}
```

5. 정책 검토를 선택합니다.
6. 정책 검토 페이지에서 이름 상자에 이름을 입력한 후 설명을 입력합니다.
7. 정책 생성(Create policy)을 선택합니다. 그러면 역할 페이지로 돌아갑니다.

S3 객체 액세스를 위한 IAM 역할 생성

다음 절차를 사용하여 S3 객체에 저장된 구성을 가져올 수 있는 IAM 역할을 생성합니다. AWS AppConfig

Amazon S3 객체 액세스를 위한 IAM 역할을 생성하려면

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 탐색 창에서 역할(Roles)을 선택한 후 역할 생성(Create role)을 선택합니다.
3. 신뢰할 수 있는 유형의 엔터티 선택 섹션에서 AWS 서비스를 선택합니다.
4. 사용 사례 선택 섹션의 일반 사용 사례에서 EC2를 선택하고 다음: 권한을 선택합니다.
5. 권한 정책 연결 페이지의 검색 상자에 이전 절차에서 만든 정책의 이름을 입력합니다.
6. 정책을 선택한 후 다음: 태그를 선택합니다.
7. 태그 추가(선택 사항) 페이지에서 키와 선택적 값을 입력하고 다음: 검토를 선택합니다.
8. 검토 페이지에서 역할 이름 필드에 이름을 입력한 후 설명을 입력합니다.
9. 역할 생성을 선택합니다. 그러면 역할 페이지로 돌아갑니다.
10. 역할 페이지에서 방금 만든 역할을 선택하여 요약 페이지를 엽니다. 역할 이름 및 역할 ARN을 메모합니다. 이 주제의 뒷부분에서 구성 프로필을 생성할 때 역할 ARN을 지정하게 됩니다.

신뢰 관계 생성

다음 절차에 따라 생성한 역할이 AWS AppConfig를 신뢰하도록 구성합니다.

신뢰 관계를 추가하려면

1. 방금 생성한 역할에 대한 요약 페이지에서 신뢰 관계 탭을 선택한 후 신뢰 관계 편집을 선택합니다.
2. 다음 예제에 표시된 대로 "ec2.amazonaws.com"을 삭제하고 "appconfig.amazonaws.com"을 추가합니다.

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

3. 신뢰 정책 업데이트를 선택합니다.

자유 형식 구성 및 구성 프로필 생성

이 섹션에서는 자유 형식 구성 및 구성 프로필을 생성하는 방법을 설명합니다. 시작하기 전에 다음 정보를 유념합니다.

- 다음 절차를 수행하려면 선택한 구성 저장소의 구성 데이터에 AWS AppConfig 가 액세스할 수 있도록 IAM 서비스 역할을 지정해야 합니다. AWS AppConfig 호스팅된 구성 스토어를 사용하는 경우에는 이 역할이 필요하지 않습니다. S3, Parameter Store, 또는 Systems Manager 문서 저장소를 선택하는 경우, 기존 IAM 역할을 선택하거나 시스템에서 자동으로 역할을 생성하도록 옵션을 선택해야 합니다. 자세한 내용은 구성 프로필 IAM 역할에 대한 정보 섹션을 참조하십시오.
- S3에 저장된 구성에 대한 구성 프로필을 생성하려면 권한을 구성해야 합니다. S3를 구성 저장소로 사용하기 위한 권한 및 기타 요구 사항에 대한 자세한 내용은 Amazon S3에 저장되는 구성 정보 섹션을 참조하십시오.
- 유효성 검사기를 사용하려는 경우 이를 사용하기 위한 세부 정보 및 요구 사항을 검토하십시오. 자세한 정보는 유효성 검사기에 대한 정보을 참조하세요.

주제

- AWS AppConfig 자유형 구성 프로파일 생성 (콘솔)
- AWS AppConfig 자유 형식 구성 프로필 생성 (명령줄)

AWS AppConfig 자유형 구성 프로파일 생성 (콘솔)

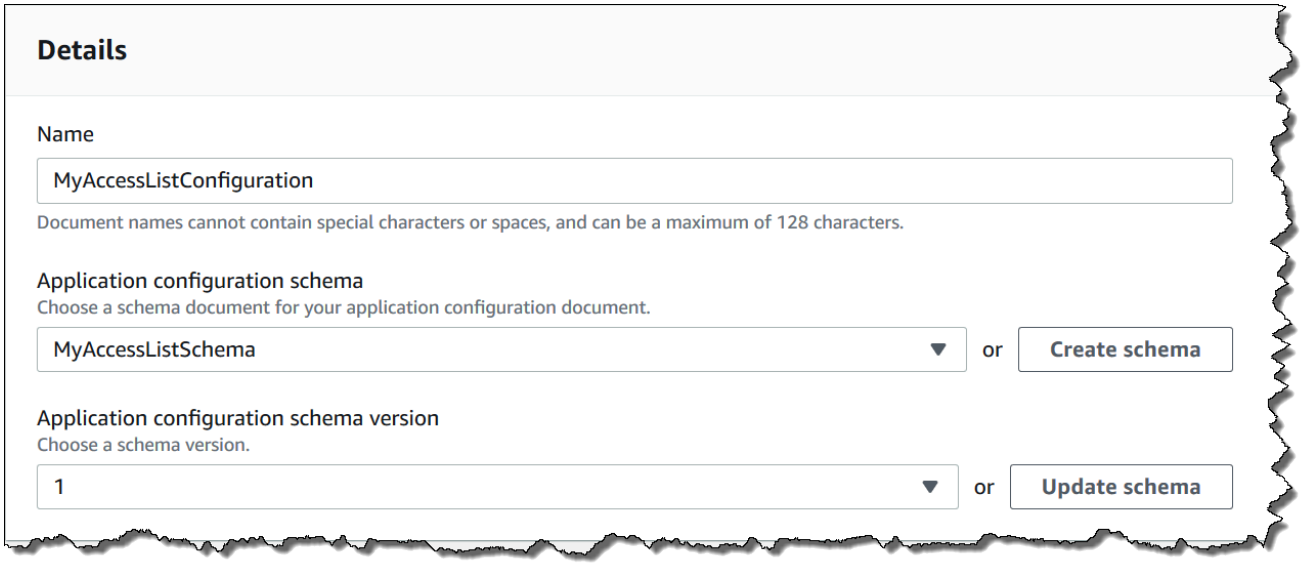
다음 절차에 따라 콘솔을 사용하여 AWS AppConfig 자유형 구성 프로파일을 생성하고 (선택 사항) 자유형 구성을 만들 수 있습니다. AWS Systems Manager

구성 프로파일을 생성하려면

1. https://console.aws.amazon.com/systems-manager/appconfig/ 에서 콘솔을 엽니다. AWS Systems Manager
2. 애플리케이션 탭에서 애플리케이션을 선택한 다음 구성 프로파일 및 기능 플래그 탭을 선택합니다.
3. 생성을 선택하세요.
4. Freeform 구성을 선택한 다음 선택을 선택합니다.
5. 이름에 구성 프로필의 이름을 입력합니다.
6. 설명에 구성 프로필에 대한 정보를 입력합니다.
7. 구성 소스 섹션에서 옵션을 선택합니다.
8.
   - AWS AppConfig 호스팅된 구성을 선택한 경우 텍스트, JSON 또는 YAML을 선택하고 필드에 구성을 입력합니다. 다음을 선택하고 이 절차의 10단계로 이동합니다.
   - Amazon S3 객체를 선택한 경우, S3 객체 원본 필드에 객체 URI를 입력한 후 [Next] 를 선택합니다.
   - AWS Systems Manager 파라미터를 선택한 경우 목록에서 파라미터 이름을 선택합니다. 다음을 선택합니다.
   - Secrets Manager 암호를 선택한 경우 암호의 이름을 입력합니다. 다음을 선택합니다.
   - AWS CodePipeline을 선택한 경우 다음을 선택하고 이 절차의 10단계로 이동합니다.
   - AWS Systems Manager 문서를 선택한 경우 다음 단계를 완료합니다.

   a. 문서 소스 섹션에서 저장된 문서 또는 새 문서를 선택합니다.
   b. 저장된 문서를 선택한 경우 목록에서 SSM 문서를 선택합니다. 새 문서를 선택한 경우 세부 정보 및 콘텐츠 섹션이 나타납니다.
   c. 세부 정보 섹션에서 새 애플리케이션 구성의 이름을 입력합니다.

d. 애플리케이션 구성 스키마 섹션의 경우 목록을 사용하여 JSON 스키마를 선택하거나 스키마 생성을 선택합니다. 스키마 생성을 선택하면 Systems Manager가 스키마 생성 페이지를 엽니다. 콘텐츠 섹션에 스키마 세부 정보를 입력한 다음 스키마 생성을 선택합니다.



e. 애플리케이션 구성 스키마 버전의 경우 목록에서 버전을 선택하거나 스키마 업데이트를 선택하여 스키마를 편집하고 새 버전을 생성합니다.

f. 콘텐츠 섹션에서 YAML 또는 JSON을 선택한 다음 필드에 구성 데이터를 입력합니다.

g. 다음을 선택합니다.

9. 서비스 역할 섹션에서 새 서비스 역할을 선택하여 구성 데이터에 대한 액세스를 제공하는 IAM 역할을 AWS AppConfig 생성합니다. AWS AppConfig 이전에 입력한 이름을 기반으로 역할 이름 필드를 자동으로 채웁니다. 또는 IAM에 이미 존재하는 역할을 선택하려면 기존 서비스 역할을 선택합니다. 역할 ARN 목록을 사용하여 역할을 선택합니다.

10. 유효성 검사기 추가 페이지에서 JSON 스키마 또는 AWS Lambda을 선택합니다. JSON 스키마를 선택하는 경우, 필드에 JSON 스키마를 입력합니다. AWS Lambda를 선택한 경우 목록에서 함수의 Amazon 리소스 이름(ARN)과 버전을 선택합니다.

> **Important**
>
> SSM 문서에 저장된 구성 데이터는 시스템에 구성을 추가하기 전에 연결된 JSON 스키마에 대해 유효성을 검사해야 합니다. SSM 파라미터에는 검증 방법이 필요하지 않지만 를 사용하여 새 SSM 파라미터 구성이나 업데이트된 SSM 파라미터 구성에 대한 검증 검사를 생성하는 것이 좋습니다. AWS Lambda

11. (선택 사항) 태그 섹션에 키와 선택적 값을 입력합니다. 하나의 리소스에 대해 최대 50개의 태그를 지정할 수 있습니다.
12. 구성 프로필 생성을 선택합니다.

> **Important**
>
> 에 대한 구성 프로필을 만든 경우 AWS CodePipeline, 다음 섹션에 설명된 대로 배포 전략을 만든 후에는 배포 AWS AppConfig 공급자로 CodePipeline 지정하는 파이프라인을 만들어야 합니다. 배포 AWS AppConfig 공급자로 지정하는 파이프라인을 생성하는 방법에 대한 자세한 내용은 사용 설명서의 자습서: 배포 AWS AppConfig 공급자로 사용하는 파이프라인 만들기를 AWS CodePipeline 참조하십시오.

AWS AppConfig에서 기능 플래그 및 구성 데이터 배포 로 이동합니다.

AWS AppConfig 자유 형식 구성 프로필 생성 (명령줄)

다음 절차는 AWS CLI (Linux 또는 Windows) 를 사용하거나 AWS AppConfig 자유형 구성 AWS Tools for PowerShell 프로필을 만드는 방법을 설명합니다. 원하는 경우 아래 나열된 명령을 AWS CloudShell 사용하여 실행할 수 있습니다. 자세한 내용은 AWS CloudShell사용 설명서의 AWS CloudShell 이란 무엇입니까? 섹션을 참조하십시오.

> **Note**
>
> 호스팅된 구성 저장소에서 AWS AppConfig 호스팅되는 자유 형식 구성의 `hosted` 경우 위치 URI를 지정합니다.

단계별 구성 프로필 생성

1. 를 엽니다. AWS CLI
2. 다음 명령을 실행하여 자유 형식 구성 프로필을 생성합니다.

   Linux

```
aws appconfig create-configuration-profile \
  --application-id The_application_ID \
  --name A_name_for_the_configuration_profile \
  --description A_description_of_the_configuration_profile \
```

```
  --location-uri A_URI_to_locate_the_configuration or hosted \
  --retrieval-role-
arn The_ARN_of_the_IAM_role_with_permission_to_access_the_configuration_at_the_specified
 \
  --tags User_defined_key_value_pair_metadata_of_the_configuration_profile \
  --validators "Content=JSON_Schema_content_or_the_ARN_of_an_AWS
 Lambda_function,Type=JSON_SCHEMA or LAMBDA"
```

Windows

```
aws appconfig create-configuration-profile ^
  --application-id The_application_ID ^
  --name A_name_for_the_configuration_profile ^
  --description A_description_of_the_configuration_profile ^
  --location-uri A_URI_to_locate_the_configuration or hosted  ^
  --retrieval-role-
arn The_ARN_of_the_IAM_role_with_permission_to_access_the_configuration_at_the_specified
 ^
  --tags User_defined_key_value_pair_metadata_of_the_configuration_profile ^
  --validators "Content=JSON_Schema_content_or_the_ARN_of_an_AWS
 Lambda_function,Type=JSON_SCHEMA or LAMBDA"
```

PowerShell

```
New-APPCConfigurationProfile `
  -Name A_name_for_the_configuration_profile `
  -ApplicationId The_application_ID `
  -Description Description_of_the_configuration_profile `
  -LocationUri A_URI_to_locate_the_configuration or hosted `
  -
RetrievalRoleArn The_ARN_of_the_IAM_role_with_permission_to_access_the_configuration_at_
 `
  -
Tag Hashtable_type_user_defined_key_value_pair_metadata_of_the_configuration_profile
 `
  -Validators "Content=JSON_Schema_content_or_the_ARN_of_an_AWS
 Lambda_function,Type=JSON_SCHEMA or LAMBDA"
```

> **Note**
>
> AWS AppConfig 호스팅된 구성 저장소에서 구성을 생성한 경우 CreateHostedConfigurationVersionAPI 작업을 사용하여 새 버전의 구성을 만들 수 있습니다. 이 API 작업에 대한 AWS CLI 세부 정보 및 샘플 명령을 보려면 AWS CLI 명령 참조를 참조하십시오 create-hosted-configuration-version.

구성 데이터의 기타 소스

이 항목에는 와 통합되는 다른 AWS 서비스에 대한 정보가 포함되어 AWS AppConfig있습니다.

AWS AppConfig 통합: AWS Secrets Manager

Secrets Manager는 데이터베이스와 다른 서비스의 자격 증명을 안전하게 암호화, 저장 및 검색하는 데 효과적입니다. 앱에서 자격 증명을 하드코딩하는 대신 필요할 때마다 Secrets Manager를 호출하여 자격 증명을 검색할 수 있습니다. Secrets Manager를 사용하면 암호에 대한 액세스를 교체 및 관리할 수 있으므로 IT 리소스 및 데이터에 대한 액세스를 보호할 수 있습니다.

자유 형식 구성 프로필을 만들 때 Secrets Manager를 구성 데이터의 소스로 선택할 수 있습니다. 구성 프로필을 생성하기 전에 Secrets Manager에 온보딩하고 암호를 생성해야 합니다. Secrets Manager에 대한 자세한 내용은 무엇입니까 AWS Secrets Manager? 를 참조하십시오. AWS Secrets Manager 사용 설명서에서. Secrets Manager를 사용하는 구성 프로필을 만드는 방법에 대한 자세한 내용은 기능 플래그 및 자유 형식 구성 데이터 생성 AWS AppConfig을 참조하십시오.

AWS AppConfig에서 기능 플래그 및 구성 데이터 배포

기능 플래그 및 자유 형식 구성 데이터를 사용하는 데 필요한 아티팩트를 생성한 후 새 배포를 만들 수 있습니다. 새로운 배포를 생성할 때 다음 정보를 지정할 수 있습니다.

- 애플리케이션 ID
- 구성 프로필 ID
- 구성 버전
- 구성 데이터를 배포할 환경 ID
- 변경 사항을 얼마나 빨리 적용할지를 정의하는 배포 전략 ID
- 고객 관리 키를 사용하여 데이터를 암호화하기 위한 AWS Key Management Service (AWS KMS) 키 ID.

StartDeployment API 작업을 호출하면 AWS AppConfig는 다음 작업을 수행합니다.

1. 구성 프로필의 위치 URI를 사용하여 기본 데이터 저장소에서 구성 데이터를 검색합니다.
2. 구성 프로필을 생성할 때 지정한 유효성 검사기를 사용하여 구성 데이터가 구문상 및 의미상 올바른지 확인합니다.
3. 애플리케이션에서 검색할 수 있도록 데이터 사본을 캐시합니다. 이 캐시된 복사본을 배포된 데이터라고 합니다.

AWS AppConfig는 Amazon CloudWatch와 통합하여 배포를 모니터링합니다. 배포로 인해 CloudWatch에서 경보가 트리거되는 경우 AWS AppConfig는 배포를 자동으로 롤백하여 애플리케이션 사용자에게 미치는 영향을 최소화합니다.

주제

- 배포 전략 작업
- 구성 배포
- CodePipeline과 AWS AppConfig 배포 통합

배포 전략 작업

배포 전략을 사용하면 몇 분 또는 몇 시간에 걸쳐 프로덕션 환경에 변경 사항을 천천히 릴리스할 수 있습니다. AWS AppConfig 배포 전략은 다음과 같이 구성 배포의 중요한 측면을 정의합니다.

<table>
<tr><th>설정</th><th>설명</th></tr>
<tr><td>배포 유형</td><td>배포 유형은 구성이 배포 또는 롤아웃되는 방법을 정의합니다. AWS AppConfig는 선형 및 지수 배포 유형을 지원합니다.

• 선형: 이 유형의 경우 AWS AppConfig는 배포 동안 균등하게 분산된 성장 계수의 증분에 따라 배포를 처리합니다. 다음은 20% 선형 성장을 사용하는 10시간 배포 타임라인의 예시입니다.

<table>
<tr><th>경과 시간</th><th>배포 진행 중</th></tr>
<tr><td>0시간</td><td>0%</td></tr>
<tr><td>2시간</td><td>20%</td></tr>
<tr><td>4시간</td><td>40%</td></tr>
<tr><td>6시간</td><td>60%</td></tr>
<tr><td>8시간</td><td>80%</td></tr>
<tr><td>10시간</td><td>100%</td></tr>
</table>
• 지수: 이 유형의 경우 AWS AppConfig는 <code>G*(2^N)</code> 공식을 사용하여 배포를 기하급수적으로 처리합니다. 이 공식에서 <code>G</code>는 사용자가 지정한 단계 비율이며 <code>N</code>은 구성이 모든 대상에 배포될 때까지의 단계 수입니다. 예를 들어, 성장 계수를 2로 지정하면 시스템은 다음과 같이 구성을 롤아웃합니다.

<pre>2*(2^0)
2*(2^1)
2*(2^2)</pre></td></tr>
</table>

설정	설명
	배포를 수치적으로 표현하면 배포는 대상의 2%, 대상의 4%, 대상의 8% 등 이런 방식으로 구성이 모든 대상에 배포될 때까지 롤아웃됩니다.
단계 비율(성장 계수)	이 설정은 배포의 각 단계에서 대상으로 지정할 호출자의 백분율을 지정합니다. **Note** SDK 및 AWS AppConfig API 참조에서 `step percentage` 를 `growth factor`라고 합니다.
배포 시간	이 설정은 AWS AppConfig이 호스트에 배포되는 시간을 지정합니다. 이는 제한 시간 값이 아닙니다. 배포가 간격에 따라 처리되는 동안의 시간입니다.
베이크 소요 시간	이 세팅은 구성이 대상의 100%에 배포된 후 배포가 완료된 것으로 간주하기 전에 AWS AppConfig가 Amazon CloudWatch 경보를 모니터링하는 시간을 지정합니다. 이 시간 동안 경보가 트리거되면 AWS AppConfig이 배포를 롤백합니다. CloudWatch 경보에 따라 AWS AppConfig가 롤백할 수 있는 권한을 구성해야 합니다. 자세한 내용은 (선택 사항) 경보를 기반으로 롤백 권한을 구성합니다. CloudWatch 섹션을 참조하세요.

AWS AppConfig에 포함된 사전 정의된 전략을 선택하거나 직접 만들 수 있습니다.

주제

- 미리 정의된 배포 전략

- 배치 전략 생성

미리 정의된 배포 전략

AWS AppConfig에는 구성을 빠르게 배포할 수 있도록 미리 정의된 배포 전략이 포함되어 있습니다. 고유한 전략을 생성하는 대신 구성을 배포할 때 다음 중 하나를 선택할 수 있습니다.

배포 전략	설명
AppConfig.Linear20PercentEvery6Minutes	AWS 권장: 이 전략은 30분 배포를 위해 6분마다 모든 대상의 20%에 구성을 배포합니다. 시스템에서 30분 동안 Amazon CloudWatch 경보를 모니터링합니다. 이 시간에 경보가 수신되지 않으면 배포가 완료된 것입니다. 이 시간 동안 경보가 트리거되면 AWS AppConfig이 배포를 롤백합니다. 이 전략은 AWS 모범 사례에 부합하며 긴 지속 시간과 베이크 소요 시간으로 인해 배포 안전에 더욱 중점을 두기 때문에 프로덕션 배포에 사용하는 것을 권장합니다.
AppConfig.Canary10Percent20Minutes	AWS 권장: 이 전략은 20분 동안 10% 성장 계수를 사용하여 배포를 기하급수적으로 처리합니다. 시스템에서 10분 동안 CloudWatch 경보를 모니터링합니다. 이 시간에 경보가 수신되지 않으면 배포가 완료된 것입니다. 이 시간 동안 경보가 트리거되면 AWS AppConfig이 배포를 롤백합니다. 이 전략은 구성 배포에 대한 AWS 모범 사례를 준수하므로 프로덕션 배포에 사용하는 것이 좋습니다.
AppConfig.AllAtOnce	신속:

배포 전략	설명
	이 전략은 구성을 모든 대상에 즉시 배포합니다. 시스템에서 10분 동안 CloudWatch 경보를 모니터링합니다. 이 시간에 경보가 수신되지 않으면 배포가 완료된 것입니다. 이 시간 동안 경보가 트리거되면 AWS AppConfig이 배포를 롤백합니다.
AppConfig.Linear50PercentEvery30Seconds	테스트/데모: 이 전략은 1분 배포를 위해 30초마다 모든 대상의 절반에 구성을 배포합니다. 시스템에서 1분 동안 Amazon CloudWatch 경보를 모니터링합니다. 이 시간에 경보가 수신되지 않으면 배포가 완료된 것입니다. 이 시간 동안 경보가 트리거되면 AWS AppConfig이 배포를 롤백합니다. 이 전략은 지속 시간과 베이크 소요 시간이 짧기 때문에 테스트 또는 데모 용도로만 사용하는 것을 권장합니다.

배치 전략 생성

최대 20개의 배포 전략을 생성할 수 있습니다. 구성을 배포할 때 애플리케이션 및 환경에 가장 적합한 배포 전략을 선택할 수 있습니다.

AWS AppConfig 배포 전략 생성 (콘솔)

다음 절차를 통해 AWS Systems Manager 콘솔을 사용하여 AWS AppConfig 애플리케이션을 생성합니다.

배치 전략을 생성하려면

1. https://console.aws.amazon.com/systems-manager/appconfig/에서 AWS Systems Manager 콘솔을 엽니다.
2. 탐색 창에서 AWS AppConfig를 선택합니다.
3. 배포 전략 탭을 선택한 다음, 배포 전략 생성을 선택합니다.

4. 이름에 배치 전략의 이름을 입력합니다.
5. 설명에 배치 전략에 대한 정보를 입력합니다.
6. 배포 유형에서 유형을 선택합니다.
7. 단계 백분율에서 배포의 각 단계 동안 대상으로 지정할 호출자의 백분율을 선택합니다.
8. 배포 시간에 배포의 총 지속 시간(분 또는 시간)을 입력합니다.
9. 베이크 소요 시간에 배포의 다음 단계로 진행하기 전이나 완료할 배포를 고려하기 전에 Amazon CloudWatch 경보를 모니터링하는 총 시간(분 또는 시간)을 입력합니다.
10. 태그 섹션에 키와 선택적 값을 입력합니다. 하나의 리소스에 대해 최대 50개의 태그를 지정할 수 있습니다.
11. 배포 전략 생성을 선택합니다.

> **Important**
>
> AWS CodePipeline에 대한 구성 프로필을 만든 경우 CodePipeline에서 AWS AppConfig를 배포 공급자로 지정하는 파이프라인을 만들어야 합니다. 구성 배포을 수행할 필요가 없습니다. 하지만 API를 직접 호출하여 구성 검색에 설명된 대로 애플리케이션 구성 업데이트를 수신하도록 클라이언트를 구성해야 합니다. AWS AppConfig를 배포 공급자로 지정하는 파이프라인을 생성하는 방법에 대한 자세한 내용은 AWS CodePipeline 사용 설명서의 자습서: AWS AppConfig를 배포 공급자로 사용하는 파이프라인 만들기를 참조하십시오.

구성 배포 항목으로 이동합니다.

AWS AppConfig 배포 전략 생성 (명령줄)

다음 절차에서는 AWS CLI(Linux 또는 Windows) 또는 AWS Tools for PowerShell을 사용하여 AWS AppConfig 배포 전략을 생성하는 방법을 설명합니다.

단계별 배포 전략 생성

1. AWS CLI을 엽니다.
2. 다음 명령을 실행하여 배포 전략을 생성합니다.

   Linux

```
aws appconfig create-deployment-strategy \
  --name A_name_for_the_deployment_strategy \
```

```
  --description A_description_of_the_deployment_strategy \
  --deployment-duration-in-minutes Total_amount_of_time_for_a_deployment_to_last
 \
  --final-bake-time-in-minutes Amount_of_time_AWS
 AppConfig_monitors_for_alarms_before_considering_the_deployment_to_be_complete
 \
  --growth-
factor The_percentage_of_targets_to_receive_a_deployed_configuration_during_each_interva
 \
  --growth-
type The_linear_or_exponential_algorithm_used_to_define_how_percentage_grows_over_time
 \
  --replicate-
to To_save_the_deployment_strategy_to_a_Systems_Manager_(SSM)_document \
  --tags User_defined_key_value_pair_metadata_of_the_deployment_strategy
```

Windows

```
aws appconfig create-deployment-strategy ^
  --name A_name_for_the_deployment_strategy ^
  --description A_description_of_the_deployment_strategy ^
  --deployment-duration-in-minutes Total_amount_of_time_for_a_deployment_to_last
 ^
  --final-bake-time-in-minutes Amount_of_time_AWS
 AppConfig_monitors_for_alarms_before_considering_the_deployment_to_be_complete
 ^
  --growth-
factor The_percentage_of_targets_to_receive_a_deployed_configuration_during_each_interva
 ^
  --growth-
type The_linear_or_exponential_algorithm_used_to_define_how_percentage_grows_over_time
 ^
  --name A_name_for_the_deployment_strategy ^
  --replicate-
to To_save_the_deployment_strategy_to_a_Systems_Manager_(SSM)_document ^
  --tags User_defined_key_value_pair_metadata_of_the_deployment_strategy
```

PowerShell

```
New-APPCDeploymentStrategy `
  --Name A_name_for_the_deployment_strategy `
  --Description A_description_of_the_deployment_strategy `
  --DeploymentDurationInMinutes Total_amount_of_time_for_a_deployment_to_last `
```

```
  --FinalBakeTimeInMinutes Amount_of_time_AWS
 AppConfig_monitors_for_alarms_before_considering_the_deployment_to_be_complete
 `
  --
GrowthFactor The_percentage_of_targets_to_receive_a_deployed_configuration_during_each_i
 `
  --
GrowthType The_linear_or_exponential_algorithm_used_to_define_how_percentage_grows_over_
 `
  --
ReplicateTo To_save_the_deployment_strategy_to_a_Systems_Manager_(SSM)_document
 `
  --
Tag Hashtable_type_User_defined_key_value_pair_metadata_of_the_deployment_strategy
```

시스템은 다음과 같은 정보를 반환합니다.

Linux

```
{
   "Id": "Id of the deployment strategy",
   "Name": "Name of the deployment strategy",
   "Description": "Description of the deployment strategy",
   "DeploymentDurationInMinutes": "Total amount of time the deployment lasted",
   "GrowthType": "The linear or exponential algorithm used to define how
 percentage grew over time",
   "GrowthFactor": "The percentage of targets that received a deployed
 configuration during each interval",
   "FinalBakeTimeInMinutes": "The amount of time AWS AppConfig monitored for
 alarms before considering the deployment to be complete",
   "ReplicateTo": "The Systems Manager (SSM) document where the deployment
 strategy is saved"
}
```

Windows

```
{
   "Id": "Id of the deployment strategy",
   "Name": "Name of the deployment strategy",
   "Description": "Description of the deployment strategy",
   "DeploymentDurationInMinutes": "Total amount of time the deployment lasted",
```

```
  "GrowthType": "The linear or exponential algorithm used to define how
 percentage grew over time",
  "GrowthFactor": "The percentage of targets that received a deployed
 configuration during each interval",
  "FinalBakeTimeInMinutes": "The amount of time AWS AppConfig monitored for
 alarms before considering the deployment to be complete",
  "ReplicateTo": "The Systems Manager (SSM) document where the deployment
 strategy is saved"
}
```

PowerShell

```
ContentLength               : Runtime of the command
DeploymentDurationInMinutes : Total amount of time the deployment lasted
Description                 : Description of the deployment strategy
FinalBakeTimeInMinutes      : The amount of time AWS AppConfig monitored for
 alarms before considering the deployment to be complete
GrowthFactor                : The percentage of targets that received a deployed
 configuration during each interval
GrowthType                  : The linear or exponential algorithm used to define
 how percentage grew over time
HttpStatusCode              : HTTP Status of the runtime
Id                          : The deployment strategy ID
Name                        : Name of the deployment strategy
ReplicateTo                 : The Systems Manager (SSM) document where the
 deployment strategy is saved
ResponseMetadata            : Runtime Metadata
```

구성 배포

기능 플래그 및 자유 형식 구성 데이터 작업에 필요한 아티팩트를 생성한 후 AWS Management Console, AWS CLI, 또는 SDK를 사용하여 새 배포를 만들 수 있습니다. AWS AppConfig에서 배포를 시작하면 StartDeployment API 작업이 호출됩니다. 이 호출에는 AWS AppConfig 애플리케이션의 ID, 환경, 구성 프로파일 및 배포할 구성 데이터 버전(선택 사항)이 포함됩니다. 호출에는 사용할 배포 전략의 ID도 포함되는데, 이에 따라 구성 데이터가 배치되는 방법이 결정됩니다.

AWS Secrets Manager에 저장된 암호, 고객 관리 키로 암호화된 Amazon Simple Storage Service(S3) 객체 또는 고객 관리 키로 암호화된 AWS Systems Manager Parameter Store에 저장된 보안 문자열 파라미터를 배포하는 경우 `KmsKeyIdentifier` 파라미터 값을 지정해야 합니다. 구성이 암호화되지 않

았거나 AWS 관리형 키로 암호화된 경우 `KmsKeyIdentifier` 파라미터 값을 지정하지 않아도 됩니다.

> **Note**
>
> `KmsKeyIdentifier`에 지정하는 값은 고객 관리형 키여야 합니다. 이는 구성을 암호화하는 데 사용한 키와 같지 않아도 됩니다.
>
> `KmsKeyIdentifier`로 배포를 시작하는 경우 AWS Identity and Access Management (IAM) 주체에 연결된 권한 정책이 `kms:GenerateDataKey` 작업을 허용해야 합니다.

AWS AppConfig은 모든 호스트에 대한 배포를 모니터링하고 상태를 보고합니다. 배포가 실패하면 AWS AppConfig는 구성을 롤백합니다.

> **Note**
>
> 하나의 환경에 한 번에 하나의 구성만 배포할 수 있습니다. 하지만 각각 다른 환경에 한 가지 구성을 동시에 배포할 수 있습니다.

구성 배포 (콘솔)

다음 절차에 따라 AWS Systems Manager 콘솔을 사용하여 AWS AppConfig 구성을 배포합니다.

콘솔을 사용하여 구성을 배포하려면

1. https://console.aws.amazon.com/systems-manager/appconfig/에서 AWS Systems Manager 콘솔을 엽니다.
2. 탐색 창에서 AWS AppConfig를 선택합니다.
3. 애플리케이션 탭에서 애플리케이션을 선택한 다음, 세부 정보 보기를 선택합니다.
4. 환경 탭에서 환경을 선택한 다음, 세부 정보 보기를 선택합니다.
5. 배포 시작을 선택합니다.
6. 구성의 목록에서 구성을 선택합니다.
7. 구성 소스에 따라 문서 버전 또는 파라미터 버전 목록을 사용하여 배포할 버전을 선택합니다.
8. 배치 전략의 목록에서 전략을 선택합니다.
9. 배치 설명에 설명을 입력합니다.

10. 태그 섹션에 키와 선택적 값을 입력합니다. 하나의 리소스에 대해 최대 50개의 태그를 지정할 수 있습니다.
11. 배포 시작을 선택합니다.

구성 배포 (명령줄)

다음 절차에서는 AWS CLI (Linux 또는 Windows) 또는 AWS Tools for PowerShell을 사용하여 AWS AppConfig 구성을 생성하는 방법을 설명합니다.

구성을 단계별로 배포하려면

1. AWS CLI을 엽니다.
2. 다음 명령을 실행하여 구성을 배포합니다.

   Linux

   ```
   aws appconfig start-deployment \
     --application-id The_application_ID \
     --environment-id The_environment_ID \
     --deployment-strategy-id The_deployment_strategy_ID \
     --configuration-profile-id The_configuration_profile_ID \
     --configuration-version The_configuration_version_to_deploy \
     --description A_description_of_the_deployment \
     --tags User_defined_key_value_pair_metadata_of_the_deployment
   ```

   Windows

   ```
   aws appconfig start-deployment ^
     --application-id The_application_ID ^
     --environment-id The_environment_ID ^
     --deployment-strategy-id The_deployment_strategy_ID ^
     --configuration-profile-id The_configuration_profile_ID ^
     --configuration-version The_configuration_version_to_deploy ^
     --description A_description_of_the_deployment ^
     --tags User_defined_key_value_pair_metadata_of_the_deployment
   ```

   PowerShell

   ```
   Start-APPCDeployment `
     -ApplicationId The_application_ID `
   ```

```
  -ConfigurationProfileId The_configuration_profile_ID `
  -ConfigurationVersion The_configuration_version_to_deploy `
  -DeploymentStrategyId The_deployment_strategy_ID `
  -Description A_description_of_the_deployment `
  -EnvironmentId The_environment_ID `
  -Tag Hashtable_type_user_defined_key_value_pair_metadata_of_the_deployment
```

시스템은 다음과 같은 정보를 반환합니다.

Linux

```
{
   "ApplicationId": "The ID of the application that was deployed",
   "EnvironmentId" : "The ID of the environment",
   "DeploymentStrategyId": "The ID of the deployment strategy that was
 deployed",
   "ConfigurationProfileId": "The ID of the configuration profile that was
 deployed",
   "DeploymentNumber": The sequence number of the deployment,
   "ConfigurationName": "The name of the configuration",
   "ConfigurationLocationUri": "Information about the source location of the
 configuration",
   "ConfigurationVersion": "The configuration version that was deployed",
   "Description": "The description of the deployment",
   "DeploymentDurationInMinutes": Total amount of time the deployment lasted,
   "GrowthType": "The linear or exponential algorithm used to define how
 percentage grew over time",
   "GrowthFactor": The percentage of targets to receive a deployed configuration
 during each interval,
   "FinalBakeTimeInMinutes": Time AWS AppConfig monitored for alarms before
 considering the deployment to be complete,
   "State": "The state of the deployment",

   "EventLog": [
      {
         "Description": "A description of the deployment event",
         "EventType": "The type of deployment event",
         "OccurredAt": The date and time the event occurred,
         "TriggeredBy": "The entity that triggered the deployment event"
      }
   ],
```

```
   "PercentageComplete": The percentage of targets for which the deployment is
 available,
   "StartedAt": The time the deployment started,
   "CompletedAt": The time the deployment completed
}
```

Windows

```
{
   "ApplicationId": "The ID of the application that was deployed",
   "EnvironmentId" : "The ID of the environment",
   "DeploymentStrategyId": "The ID of the deployment strategy that was
 deployed",
   "ConfigurationProfileId": "The ID of the configuration profile that was
 deployed",
   "DeploymentNumber": The sequence number of the deployment,
   "ConfigurationName": "The name of the configuration",
   "ConfigurationLocationUri": "Information about the source location of the
 configuration",
   "ConfigurationVersion": "The configuration version that was deployed",
   "Description": "The description of the deployment",
   "DeploymentDurationInMinutes": Total amount of time the deployment lasted,
   "GrowthType": "The linear or exponential algorithm used to define how
 percentage grew over time",
   "GrowthFactor": The percentage of targets to receive a deployed configuration
 during each interval,
   "FinalBakeTimeInMinutes": Time AWS AppConfig monitored for alarms before
 considering the deployment to be complete,
   "State": "The state of the deployment",

   "EventLog": [
      {
         "Description": "A description of the deployment event",
         "EventType": "The type of deployment event",
         "OccurredAt": The date and time the event occurred,
         "TriggeredBy": "The entity that triggered the deployment event"
      }
   ],

   "PercentageComplete": The percentage of targets for which the deployment is
 available,
   "StartedAt": The time the deployment started,
   "CompletedAt": The time the deployment completed
```

```
}
```

PowerShell

```
ApplicationId               : The ID of the application that was deployed
CompletedAt                 : The time the deployment completed
ConfigurationLocationUri    : Information about the source location of the
 configuration
ConfigurationName           : The name of the configuration
ConfigurationProfileId      : The ID of the configuration profile that was
 deployed
ConfigurationVersion        : The configuration version that was deployed
ContentLength               : Runtime of the deployment
DeploymentDurationInMinutes : Total amount of time the deployment lasted
DeploymentNumber            : The sequence number of the deployment
DeploymentStrategyId        : The ID of the deployment strategy that was
 deployed
Description                 : The description of the deployment
EnvironmentId               : The ID of the environment that was deployed
EventLog                    : {Description : A description of the deployment
 event, EventType : The type of deployment event, OccurredAt : The date and time
 the event occurred,
         TriggeredBy : The entity that triggered the deployment event}
FinalBakeTimeInMinutes      : Time AWS AppConfig monitored for alarms before
 considering the deployment to be complete
GrowthFactor                : The percentage of targets to receive a deployed
 configuration during each interval
GrowthType                  : The linear or exponential algorithm used to define
 how percentage grew over time
HttpStatusCode              : HTTP Status of the runtime
PercentageComplete          : The percentage of targets for which the deployment
 is available
ResponseMetadata            : Runtime Metadata
StartedAt                   : The time the deployment started
State                       : The state of the deployment
```

CodePipeline과 AWS AppConfig 배포 통합

AWS AppConfig는 AWS CodePipeline (CodePipeline)에 대한 통합 배포 액션입니다. CodePipeline은 빠르고 안정적인 애플리케이션 및 인프라 업데이트를 위해 릴리스 파이프라인을 자동화하는 데 사용할 수 있는 완전관리형 지속적 제공 서비스입니다. CodePipeline은 정의한 릴리스 모델을 기반으로 코

드 변경이 있을 때마다 릴리스 프로세스의 구축, 테스트 및 배포 단계를 자동화합니다. 자세한 내용은 AWS CodePipeline란 무엇입니까?를 참조하세요.

AWS AppConfig와 CodePipeline의 통합은 다음과 같은 이점을 제공합니다.

- CodePipeline을 사용하여 오케스트레이션을 관리하는 고객은 이제 전체 코드베이스를 배포할 필요 없이 애플리케이션에 구성 변경 사항을 간단히 배포할 수 있습니다.
- AWS AppConfig를 사용하여 구성 배포를 관리하고 싶지만 AWS AppConfig가 현재 코드 또는 구성 스토어를 지원하지 않아 제한을 받는 고객에게 이제 추가 옵션이 있습니다. CodePipeline은 AWS CodeCommit, GitHub, BitBucket 등을 지원합니다.

> **Note**
>
> CodePipeline과 AWS AppConfig 통합은 CodePipeline을 사용할 수 있는 AWS 리전에 한해 지원됩니다.

통합의 작동 방식

CodePipeline을 설정하고 구성하는 것부터 시작합니다. 여기에는 CodePipeline 지원 코드 저장소에 구성을 추가하는 것도 포함됩니다. 다음으로, 다음 작업을 수행하여 AWS AppConfig 환경을 설정합니다.

- 네임스페이스와 구성 프로필 생성
- 사전 정의된 배포 전략을 선택 또는 직접 생성

이 작업을 완료한 후 AWS AppConfig를 배포 공급자로 지정하는 파이프라인을 CodePipeline에서 생성합니다. 그런 다음 구성을 변경하여 CodePipeline 코드 저장소에 업로드할 수 있습니다. 새 구성을 업로드하면 CodePipeline에서 새 배포가 자동으로 시작됩니다. 배포가 완료된 후 변경을 확인할 수 있습니다. AWS AppConfig를 배포 공급자로 지정하는 파이프라인을 생성하는 방법에 대한 자세한 내용은 AWS CodePipeline 사용 설명서의 자습서: AWS AppConfig를 배포 공급자로 사용하는 파이프라인 만들기를 참조하십시오.

AWS AppConfig에서 기능 플래그 및 구성 데이터 검색

애플리케이션은 데이터 서비스를 사용하여 구성 세션을 설정하여 기능 플래그와 자유 형식 구성 AWS AppConfig 데이터를 검색합니다. 이 섹션에 설명된 간소화된 검색 방법 중 하나를 사용하는 경우 AWS AppConfig Agent Lambda 확장 프로그램 AWS AppConfig 또는 에이전트가 사용자를 대신하여 일련의 API 호출 및 세션 토큰을 관리합니다. AWS AppConfig 에이전트를 로컬 호스트로 구성하고 에이전트가 구성 업데이트를 AWS AppConfig 위해 폴링하도록 합니다. 에이전트는 StartConfigurationSession및 GetLatestConfigurationAPI 작업을 호출하고 구성 데이터를 로컬에 캐시합니다. 데이터를 검색하기 위해 애플리케이션은 로컬 호스트 서버에 HTTP 호출을 수행합니다. AWS AppConfig 에이전트는 에 간소화된 검색 방법 설명된 대로 여러 사용 사례를 지원합니다.

원한다면 이러한 API 작업을 수동으로 호출하여 구성을 검색할 수 있습니다. API 프로세스는 다음과 같이 작동합니다.

애플리케이션은 `StartConfigurationSession` API 작업을 사용하여 구성 세션을 설정합니다. 그러면 세션 클라이언트가 `GetLatestConfiguration`를 정기적으로 호출하여 사용 가능한 최신 데이터를 확인하고 검색합니다.

`StartConfigurationSession`호출할 때 코드는 세션이 추적하는 AWS AppConfig 애플리케이션, 환경 및 구성 프로필의 식별자 (ID 또는 이름) 를 보냅니다.

이에 대한 `InitialConfigurationToken` 응답으로 세션의 클라이언트에 a를 AWS AppConfig 제공하여 해당 세션을 처음 `GetLatestConfiguration` 호출할 때 사용합니다.

`GetLatestConfiguration`를 호출하면 클라이언트 코드는 가장 최근의 `ConfigurationToken` 값을 보내고 이에 대한 응답으로 수신합니다.

- `NextPollConfigurationToken`: 다음에 `GetLatestConfiguration` 호출 시 사용할 `ConfigurationToken` 값입니다.
- 구성: 세션에 사용할 최신 데이터. 클라이언트에 이미 최신 버전의 구성이 있는 경우 비어 있을 수 있습니다.

이 섹션에는 다음 정보가 포함됩니다.

내용

- AWS AppConfig 데이터 플레인 서비스 정보
- 간소화된 검색 방법
- API를 직접 호출하여 구성 검색

AWS AppConfig 데이터 플레인 서비스 정보

2021년 11월 18일에 새로운 데이터 플레인 서비스가 AWS AppConfig 출시되었습니다. 이 서비스는 `GetConfiguration` API 작업을 사용하여 구성 데이터를 검색하는 이전 프로세스를 대체합니다. 데이터 플레인 서비스는 두 개의 새로운 API 작업, StartConfigurationSession및 GetLatestConfiguration을 사용합니다. 또한 데이터 영역 서비스는 새 엔드포인트를 사용합니다.

2022년 1월 28일 AWS AppConfig 이전에 사용을 시작한 경우 서비스가 `GetConfiguration` API 작업을 직접 호출하거나 AWS AppConfig Agent Lambda 확장 프로그램과 같이 에서 제공하는 AWS클라이언트를 사용하여 이 API 작업을 호출할 수 있습니다. `GetConfiguration` API 작업을 직접 호출하는 경우 `StartConfigurationSession` 및 `GetLatestConfiguration` API 작업을 사용하기 위한 조치를 취하십시오. AWS AppConfig Agent Lambda 확장 프로그램을 사용하는 경우 이 항목의 뒷부분에 있는 이 변경이 AWS AppConfig Agent Lambda 확장 프로그램에 미치는 영향이라는 제목의 섹션을 참조하십시오.

새 데이터 영역 API 작업은 현재 더 이상 사용되지 않는 `GetConfiguration` API 작업에 비해 다음과 같은 이점을 제공합니다.

1. `ClientID` 파라미터를 관리할 필요가 없습니다. 데이터 영역 서비스의 경우 `StartConfigurationSession`에서 생성한 세션 토큰을 통해 `ClientID`가 내부적으로 관리됩니다.
2. 더 이상 구성 데이터의 캐시된 버전을 표시하기 위해 `ClientConfigurationVersion`를 포함시킬 필요가 없습니다. 데이터 영역 서비스의 경우 `StartConfigurationSession`에서 생성한 세션 토큰을 통해 `ClientConfigurationVersion`가 내부적으로 관리됩니다.
3. 데이터 영역 API 직접 호출을 위한 새로운 전용 엔드포인트는 컨트롤 플레인 및 데이터 영역 호출을 분리하여 코드 구조를 개선합니다.
4. 새 데이터 영역 서비스는 데이터 영역 작업의 향후 확장성을 개선합니다. 구성 데이터 검색을 관리하는 구성 세션을 활용하여 AWS AppConfig 팀은 향후 더욱 강력한 개선 사항을 만들 수 있습니다.

`GetConfiguration`에서 **`GetLatestConfiguration`**로 마이그레이션

새 데이터 영역 서비스를 사용하려면 `GetConfiguration` API 작업을 호출하는 코드를 업데이트해야 합니다. `StartConfigurationSession` API 작업을 사용하여 구성 세션을 시작한 다음 `GetLatestConfiguration` API 작업을 호출하여 구성 데이터를 검색합니다. 성능 개선을 위해 구성 데이터를 로컬에서 캐시하는 것이 좋습니다. 자세한 정보는 API를 직접 호출하여 구성 검색을 참조하세요.

이 변경이 AWS AppConfig 에이전트 Lambda 확장 프로그램에 미치는 영향

이 변경은 AWS AppConfig Agent Lambda 확장 프로그램의 작동 방식에 직접적인 영향을 주지 않습니다. AWS AppConfig 에이전트 Lambda 확장 프로그램의 이전 버전은 사용자를 `GetConfiguration` 대신하여 API 작업을 호출했습니다. 최신 버전은 데이터 영역 API 작업을 호출합니다. AWS AppConfig Lambda 확장을 사용하는 경우 확장을 최신 Amazon 리소스 이름(ARN)으로 업데이트하고 새 API 직접 호출에 대한 권한을 업데이트하는 것이 좋습니다. 자세한 내용은 AWS AppConfig 에이전트 Lambda 확장 프로그램을 사용하여 구성 데이터 검색 섹션을 참조하십시오.

간소화된 검색 방법

AWS AppConfig 구성 데이터를 검색하는 몇 가지 간소화된 방법을 제공합니다. AWS Lambda 함수에 AWS AppConfig 기능 플래그 또는 자유 형식 구성 데이터를 사용하는 경우 AWS AppConfig Agent Lambda 확장 프로그램을 사용하여 구성을 검색할 수 있습니다. Amazon EC2 인스턴스에서 실행되는 애플리케이션이 있는 경우 AWS AppConfig 에이전트를 사용하여 구성을 검색할 수 있습니다. AWS AppConfig 에이전트는 또한 아마존 엘라스틱 쿠버네티스 서비스 (Amazon EKS) 또는 아마존 Elastic Container Service (아마존 ECS) 컨테이너 이미지에서 실행되는 애플리케이션을 지원합니다.

초기 설정을 완료한 후 구성 데이터를 검색하는 이러한 방법은 API를 직접 호출하는 것보다 더 간단합니다. AWS AppConfig 자동으로 모범 사례를 구현하고 구성을 검색하기 위한 API 호출 횟수가 줄어들어 사용 AWS AppConfig 비용을 낮출 수 있습니다.

주제

- AWS AppConfig 에이전트 Lambda 확장 프로그램을 사용하여 구성 데이터 검색
- Amazon EC2 인스턴스에서 구성 데이터 검색
- Amazon ECS 및 Amazon EKS 에서 구성 데이터 검색
- 추가 검색 기능
- AWS AppConfig 에이전트 로컬 개발

AWS AppConfig 에이전트 Lambda 확장 프로그램을 사용하여 구성 데이터 검색

AWS Lambda 확장은 Lambda 함수의 기능을 보강하는 동반 프로세스입니다. 확장은 함수가 간접적으로 호출되기 전에 시작되어 함수와 병렬로 실행되고 함수 호출이 처리된 후에도 계속 실행될 수 있습니다. 기본적으로, Lambda 확장은 Lambda 호출과 병렬로 실행되는 클라이언트와 같습니다. 이 병렬 클라이언트는 수명 주기 중 언제든지 함수와 연결될 수 있습니다.

Lambda 함수에서 AWS AppConfig 기능 플래그 또는 기타 동적 구성 데이터를 사용하는 경우, Lambda 함수에 AWS AppConfig 에이전트 Lambda 확장 프로그램을 계층으로 추가하는 것이 좋습니다. 이렇게 하면 기능 플래그를 더 간단하게 호출할 수 있으며 확장 프로그램 자체에는 사용을 단순화하는 동시에 비용을 절감하는 모범 사례가 포함됩니다. AWS AppConfig AWS AppConfig 서비스에 대한 API 호출이 줄어들고 Lambda 함수 처리 시간이 단축되어 비용이 절감됩니다. Lambda 확장에 대한 자세한 내용은 AWS Lambda 개발자 안내서의 Lambda 확장 을 참조하십시오.

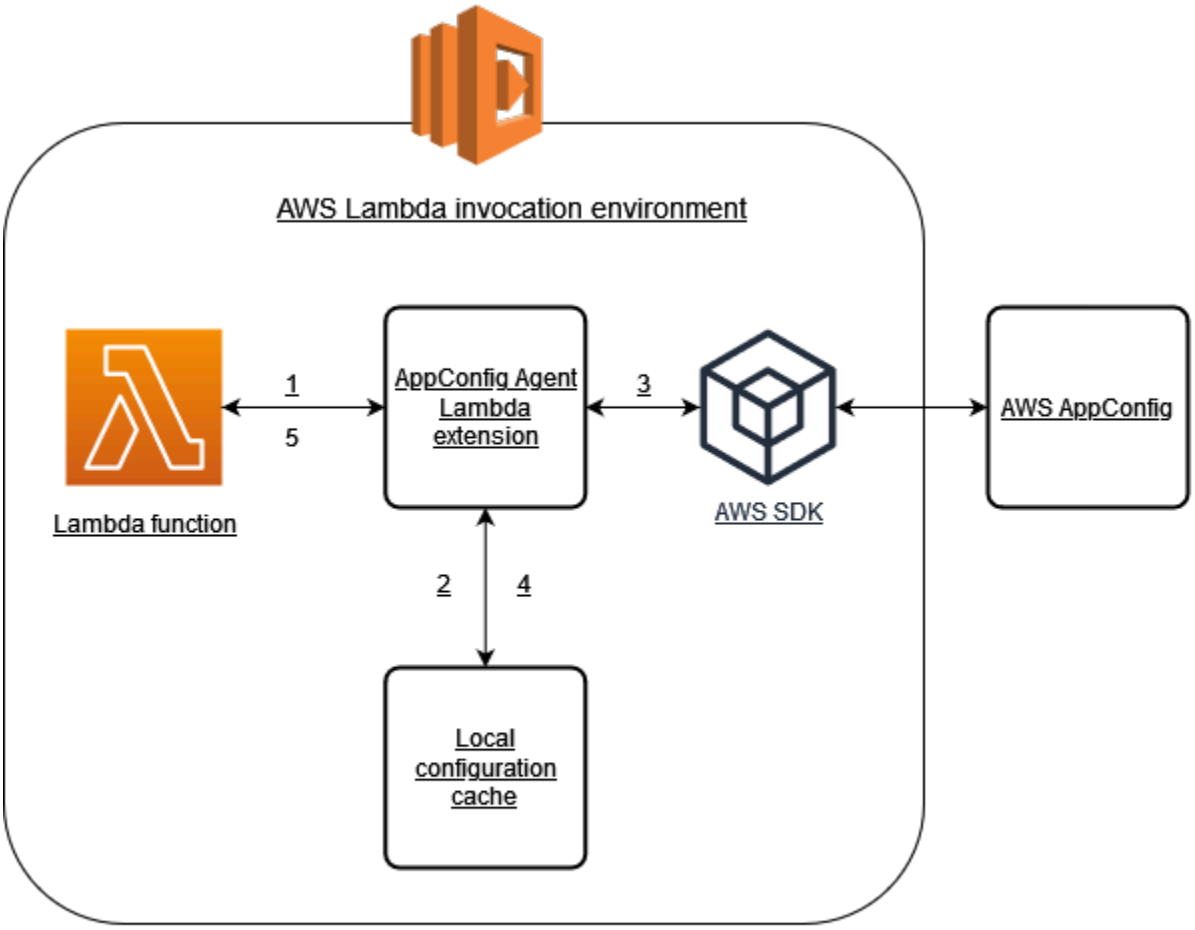
> **Note**
>
> AWS AppConfig 요금은 구성이 호출되고 수신된 횟수를 기준으로 합니다. Lambda가 여러 번의 콜드 스타트를 수행하고 새 구성 데이터를 자주 검색하면 비용이 증가합니다.

이 주제에는 AWS AppConfig 에이전트 Lambda 확장 프로그램에 대한 정보와 Lambda 함수와 함께 작동하도록 확장 프로그램을 구성하는 방법에 대한 절차가 포함되어 있습니다.

작동 방식

Lambda 확장 프로그램이 없는 Lambda 함수의 구성을 관리하는 AWS AppConfig 데 사용하는 경우, 및 API 작업과 통합하여 구성 업데이트를 수신하도록 Lambda 함수를 구성해야 합니다. StartConfigurationSessionGetLatestConfiguration

AWS AppConfig 에이전트 Lambda 확장 프로그램을 Lambda 함수와 통합하면 이 프로세스가 간소화됩니다. 확장 프로그램은 서비스를 호출하고, 검색된 데이터의 로컬 캐시를 관리하고, 다음 AWS AppConfig 서비스 호출에 필요한 구성 토큰을 추적하고, 백그라운드에서 구성 업데이트를 정기적으로 확인합니다. 다음 다이어그램은 작동 방식을 보여 줍니다.



1. AWS AppConfig 에이전트 Lambda 확장 프로그램을 Lambda 함수의 계층으로 구성합니다.
2. 구성 데이터에 액세스하기 위해 함수는 에서 실행되는 HTTP 엔드포인트에서 AWS AppConfig 확장 프로그램을 호출합니다. `localhost:2772`
3. 확장은 구성 데이터의 로컬 캐시를 유지합니다. 데이터가 캐시에 없는 경우 확장 프로그램은 구성 데이터를 가져오기 AWS AppConfig 위해 호출합니다.
4. 서비스로부터 구성을 수신하면 확장은 구성을 로컬 캐시에 저장하고 Lambda 함수에 전달합니다.
5. AWS AppConfig 에이전트 Lambda 확장 프로그램은 백그라운드에서 구성 데이터에 대한 업데이트를 정기적으로 확인합니다. Lambda 함수가 간접적으로 호출될 때마다 확장은 구성을 검색한 이후 경과된 시간을 확인합니다. 경과된 시간이 구성된 폴링 간격보다 길면 확장 프로그램은 새로 배포된 데이터를 확인하기 AWS AppConfig 위해 호출하고, 변경 사항이 있는 경우 로컬 캐시를 업데이트하고, 경과 시간을 재설정합니다.

> **Note**
>
> - Lambda는 함수에 필요한 동시성 레벨에 해당하는 별도의 인스턴스를 인스턴스화합니다. 각 인스턴스는 격리되며 구성 데이터의 자체 로컬 캐시를 유지합니다. Lambda 인스턴스 및 동시성에 대한 자세한 내용은 Lambda 함수의 동시성 관리를 참조하십시오.
> - 에서 업데이트된 구성을 배포한 후 Lambda 함수에 구성 변경 사항이 나타나는 데 걸리는 시간은 배포에 사용한 배포 전략과 확장 프로그램에 AWS AppConfig구성한 폴링 간격에 따라 다릅니다.

시작하기 전 준비 사항

AWS AppConfig 에이전트 Lambda 확장 프로그램을 활성화하기 전에 다음을 수행하십시오.

- Lambda 함수에서 구성을 조직하여 AWS AppConfig로 외부화할 수 있습니다.
- 기능 플래그 또는 자유 형식 구성 데이터를 비롯한 AWS AppConfig 아티팩트와 구성 데이터를 생성합니다. 자세한 정보는 기능 플래그 및 자유 형식 구성 데이터 생성 AWS AppConfig을 참조하세요.
- Lambda 함수 실행 역할에서 사용하는 AWS Identity and Access Management (IAM) 정책에 `appconfig:StartConfigurationSession` 및 `appconfig:GetLatestConfiguration` 를 추가합니다. 자세한 내용은 AWS Lambda 개발자 안내서의 AWS Lambda 실행 역할을 참조하십시오. AWS AppConfig 권한에 대한 자세한 내용은 서비스 권한 부여 참조에서 AWS AppConfig에 대한 액션, 리소스 및 조건 키를 참조하십시오.

AWS AppConfig 에이전트 Lambda 확장 프로그램 추가

AWS AppConfig 에이전트 Lambda 확장 프로그램을 사용하려면 Lambda에 확장 프로그램을 추가해야 합니다. Lambda 함수에 AWS AppConfig 에이전트 Lambda 확장 프로그램을 계층으로 추가하거나 Lambda 함수에서 확장을 컨테이너 이미지로 활성화하여 이 작업을 수행할 수 있습니다.

> **Note**
>
> AWS AppConfig 확장 프로그램은 런타임에 구애받지 않으며 모든 런타임을 지원합니다.

계층과 ARN을 사용하여 AWS AppConfig 에이전트 Lambda 확장 추가

AWS AppConfig 에이전트 Lambda 확장 프로그램을 사용하려면 확장 프로그램을 Lambda 함수에 계층으로 추가합니다. 함수에 계층을 추가하는 방법에 대한 자세한 내용은 AWS Lambda 개발자 안내서의 확장 구성을 참조하십시오. AWS Lambda 콘솔에 있는 확장 프로그램의 이름은 - -Extension입니다.AWS AppConfig 또한 Lambda에 확장을 계층으로 추가하는 경우 Amazon 리소스 이름(ARN)을 지정해야 합니다. 다음 목록 중 플랫폼 및 Lambda를 생성한 AWS 리전 위치에 해당하는 ARN을 선택하십시오.

- x86-64 플랫폼
- ARM64 플랫폼

함수에 추가하기 전에 확장을 테스트하려는 경우 다음 코드 예제를 사용하여 확장이 작동하는지 확인할 수 있습니다.

```
import urllib.request


def lambda_handler(event, context):
    url = f'http://localhost:2772/applications/application_name/
environments/environment_name/configurations/configuration_name'
    config = urllib.request.urlopen(url).read()
    return config
```

테스트하려면 Python용 Lambda 함수를 새로 생성하고 확장을 추가한 다음 Lambda 함수를 실행하십시오. Lambda 함수를 실행한 후 AWS AppConfig Lambda 함수는 http://localhost:2772 경로에 지정된 구성을 반환합니다. Lambda 함수 생성에 관한 정보는 AWS Lambda 개발자 안내서의 콘솔로 Lambda 함수 생성을 참조하십시오.

AWS AppConfig Agent Lambda 확장 프로그램을 컨테이너 이미지로 추가하려면 을 참조하십시오. 컨테이너 이미지를 사용하여 AWS AppConfig 에이전트 Lambda 확장 프로그램 추가

AWS AppConfig 에이전트 Lambda 확장 구성

다음 AWS Lambda 환경 변수를 변경하여 확장을 구성할 수 있습니다. 자세한 내용은 AWS Lambda 개발자 안내서의 AWS Lambda 환경 변수 사용을 참조하십시오.

구성 데이터 프리페칭

환경 변수 AWS_APPCONFIG_EXTENSION_PREFETCH_LIST는 함수 시작 시간을 개선할 수 있습니다. AWS AppConfig Agent Lambda 확장 프로그램이 초기화되면 Lambda가 함수를 초기화하고 핸들러를 AWS AppConfig 호출하기 시작하기 전부터 지정된 구성을 검색합니다. 함수가 요청하기 전에 로컬 캐시에서 구성 데이터를 이미 사용할 수 있는 경우도 있습니다.

프리페치 기능을 사용하려면 환경 변수 값을 구성 데이터에 해당하는 경로로 설정하십시오. 예를 들어 구성이 각각 “my_application”, “my_environment” 및 “my_configuration_data”라는 이름의 애플리케이션, 환경 및 구성 프로파일에 해당하는 경우 경로는 /applications/my_application/environments/my_environment/configurations/my_configuration_data입니다. 여러 구성 항목을 쉼표로 구분된 목록으로 나열하여 지정할 수 있습니다(리소스 이름에 쉼표가 포함된 경우 해당 이름 대신 리소스의 ID 값을 사용하십시오.).

다른 계정에서 구성 데이터에 액세스

AWS AppConfig Agent Lambda 확장 프로그램은 데이터에 권한을 부여하는 IAM 역할을 지정하여 다른 계정에서 구성 데이터를 검색할 수 있습니다. 이를 설정하려면 다음 단계를 수행합니다.

1. 구성 데이터를 관리하는 데 사용되는 계정에서 Lambda 함수를 실행하는 계정에 구성 리소스에 해당하는 부분 또는 전체 ARN과 함께 appconfig:GetLatestConfiguration 및 작업에 대한 액세스 권한을 부여하는 appconfig:StartConfigurationSession 신뢰 정책이 포함된 역할을 생성합니다. AWS AppConfig AWS AppConfig
2. Lambda 함수를 실행하는 계정에서 1단계에서 생성한 역할의 ARN을 사용하여 Lambda 함수에 AWS_APPCONFIG_EXTENSION_ROLE_ARN 환경 변수를 추가합니다.
3. (선택 사항) 필요한 경우 AWS_APPCONFIG_EXTENSION_ROLE_EXTERNAL_ID 환경 변수를 사용하여 외부 ID를 지정할 수 있습니다. 마찬가지로 AWS_APPCONFIG_EXTENSION_ROLE_SESSION_NAME 환경 변수를 사용하여 세션 이름을 구성할 수 있습니다.

> **Note**
>
> 다음 정보를 참고하세요.
>
> - AWS AppConfig 에이전트 Lambda 확장 프로그램은 한 계정에서만 데이터를 검색할 수 있습니다. IAM 역할을 지정하는 경우 확장은 Lambda 함수가 실행되는 계정에서 구성 데이터를 검색할 수 없습니다.
> - AWS Lambda Amazon Logs를 사용하여 AWS AppConfig 에이전트 Lambda 확장 프로그램 및 Lambda 함수에 대한 정보를 기록합니다. CloudWatch

환경 변수	Details	기본값
AWS_APPCONFIG_EXTENSION_HTTP_PORT	이 환경 변수는 확장을 호스팅하는 로컬 HTTP 서버가 실행되는 포트를 지정합니다.	2772
AWS_APPCONFIG_EXTENSION_LOG_LEVEL	이 환경 변수는 함수에 대해 Amazon Logs로 전송할 AWS AppConfig 확장별 CloudWatch 로그를 지정합니다. 대/소문자를 구분하지 않는 유효한 값은 debug, info, warn, error 및 none입니다. 디버그에는 타이밍 정보를 비롯한 확장에 대한 세부 정보가 포함됩니다.	info
AWS_APPCONFIG_EXTENSION_MAX_CONNECTIONS	이 환경 변수는 확장이 AWS AppConfig에서 구성을 검색하는 데 사용하는 최대 연결 수를 구성합니다.	3
AWS_APPCONFIG_EXTENSION_POLL_INTERVAL_SECONDS	이 환경 변수는 확장 프로그램이 업데이트된 구성을 AWS AppConfig 폴링하는 빈도를 초 단위로 제어합니다.	45
AWS_APPCONFIG_EXTENSION_POLL_TIMEOUT_MILLIS	이 환경 변수는 확장 프로그램이 캐시의 데이터를 새로 고칠 AWS AppConfig 때 응답을 기다리는 최대 시간 (밀리초) 을 제어합니다. 지정된 시간 내에 AWS AppConfig 응답하지 않으면 확장 프로그램은 이 폴링 간격을 건너뛰고 이전에 업데이트된 캐시된 데이터를 반환합니다.	3000

환경 변수	Details	기본값
AWS_APPCONFIG_EXTENSION_PREFETCH_LIST	이 환경 변수는 함수가 초기화되고 처리기가 실행되기 전에 확장에서 검색을 시작하는 구성 데이터를 지정합니다. 이렇게 하면 함수의 콜드 스타트 시간을 크게 줄일 수 있습니다.	None
AWS_APPCONFIG_EXTENSION_PROXY_HEADERS	이 환경 변수는 AWS_APPCONFIG_EXTENSION_PROXY_URL 환경 변수에서 참조되는 프록시에 필요한 헤더를 지정합니다. 값은 쉼표로 구분된 헤더 목록입니다. 각 헤더는 다음 형식을 사용합니다. `"header: value"`	None
AWS_APPCONFIG_EXTENSION_PROXY_URL	이 환경 변수는 AWS AppConfig 내선에서 확장으로의 연결에 사용할 프록시 URL을 지정합니다. AWS 서비스HTTPS그리고 HTTP URL도 지원됩니다.	None
AWS_APPCONFIG_EXTENSION_ROLE_ARN	이 환경 변수는 구성을 검색하기 위해 확장 프로그램이 맡아야 하는 역할에 해당하는 IAM 역할 AWS AppConfig ARN을 지정합니다.	None
AWS_APPCONFIG_EXTENSION_ROLE_EXTERNAL_ID	이 환경 변수는 수임된 역할 ARN과 함께 사용할 외부 ID를 지정합니다.	None

환경 변수	Details	기본값
AWS_APPCONFIG_EXTENSION_ROLE_SESSION_NAME	이 환경 변수는 수임된 IAM 역할의 자격 증명과 연결할 세션 이름을 지정합니다.	None
AWS_APPCONFIG_EXTENSION_SERVICE_REGION	이 환경 변수는 확장 프로그램이 서비스를 호출하는 데 사용해야 하는 대체 지역을 지정합니다. AWS AppConfig 정의되지 않은 경우 확장은 현재 지역의 엔드포인트를 사용합니다.	None
AWS_APPCONFIG_EXTENSION_MANIFEST	이 환경 변수는 다중 계정 검색과 같은 추가 구성별 기능을 활용하고 구성을 디스크에 저장하도록 AWS AppConfig 에이전트를 구성합니다. 다음 값 중 하나를 입력할 수 있습니다. • "app:env:manifest-config" • "file:/fully/qualified/path/to/manifest.json" 이러한 기능에 대한 자세한 내용은 추가 검색 기능 섹션을 참조하세요.	true
AWS_APPCONFIG_EXTENSION_WAIT_ON_MANIFEST	이 환경 변수는 AWS AppConfig 에이전트가 시작을 완료하기 전에 매니페스트가 처리될 때까지 기다리도록 구성합니다.	true

기능 플래그 구성에서 하나 이상의 플래그 검색

기능 플래그 구성(`AWS.AppConfig.FeatureFlags` 유형의 구성)의 경우, Lambda 확장을 사용하면 구성에서 단일 플래그 또는 플래그의 하위 집합을 검색할 수 있습니다. Lambda가 구성 프로필에서 몇 개의 플래그만 사용해야 하는 경우 하나 또는 두 개의 플래그를 검색하는 것이 유용합니다. 다음 예에서는 Python을 사용합니다.

> **Note**
>
> 구성에서 단일 기능 플래그 또는 일부 플래그를 호출하는 기능은 Agent AWS AppConfig Lambda 확장 버전 2.0.45 이상에서만 사용할 수 있습니다.

로컬 HTTP 엔드포인트에서 AWS AppConfig 구성 데이터를 검색할 수 있습니다. 특정 플래그 또는 플래그 목록에 액세스하려면 AWS AppConfig 구성 프로필의 `?flag=flag_name` 쿼리 파라미터를 사용하십시오.

단일 플래그와 해당 속성에 액세스하려면

```
import urllib.request

def lambda_handler(event, context):
    url = f'http://localhost:2772/applications/application_name/
environments/environment_name/configurations/configuration_name?flag=flag_name'
    config = urllib.request.urlopen(url).read()
    return config
```

여러 플래그와 해당 속성에 액세스하려면

```
import urllib.request

def lambda_handler(event, context):
    url = f'http://localhost:2772/applications/application_name/
environments/environment_name/configurations/configuration_name?
flag=flag_name_one&flag=flag_name_two'
    config = urllib.request.urlopen(url).read()
    return config
```

사용 가능한 AWS AppConfig 에이전트 Lambda 확장 프로그램 버전

이 주제에는 AWS AppConfig 에이전트 Lambda 확장 버전에 대한 정보가 포함되어 있습니다. AWS AppConfig 에이전트 Lambda 확장은 x86-64 및 ARM64 (Graviton2) 플랫폼용으로 개발된 Lambda 함수를 지원합니다. 제대로 작동하려면 Lambda 함수가 현재 호스팅된 위치에 대한 AWS 리전 특정 Amazon 리소스 이름 (ARN) 을 사용하도록 구성해야 합니다. 이 섹션의 뒷부분에서 ARN 세부 정보를 볼 AWS 리전 수 있습니다.

Important

AWS AppConfig 에이전트 Lambda 확장 프로그램에 대한 다음과 같은 중요한 세부 정보를 참고하십시오.

- `GetConfiguration` API 작업은 2022년 1월 28일 날짜로 더 이상 사용되지 않습니다. 구성 데이터를 수신하기 위한 호출은 대신 `StartConfigurationSession` 및 `GetLatestConfiguration` API를 사용해야 합니다. 2022년 1월 28일 이전에 생성된 AWS AppConfig Agent Lambda 확장 프로그램 버전을 사용하는 경우 새 API에 대한 권한을 구성해야 할 수 있습니다. 자세한 정보는 AWS AppConfig 데이터 플레인 서비스 정보을 참조하세요.
- AWS AppConfig 에 나열된 모든 버전을 지원합니다. 이전 확장 버전 확장 개선 사항을 활용하려면 주기적으로 최신 버전으로 업데이트하는 것을 권장합니다.

주제

- AWS AppConfig 에이전트 Lambda 확장 릴리스 노트
- Lambda 확장 버전 번호 찾기
- x86-64 플랫폼
- ARM64 플랫폼
- 이전 확장 버전

AWS AppConfig 에이전트 Lambda 확장 릴리스 노트

다음 표는 최신 버전의 AWS AppConfig Lambda 확장 프로그램에 적용된 변경 사항을 설명합니다.

버전	시작 날짜	참고
2.0.358	12/01/2023년	다음 검색 기능에 대한 지원이 추가되었습니다. • 다중 계정 검색: 기본 계정 또는 검색에서 AWS AppConfig 에이전트를 사용하여 여러 공급업체 AWS 계정 계정에서 구성 데이터를 검색합니다. • 구성 복사본을 디스크에 쓰기: AWS AppConfig 에이전트를 사용하여 구성 데이터를 디스크에 기록합니다. 이 기능을 사용하면 디스크에서 구성 데이터를 읽는 애플리케이션을 사용하는 고객이 이 기능을 통합하여 사용할 수 AWS AppConfig있습니다.
2.0.181	08/14/2023년	이스라엘 (텔아비브) AWS 리전 il-central-1에 대한 지원이 추가되었습니다.
2.0.165	2023년 2월 21일	사소한 버그가 수정됨. 더 이상 콘솔을 통한 확장 프로그램 사용을 특정 런타임 버전으로 제한하지 않습니다. AWS Lambda 다음 AWS 리전에 대한 지원이 추가됨. • 중동(UAE), me-central-1 • 아시아 태평양(하이데라바드) ap-south-2

버전	시작 날짜	참고
		• 아시아 태평양(멜버른) ap-southeast-4 • 유럽(스페인), eu-south-2 • 유럽(취리히) eu-central-2
2.0.122	08/23/2022	`AWS_APPCONFIG_EXTENSION_PROXY_URL` 및 `AWS_APPCONFIG_EXTENSION_PROXY_HEADERS` 환경 변수로 구성할 수 있는 터널링 프록시에 대한 지원이 추가됨. .NET 6을 런타임으로 추가됨. 환경 변수에 대한 자세한 내용은 AWS AppConfig 에이전트 Lambda 확장 구성 섹션을 참조하십시오.
2.0.58	05/03/2022	Lambda의 Graviton2 (ARM64) 프로세서에 대한 지원이 개선됨.
2.0.45	03/15/2022	단일 기능 플래그 호출에 대한 지원이 추가됨. 이전에는 고객이 구성 프로필로 그룹화된 기능 플래그를 호출하고 클라이언트 측 응답을 파싱해야 했습니다. 이번 릴리스부터 고객은 HTTP localhost 엔드포인트를 호출할 때 `flag=<flag-name>` 파라미터를 사용하여 단일 플래그의 값을 가져올 수 있습니다. Graviton2 (ARM64) 프로세서에 대한 초기 지원도 추가되었습니다.

Lambda 확장 버전 번호 찾기

다음 절차를 사용하여 현재 구성된 AWS AppConfig Agent Lambda 확장 프로그램의 버전 번호를 찾을 수 있습니다. 제대로 작동하려면 Lambda 함수가 현재 호스팅된 위치에 대한 AWS 리전 특정 Amazon 리소스 이름 (ARN) 을 사용하도록 구성해야 합니다.

1. AWS Management Console 로그인하고 https://console.aws.amazon.com/lambda/ 에서 AWS Lambda 콘솔을 엽니다.
2. `AWS-AppConfig-Extension` 계층을 추가하려는 Lambda 함수를 선택합니다.
3. 계층 섹션에서 계층 추가를 선택합니다.
4. 레이어 선택 섹션의 AWS 레이어 목록에서 AWS- AppConfig -Extension을 선택합니다.
5. 버전 목록을 사용하여 버전 번호를 선택합니다.
6. 추가를 선택합니다.
7. 테스트를 사용해 함수를 테스트합니다.
8. 테스트가 완료되면 로그 출력을 확인하십시오. 실행 세부 정보 섹션에서 AWS AppConfig 에이전트 Lambda 확장 프로그램 버전을 찾으십시오. 이 버전은 해당 버전의 필수 URL과 일치해야 합니다.

x86-64 플랫폼

Lambda에 확장을 계층으로 추가하는 경우 ARN을 지정해야 합니다. 다음 표에서 Lambda를 생성한 AWS 리전 위치에 해당하는 ARN을 선택합니다. 이러한 ARN은 x86-64 플랫폼용으로 개발된 Lambda 함수를 위한 것입니다.

버전 2.0.358

지역	ARN
미국 동부(버지니아 북부)	`arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension:128`
미국 동부(오하이오)	`arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension:93`

지역	ARN
미국 서부(캘리포니아 북부)	arn:aws:lambda:us-west-1:958113053741:layer:AWS-AppConfig-Extension:141
미국 서부(오레곤)	arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension:161
캐나다(중부)	arn:aws:lambda:ca-central-1:039592058896:layer:AWS-AppConfig-Extension:93
유럽(프랑크푸르트)	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension:106
유럽(취리히)	arn:aws:lambda:eu-central-2:758369105281:layer:AWS-AppConfig-Extension:47
유럽(아일랜드)	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension:125
유럽(런던)	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension:93
유럽(파리)	arn:aws:lambda:eu-west-3:493207061005:layer:AWS-AppConfig-Extension:98
유럽(스톡홀름)	arn:aws:lambda:eu-north-1:646970417810:layer:AWS-AppConfig-Extension:159

지역	ARN
유럽(밀라노)	arn:aws:lambda:eu-south-1:203683718741:layer:AWS-AppConfig-Extension:83
유럽(스페인)	arn:aws:lambda:eu-south-2:586093569114:layer:AWS-AppConfig-Extension:44
중국(베이징)	arn:aws-cn:lambda:cn-north-1:615057806174:layer:AWS-AppConfig-Extension:76
중국(닝샤)	arn:aws-cn:lambda:cn-northwest-1:615084187847:layer:AWS-AppConfig-Extension:76
아시아 태평양(홍콩)	arn:aws:lambda:ap-east-1:630222743974:layer:AWS-AppConfig-Extension:83
아시아 태평양(도쿄)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension:98
아시아 태평양(서울)	arn:aws:lambda:ap-northeast-2:826293736237:layer:AWS-AppConfig-Extension:108
아시아 태평양(오사카)	arn:aws:lambda:ap-northeast-3:706869817123:layer:AWS-AppConfig-Extension:101
아시아 태평양(싱가포르)	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension:106

지역	ARN
아시아 태평양(시드니)	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension:106
아시아 태평양(자카르타)	arn:aws:lambda:ap-southeast-3:418787028745:layer:AWS-AppConfig-Extension:79
아시아 태평양(멜버른)	arn:aws:lambda:ap-southeast-4:307021474294:layer:AWS-AppConfig-Extension:20
아시아 태평양(뭄바이)	arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension:107
아시아 태평양(하이데라바드)	arn:aws:lambda:ap-south-2:489524808438:layer:AWS-AppConfig-Extension:47
남아메리카(상파울루)	arn:aws:lambda:sa-east-1:000010852771:layer:AWS-AppConfig-Extension:128
아프리카(케이프타운)	arn:aws:lambda:af-south-1:574348263942:layer:AWS-AppConfig-Extension:83
이스라엘(텔아비브)	arn:aws:lambda:il-central-1:895787185223:layer:AWS-AppConfig-Extension:22
중동(UAE)	arn:aws:lambda:me-central-1:662846165436:layer:AWS-AppConfig-Extension:49

지역	ARN
중동(바레인)	arn:aws:lambda:me-south-1:559955524753:layer:AWS-AppConfig-Extension:85
AWS GovCloud (미국 동부)	arn:aws-us-gov:lambda:us-gov-east-1:946561847325:layer:AWS-AppConfig-Extension:54
AWS GovCloud (미국 서부)	arn:aws-us-gov:lambda:us-gov-west-1:946746059096:layer:AWS-AppConfig-Extension:54

ARM64 플랫폼

Lambda에 확장을 계층으로 추가하는 경우 ARN을 지정해야 합니다. 다음 표에서 Lambda를 생성한 AWS 리전 위치에 해당하는 ARN을 선택합니다. 이러한 ARN은 ARM64 플랫폼용으로 개발된 Lambda 함수를 위한 것입니다.

버전 2.0.358

지역	ARN
미국 동부(버지니아 북부)	arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension-Arm64:61
미국 동부(오하이오)	arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension-Arm64:45
미국 서부(캘리포니아 북부)	arn:aws:lambda:us-west-1:958113053741:layer:AWS-AppConfig-Extension-Arm64:18

지역	ARN
미국 서부(오레곤)	arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension-Arm64:63
캐나다(중부)	arn:aws:lambda:ca-central-1:039592058896:layer:AWS-AppConfig-Extension-Arm64:13
유럽(프랑크푸르트)	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension-Arm64:49
유럽(취리히)	arn:aws:lambda:eu-central-2:758369105281:layer:AWS-AppConfig-Extension-Arm64:5
유럽(아일랜드)	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension-Arm64:63
유럽(런던)	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension-Arm64:45
유럽(파리)	arn:aws:lambda:eu-west-3:493207061005:layer:AWS-AppConfig-Extension-Arm64:17
유럽(스톡홀름)	arn:aws:lambda:eu-north-1:646970417810:layer:AWS-AppConfig-Extension-Arm64:18
유럽(밀라노)	arn:aws:lambda:eu-south-1:203683718741:layer:AWS-AppConfig-Extension-Arm64:11

지역	ARN
유럽(스페인)	arn:aws:lambda:eu-south-2:586093569114:layer:AWS-AppConfig-Extension-Arm64:5
아시아 태평양(홍콩)	arn:aws:lambda:ap-east-1:630222743974:layer:AWS-AppConfig-Extension-Arm64:11
아시아 태평양(도쿄)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension-Arm64:51
아시아 태평양(서울)	arn:aws:lambda:ap-northeast-2:826293736237:layer:AWS-AppConfig-Extension-Arm64:16
아시아 태평양(오사카)	arn:aws:lambda:ap-northeast-3:706869817123:layer:AWS-AppConfig-Extension-Arm64:16
아시아 태평양(싱가포르)	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension-Arm64:58
아시아 태평양(시드니)	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension-Arm64:49
아시아 태평양(자카르타)	arn:aws:lambda:ap-southeast-3:418787028745:layer:AWS-AppConfig-Extension-Arm64:16
아시아 태평양(멜버른)	arn:aws:lambda:ap-southeast-4:307021474294:layer:AWS-AppConfig-Extension-Arm64:5

지역	ARN
아시아 태평양(뭄바이)	arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension-Arm64:49
아시아 태평양(하이데라바드)	arn:aws:lambda:ap-south-2:489524808438:layer:AWS-AppConfig-Extension-Arm64:5
남아메리카(상파울루)	arn:aws:lambda:sa-east-1:000010852771:layer:AWS-AppConfig-Extension-Arm64:16
아프리카(케이프타운)	arn:aws:lambda:af-south-1:574348263942:layer:AWS-AppConfig-Extension-Arm64:11
중동(UAE)	arn:aws:lambda:me-central-1:662846165436:layer:AWS-AppConfig-Extension-Arm64:5
중동(바레인)	arn:aws:lambda:me-south-1:559955524753:layer:AWS-AppConfig-Extension-Arm64:13
이스라엘(텔아비브)	arn:aws:lambda:il-central-1:895787185223:layer:AWS-AppConfig-Extension-Arm64:5

이전 확장 버전

이 섹션에는 ARN과 AWS AppConfig Lambda AWS 리전 확장 프로그램의 이전 버전에 대한 목록이 나와 있습니다. 이 목록에는 AWS AppConfig 에이전트 Lambda 확장의 모든 이전 버전에 대한 정보가 포함되어 있지는 않지만 새 버전이 출시되면 업데이트됩니다.

이전 확장 버전 (x86-64 플랫폼)

다음 표에는 ARN과 x86-64 AWS 리전 플랫폼용으로 개발된 AWS AppConfig 에이전트 Lambda 확장 프로그램의 이전 버전용 목록이 나와 있습니다.

날짜가 새 확장 프로그램으로 대체됨: 2023년 12월 1일

버전 2.0.181

지역	ARN
미국 동부(버지니아 북부)	arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension:113
미국 동부(오하이오)	arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension:81
미국 서부(캘리포니아 북부)	arn:aws:lambda:us-west-1:958113053741:layer:AWS-AppConfig-Extension:124
미국 서부(오레곤)	arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension:146
캐나다(중부)	arn:aws:lambda:ca-central-1:039592058896:layer:AWS-AppConfig-Extension:81
유럽(프랑크푸르트)	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension:93
유럽(취리히)	arn:aws:lambda:eu-central-2:758369105281:layer:AWS-AppConfig-Extension:32

지역	ARN
유럽(아일랜드)	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension:110
유럽(런던)	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension:81
유럽(파리)	arn:aws:lambda:eu-west-3:493207061005:layer:AWS-AppConfig-Extension:82
유럽(스톡홀름)	arn:aws:lambda:eu-north-1:646970417810:layer:AWS-AppConfig-Extension:142
유럽(밀라노)	arn:aws:lambda:eu-south-1:203683718741:layer:AWS-AppConfig-Extension:73
유럽(스페인)	arn:aws:lambda:eu-south-2:586093569114:layer:AWS-AppConfig-Extension:29
중국(베이징)	arn:aws-cn:lambda:cn-north-1:615057806174:layer:AWS-AppConfig-Extension:68
중국(닝샤)	arn:aws-cn:lambda:cn-northwest-1:615084187847:layer:AWS-AppConfig-Extension:68
아시아 태평양(홍콩)	arn:aws:lambda:ap-east-1:630222743974:layer:AWS-AppConfig-Extension:73

지역	ARN
아시아 태평양(도쿄)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension:84
아시아 태평양(서울)	arn:aws:lambda:ap-northeast-2:826293736237:layer:AWS-AppConfig-Extension:93
아시아 태평양(오사카)	arn:aws:lambda:ap-northeast-3:706869817123:layer:AWS-AppConfig-Extension:86
아시아 태평양(싱가포르)	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension:91
아시아 태평양(시드니)	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension:93
아시아 태평양(자카르타)	arn:aws:lambda:ap-southeast-3:418787028745:layer:AWS-AppConfig-Extension:64
아시아 태평양(멜버른)	arn:aws:lambda:ap-southeast-4:307021474294:layer:AWS-AppConfig-Extension:5
아시아 태평양(뭄바이)	arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension:94
아시아 태평양(하이데라바드)	arn:aws:lambda:ap-south-2:489524808438:layer:AWS-AppConfig-Extension:32

지역	ARN
남아메리카(상파울루)	arn:aws:lambda:sa-east-1:000010852771:layer:AWS-AppConfig-Extension:113
아프리카(케이프타운)	arn:aws:lambda:af-south-1:574348263942:layer:AWS-AppConfig-Extension:73
이스라엘(텔아비브)	arn:aws:lambda:il-central-1:895787185223:layer:AWS-AppConfig-Extension:7
중동(UAE)	arn:aws:lambda:me-central-1:662846165436:layer:AWS-AppConfig-Extension:34
중동(바레인)	arn:aws:lambda:me-south-1:559955524753:layer:AWS-AppConfig-Extension:73
AWS GovCloud (미국 동부)	arn:aws-us-gov:lambda:us-gov-east-1:946561847325:layer:AWS-AppConfig-Extension:46
AWS GovCloud (미국 서부)	arn:aws-us-gov:lambda:us-gov-west-1:946746059096:layer:AWS-AppConfig-Extension:46

최신 확장으로 대체된 날짜: 2023/08/14

버전 2.0.165

지역	ARN
미국 동부(버지니아 북부)	arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension:110
미국 동부(오하이오)	arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension:79
미국 서부(캘리포니아 북부)	arn:aws:lambda:us-west-1:958113053741:layer:AWS-AppConfig-Extension:121
미국 서부(오레곤)	arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension:143
캐나다(중부)	arn:aws:lambda:ca-central-1:039592058896:layer:AWS-AppConfig-Extension:79
유럽(프랑크푸르트)	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension:91
유럽(취리히)	arn:aws:lambda:eu-central-2:758369105281:layer:AWS-AppConfig-Extension:29
유럽(아일랜드)	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension:108
유럽(런던)	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension:79

지역	ARN
유럽(파리)	arn:aws:lambda:eu-west-3:493207061005:layer:AWS-AppConfig-Extension:80
유럽(스톡홀름)	arn:aws:lambda:eu-north-1:646970417810:layer:AWS-AppConfig-Extension:139
유럽(밀라노)	arn:aws:lambda:eu-south-1:203683718741:layer:AWS-AppConfig-Extension:71
유럽(스페인)	arn:aws:lambda:eu-south-2:586093569114:layer:AWS-AppConfig-Extension:26
중국(베이징)	arn:aws-cn:lambda:cn-north-1:615057806174:layer:AWS-AppConfig-Extension:66
중국(닝샤)	arn:aws-cn:lambda:cn-northwest-1:615084187847:layer:AWS-AppConfig-Extension:66
아시아 태평양(홍콩)	arn:aws:lambda:ap-east-1:630222743974:layer:AWS-AppConfig-Extension:71
아시아 태평양(도쿄)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension:82
아시아 태평양(서울)	arn:aws:lambda:ap-northeast-2:826293736237:layer:AWS-AppConfig-Extension:91

지역	ARN
아시아 태평양(오사카)	arn:aws:lambda:ap-northeast-3:706869817123:layer:AWS-AppConfig-Extension:84
아시아 태평양(싱가포르)	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension:89
아시아 태평양(시드니)	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension:91
아시아 태평양(자카르타)	arn:aws:lambda:ap-southeast-3:418787028745:layer:AWS-AppConfig-Extension:60
아시아 태평양(멜버른)	arn:aws:lambda:ap-southeast-4:307021474294:layer:AWS-AppConfig-Extension:2
아시아 태평양(뭄바이)	arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension:92
아시아 태평양(하이데라바드)	arn:aws:lambda:ap-south-2:489524808438:layer:AWS-AppConfig-Extension:29
남아메리카(상파울루)	arn:aws:lambda:sa-east-1:000010852771:layer:AWS-AppConfig-Extension:110
아프리카(케이프타운)	arn:aws:lambda:af-south-1:574348263942:layer:AWS-AppConfig-Extension:71

지역	ARN
중동(UAE)	arn:aws:lambda:me-central-1:662846165436:layer:AWS-AppConfig-Extension:31
중동(바레인)	arn:aws:lambda:me-south-1:559955524753:layer:AWS-AppConfig-Extension:71
AWS GovCloud (미국 동부)	arn:aws-us-gov:lambda:us-gov-east-1:946561847325:layer:AWS-AppConfig-Extension:44
AWS GovCloud (미국 서부)	arn:aws-us-gov:lambda:us-gov-west-1:946746059096:layer:AWS-AppConfig-Extension:44

최신 확장으로 대체된 날짜: 2023/02/21

버전 2.0.122

지역	ARN
미국 동부(버지니아 북부)	arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension:82
미국 동부(오하이오)	arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension:59
미국 서부(캘리포니아 북부)	arn:aws:lambda:us-west-1:958113053741:layer:AWS-AppConfig-Extension:93

지역	ARN
미국 서부(오레곤)	arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension:114
캐나다(중부)	arn:aws:lambda:ca-central-1:039592058896:layer:AWS-AppConfig-Extension:59
유럽(프랑크푸르트)	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension:70
유럽(아일랜드)	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension:82
유럽(런던)	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension:59
유럽(파리)	arn:aws:lambda:eu-west-3:493207061005:layer:AWS-AppConfig-Extension:60
유럽(스톡홀름)	arn:aws:lambda:eu-north-1:646970417810:layer:AWS-AppConfig-Extension:111
유럽(밀라노)	arn:aws:lambda:eu-south-1:203683718741:layer:AWS-AppConfig-Extension:54
중국(베이징)	arn:aws-cn:lambda:cn-north-1:615057806174:layer:AWS-AppConfig-Extension:52

지역	ARN
중국(닝샤)	arn:aws-cn:lambda:cn-northwest-1:615084187847:layer:AWS-AppConfig-Extension:52
아시아 태평양(홍콩)	arn:aws:lambda:ap-east-1:630222743974:layer:AWS-AppConfig-Extension:54
아시아 태평양(도쿄)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension:62
아시아 태평양(서울)	arn:aws:lambda:ap-northeast-2:826293736237:layer:AWS-AppConfig-Extension:70
아시아 태평양(오사카)	arn:aws:lambda:ap-northeast-3:706869817123:layer:AWS-AppConfig-Extension:59
아시아 태평양(싱가포르)	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension:64
아시아 태평양(시드니)	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension:70
아시아 태평양(자카르타)	arn:aws:lambda:ap-southeast-3:418787028745:layer:AWS-AppConfig-Extension:37
아시아 태평양(뭄바이)	arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension:71

지역	ARN
남아메리카(상파울루)	arn:aws:lambda:sa-east-1:000010852771:layer:AWS-AppConfig-Extension:82
아프리카(케이프타운)	arn:aws:lambda:af-south-1:574348263942:layer:AWS-AppConfig-Extension:54
중동(바레인)	arn:aws:lambda:me-south-1:559955524753:layer:AWS-AppConfig-Extension:54
AWS GovCloud (미국 동부)	arn:aws-us-gov:lambda:us-gov-east-1:946561847325:layer:AWS-AppConfig-Extension:29
AWS GovCloud (미국 서부)	arn:aws-us-gov:lambda:us-gov-west-1:946746059096:layer:AWS-AppConfig-Extension:29

최신 확장으로 대체된 날짜: 2022/08/23

버전 2.0.58

지역	ARN
미국 동부(버지니아 북부)	arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension:69
미국 동부(오하이오)	arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension:50

지역	ARN
미국 서부(캘리포니아 북부)	arn:aws:lambda:us-west-1:958113053741:layer:AWS-AppConfig-Extension:78
미국 서부(오레곤)	arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension:101
캐나다(중부)	arn:aws:lambda:ca-central-1:039592058896:layer:AWS-AppConfig-Extension:50
유럽(프랑크푸르트)	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension:59
유럽(아일랜드)	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension:69
유럽(런던)	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension:50
유럽(파리)	arn:aws:lambda:eu-west-3:493207061005:layer:AWS-AppConfig-Extension:51
유럽(스톡홀름)	arn:aws:lambda:eu-north-1:646970417810:layer:AWS-AppConfig-Extension:98
유럽(밀라노)	arn:aws:lambda:eu-south-1:203683718741:layer:AWS-AppConfig-Extension:47

지역	ARN
중국(베이징)	arn:aws-cn:lambda:cn-north-1:615057806174:layer:AWS-AppConfig-Extension:46
중국(닝샤)	arn:aws-cn:lambda:cn-northwest-1:615084187847:layer:AWS-AppConfig-Extension:46
아시아 태평양(홍콩)	arn:aws:lambda:ap-east-1:630222743974:layer:AWS-AppConfig-Extension:47
아시아 태평양(도쿄)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension:49
아시아 태평양(서울)	arn:aws:lambda:ap-northeast-2:826293736237:layer:AWS-AppConfig-Extension:59
아시아 태평양(오사카)	arn:aws:lambda:ap-northeast-3:706869817123:layer:AWS-AppConfig-Extension:46
아시아 태평양(싱가포르)	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension:51
아시아 태평양(시드니)	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension:59
아시아 태평양(자카르타)	arn:aws:lambda:ap-southeast-3:418787028745:layer:AWS-AppConfig-Extension:24

지역	ARN
아시아 태평양(뭄바이)	arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension:60
남아메리카(상파울루)	arn:aws:lambda:sa-east-1:000010852771:layer:AWS-AppConfig-Extension:69
아프리카(케이프타운)	arn:aws:lambda:af-south-1:574348263942:layer:AWS-AppConfig-Extension:47
중동(바레인)	arn:aws:lambda:me-south-1:559955524753:layer:AWS-AppConfig-Extension:47
AWS GovCloud (미국 동부)	arn:aws-us-gov:lambda:us-gov-east-1:946561847325:layer:AWS-AppConfig-Extension:23
AWS GovCloud (미국 서부)	arn:aws-us-gov:lambda:us-gov-west-1:946746059096:layer:AWS-AppConfig-Extension:23

최신 확장으로 대체된 날짜: 2022/04/21

버전 2.0.45

지역	ARN
미국 동부(버지니아 북부)	arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension:68

지역	ARN
미국 동부(오하이오)	arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension:49
미국 서부(캘리포니아 북부)	arn:aws:lambda:us-west-1:958113053741:layer:AWS-AppConfig-Extension:77
미국 서부(오레곤)	arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension:100
캐나다(중부)	arn:aws:lambda:ca-central-1:039592058896:layer:AWS-AppConfig-Extension:49
유럽(프랑크푸르트)	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension:58
유럽(아일랜드)	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension:68
유럽(런던)	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension:49
유럽(파리)	arn:aws:lambda:eu-west-3:493207061005:layer:AWS-AppConfig-Extension:50
유럽(스톡홀름)	arn:aws:lambda:eu-north-1:646970417810:layer:AWS-AppConfig-Extension:97

지역	ARN
유럽(밀라노)	arn:aws:lambda:eu-south-1:203683718741:layer:AWS-AppConfig-Extension:46
중국(베이징)	arn:aws-cn:lambda:cn-north-1:615057806174:layer:AWS-AppConfig-Extension:45
중국(닝샤)	arn:aws-cn:lambda:cn-northwest-1:615084187847:layer:AWS-AppConfig-Extension:45
아시아 태평양(홍콩)	arn:aws:lambda:ap-east-1:630222743974:layer:AWS-AppConfig-Extension:46
아시아 태평양(도쿄)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension:48
아시아 태평양(서울)	arn:aws:lambda:ap-northeast-2:826293736237:layer:AWS-AppConfig-Extension:58
아시아 태평양(오사카)	arn:aws:lambda:ap-northeast-3:706869817123:layer:AWS-AppConfig-Extension:45
아시아 태평양(싱가포르)	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension:50
아시아 태평양(시드니)	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension:58

지역	ARN
아시아 태평양(자카르타)	arn:aws:lambda:ap-southeast-3:418787028745:layer:AWS-AppConfig-Extension:23
아시아 태평양(뭄바이)	arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension:59
남아메리카(상파울루)	arn:aws:lambda:sa-east-1:000010852771:layer:AWS-AppConfig-Extension:68
아프리카(케이프타운)	arn:aws:lambda:af-south-1:574348263942:layer:AWS-AppConfig-Extension:46
중동(바레인)	arn:aws:lambda:me-south-1:559955524753:layer:AWS-AppConfig-Extension:46
AWS GovCloud (미국 동부)	arn:aws-us-gov:lambda:us-gov-east-1:946561847325:layer:AWS-AppConfig-Extension:22
AWS GovCloud (미국 서부)	arn:aws-us-gov:lambda:us-gov-west-1:946746059096:layer:AWS-AppConfig-Extension:22

최신 확장으로 대체된 날짜: 2022/03/15

버전 2.0.30

지역	ARN
미국 동부(버지니아 북부)	arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension:61
미국 동부(오하이오)	arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension:47
미국 서부(캘리포니아 북부)	arn:aws:lambda:us-west-1:958113053741:layer:AWS-AppConfig-Extension:61
미국 서부(오레곤)	arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension:89
캐나다(중부)	arn:aws:lambda:ca-central-1:039592058896:layer:AWS-AppConfig-Extension:47
유럽(프랑크푸르트)	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension:54
유럽(아일랜드)	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension:59
유럽(런던)	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension:47
유럽(파리)	arn:aws:lambda:eu-west-3:493207061005:layer:AWS-AppConfig-Extension:48

지역	ARN
유럽(스톡홀름)	arn:aws:lambda:eu-north-1:646970417810:layer:AWS-AppConfig-Extension:86
유럽(밀라노)	arn:aws:lambda:eu-south-1:203683718741:layer:AWS-AppConfig-Extension:44
중국(베이징)	arn:aws-cn:lambda:cn-north-1:615057806174:layer:AWS-AppConfig-Extension:43
중국(닝샤)	arn:aws-cn:lambda:cn-northwest-1:615084187847:layer:AWS-AppConfig-Extension:43
아시아 태평양(홍콩)	arn:aws:lambda:ap-east-1:630222743974:layer:AWS-AppConfig-Extension:44
아시아 태평양(도쿄)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension:45
아시아 태평양(오사카)	arn:aws:lambda:ap-northeast-3:706869817123:layer:AWS-AppConfig-Extension:42
아시아 태평양(서울)	arn:aws:lambda:ap-northeast-2:826293736237:layer:AWS-AppConfig-Extension:54
아시아 태평양(싱가포르)	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension:45

지역	ARN
아시아 태평양(시드니)	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension:54
아시아 태평양(자카르타)	arn:aws:lambda:ap-southeast-3:418787028745:layer:AWS-AppConfig-Extension:13
아시아 태평양(뭄바이)	arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension:55
남아메리카(상파울루)	arn:aws:lambda:sa-east-1:000010852771:layer:AWS-AppConfig-Extension:61
아프리카(케이프타운)	arn:aws:lambda:af-south-1:574348263942:layer:AWS-AppConfig-Extension:44
중동(바레인)	arn:aws:lambda:me-south-1:559955524753:layer:AWS-AppConfig-Extension:44
AWS GovCloud (미국 동부)	arn:aws-us-gov:lambda:us-gov-east-1:946561847325:layer:AWS-AppConfig-Extension:20
AWS GovCloud (미국 서부)	arn:aws-us-gov:lambda:us-gov-west-1:946746059096:layer:AWS-AppConfig-Extension:20

이전 확장 버전 (ARM64 플랫폼)

다음 표에는 ARM64 AWS 리전 플랫폼용으로 개발된 AWS AppConfig 에이전트 Lambda 확장 프로그램의 이전 버전용 ARN 및 목록이 나와 있습니다.

새 확장 프로그램으로 대체된 날짜: 2023년 12월 1일

버전 2.0.181

지역	ARN
미국 동부(버지니아 북부)	arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension-Arm64:46
미국 동부(오하이오)	arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension-Arm64:33
미국 서부(캘리포니아 북부)	arn:aws:lambda:us-west-1:958113053741:layer:AWS-AppConfig-Extension-Arm64:1
미국 서부(오레곤)	arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension-Arm64:48
캐나다(중부)	arn:aws:lambda:ca-central-1:039592058896:layer:AWS-AppConfig-Extension-Arm64:1
유럽(프랑크푸르트)	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension-Arm64:36
유럽(아일랜드)	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension-Arm64:48
유럽(런던)	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension-Arm64:33

지역	ARN
유럽(파리)	arn:aws:lambda:eu-west-3:493207061005:layer:AWS-AppConfig-Extension-Arm64:1
유럽(스톡홀름)	arn:aws:lambda:eu-north-1:646970417810:layer:AWS-AppConfig-Extension-Arm64:1
유럽(밀라노)	arn:aws:lambda:eu-south-1:203683718741:layer:AWS-AppConfig-Extension-Arm64:1
아시아 태평양(홍콩)	arn:aws:lambda:ap-east-1:630222743974:layer:AWS-AppConfig-Extension-Arm64:1
아시아 태평양(도쿄)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension-Arm64:37
아시아 태평양(서울)	arn:aws:lambda:ap-northeast-2:826293736237:layer:AWS-AppConfig-Extension-Arm64:1
아시아 태평양(오사카)	arn:aws:lambda:ap-northeast-3:706869817123:layer:AWS-AppConfig-Extension-Arm64:1
아시아 태평양(싱가포르)	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension-Arm64:43
아시아 태평양(시드니)	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension-Arm64:36

지역	ARN
아시아 태평양(자카르타)	arn:aws:lambda:ap-southeast-3:418787028745:layer:AWS-AppConfig-Extension-Arm64:1
아시아 태평양(뭄바이)	arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension-Arm64:36
남아메리카(상파울루)	arn:aws:lambda:sa-east-1:000010852771:layer:AWS-AppConfig-Extension-Arm64:1
아프리카(케이프타운)	arn:aws:lambda:af-south-1:574348263942:layer:AWS-AppConfig-Extension-Arm64:1
중동(바레인)	arn:aws:lambda:me-south-1:559955524753:layer:AWS-AppConfig-Extension-Arm64:1

최신 확장으로 대체된 날짜: 2023/03/30

버전 2.0.165

지역	ARN
미국 동부(버지니아 북부)	arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension-Arm64:43
미국 동부(오하이오)	arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension-Arm64:31

지역	ARN
미국 서부(오레곤)	arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension-Arm64:45
유럽(프랑크푸르트)	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension-Arm64:34
유럽(아일랜드)	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension-Arm64:46
유럽(런던)	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension-Arm64:31
아시아 태평양(도쿄)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension-Arm64:35
아시아 태평양(싱가포르)	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension-Arm64:41
아시아 태평양(시드니)	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension-Arm64:34
아시아 태평양(뭄바이)	arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension-Arm64:34

최신 확장으로 대체된 날짜: 2023/02/21

버전 2.0.122

지역	ARN
미국 동부(버지니아 북부)	arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension-Arm64:15
미국 동부(오하이오)	arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension-Arm64:11
미국 서부(오레곤)	arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension-Arm64:16
유럽(프랑크푸르트)	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension-Arm64:13
유럽(아일랜드)	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension-Arm64:20
유럽(런던)	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension-Arm64:11
아시아 태평양(도쿄)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension-Arm64:15
아시아 태평양(싱가포르)	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension-Arm64:16
아시아 태평양(시드니)	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension-Arm64:13

지역	ARN
아시아 태평양(뭄바이)	arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension-Arm64:13

최신 확장으로 대체된 날짜: 2022/08/23

버전 2.0.58

지역	ARN
미국 동부(버지니아 북부)	arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension-Arm64:2
미국 동부(오하이오)	arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension-Arm64:2
미국 서부(오레곤)	arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension-Arm64:3
유럽(프랑크푸르트)	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension-Arm64:2
유럽(아일랜드)	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension-Arm64:7
유럽(런던)	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension-Arm64:2

지역	ARN
아시아 태평양(도쿄)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension-Arm64:2
아시아 태평양(싱가포르)	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension-Arm64:3
아시아 태평양(시드니)	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension-Arm64:2
아시아 태평양(뭄바이)	arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension-Arm64:2

최신 확장으로 대체된 날짜: 2022/04/21

버전 2.0.45

지역	ARN
미국 동부(버지니아 북부)	arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension-Arm64:1
미국 동부(오하이오)	arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension-Arm64:1
미국 서부(오레곤)	arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension-Arm64:2

지역	ARN
유럽(프랑크푸르트)	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension-Arm64:1
유럽(아일랜드)	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension-Arm64:6
유럽(런던)	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension-Arm64:1
아시아 태평양(도쿄)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension-Arm64:1
아시아 태평양(싱가포르)	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension-Arm64:2
아시아 태평양(시드니)	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension-Arm64:1
아시아 태평양(뭄바이)	arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension-Arm64:1

컨테이너 이미지를 사용하여 AWS AppConfig 에이전트 Lambda 확장 프로그램 추가

AWS AppConfig 에이전트 Lambda 확장 프로그램을 컨테이너 이미지로 패키징하여 Amazon Elastic Container Registry (Amazon ECR) 에서 호스팅되는 컨테이너 레지스트리에 업로드할 수 있습니다.

AWS AppConfig 에이전트 Lambda 확장 프로그램을 Lambda 컨테이너 이미지로 추가하려면

1. 아래와 같이 () 에 AWS 리전 와 Amazon 리소스 이름 (ARN AWS CLI) 을 AWS Command Line Interface 입력합니다. 지역 및 ARN 값을 사용자의 지역 및 일치하는 ARN으로 대체하여 Lambda 계층의 사본을 검색하십시오.

```
aws lambda get-layer-version-by-arn \
  --region us-east-1 \
  --arn arn:aws:lambda::layer:AWS-AppConfig-Extension:00
```

   다음은 응답입니다.

```
{
  "LayerVersionArn": "arn:aws:lambda::layer:AWS-AppConfig-Extension:00",
  "Description": "AWS AppConfig extension: Use dynamic configurations deployed via
 AWS AppConfig for your AWS Lambda functions",
  "CreatedDate": "2021-04-01T02:37:55.339+0000",
  "LayerArn": "arn:aws:lambda::layer:AWS-AppConfig-Extension",

  "Content": {
    "CodeSize": 5228073,
    "CodeSha256": "8otOgbLQbexpUm3rKlMhvcE6Q5TvwcLCKrc4Oe+vmMY=",
    "Location" : "S3-Bucket-Location-URL"
  },

  "Version": 30,
  "CompatibleRuntimes": [
    "python3.8",
    "python3.7",
    "nodejs12.x",
    "ruby2.7"
  ],
}
```

2. 위 응답에서 `Location`에 대해 반환된 값은 Lambda 확장을 포함하는 Amazon Simple Storage Service(S3) 버킷의 URL입니다. URL을 웹 브라우저에 붙여 넣어 Lambda 확장 .zip 파일을 다운로드합니다.

   > **Note**
   >
   > Amazon S3 버킷 URL은 10분 동안만 사용할 수 있습니다.

(선택 사항) 또는 다음 curl 명령을 사용하여 Lambda 확장을 다운로드할 수 있습니다.

```
curl -o extension.zip "S3-Bucket-Location-URL"
```

(선택 사항) 또는 1단계와 2단계를 결합하여 ARN을 검색하고 .zip 확장 파일을 한꺼번에 다운로드할 수도 있습니다.

```
aws lambda get-layer-version-by-arn \
  --arn arn:aws:lambda::layer:AWS-AppConfig-Extension:00 \
  | jq -r '.Content.Location' \
  | xargs curl -o extension.zip
```

3. Dockerfile에 다음 줄을 추가하여 컨테이너 이미지에 확장을 추가합니다.

```
COPY extension.zip extension.zip
RUN yum install -y unzip \
  && unzip extension.zip /opt \
  && rm -f extension.zip
```

4. Lambda 함수 실행 역할에 GetConfigurationappconfig: 권한 세트가 있는지 확인하십시오.

예

이 섹션에는 컨테이너 이미지 기반 Python Lambda 함수에서 AWS AppConfig 에이전트 Lambda 확장을 활성화하는 예제가 포함되어 있습니다.

1. 다음과 유사한 Dockerfile을 생성합니다.

```
FROM public.ecr.aws/lambda/python:3.8 AS builder
COPY extension.zip extension.zip
RUN yum install -y unzip \
  && unzip extension.zip -d /opt \
  && rm -f extension.zip

FROM public.ecr.aws/lambda/python:3.8
COPY --from=builder /opt /opt
COPY index.py ${LAMBDA_TASK_ROOT}
CMD [ "index.handler" ]
```

2. 확장 레이어를 Dockerfile과 동일한 디렉터리에 다운로드합니다.

```
aws lambda get-layer-version-by-arn \
  --arn arn:aws:lambda::layer:AWS-AppConfig-Extension:00 \
  | jq -r '.Content.Location' \
  | xargs curl -o extension.zip
```

3. Dockerfile과 동일한 디렉터리에서 index.py라는 이름의 Python 파일을 만듭니다.

```
import urllib.request

def handler(event, context):
    return {
       # replace parameters here with your application, environment, and
 configuration names
       'configuration': get_configuration('myApp', 'myEnv', 'myConfig'),
    }

def get_configuration(app, env, config):
    url = f'http://localhost:2772/applications/{app}/environments/{env}/
configurations/{config}'
    return urllib.request.urlopen(url).read()
```

4. 다음 단계를 실행함으로써 docker 이미지를 구축하여 Amazon ECR에 업로드합니다.

```
// set environment variables
export ACCOUNT_ID = <YOUR_ACCOUNT_ID>
export REGION = <AWS_REGION>

// create an ECR repository
aws ecr create-repository --repository-name test-repository

// build the docker image
docker build -t test-image .

// sign in to AWS
aws ecr get-login-password \
  | docker login \
  --username AWS \
  --password-stdin "$ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.com"

// tag the image
docker tag test-image:latest "$ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.com/test-
repository:latest"
```

```
// push the image to ECR
docker push "$ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.com/test-repository:latest"
```

5. 위에서 생성한 Amazon ECR 이미지를 사용하여 Lambda 함수를 생성합니다. 컨테이너로서의 Lambda 함수에 대한 자세한 내용은 컨테이너 이미지로 정의된 Lambda 함수 생성을 참조하십시오.
6. Lambda 함수 실행 역할에 GetConfigurationappconfig: 권한 세트가 있는지 확인하십시오.

OpenAPI와 통합하기

OpenAPI에 대한 다음 YAML 사양을 사용하여 OpenAPI 생성기와 같은 도구를 사용하여 SDK를 생성할 수 있습니다. 애플리케이션, 환경 또는 구성의 하드코딩된 값을 포함하도록 이 사양을 업데이트할 수 있습니다. 또한 경로를 추가하고 (구성 유형이 여러 개인 경우) 구성 스키마를 포함하여 SDK 클라이언트의 구성별 유형 모델을 생성할 수 있습니다. OpenAPI(스웨거라고도 함)에 대한 자세한 내용은 OpenAPI 사양을 참조하십시오.

```
openapi: 3.0.0
info:
  version: 1.0.0
  title: AppConfig Agent Lambda extension API
  description: An API model for the AppConfig Agent Lambda extension.
servers:
  - url: https://localhost:{port}/
    variables:
      port:
        default:
          '2772'
paths:
  /applications/{Application}/environments/{Environment}/configurations/
{Configuration}:
    get:
      operationId: getConfiguration
      tags:
        - configuration
      parameters:
        - in: path
          name: Application
          description: The application for the configuration to get. Specify either the
 application name or the application ID.
          required: true
          schema:
```

```
            type: string
        - in: path
          name: Environment
          description: The environment for the configuration to get. Specify either the
environment name or the environment ID.
          required: true
          schema:
            type: string
        - in: path
          name: Configuration
          description: The configuration to get. Specify either the configuration name
or the configuration ID.
          required: true
          schema:
            type: string
      responses:
        200:
          headers:
            ConfigurationVersion:
              schema:
                type: string
          content:
            application/octet-stream:
              schema:
                type: string
                format: binary
          description: successful config retrieval
        400:
          description: BadRequestException
          content:
            application/text:
              schema:
                $ref: '#/components/schemas/Error'
        404:
          description: ResourceNotFoundException
          content:
            application/text:
              schema:
                $ref: '#/components/schemas/Error'
        500:
          description: InternalServerException
          content:
            application/text:
              schema:
```

```
                $ref: '#/components/schemas/Error'
        502:
          description: BadGatewayException
          content:
            application/text:
              schema:
                $ref: '#/components/schemas/Error'
        504:
          description: GatewayTimeoutException
          content:
            application/text:
              schema:
                $ref: '#/components/schemas/Error'

components:
  schemas:
    Error:
      type: string
      description: The response error
```

Amazon EC2 인스턴스에서 구성 데이터 검색

에이전트를 사용하여 Amazon Elastic Compute Cloud (Amazon EC2) Linux 인스턴스에서 실행되는 AWS AppConfig 애플리케이션과 통합할 수 있습니다. AWS AppConfig 에이전트는 다음과 같은 방법으로 애플리케이션 처리 및 관리를 개선합니다.

- 에이전트는 AWS Identity and Access Management (IAM) 역할을 사용하고 구성 데이터의 로컬 캐시를 관리하여 사용자를 대신하여 전화를 겁니다 AWS AppConfig . 로컬 캐시에서 구성 데이터를 가져오면 애플리케이션에서 구성 데이터를 관리하는 데 필요한 코드 업데이트 횟수가 줄어들고, 구성 데이터를 밀리초 단위로 검색할 수 있으며, 이러한 데이터에 대한 호출을 방해할 수 있는 네트워크 문제의 영향을 받지 않습니다.*
- 에이전트는 기능 플래그를 검색하고 해결하기 AWS AppConfig 위한 기본 환경을 제공합니다.
- 에이전트는 기본적으로 캐싱 전략, 폴링 간격, 로컬 구성 데이터의 가용성에 대한 모범 사례를 제공하는 동시에 후속 서비스 호출에 필요한 구성 토큰을 추적합니다.
- 에이전트는 백그라운드에서 실행되는 동안 정기적으로 AWS AppConfig 데이터 플레인을 폴링하여 구성 데이터 업데이트를 확인합니다. 애플리케이션은 포트 2772 (사용자 지정 가능한 기본 포트 값)에서 localhost에 연결하고 HTTP GET을 호출하여 데이터를 검색함으로써 데이터를 검색할 수 있습니다.

*AWS AppConfig 에이전트는 서비스가 구성 데이터를 처음 검색할 때 데이터를 캐시합니다. 이러한 이유로 데이터를 검색하기 위한 첫 번째 호출은 후속 호출보다 느립니다.

주제

- 1단계: (필수) 리소스 생성 및 권한 구성
- 2단계: (필수) Amazon EC2 AWS AppConfig 인스턴스에 에이전트 설치 및 시작
- 3단계: (선택 사항이지만 권장됨) CloudWatch Logs에 로그 파일 전송
- 4단계: (선택 사항) 환경 변수를 사용하여 Amazon EC2용 AWS AppConfig 에이전트 구성
- 5단계: (필수) 구성 데이터 검색
- 6단계 (선택 사항이지만 권장): 에이전트 업데이트 자동화 AWS AppConfig

1단계: (필수) 리소스 생성 및 권한 구성

Amazon EC2 인스턴스에서 실행되는 AWS AppConfig 애플리케이션과 통합하려면 기능 플래그 또는 자유 형식 구성 데이터를 비롯한 AWS AppConfig 아티팩트와 구성 데이터를 생성해야 합니다. 자세한 정보는 기능 플래그 및 자유 형식 구성 데이터 생성 AWS AppConfig을 참조하세요.

에서 AWS AppConfig호스팅하는 구성 데이터를 검색하려면 애플리케이션이 데이터 플레인에 액세스할 수 있도록 구성되어 있어야 합니다. AWS AppConfig 애플리케이션에 액세스 권한을 부여하려면 Amazon EC2 인스턴스 역할에 할당된 IAM 권한 정책을 업데이트하십시오. 특히, 정책에 `appconfig:StartConfigurationSession` 및 `appconfig:GetLatestConfiguration` 액션을 추가해야 합니다. 예:

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

정책에 권한을 추가하는 것에 대한 자세한 내용은 IAM 사용 설명서의 IAM 자격 증명 권한 추가 및 제거를 참조하십시오.

2단계: (필수) Amazon EC2 AWS AppConfig 인스턴스에 에이전트 설치 및 시작

AWS AppConfig 에이전트는 에서 관리하는 Amazon Simple Storage 서비스 (Amazon S3) 버킷에서 호스팅됩니다. AWS다음 절차를 사용하여 Linux 인스턴스에 최신 버전의 에이전트를 설치합니다. 애플리케이션이 여러 인스턴스에 분산되어 있는 경우 애플리케이션을 호스팅하는 각 인스턴스에서 이 절차를 수행해야 합니다.

> **Note**
>
> 다음과 같은 정보를 참고합니다.
>
> - AWS AppConfig 에이전트는 커널 버전 4.15 이상을 실행하는 Linux 운영 체제에서 사용할 수 있습니다. Ubuntu와 같은 Debian 기반 시스템은 지원되지 않습니다.
> - 에이전트는 x86_64 및 ARM64 아키텍처를 지원합니다.
> - 분산 애플리케이션의 경우 Auto Scaling 그룹의 Amazon EC2 사용자 데이터에 설치 및 시작 명령을 추가하는 것이 좋습니다. 그러면 각 인스턴스가 명령을 자동으로 실행합니다. 자세한 정보는 Linux 인스턴스용 Amazon EC2 사용 설명서의 시작 시 Linux 인스턴스에서 명령 실행을 참조하십시오. 추가로 Amazon EC2 Auto Scaling 사용 설명서의 자습서: 인스턴스 메타데이터를 통해 대상 수명 주기 상태를 검색하기 위한 사용자 데이터 구성하기를 참조하십시오.
> - 이 주제 전체의 절차에서는 인스턴스에 로그인하여 명령을 실행하여 에이전트를 설치하는 등의 액션을 수행하는 방법을 설명합니다. 의 AWS Systems Manager기능인 Run Command를 사용하여 로컬 클라이언트 시스템에서 명령을 실행하고 하나 이상의 인스턴스를 대상으로 지정할 수 있습니다. 자세한 내용은 AWS Systems Manager 사용 설명서에서 AWS Systems Manager Run Command를 참조하십시오.
> - AWS AppConfig Amazon EC2 Linux 인스턴스의 에이전트는 서비스입니다. `systemd`

인스턴스에 AWS AppConfig 에이전트를 설치하고 시작하려면

1. Linux 인스턴스에 로그인합니다.
2. 터미널을 열고 x86_64 아키텍처에 대한 관리자 권한으로 다음 명령을 실행합니다.

```
sudo yum install https://s3.amazonaws.com/aws-appconfig-downloads/aws-appconfig-agent/linux/x86_64/latest/aws-appconfig-agent.rpm
```

ARM64 아키텍처의 경우 다음 명령을 실행합니다.

```
sudo yum install https://s3.amazonaws.com/aws-appconfig-downloads/aws-appconfig-agent/linux/arm64/latest/aws-appconfig-agent.rpm
```

특정 버전의 AWS AppConfig 에이전트를 설치하려면 URL을 특정 버전 번호로 `latest` 바꾸십시오. 다음은 x86_64의 예시입니다.

```
sudo yum install https://s3.amazonaws.com/aws-appconfig-downloads/aws-appconfig-agent/linux/x86_64/2.0.2/aws-appconfig-agent.rpm
```

3. 에이전트를 시작하려면 다음 명령을 실행합니다.

```
sudo systemctl start aws-appconfig-agent
```

4. 다음 명령을 실행하여 에이전트가 실행 중인지 확인합니다.

```
sudo systemctl status aws-appconfig-agent
```

명령이 제대로 실행되면 다음과 비슷한 정보를 반환합니다.

```
aws-appconfig-agent.service - aws-appconfig-agent
  ...
  Active: active (running) since Mon 2023-07-26 00:00:00 UTC; 0s ago
  ...
```

> **Note**
>
> 에이전트를 중지하려면 다음 명령을 실행합니다.
>
> ```
> sudo systemctl stop aws-appconfig-agent
> ```

3단계: (선택 사항이지만 권장됨) CloudWatch Logs에 로그 파일 전송

기본적으로 AWS AppConfig 에이전트는 STDERR에 로그를 게시합니다. Systemd는 Linux 인스턴스에서 실행되는 모든 서비스의 STDOUT 및 STDERR을 시스템 저널로 리디렉션합니다. 하나 또는 두 개의 인스턴스에서만 AWS AppConfig 에이전트를 실행하는 경우 systemd 저널에서 로그 데이터를 보고 관리할 수 있습니다. 분산 애플리케이션에 강력히 권장되는 더 나은 솔루션은 디스크에 로그 파일을 기록한 다음 Amazon CloudWatch 에이전트를 사용하여 로그 데이터를 AWS 클라우드에 업로드하는 것입니다. 또한 인스턴스에서 오래된 로그 파일을 삭제하도록 CloudWatch 에이전트를 구성하여 인스턴스의 디스크 공간이 부족해지는 것을 방지할 수 있습니다.

디스크 로깅을 활성화하려면 4단계: (선택 사항) 환경 변수를 사용하여 Amazon EC2용 AWS AppConfig 에이전트 구성에 설명된 대로 `LOG_PATH` 환경 변수를 설정해야 합니다.

에이전트를 시작하려면 Amazon 사용 설명서의 CloudWatch 에이전트를 사용하여 Amazon EC2 인스턴스 및 온프레미스 서버에서 지표 및 로그 수집을 참조하십시오. CloudWatch CloudWatch Systems Manager의 기능인 빠른 설치를 사용하여 CloudWatch 에이전트를 빠르게 설치할 수 있습니다. 자세한 내용은 AWS Systems Manager 사용 설명서의 빠른 설정 호스트 관리를 참조하십시오.

> **Warning**
>
> CloudWatch 에이전트를 사용하지 않고 디스크에 로그 파일을 쓰도록 선택한 경우 이전 로그 파일을 삭제해야 합니다. AWS AppConfig 에이전트는 1시간마다 로그 파일을 자동으로 교체합니다. 오래된 로그 파일을 삭제하지 않으면 인스턴스의 디스크 공간이 부족해질 수 있습니다.

인스턴스에 CloudWatch 에이전트를 설치한 후 CloudWatch 에이전트 구성 파일을 생성합니다. 구성 파일은 CloudWatch 에이전트에게 AWS AppConfig 에이전트 로그 파일 사용 방법을 안내합니다. 에이전트 구성 파일을 만드는 방법에 대한 자세한 내용은 CloudWatch 에이전트 구성 파일 만들기를 참조하십시오. CloudWatch

인스턴스의 CloudWatch 에이전트 구성 파일에 다음 `logs` 섹션을 추가하고 변경 내용을 저장합니다.

```
"logs": {
  "logs_collected": {
    "files": {
      "collect_list": [
        {
          "file_path": "/path_you_specified_for_logging",
```

```
          "log_group_name": "${YOUR_LOG_GROUP_NAME}/aws-appconfig-agent.log",
          "auto_removal": true
        },
        ...
      ]
    },
    ...
  },
  ...
}
```

값이 인 경우 `auto_removal` 에이전트는 `true` 순환된 CloudWatch AWS AppConfig 에이전트 로그 파일을 자동으로 삭제합니다.

4단계: (선택 사항) 환경 변수를 사용하여 Amazon EC2용 AWS AppConfig 에이전트 구성

환경 변수를 사용하여 Amazon EC2용 AWS AppConfig 에이전트를 구성할 수 있습니다. `systemd` 서비스의 환경 변수를 설정하려면 드롭인 단위 파일을 생성합니다. 다음 예제는 드롭인 유닛 파일을 생성하여 AWS AppConfig 에이전트 로깅 수준을 로 설정하는 방법을 보여줍니다. `DEBUG`

환경 변수에 대한 드롭인 단위 파일을 만드는 방법의 예

1. Linux 인스턴스에 로그인합니다.
2. 터미널을 열고 다음 명령을 관리자 권한으로 실행합니다. 이 명령은 구성 디렉터리를 생성합니다.

   ```
   sudo mkdir /etc/systemd/system/aws-appconfig-agent.service.d
   ```

3. 다음 명령을 실행해 드롭인 단위 파일을 생성합니다. *`file_name`*을 파일 이름으로 바꿉니다. 확장자는 다음과 같아야 합니다: `.conf`

   ```
   sudo touch /etc/systemd/system/aws-appconfig-agent.service.d/file_name.conf
   ```

4. 드롭인 단위 파일에 정보를 입력합니다. 다음 예제에서는 환경 변수를 정의하는 `Service` 섹션을 추가합니다. 이 예제에서는 AWS AppConfig 에이전트 로그 수준을 `DEBUG`로 설정합니다.

   ```
   [Service]
   Environment=LOG_LEVEL=DEBUG
   ```

5. 다음 명령을 실행하여 systemd 구성을 다시 로드합니다.

```
sudo systemctl daemon-reload
```

6. 다음 명령을 실행하여 AWS AppConfig 에이전트를 다시 시작합니다.

```
sudo systemctl restart aws-appconfig-agent
```

드롭인 단위 파일에 다음 환경 변수를 지정하여 Amazon EC2용 AWS AppConfig 에이전트를 구성할 수 있습니다.

환경 변수	Details	기본값
ACCESS_TOKEN	이 환경 변수는 에이전트 HTTP 서버에 구성 데이터를 요청할 때 제공해야 하는 토큰을 정의합니다. 토큰 값은 권한 부여 유형이 Bearer인 HTTP 요청 승인 헤더에서 설정해야 합니다. 의 예는 다음과 같습니다. `GET /applications/my_app/... Host: localhost:2772 Authorization: Bearer <token value>`	None
BACKUP_DIRECTORY	이 환경 변수를 사용하면 AWS AppConfig 에이전트가 검색한 각 구성의 백업을 지정된 디렉터리에 저장할 수 있습니다. **Important** 디스크에 백업된 구성은 암호화되지 않습니	None

환경 변수	Details	기본값
	다. 구성에 민감한 데이터가 포함된 경우 파일 시스템 권한에 대한 최소 권한 원칙을 적용하는 것이 좋습니다. AWS AppConfig 자세한 정보는 AWS AppConfig의 보안을 참조하세요.	
HTTP_PORT	이 환경 변수는 에이전트의 HTTP 서버가 실행되는 포트를 지정합니다.	2772
LOG_LEVEL	이 환경 변수는 에이전트가 기록하는 세부 정보 수준을 지정합니다. 각 레벨에는 현재 레벨과 모든 상위 레벨이 포함됩니다. 변수는 대소문자를 구분합니다. 로그 수준은 가장 세부적인 것부터 세부적이지 않은 것까지 debug, info, warn, error, 및 none 입니다. Debug는 타이밍 정보를 비롯한 에이전트에 대한 자세한 정보가 포함됩니다.	info
LOG_PATH	로그가 기록되는 디스크 위치. 지정하지 않으면 로그가 stderr에 기록됩니다.	None

환경 변수	Details	기본값
MANIFEST	이 환경 변수는 AWS AppConfig 에이전트가 다중 계정 검색과 같은 추가 구성별 기능을 활용하고 구성을 디스크에 저장하도록 구성합니다. 다음 값 중 하나를 입력할 수 있습니다. • "app:env:manifest-config" • "file:/fully/qualified/path/to/manifest.json" 이러한 기능에 대한 자세한 내용은 추가 검색 기능 섹션을 참조하세요.	true
MAX_CONNECTIONS	이 환경 변수는 에이전트가 AWS AppConfig에서 구성을 검색하는 데 사용하는 최대 연결 수를 구성합니다.	3

환경 변수	Details	기본값
POLL_INTERVAL	이 환경 변수는 에이전트가 업데이트된 구성 데이터를 AWS AppConfig 폴링하는 빈도를 제어합니다. 간격을 초 단위로 지정할 수 있습니다. 시간 단위를 사용하여 숫자를 지정할 수도 있습니다. 초는 s, 분은 m, 시간은 h입니다. 단위를 지정하지 않으면 에이전트의 기본값은 초로 설정됩니다. 예를 들어 60, 60초, 1분은 폴링 간격이 동일합니다.	45초
PREFETCH_LIST	이 환경 변수는 에이전트가 AWS AppConfig 시작하자마자 요청하는 구성 데이터를 지정합니다.	None
PRELOAD_BACKUPS	로 `true` 설정하면 AWS AppConfig 에이전트는 에서 찾은 구성 백업을 BACKUP_DIRECTORY 메모리로 로드하고 서비스에 새 버전이 있는지 즉시 확인합니다. 로 `false` 설정된 경우 AWS AppConfig 에이전트는 서비스에서 구성 데이터를 검색할 수 없는 경우 (예: 네트워크에 문제가 있는 경우) 에만 구성 백업의 내용을 로드합니다.	true

환경 변수	Details	기본값
PROXY_HEADERS	이 환경 변수는 PROXY_URL 환경 변수에서 참조되는 프록시에 필요한 헤더를 지정합니다. 값은 쉼표로 구분된 헤더 목록입니다. 각 헤더는 다음 형식을 사용합니다. `"header: value"`	None
PROXY_URL	이 환경 변수는 다음을 포함하여 에이전트와의 연결에 사용할 프록시 URL을 지정합니다 AWS AppConfig. AWS 서비스HTTPS및 HTTP URL이 지원됩니다.	None

환경 변수	Details	기본값
REQUEST_TIMEOUT	이 환경 변수는 에이전트가 응답을 기다리는 시간을 제어합니다. AWS AppConfig서비스가 응답하지 않으면 요청이 실패합니다. 초기 데이터 검색을 위한 요청인 경우 에이전트는 애플리케이션에 오류를 반환합니다. 업데이트된 데이터에 대한 백그라운드 확인 중에 제한 시간이 초과되면 에이전트는 오류를 기록하고 잠시 후 다시 시도합니다. 제한 시간을 밀리초로 지정할 수 있습니다. 시간 단위로 숫자를 지정할 수도 있습니다. 밀리초는 ms이고 초는 s입니다. 단위를 지정하지 않으면 에이전트의 기본값은 밀리초로 설정됩니다. 예를 들어 5000, 5000ms 및 5s의 경우 요청 제한 시간 값이 동일합니다.	3000 밀리초
ROLE_ARN	이 환경 변수는 IAM 역할의 Amazon 리소스 이름 (ARN) 을 지정합니다. AWS AppConfig 에이전트는 이 역할을 맡아 구성 데이터를 검색합니다.	None
ROLE_EXTERNAL_ID	이 환경 변수는 수임된 역할 ARN과 함께 사용할 외부 ID를 지정합니다.	None

환경 변수	Details	기본값
ROLE_SESSION_NAME	이 환경 변수는 수임된 IAM 역할의 자격 증명과 연결할 세션 이름을 지정합니다.	None
SERVICE_REGION	이 환경 변수는 AWS AppConfig 에이전트가 AWS AppConfig 서비스를 AWS 리전 호출하는 데 사용하는 대안을 지정합니다. 정의되지 않은 상태로 두면 에이전트는 현재 리전을 확인하려고 시도합니다. 그렇게 할 수 없는 경우 에이전트가 시작되지 않습니다.	None
WAIT_ON_MANIFEST	이 환경 변수는 AWS AppConfig 에이전트가 시작을 완료하기 전에 매니페스트가 처리될 때까지 기다리도록 구성합니다.	true

5단계: (필수) 구성 데이터 검색

HTTP localhost 호출을 사용하여 AWS AppConfig 에이전트에서 구성 데이터를 검색할 수 있습니다. 다음 예제는 HTTP 클라이언트와 `curl`을 함께 사용합니다. 애플리케이션 언어에서 지원하는 사용 가능한 모든 HTTP 클라이언트나 AWS SDK를 비롯한 사용 가능한 라이브러리를 사용하여 에이전트를 호출할 수 있습니다.

배포된 구성의 전체 내용을 검색하려면

```
$ curl "http://localhost:2772/applications/application_name/
environments/environment_name/configurations/configuration_name"
```

`Feature Flag` 유형의 AWS AppConfig 구성에서 단일 플래그와 해당 속성을 검색하려면

```
$ curl "http://localhost:2772/applications/application_name/
environments/environment_name/configurations/configuration_name?flag=flag_name"
```

`Feature Flag` 유형의 AWS AppConfig 구성에서 여러 플래그와 해당 속성에 액세스하려면

```
$ curl "http://localhost:2772/applications/application_name/
environments/environment_name/configurations/configuration_name?
flag=flag_name_one&flag=flag_name_two"
```

6단계 (선택 사항이지만 권장): 에이전트 업데이트 자동화 AWS AppConfig

AWS AppConfig 에이전트는 정기적으로 업데이트됩니다. 인스턴스에서 최신 버전의 AWS AppConfig 에이전트를 실행하려면 Amazon EC2 사용자 데이터에 다음 명령을 추가하는 것이 좋습니다. 인스턴스 또는 EC2 Auto Scaling 그룹의 사용자 데이터에 명령을 추가할 수 있습니다. 스크립트는 인스턴스가 시작되거나 재부팅될 때마다 에이전트의 최신 버전을 설치하고 시작합니다.

```
#!/bin/bash
# install the latest version of the agent
yum install -y https://s3.amazonaws.com/aws-appconfig-downloads/aws-appconfig-agent/
linux/x86_64/latest/aws-appconfig-agent.rpm
# optional: configure the agent
mkdir /etc/systemd/system/aws-appconfig-agent.service.d
echo "${MY_AGENT_CONFIG}" > /etc/systemd/system/aws-appconfig-agent.service.d/
overrides.conf
systemctl daemon-reload
# start the agent
systemctl start aws-appconfig-agent
```

Amazon ECS 및 Amazon EKS 에서 구성 데이터 검색

에이전트를 AWS AppConfig 사용하여 아마존 Elastic Container Service (Amazon ECS) 및 아마존 Elastic Kubernetes 서비스 (Amazon EKS) 와 통합할 수 있습니다. AWS AppConfig 에이전트는 Amazon ECS 및 Amazon EKS 컨테이너 애플리케이션과 함께 실행되는 사이드카 컨테이너 역할을 합니다. 에이전트는 다음과 같은 방식으로 컨테이너식 애플리케이션 처리 및 관리를 개선합니다.

- 에이전트는 AWS Identity and Access Management (IAM) 역할을 사용하고 구성 데이터의 로컬 캐시를 관리하여 사용자를 대신하여 전화를 겁니다 AWS AppConfig . 로컬 캐시에서 구성 데이터를 가져오면 애플리케이션에서 구성 데이터를 관리하는 데 필요한 코드 업데이트 횟수가 줄어들고, 구성 데이터를 밀리초 단위로 검색할 수 있으며, 이러한 데이터에 대한 호출을 방해할 수 있는 네트워크 문제의 영향을 받지 않습니다.*
- 에이전트는 기능 플래그를 검색하고 해결하기 AWS AppConfig 위한 기본 환경을 제공합니다.

- 에이전트는 기본적으로 캐싱 전략, 폴링 간격, 로컬 구성 데이터 가용성에 대한 모범 사례를 제공하는 동시에 후속 서비스 호출에 필요한 구성 토큰을 추적합니다.
- 에이전트는 백그라운드에서 실행되는 동안 정기적으로 AWS AppConfig 데이터 플레인을 폴링하여 구성 데이터 업데이트를 확인합니다. 컨테이너화된 애플리케이션은 포트 2772(사용자 지정 가능한 기본 포트 값)에서 localhost에 연결하고 HTTP GET을 호출하여 데이터를 검색함으로써 데이터를 검색할 수 있습니다.
- AWS AppConfig 에이전트는 컨테이너를 다시 시작하거나 재활용할 필요 없이 컨테이너의 구성 데이터를 업데이트합니다.

*AWS AppConfig 에이전트는 서비스가 구성 데이터를 처음 검색할 때 데이터를 캐시합니다. 이러한 이유로 데이터를 검색하기 위한 첫 번째 호출은 후속 호출보다 느립니다.

주제

- 시작하기 전 준비 사항
- Amazon ECS 통합을 위한 AWS AppConfig 에이전트 시작
- Amazon EKS 통합을 위한 AWS AppConfig 에이전트 시작
- 환경 변수를 사용하여 Amazon ECS 및 Amazon EKS용 AWS AppConfig 에이전트를 구성합니다.
- 구성 데이터 검색

시작하기 전 준비 사항

컨테이너 AWS AppConfig 애플리케이션과 통합하려면 기능 플래그 또는 자유 형식 구성 데이터를 비롯한 AWS AppConfig 아티팩트와 구성 데이터를 생성해야 합니다. 자세한 정보는 기능 플래그 및 자유 형식 구성 데이터 생성 AWS AppConfig을 참조하세요.

에서 AWS AppConfig호스팅되는 구성 데이터를 검색하려면 데이터 플레인에 액세스할 수 있도록 컨테이너 애플리케이션을 구성해야 합니다. AWS AppConfig 애플리케이션에 액세스 권한을 부여하려면 컨테이너 서비스 IAM 역할이 사용하는 IAM 권한 정책을 업데이트하십시오. 특히, 정책에 `appconfig:StartConfigurationSession` 및 `appconfig:GetLatestConfiguration` 액션을 추가해야 합니다. 컨테이너 서비스 IAM 역할에는 다음이 포함됩니다.

- Amazon ECS 태스크 역할
- Amazon EKS 노드 역할
- AWS Fargate (Fargate) 포드 실행 역할 (Amazon EKS 컨테이너가 컴퓨팅 프로세싱에 Fargate를 사용하는 경우)

정책에 권한을 추가하는 것에 대한 자세한 내용은 IAM 사용 설명서의 IAM 자격 증명 권한 추가 및 제거를 참조하십시오.

Amazon ECS 통합을 위한 AWS AppConfig 에이전트 시작

AWS AppConfig 에이전트 사이드카 컨테이너는 Amazon ECS 환경에서 자동으로 사용할 수 있습니다. AWS AppConfig 에이전트 사이드카 컨테이너를 사용하려면 해당 컨테이너를 시작해야 합니다.

Amazon ECS(콘솔)를 시작하려면

1. https://console.aws.amazon.com/ecs/v2에서 콘솔을 엽니다.
2. 탐색 창에서 작업 정의를 선택합니다.
3. 애플리케이션의 작업 정의를 선택한 다음 최신 수정 버전을 선택합니다.
4. 새 개정 생성, 새 개정 생성을 선택합니다.
5. 컨테이너 추가를 선택합니다.
6. 이름에는 AWS AppConfig 에이전트 컨테이너의 고유한 이름을 입력합니다.
7. 이미지 URI에 다음을 입력합니다. **`public.ecr.aws/aws-appconfig/aws-appconfig-agent:2.x`**
8. 필수 컨테이너의 경우 예를 선택합니다.
9. 포트 매핑 섹션에서, 포트 매핑 추가를 선택합니다.
10. 컨테이너 포트에 **2772**를 입력합니다.

> **Note**
>
> AWS AppConfig 에이전트는 기본적으로 포트 2772에서 실행됩니다. 다른 포트를 지정할 수 있습니다.

11. 생성을 선택하세요. Amazon ECS는 새 컨테이너 개정을 생성하고 세부 정보를 표시합니다.
12. 탐색 창에서 클러스터를 선택한 다음 목록에서 애플리케이션 클러스터를 선택합니다.
13. 서비스 탭에서 애플리케이션에 맞는 서비스를 선택합니다.
14. 업데이트를 선택합니다.
15. 배포 구성에서 개정에 대해 최신 버전을 선택합니다.
16. 업데이트를 선택합니다. Amazon ECS는 최신 작업 정의를 배포합니다.
17. 배포가 완료되면 구성 및 작업 탭에서 AWS AppConfig 에이전트가 실행 중인지 확인할 수 있습니다. 작업 탭에서 실행 중인 작업을 선택합니다.

18. 컨테이너 섹션에서 AWS AppConfig 에이전트 컨테이너가 나열되어 있는지 확인합니다.
19. AWS AppConfig 에이전트가 시작되었는지 확인하려면 로그 탭을 선택합니다. AWS AppConfig 에이전트 컨테이너에 대해 다음과 같은 명령문을 찾으십시오. `[appconfig agent] 1970/01/01 00:00:00 INFO serving on localhost:2772`

> **Note**
>
> 환경 변수를 입력하거나 변경하여 AWS AppConfig 에이전트의 기본 동작을 조정할 수 있습니다. 사용할 수 있는 환경 변수에 대한 자세한 내용은 환경 변수를 사용하여 Amazon ECS 및 Amazon EKS용 AWS AppConfig 에이전트를 구성합니다. 섹션을 참조하십시오. Amazon ECS에서 환경 변수를 변경하는 방법에 대한 자세한 내용은 Amazon Elastic Container Service 개발자 안내서의 컨테이너에 환경 변수 전달을 참조하십시오.

Amazon EKS 통합을 위한 AWS AppConfig 에이전트 시작

AWS AppConfig 에이전트 사이드카 컨테이너는 Amazon EKS 환경에서 자동으로 사용할 수 있습니다. AWS AppConfig 에이전트 사이드카 컨테이너를 사용하려면 해당 컨테이너를 시작해야 합니다. 다음 절차에서는 Amazon EKS `kubectl` 명령줄 도구를 사용하여 컨테이너 애플리케이션에서 `kubeconfig` 파일을 변경하는 방법을 설명합니다. `kubeconfig` 파일 생성 또는 편집에 대한 자세한 내용은 Amazon EKS 사용 설명서의 Amazon EKS 클러스터용 kubeconfig 파일 생성 또는 업데이트를 참조하십시오.

AWS AppConfig 에이전트를 시작하려면 (kubectl 명령줄 도구)

1. `kubeconfig` 파일을 열고 Amazon EKS 애플리케이션이 단일 컨테이너 배포로 실행되고 있는지 확인합니다. 파일 콘텐츠는 다음과 비슷해야 합니다.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
  namespace: my-namespace
  labels:
    app: my-application-label
spec:
  replicas: 1
  selector:
    matchLabels:
```

```
      app: my-application-label
  template:
    metadata:
      labels:
        app: my-application-label
    spec:
      containers:
      - name: my-app
        image: my-repo/my-image
        imagePullPolicy: IfNotPresent
```

2. YAML AWS AppConfig 배포 파일에 에이전트 컨테이너 정의 세부 정보를 추가합니다.

```
- name: appconfig-agent
        image: public.ecr.aws/aws-appconfig/aws-appconfig-agent:2.x
        ports:
        - name: http
          containerPort: 2772
          protocol: TCP
        env:
        - name: SERVICE_REGION
          value: region
        imagePullPolicy: IfNotPresent
```

> **Note**
>
> 다음 정보를 참고하세요.
>
> - AWS AppConfig 에이전트는 기본적으로 포트 2772에서 실행됩니다. 다른 포트를 지정할 수 있습니다.
> - 환경 변수를 입력하여 AWS AppConfig 에이전트의 기본 동작을 조정할 수 있습니다. 자세한 정보는 환경 변수를 사용하여 Amazon ECS 및 Amazon EKS용 AWS AppConfig 에이전트를 구성합니다.을 참조하세요.
> - *SERVICE_REGION#* 경우 AWS AppConfig 에이전트가 구성 데이터를 검색하는 AWS 리전 코드 (예:us-west-1) 를 지정합니다.

3. kubectl 도구에서 다음 명령을 실행하여 클러스터에 변경 내용을 적용합니다.

```
kubectl apply -f my-deployment.yml
```

4. 배포가 완료된 후 에이전트가 실행 중인지 확인합니다 AWS AppConfig . 다음 명령을 사용하여 Application Pod 로그 파일을 확인합니다.

```
kubectl logs -n my-namespace -c appconfig-agent my-pod
```

AWS AppConfig 에이전트 컨테이너에 대한 다음과 같은 명령문을 찾으십시오. `[appconfig agent] 1970/01/01 00:00:00 INFO serving on localhost:2772`

> **Note**
>
> 환경 변수를 입력하거나 변경하여 AWS AppConfig 에이전트의 기본 동작을 조정할 수 있습니다. 사용할 수 있는 환경 변수에 대한 자세한 내용은 환경 변수를 사용하여 Amazon ECS 및 Amazon EKS용 AWS AppConfig 에이전트를 구성합니다. 섹션을 참조하십시오.

환경 변수를 사용하여 Amazon ECS 및 Amazon EKS용 AWS AppConfig 에이전트를 구성합니다.

AWS AppConfig 에이전트 컨테이너의 다음 환경 변수를 변경하여 에이전트를 구성할 수 있습니다.

환경 변수	Details	기본값
ACCESS_TOKEN	이 환경 변수는 에이전트 HTTP 서버에 구성 데이터를 요청할 때 제공해야 하는 토큰을 정의합니다. 토큰 값은 권한 부여 유형이 Bearer인 HTTP 요청 승인 헤더에서 설정해야 합니다. 의 예는 다음과 같습니다. `GET /applications/my_app/... Host: localhost:2772`	None

환경 변수	Details	기본값
	`Authorization: Bearer <token value>`	
`BACKUP_DIRECTORY`	이 환경 변수를 사용하면 AWS AppConfig 에이전트가 검색한 각 구성의 백업을 지정된 디렉터리에 저장할 수 있습니다. **Important** 디스크에 백업된 구성은 암호화되지 않습니다. 구성에 민감한 데이터가 포함된 경우 파일 시스템 권한에 대한 최소 권한 원칙을 적용하는 것이 좋습니다. AWS AppConfig 자세한 정보는 AWS AppConfig의 보안을 참조하세요.	None
`HTTP_PORT`	이 환경 변수는 에이전트의 HTTP 서버가 실행되는 포트를 지정합니다.	2772

환경 변수	Details	기본값
LOG_LEVEL	이 환경 변수는 에이전트가 기록하는 세부 정보 수준을 지정합니다. 각 레벨에는 현재 레벨과 모든 상위 레벨이 포함됩니다. 변수는 대소문자를 구분합니다. 로그 수준은 가장 세부적인 것부터 세부적이지 않은 것까지 debug, info, warn, error, 및 none 입니다. Debug는 타이밍 정보를 비롯한 에이전트에 대한 자세한 정보가 포함됩니다.	info
MANIFEST	이 환경 변수는 AWS AppConfig 에이전트가 다중 계정 검색과 같은 추가 구성별 기능을 활용하고 구성을 디스크에 저장하도록 구성합니다. 다음 값 중 하나를 입력할 수 있습니다. • "app:env:manifest-config" • "file:/fully/qualified/path/to/manifest.json" 이러한 기능에 대한 자세한 내용은 추가 검색 기능 섹션을 참조하세요.	true

환경 변수	Details	기본값
MAX_CONNECTIONS	이 환경 변수는 에이전트가 AWS AppConfig에서 구성을 검색하는 데 사용하는 최대 연결 수를 구성합니다.	3
POLL_INTERVAL	이 환경 변수는 에이전트가 업데이트된 구성 데이터를 AWS AppConfig 폴링하는 빈도를 제어합니다. 간격을 초 단위로 지정할 수 있습니다. 시간 단위를 사용하여 숫자를 지정할 수도 있습니다. 초는 s, 분은 m, 시간은 h입니다. 단위를 지정하지 않으면 에이전트의 기본값은 초로 설정됩니다. 예를 들어 60, 60초, 1분은 폴링 간격이 동일합니다.	45초
PREFETCH_LIST	이 환경 변수는 에이전트가 AWS AppConfig 시작하자마자 요청하는 구성 데이터를 지정합니다.	None
PRELOAD_BACKUPS	로 `true` 설정하면 AWS AppConfig 에이전트는 에서 찾은 구성 백업을 BACKUP_DIRECTORY 메모리로 로드하고 서비스에 새 버전이 있는지 즉시 확인합니다. 로 `false` 설정된 경우 AWS AppConfig 에이전트는 서비스에서 구성 데이터를 검색할 수 없는 경우 (예: 네트워크에 문제가 있는 경우) 에만 구성 백업의 내용을 로드합니다.	true

환경 변수	Details	기본값
PROXY_HEADERS	이 환경 변수는 PROXY_URL 환경 변수에서 참조되는 프록시에 필요한 헤더를 지정합니다. 값은 쉼표로 구분된 헤더 목록입니다. 각 헤더는 다음 형식을 사용합니다. `"header: value"`	None
PROXY_URL	이 환경 변수는 다음을 포함하여 에이전트와의 연결에 사용할 프록시 URL을 지정합니다 AWS AppConfig. AWS 서비스HTTPS및 HTTP URL이 지원됩니다.	None

환경 변수	Details	기본값
REQUEST_TIMEOUT	이 환경 변수는 에이전트가 응답을 기다리는 시간을 제어합니다. AWS AppConfig서비스가 응답하지 않으면 요청이 실패합니다. 초기 데이터 검색을 위한 요청인 경우 에이전트는 애플리케이션에 오류를 반환합니다. 업데이트된 데이터에 대한 백그라운드 확인 중에 제한 시간이 초과되면 에이전트는 오류를 기록하고 잠시 후 다시 시도합니다. 제한 시간을 밀리초로 지정할 수 있습니다. 시간 단위로 숫자를 지정할 수도 있습니다. 밀리초는 ms이고 초는 s입니다. 단위를 지정하지 않으면 에이전트의 기본값은 밀리초로 설정됩니다. 예를 들어 5000, 5000ms 및 5s의 경우 요청 제한 시간 값이 동일합니다.	3000 밀리초
ROLE_ARN	이 환경 변수는 IAM 역할의 Amazon 리소스 이름 (ARN) 을 지정합니다. AWS AppConfig 에이전트는 이 역할을 맡아 구성 데이터를 검색합니다.	None
ROLE_EXTERNAL_ID	이 환경 변수는 수임된 역할 ARN과 함께 사용할 외부 ID를 지정합니다.	None

환경 변수	Details	기본값
ROLE_SESSION_NAME	이 환경 변수는 수임된 IAM 역할의 자격 증명과 연결할 세션 이름을 지정합니다.	None
SERVICE_REGION	이 환경 변수는 AWS AppConfig 에이전트가 AWS AppConfig 서비스를 AWS 리전 호출하는 데 사용하는 대안을 지정합니다. 정의되지 않은 상태로 두면 에이전트는 현재 리전을 확인하려고 시도합니다. 그렇게 할 수 없는 경우 에이전트가 시작되지 않습니다.	None
WAIT_ON_MANIFEST	이 환경 변수는 AWS AppConfig 에이전트가 시작을 완료하기 전에 매니페스트가 처리될 때까지 기다리도록 구성합니다.	true

구성 데이터 검색

HTTP localhost 호출을 사용하여 AWS AppConfig 에이전트에서 구성 데이터를 검색할 수 있습니다. 다음 예제는 HTTP 클라이언트와 `curl`을 함께 사용합니다. 애플리케이션 언어 또는 사용 가능한 라이브러리에서 지원하는 사용 가능한 모든 HTTP 클라이언트를 사용하여 에이전트를 호출할 수 있습니다.

> **Note**
>
> 애플리케이션에서 순방향 슬래시를 사용하는 경우 (예: “test-backend/test-service”) 구성 데이터를 검색하려면 URL 인코딩을 사용해야 합니다.

배포된 구성의 전체 내용을 검색하려면

```
$ curl "http://localhost:2772/applications/application_name/
environments/environment_name/configurations/configuration_name"
```

Feature Flag 유형의 AWS AppConfig 구성에서 단일 플래그와 해당 속성을 검색하려면

```
$ curl "http://localhost:2772/applications/application_name/
environments/environment_name/configurations/configuration_name?flag=flag_name"
```

Feature Flag 유형의 AWS AppConfig 구성에서 여러 플래그와 해당 속성에 액세스하려면

```
$ curl "http://localhost:2772/applications/application_name/
environments/environment_name/configurations/configuration_name?
flag=flag_name_one&flag=flag_name_two"
```

추가 검색 기능

AWS AppConfig 에이전트는 애플리케이션의 구성을 검색하는 데 도움이 되는 다음과 같은 추가 기능을 제공합니다.

- 다중 계정 검색: 기본 또는 AWS 계정 검색에서 AWS AppConfig 에이전트를 사용하여 여러 공급업체 계정에서 구성 데이터를 검색하십시오.
- 구성 사본을 디스크에 쓰기: AWS AppConfig 에이전트를 사용하여 구성 데이터를 디스크에 기록합니다. 이 기능을 사용하면 디스크에서 구성 데이터를 읽는 애플리케이션을 보유한 고객이 통합할 수 AWS AppConfig있습니다.

에이전트 매니페스트에 대한 정보

이러한 AWS AppConfig 에이전트 기능을 활성화하려면 매니페스트를 생성해야 합니다. 매니페스트는 에이전트가 수행할 수 있는 작업을 제어하기 위해 제공하는 구성 데이터 세트입니다. 매니페스트는 JSON으로 작성됩니다. 여기에는 사용자가 사용하여 배포한 다양한 구성에 해당하는 최상위 키 세트가 포함되어 있습니다. AWS AppConfig

매니페스트에는 여러 구성이 포함될 수 있습니다. 또한 매니페스트의 각 구성은 지정된 구성에 사용할 하나 이상의 에이전트 기능을 식별할 수 있습니다. 매니페스트의 내용은 다음 형식을 사용합니다.

```
{
    "application_name:environment_name:configuration_name": {
        "agent_feature_to_enable_1": {
```

```
            "feature-setting-key": "feature-setting-value"
        },
        "agent_feature_to_enable_2": {
            "feature-setting-key": "feature-setting-value"
        }
    }
}
```

다음은 두 가지 구성이 있는 매니페스트의 JSON 예시입니다. 첫 번째 구성 (*MyApp*) 은 AWS AppConfig 에이전트 기능을 사용하지 않습니다. 두 번째 구성 (*My2ndApp*) 은 디스크에 구성 쓰기 복사와 다중 계정 검색 기능을 사용합니다.

```
{
        "MyApp:Test:MyAllowListConfiguration": {},

        "My2ndApp:Beta:MyEnableMobilePaymentsFeatureFlagConfiguration": {
            "credentials": {
                "roleArn": "arn:us-west-1:iam::123456789012:role/MyTestRole",
                "roleExternalId": "00b148e2-4ea4-46a1-ab0f-c422b54d0aac",
                "roleSessionName": "AwsAppConfigAgent",
                "credentialsDuration": "2h"
            },
            "writeTo": {
                "path": "/tmp/aws-appconfig/my-2nd-app/beta/my-enable-payments-feature-
flag-configuration.json"
            }
        }
    }
```

에이전트 매니페스트를 제공하는 방법

매니페스트를 AWS AppConfig 에이전트가 읽을 수 있는 위치에 파일로 저장할 수 있습니다. 또는 매니페스트를 구성으로 저장한 다음 에이전트가 해당 AWS AppConfig 구성을 가리키도록 할 수 있습니다. 에이전트 매니페스트를 제공하려면 다음 값 중 하나를 사용하여 `MANIFEST` 환경 변수를 설정해야 합니다.

매니페스트 위치	환경 변수 값	사용 사례
파일	파일: /경로/to/agent-manifest.json	매니페스트가 자주 변경되지 않을 경우 이 방법을 사용하세요.

매니페스트 위치	환경 변수 값	사용 사례
AWS AppConfig 구성	*###### ##: ## ##: ## # #*	동적 업데이트에는 이 방법을 사용합니다. 다른 AWS AppConfig 구성을 저장하는 것과 동일한 방식으로 AWS AppConfig 구성으로 저장된 매니페스트를 업데이트하고 배포할 수 있습니다.
환경 변수	매니페스트 콘텐츠 (JSON)	매니페스트가 자주 변경되지 않을 경우 이 방법을 사용하세요. 이 방법은 파일을 노출하는 것보다 환경 변수를 설정하는 것이 더 쉬운 컨테이너 환경에서 유용합니다.

AWS AppConfig 에이전트의 변수 설정에 대한 자세한 내용은 사용 사례의 관련 주제를 참조하십시오.

- AWS AppConfig 에이전트 Lambda 확장 구성
- Amazon EC2에서 AWS AppConfig 에이전트 사용
- Amazon ECS 및 Amazon EKS와 함께 AWS AppConfig 에이전트 사용

다중 계정 검색

AWS AppConfig 에이전트 매니페스트에 자격 증명 재정의를 AWS 계정 입력하여 여러 구성에서 구성을 검색하도록 에이전트를 구성할 수 있습니다. AWS AppConfig 자격 증명 재정의에는 (IAM) 역할의 Amazon 리소스 이름 AWS Identity and Access Management (ARN), 역할 ID, 세션 이름, 에이전트가 역할을 맡을 수 있는 기간이 포함됩니다.

매니페스트의 “자격 증명” 섹션에 이러한 세부 정보를 입력합니다. “자격 증명” 섹션은 다음 형식을 사용합니다.

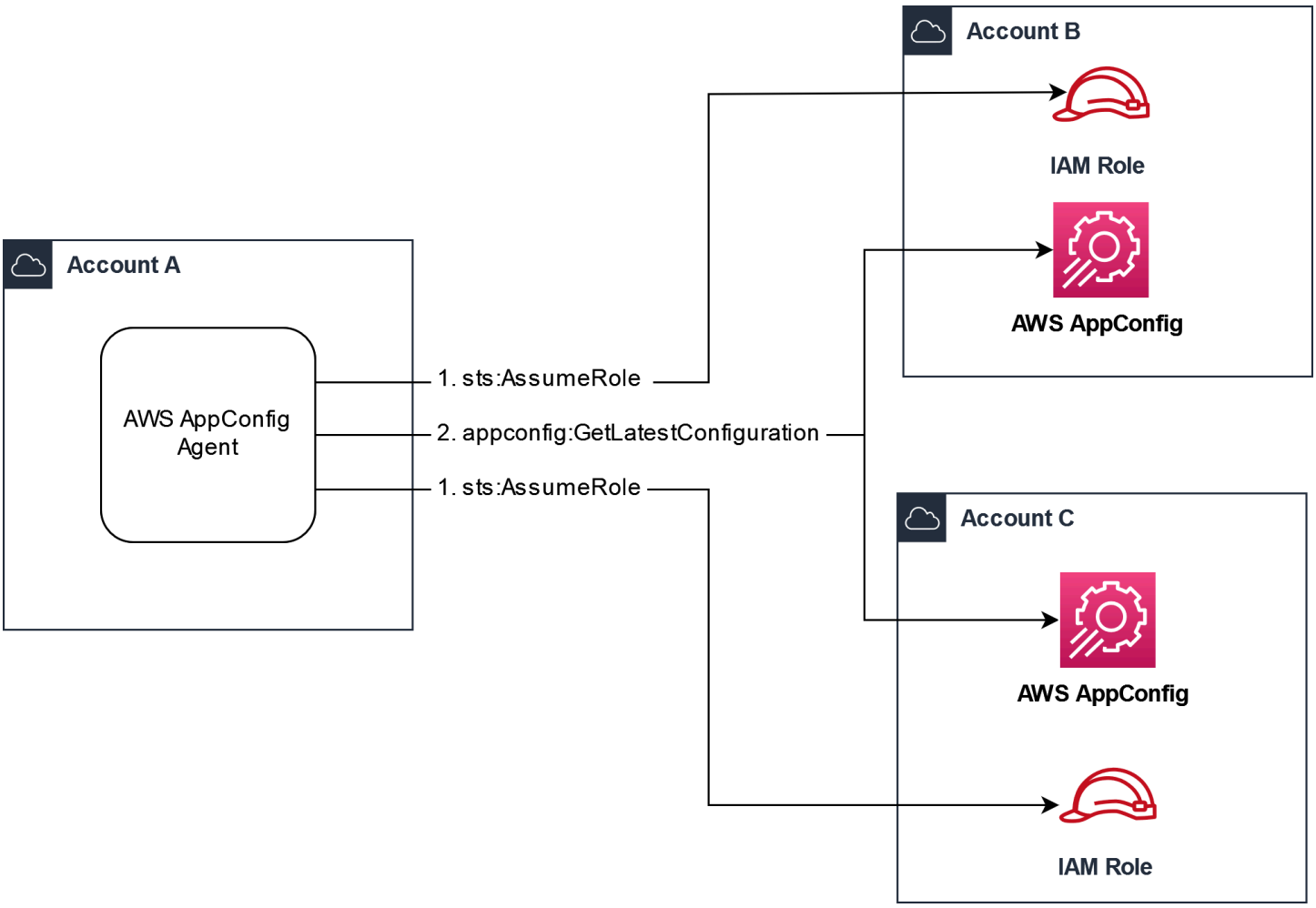
```
{
    "application_name:environment_name:configuration_name": {
        "credentials": {
            "roleArn": "arn:partition:iam::account_ID:role/roleName",
```

```
            "roleExternalId": "string",
            "roleSessionName": "string",
            "credentialsDuration": "time_in_hours"
        }
    }
}
```

예:

```
{
    "My2ndApp:Beta:MyEnableMobilePaymentsFeatureFlagConfiguration": {
        "credentials": {
            "roleArn": "arn:us-west-1:iam::123456789012:role/MyTestRole",
            "roleExternalId": "00b148e2-4ea4-46a1-ab0f-c422b54d0aac",
            "roleSessionName": "AWSAppConfigAgent",
            "credentialsDuration": "2h"
        }
    }
}
```

구성을 검색하기 전에 에이전트는 매니페스트에서 구성에 대한 자격 증명 세부 정보를 읽은 다음 해당 구성에 지정된 IAM 역할을 맡습니다. 단일 매니페스트에서 여러 구성에 대해 서로 다른 자격 증명 재정의 세트를 지정할 수 있습니다. 다음 다이어그램은 AWS AppConfig 에이전트가 계정 A (검색 계정) 에서 실행되는 동안 계정 B와 C (공급업체 계정) 에 지정된 별도의 역할을 맡은 다음 GetLatestConfigurationAPI 작업을 호출하여 해당 계정에서 실행되는 구성 데이터를 검색하는 방법을 보여줍니다. AWS AppConfig



공급업체 계정에서 구성 데이터를 검색할 수 있는 권한을 구성합니다.

AWS AppConfig 검색 계정에서 실행되는 에이전트는 공급업체 계정에서 구성 데이터를 검색할 수 있는 권한이 필요합니다. 각 공급업체 계정에서 AWS Identity and Access Management (IAM) 역할을 생성하여 에이전트에게 권한을 부여합니다. AWS AppConfig 검색 계정의 에이전트는 이 역할을 맡아 공급업체 계정으로부터 데이터를 가져옵니다. 이 섹션의 절차를 완료하여 IAM 권한 정책, IAM 역할을 생성하고 매니페스트에 에이전트 재정의를 추가하세요.

시작하기 전 준비 사항

IAM에서 권한 정책과 역할을 생성하기 전에 다음 정보를 수집하십시오.

- 각 AWS 계정 ID의 ID. 검색 계정은 구성 데이터를 받기 위해 다른 계정을 호출하는 계정입니다. 공급업체 계정은 검색 계정에 구성 데이터를 제공하는 계정입니다.
- 검색 AWS AppConfig 계정에서 사용하는 IAM 역할의 이름. 에서 기본적으로 AWS AppConfig사용하는 역할 목록은 다음과 같습니다.

- Amazon Elastic Compute Cloud (Amazon EC2) 의 AWS AppConfig 경우 인스턴스 역할을 사용합니다.
- 의 경우 AWS Lambda, Lambda 실행 역할을 AWS AppConfig 사용합니다.
- 아마존 Elastic Container Service (Amazon ECS) 및 아마존 Elastic Kubernetes 서비스 (Amazon EKS) 의 경우 컨테이너 역할을 사용합니다. AWS AppConfig

`ROLE_ARN`환경 변수를 지정하여 AWS AppConfig 에이전트가 다른 IAM 역할을 사용하도록 구성한 경우 해당 이름을 기록해 두십시오.

권한 정책 생성

다음 절차를 사용하여 IAM 콘솔을 사용하여 권한 정책을 생성합니다. 검색 계정의 구성 데이터를 AWS 계정 제공하는 각 절차를 완료하십시오.

IAM 정책을 만들려면

1. 공급업체 계정으로 AWS Management Console 로그인합니다.
2. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
3. 탐색 창에서 정책을 선택한 후 정책 생성을 선택합니다.
4. JSON 옵션을 선택합니다.
5. 정책 편집기에서 기본 JSON을 다음 정책 설명으로 대체합니다. 각 *## ### ####### #### ## ##* 정보로 업데이트하세요.

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "appconfig:StartConfigurationSession",
            "appconfig:GetLatestConfiguration"
        ],
        "Resource":
 "arn:partition:appconfig:region:vendor_account_ID:application/
vendor_application_ID/environment/vendor_environment_ID/
configuration/vendor_configuration_ID"
    }
   ]
}
```

다음은 그 예입니다.

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "appconfig:StartConfigurationSession",
            "appconfig:GetLatestConfiguration"
        ],
        "Resource": "arn:aws:appconfig:us-east-2:111122223333:application/abc123/
environment/def456/configuration/hij789"
    }
   ]
}
```

6. 다음을 선택합니다.
7. 정책 이름 필드에 이름을 입력합니다.
8. (선택 사항) [Add tags] 에서 하나 이상의 태그-키 값 쌍을 추가하여 이 정책에 대한 액세스를 구성, 추적 또는 제어합니다.
9. 정책 생성(Create policy)을 선택합니다. 시스템에서 정책 페이지로 돌아갑니다.
10. 검색 계정의 구성 데이터를 제공할 각 AWS 계정 계정에서 이 절차를 반복합니다.

IAM 역할 생성

다음 절차를 사용하여 IAM 콘솔을 사용하여 IAM 역할을 생성합니다. 검색 계정의 구성 데이터를 AWS 계정 제공하는 각 절차를 완료하십시오.

IAM 역할을 생성하려면

1. 공급업체 계정으로 AWS Management Console 로그인합니다.
2. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
3. 탐색 창에서 역할을 선택한 다음 정책 생성을 선택합니다.
4. 신뢰할 수 있는 엔터티 유형에 AWS 계정를 선택합니다.
5. AWS 계정섹션에서 기타를 선택합니다 AWS 계정.
6. 계정 ID 필드에 검색 계정 ID를 입력합니다.
7. (선택 사항) 이 역할 수임에 대한 보안 모범 사례로 외부 ID 필요를 선택하고 문자열을 입력합니다.

8. 다음을 선택합니다.
9. 권한 추가 페이지에서 검색 필드를 사용하여 이전 절차에서 만든 정책을 찾을 수 있습니다. 이름 옆의 확인란을 선택합니다.
10. 다음을 선택합니다.
11. 역할 이름에 이름을 입력합니다.
12. (선택 사항) 설명에 설명을 입력합니다.
13. 1단계: 신뢰할 수 있는 엔티티 선택에서 편집을 선택합니다. 기본 JSON 신뢰 정책을 다음 정책으로 대체합니다. 각 *## ### #######* 검색 계정의 정보로 업데이트하세요.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS":
 "arn:aws:iam::retrieval_account_ID:role/appconfig_role_in_retrieval_account"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

14. (선택 사항) Tags(태그)에서 이 역할에 대한 액세스를 구성, 추적 또는 제어할 태그-키 값 페어를 하나 이상 추가합니다.
15. 역할 생성을 선택합니다. 그러면 역할 페이지로 돌아갑니다.
16. 방금 생성한 역할을 검색하세요. 이를 선택합니다. ARN 섹션에서 ARN을 복사합니다. 다음 절차에서 이 정보를 지정하겠습니다.

매니페스트에 자격 증명 재정의를 추가합니다.

공급업체 계정에서 IAM 역할을 생성한 후 검색 계정에서 매니페스트를 업데이트하십시오. 특히 공급업체 계정에서 구성 데이터를 검색하기 위한 자격 증명 블록과 IAM 역할 ARN을 추가합니다. JSON 형식은 다음과 같습니다.

```
{
    "vendor_application_name:vendor_environment_name:vendor_configuration_name": {
        "credentials": {
```

```
            "roleArn":
 "arn:partition:iam::vendor_account_ID:role/name_of_role_created_in_vendor_account",
            "roleExternalId": "string",
            "roleSessionName": "string",
            "credentialsDuration": "time_in_hours"
        }
    }
}
```

예:

```
{
    "My2ndApp:Beta:MyEnableMobilePaymentsFeatureFlagConfiguration": {
        "credentials": {
            "roleArn": "arn:us-west-1:iam::123456789012:role/MyTestRole",
            "roleExternalId": "00b148e2-4ea4-46a1-ab0f-c422b54d0aac",
            "roleSessionName": "AwsAppConfigAgent",
            "credentialsDuration": "2h"
        }
    }
}
```

다중 계정 검색이 제대로 작동하는지 확인

에이전트 로그를 검토하여 해당 에이전트가 여러 계정에서 구성 데이터를 검색할 수 있는지 확인할 수 있습니다. AWS AppConfig 'YourApplicationName:YourEnvironmentName:YourConfigurationName '에 대해 검색된 초기 데이터의 INFO 레벨 로그는 성공적인 검색을 위한 최상의 지표입니다. 검색이 실패하는 경우 실패 이유를 나타내는 ERROR 레벨 로그가 표시됩니다. 공급업체 계정에서 성공적으로 검색한 예는 다음과 같습니다.

```
[appconfig agent] 2023/11/13 11:33:27 INFO AppConfig Agent 2.0.x
[appconfig agent] 2023/11/13 11:33:28 INFO serving on localhost:2772
[appconfig agent] 2023/11/13 11:33:28 INFO retrieved initial data for
 'MyTestApplication:MyTestEnvironment:MyDenyListConfiguration' in XX.Xms
```

구성 사본을 디스크에 쓰기

구성 사본을 디스크에 일반 텍스트로 자동 저장하도록 AWS AppConfig 에이전트를 구성할 수 있습니다. 이 기능을 사용하면 디스크에서 구성 데이터를 읽는 애플리케이션을 사용하는 고객이 통합할 수 AWS AppConfig있습니다.

이 기능은 구성 백업 기능으로 사용하도록 설계되지 않았습니다. AWS AppConfig 에이전트는 디스크에 복사된 구성 파일을 읽지 않습니다. 구성을 디스크에 백업하려면 Amazon EC2에서 에이전트 사용 또는 Amazon ECS BACKUP_DIRECTORY 및 Amazon EKS에서 AWS AppConfigAWS AppConfig 에이전트 사용에 대한 및 PRELOAD_BACKUP 환경 변수를 참조하십시오.

> **Warning**
>
> 이 기능에 대한 다음과 같은 중요 정보를 참고하십시오.
>
> - 디스크에 저장된 구성은 일반 텍스트로 저장되며 사람이 읽을 수 있습니다. 민감한 데이터가 포함된 구성에는 이 기능을 활성화하지 마십시오.
> - 이 기능은 로컬 디스크에 기록합니다. 파일 시스템 권한에는 최소 권한 원칙을 사용하십시오. 자세한 정보는 최소 권한 액세스 구현을 참조하세요.

쓰기 구성을 활성화하려면 디스크에 구성을 복사하십시오.

1. 매니페스트를 편집합니다.
2. 디스크에 AWS AppConfig 기록하려는 구성을 선택하고 `writeTo` 요소를 추가합니다. 예:

```
{
    "application_name:environment_name:configuration_name": {
        "writeTo": {
            "path": "path_to_configuration_file"
        }
    }
}
```

예:

```
{
    "MyTestApp:MyTestEnvironment:MyNewConfiguration": {
        "writeTo": {
            "path": "/tmp/aws-appconfig/mobile-app/beta/enable-mobile-payments"
        }
    }
}
```

3. 변경 내용을 저장합니다. configuation.json 파일은 새 구성 데이터가 배포될 때마다 업데이트됩니다.

디스크에 구성 복사본 쓰기가 제대로 작동하는지 확인합니다.

AWS AppConfig 에이전트 로그를 검토하여 구성 사본이 디스크에 기록되고 있는지 확인할 수 있습니다. “정보 작성 구성 '######: ##: 구성' to *file_path*”라는 구문이 있는 INFO 로그 항목은 AWS AppConfig 에이전트가 ## 복사본을 디스크에 기록했음을 나타냅니다.

예:

```
[appconfig agent] 2023/11/13 11:33:27 INFO AppConfig Agent 2.0.x
[appconfig agent] 2023/11/13 11:33:28 INFO serving on localhost:2772
[appconfig agent] 2023/11/13 11:33:28 INFO retrieved initial data for
 'MobileApp:Beta:EnableMobilePayments' in XX.Xms
[appconfig agent] 2023/11/13 17:05:49 INFO wrote configuration
 'MobileApp:Beta:EnableMobilePayments' to /tmp/configs/your-app/your-env/your-
config.json
```

AWS AppConfig 에이전트 로컬 개발

AWS AppConfig 에이전트는 로컬 개발 모드를 지원합니다. 로컬 개발 모드를 활성화하면 에이전트는 디스크의 지정된 디렉터리에서 구성 데이터를 읽습니다. 에서는 구성 데이터를 검색하지 않습니다 AWS AppConfig. 지정된 디렉터리의 파일을 업데이트하여 구성 배포를 시뮬레이션할 수 있습니다. 다음 사용 사례에는 로컬 개발 모드를 사용하는 것이 좋습니다.

- 를 사용하여 AWS AppConfig배포하기 전에 다양한 구성 버전을 테스트하십시오.
- 코드 리포지토리에 변경 사항을 적용하기 전에 새 기능에 대한 다양한 구성 옵션을 테스트하세요.
- 다양한 구성 시나리오를 테스트하여 예상대로 작동하는지 확인하세요.

> **Warning**
>
> 프로덕션 환경에서는 로컬 개발 모드를 사용하지 마세요. 이 모드는 배포 검증 및 자동 롤백과 같은 중요한 AWS AppConfig 안전 기능을 지원하지 않습니다.

다음 절차를 사용하여 AWS AppConfig 에이전트를 로컬 개발 모드에 맞게 구성하십시오.

AWS AppConfig 에이전트를 로컬 개발 모드에 맞게 구성하려면

1. 컴퓨팅 환경에 설명된 방법을 사용하여 에이전트를 설치합니다. AWS AppConfig 에이전트는 AWS 서비스다음과 같은 작업을 수행합니다.

- AWS Lambda
- Amazon EC2
- 아마존 ECS 및 아마존 EKS

2. 에이전트가 실행 중이면 중지하십시오.
3. 환경 변수 목록에 LOCAL_DEVELOPMENT_DIRECTORY 추가합니다. 에이전트에게 읽기 권한을 제공하는 파일 시스템의 디렉터리를 지정합니다. 예를 들어 /tmp/local_configs입니다.
4. 디렉터리에 파일을 생성합니다. 파일 이름은 다음 형식을 사용해야 합니다.

```
application_name:environment_name:configuration_profile_name
```

예:

```
Mobile:Development:EnableMobilePaymentsFeatureFlagConfiguration
```

> **Note**
>
> (선택 사항) 파일에 제공하는 확장자를 기반으로 구성 데이터에 대해 에이전트가 반환하는 콘텐츠 유형을 제어할 수 있습니다. 예를 들어 확장명이.json인 파일 이름을 지정하면 에이전트는 애플리케이션이 요청할 application/json 때 콘텐츠 유형을 반환합니다. 확장자를 생략하면 에이전트는 해당 콘텐츠 application/octet-stream 유형에 사용합니다. 정밀한 제어가 필요한 경우 해당 형식의 .*type*%*subtype* 확장자를 제공할 수 있습니다. 에이전트는 의 콘텐츠 유형을 반환합니다.type/subtype.

5. 다음 명령을 실행하여 에이전트를 다시 시작하고 구성 데이터를 요청합니다.

```
curl http://localhost:2772/applications/application_name/
environments/environment_name/configurations/configuration_name
```

에이전트는 에이전트에 지정된 폴링 간격에 따라 로컬 파일의 변경 사항을 확인합니다. 폴링 간격을 지정하지 않은 경우 에이전트는 기본 간격인 45초를 사용합니다. 폴링 간격에 이 검사를 통해 에이전트가 AWS AppConfig 서비스와 상호 작용하도록 구성되었을 때와 마찬가지로 로컬 개발 환경에서도 동일하게 동작하는지 확인할 수 있습니다.

> **Note**
>
> 로컬 개발 구성 파일의 새 버전을 배포하려면 파일을 새 데이터로 업데이트하십시오.

API를 직접 호출하여 구성 검색

애플리케이션은 먼저 StartConfigurationSessionAPI 작업을 사용하여 구성 세션을 설정하여 구성 데이터를 검색합니다. 그러면 세션 클라이언트가 를 정기적으로 GetLatestConfiguration호출하여 사용 가능한 최신 데이터를 확인하고 검색합니다.

`StartConfigurationSession`이 호출되면 코드에서 다음 정보를 보냅니다.

- 세션이 추적하는 AWS AppConfig 애플리케이션, 환경 및 구성 프로필의 식별자 (ID 또는 이름).
- (선택 사항) 세션 클라이언트가 `GetLatestConfiguration` 호출 사이에 대기해야 하는 최소 시간입니다.

이에 대한 `InitialConfigurationToken` 응답으로 세션 클라이언트에 a를 AWS AppConfig 제공하여 해당 세션을 처음 `GetLatestConfiguration` 호출할 때 사용합니다.

> **Important**
>
> 이 토큰은 `GetLatestConfiguration`을 처음 호출할 때 한 번만 사용해야 합니다. 이 후에 `GetLatestConfiguration`을 호출할 때마다 `GetLatestConfiguration`응답 (`NextPollConfigurationToken`)의 새 토큰을 사용해야 합니다. 긴 폴링 사용 사례를 지원하기 위해 토큰은 최대 24시간 동안 유효합니다. `GetLatestConfiguration` 호출에서 만료된 토큰을 사용하는 경우 시스템은 `BadRequestException`을 반환합니다.

`GetLatestConfiguration`를 호출하면 클라이언트 코드는 가장 최근의 `ConfigurationToken` 값을 보내고 이에 대한 응답으로 수신합니다.

- `NextPollConfigurationToken`: 다음에 `GetLatestConfiguration` 호출 시 사용할 `ConfigurationToken` 값입니다.
- `NextPollIntervalInSeconds`: 클라이언트가 `GetLatestConfiguration`에 다음 호출을 하기 전에 기다려야 하는 시간.

- 구성: 세션에 사용할 최신 데이터. 클라이언트에 이미 최신 버전의 구성이 있는 경우 비어 있을 수 있습니다.

> **Important**
>
> 다음 중요 정보를 기록해 둡니다.
>
> - StartConfigurationSessionAPI는 애플리케이션, 환경, 구성 프로필 및 클라이언트당 한 번만 호출하여 서비스와의 세션을 설정해야 합니다. 이는 일반적으로 애플리케이션을 시작할 때 또는 구성을 처음 검색하기 직전에 수행됩니다.
> - `KmsKeyIdentifier`를 사용하여 구성을 배포하는 경우 구성 수신 요청에 `kms:Decrypt` 호출 권한이 포함되어야 합니다. 자세한 내용은 AWS Key Management Service API 참조의 복호화 섹션을 참조하십시오.
> - 이전에 구성 데이터를 검색하는 데 사용했던 API `GetConfiguration` 작업인 은 더 이상 사용되지 않습니다. `GetConfiguration` API 작업은 암호화된 구성을 지원하지 않습니다.

구성 예제 검색

다음 AWS CLI 예제는 데이터 `StartConfigurationSession` 및 `GetLatestConfiguration` API 작업을 사용하여 구성 AWS AppConfig 데이터를 검색하는 방법을 보여줍니다. 첫 번째 명령은 구성 세션을 시작합니다. 이 호출에는 AWS AppConfig 애플리케이션, 환경 및 구성 프로필의 ID (또는 이름)가 포함됩니다. API는 구성 데이터를 가져오는 데 사용된 `InitialConfigurationToken`을 반환합니다.

```
aws appconfigdata start-configuration-session \
    --application-identifier application_name_or_ID \
    --environment-identifier environment_name_or_ID \
    --configuration-profile-identifier configuration_profile_name_or_ID
```

시스템은 다음과 같은 형식의 정보로 응답합니다.

```
{
   "InitialConfigurationToken": initial configuration token
}
```

세션을 시작한 후에는 InitialConfigurationToken를 GetLatestConfiguration호출하여 구성 데이터를 가져오십시오. 구성 데이터가 `mydata.json` 파일에 저장됩니다.

```
aws appconfigdata get-latest-configuration \
    --configuration-token initial configuration token mydata.json
```

GetLatestConfiguration을 처음 호출할 때는 StartConfigurationSession에서 확보한 ConfigurationToken을 사용합니다. 다음 정보가 반환됩니다.

```
{
    "NextPollConfigurationToken" : next configuration token,
    "ContentType" : content type of configuration,
    "NextPollIntervalInSeconds" : 60
}
```

GetLatestConfiguration에 대한 후속 호출은 이전 응답의 NextPollConfigurationToken을 제공해야 합니다.

```
aws appconfigdata get-latest-configuration \
    --configuration-token next configuration token mydata.json
```

Important

GetLatestConfiguration API 작업에 대한 다음과 같은 중요 세부 정보에 주의하십시오.

- GetLatestConfiguration 응답에는 구성 데이터를 보여주는 Configuration 섹션이 포함됩니다. 이 Configuration 섹션은 시스템에서 새 구성 데이터 또는 업데이트된 구성 데이터를 찾은 경우에만 나타납니다. 시스템에서 새 구성 데이터나 업데이트된 구성 데이터를 찾지 못하면 해당 Configuration 데이터는 비어 있습니다.
- 모든 ConfigurationToken 응답에서 새 GetLatestConfiguration을 받습니다.
- 예산, 구성 배포의 예상 빈도, 구성할 대상 수를 기준으로 GetLatestConfiguration API 직접 호출의 폴링 빈도를 조정하는 것이 좋습니다.

확장을 사용하여 워크플로우 확장

확장 프로그램은 구성을 만들거나 배포하는 AWS AppConfig 워크플로우 중에 다양한 지점에 로직 또는 동작을 삽입하는 기능을 강화합니다. 예를 들어 확장 작업을 사용하여 다음 유형의 작업을 수행할 수 있습니다.

- 구성 프로파일이 배포되면 Amazon Simple Notification Service(SNS) 주제에 알림을 전송합니다.
- 배포를 시작하기 전에 민감한 데이터의 구성 프로필 내용을 스크러빙합니다.
- 기능 플래그가 변경될 때마다 Atlassian Jira 이슈를 생성하거나 업데이트합니다.
- 배포를 시작할 때 서비스 또는 데이터 소스의 콘텐츠를 구성 데이터에 병합합니다.
- 구성이 배포될 때마다 Amazon Simple Storage Service(S3) 버킷으로 구성을 백업합니다.

이러한 유형의 작업을 AWS AppConfig 응용 프로그램, 환경 및 구성 프로필과 연결할 수 있습니다.

내용

- AWS AppConfig 확장 프로그램 정보
- AWS 작성 확장 작업
- 안내: 사용자 지정 확장 만들기 AWS AppConfig
- AWS AppConfig Atlassian Jira와의 확장 통합

AWS AppConfig 확장 프로그램 정보

이 항목에서는 AWS AppConfig 확장 개념과 용어를 소개합니다. 이 정보는 AWS AppConfig 확장 설정 및 사용에 필요한 각 단계의 맥락에서 설명됩니다.

주제

- 1단계: 확장으로 수행할 작업 결정
- 2단계: 확장 실행 시기 결정
- 3단계: 확장 연결 생성
- 4단계: 구성 배포 및 확장 액션이 수행되었는지 확인

1단계: 확장으로 수행할 작업 결정

AWS AppConfig 배포가 완료될 때마다 Slack으로 메시지를 보내는 알림을 웹훅으로 수신하시겠습니까? 구성을 배포하기 전에 Amazon Simple Storage Service(S3) 버킷에 구성 프로필을 백업하시겠습니까? 구성을 배포하기 전에 민감한 정보의 구성 데이터를 스크러빙 하시겠습니까? 확장을 사용하여 이러한 유형의 작업 등을 수행할 수 있습니다. 사용자 지정 확장을 만들거나 에 포함된 AWS 작성된 확장을 사용할 수 있습니다. AWS AppConfig

> **Note**
>
> 대부분의 사용 사례에서 사용자 지정 확장을 만들려면 확장에 정의된 모든 계산 및 처리를 수행하는 AWS Lambda 함수를 만들어야 합니다. 자세한 설명은 안내: 사용자 지정 확장 만들기 AWS AppConfig 섹션을 참조하세요.

다음과 같이 AWS 작성된 확장을 사용하면 구성 배포를 다른 서비스와 빠르게 통합할 수 있습니다. AWS AppConfig 콘솔에서 또는 AWS CLI AWS Tools for PowerShell, 또는 SDK에서 직접 확장 API 작업을 호출하여 이러한 확장을 사용할 수 있습니다.

확장	설명
아마존 CloudWatch 에비빌리 A/B 테스트	이 확장 프로그램을 사용하면 애플리케이션이 작업을 호출하는 대신 로컬에서 사용자 세션에 변형을 할당할 수 있습니다. EvaluateFeature 자세한 설명은 아마존 CloudWatch 에비빌리티 확장 프로그램 사용하기 섹션을 참조하세요.
AWS AppConfig 배포 이벤트 대상 EventBridge	이 확장은 구성이 배포될 때 EventBridge 기본 이벤트 버스로 이벤트를 보냅니다.
AWS AppConfig 아마존 심플 알림 서비스 (아마존 SNS) 에 이벤트 배포	이 확장은 구성을 배포할 때 지정한 Amazon SNS 주제로 메시지를 전송합니다.
AWS AppConfig 아마존 심플 큐 서비스 (Amazon SQS) 로의 배포 이벤트	이 확장은 구성이 배포될 때 메시지를 Amazon SQS 대기열에 넣습니다.

확장	설명
통합 확장-Atlassian Jira	이 확장을 사용하면 기능 플래그를 변경할 때마다 AWS AppConfig 이슈를 생성하고 업데이트할 수 있습니다.

2단계: 확장 실행 시기 결정

확장은 AWS AppConfig 워크플로 중에 수행하는 하나 이상의 작업을 정의합니다. 예를 들어, AWS 작성된 `AWS AppConfig deployment events to Amazon SNS` 확장 프로그램에는 Amazon SNS 주제에 알림을 보내는 작업이 포함됩니다. 각 작업은 사용자와 상호 작용할 때 AWS AppConfig 또는 사용자를 대신하여 프로세스를 수행할 때 AWS AppConfig 호출됩니다. 이를 액션 포인트라고 합니다. AWS AppConfig 확장 프로그램은 다음과 같은 액션 포인트를 지원합니다.

- `PRE_CREATE_HOSTED_CONFIGURATION_VERSION`
- `PRE_START_DEPLOYMENT`
- `ON_DEPLOYMENT_START`
- `ON_DEPLOYMENT_STEP`
- `ON_DEPLOYMENT_BAKING`
- `ON_DEPLOYMENT_COMPLETE`
- `ON_DEPLOYMENT_ROLLED_BACK`

`PRE_*`액션 포인트에 구성된 확장 액션은 요청 유효성 검사 후 적용되지만 액션 포인트 이름에 해당하는 액티비티를 AWS AppConfig 수행하기 전에 적용됩니다. 이러한 액션은 요청과 동시에 처리됩니다. 요청이 두 개 이상 이루어진 경우 액션 간접 호출은 순차적으로 실행됩니다. 또한 `PRE_*` 액션 포인트는 구성 내용을 수신하고 변경할 수 있다는 점에 유의하십시오. `PRE_*` 액션 포인트는 오류에 대응하여 조치가 취해지는 것을 방지할 수도 있습니다.

`ON_*`작업 지점을 사용하여 확장 프로그램을 AWS AppConfig 워크플로와 병렬로 실행할 수도 있습니다. `ON_*`액션 포인트는 비동기적으로 호출됩니다. `ON_*`액션 포인트는 구성의 내용을 수신하지 않습니다. `ON_*` 액션 포인트 중에 확장에 오류가 발생하는 경우 서비스는 오류를 무시하고 워크플로우를 계속합니다.

3단계: 확장 연결 생성

확장을 만들거나 AWS 작성된 확장을 구성하려면 특정 AWS AppConfig 리소스가 사용될 때 확장 프로그램을 호출하는 작업 지점을 정의합니다. 예를 들어, 특정 애플리케이션에 대한 구성 배포가 시작될 때마다 `AWS AppConfig deployment events to Amazon SNS` 확장을 실행하고 Amazon SNS 주제에 대한 알림을 수신하도록 선택할 수 있습니다. 특정 AWS AppConfig 리소스에 대한 확장 프로그램을 호출하는 작업 지점을 정의하는 것을 확장 연결이라고 합니다. 확장 연결은 확장 프로그램과 AWS AppConfig 리소스 (예: 응용 프로그램 또는 구성 프로필) 간의 지정된 관계입니다.

단일 AWS AppConfig 응용 프로그램에는 여러 환경 및 구성 프로필이 포함될 수 있습니다. 확장을 응용 프로그램이나 환경에 연결하는 경우 응용 프로그램 또는 환경 리소스와 관련된 모든 워크플로에 대해 확장을 AWS AppConfig 호출합니다 (해당하는 경우).

예를 들어, 라는 구성 프로필이 포함된 AWS AppConfig 응용 프로그램이 MobileApps 있다고 가정해 보겠습니다. AccessList 그리고 MobileApps 애플리케이션에 베타, 통합 및 프로덕션 환경이 포함되어 있다고 가정해 보겠습니다. AWS 작성한 Amazon SNS 알림 확장에 대한 확장 프로그램 연결을 생성하고 확장 프로그램을 애플리케이션에 연결합니다. MobileApps Amazon SNS 알림 확장은 애플리케이션 구성이 세 가지 환경 중 하나에 배포될 때마다 간접적으로 호출됩니다.

> **Note**
>
> AWS 작성된 확장을 사용하기 위해 확장을 생성할 필요는 없지만 확장 연결을 생성해야 합니다.

4단계: 구성 배포 및 확장 액션이 수행되었는지 확인

연결을 만든 후 호스팅된 구성을 만들거나 구성을 배포할 때 확장을 AWS AppConfig 호출하고 지정된 작업을 수행합니다. 확장을 호출할 때 `PRE-*` 작업 지점 중에 시스템에서 오류가 발생하면 해당 오류에 대한 정보가 AWS AppConfig 반환됩니다.

AWS 작성 확장 작업

AWS AppConfig 다음과 같은 AWS 작성된 확장이 포함됩니다. 이러한 확장은 AWS AppConfig 워크플로를 다른 서비스와 통합하는 데 도움이 될 수 있습니다. AWS CLI AWS Tools for PowerShell, AWS Management Console 또는 SDK에서 직접 확장 API 작업을 호출하여 OR 내에서 이러한 확장을 사용할 수 있습니다.

확장	설명
아마존 CloudWatch 에비빌리 A/B 테스트	이 확장 프로그램을 사용하면 애플리케이션이 작업을 호출하는 대신 로컬에서 사용자 세션에 변형을 할당할 수 있습니다. EvaluateFeature 자세한 설명은 아마존 CloudWatch 에비빌리티 확장 프로그램 사용하기 섹션을 참조하세요.
AWS AppConfig 배포 이벤트 대상 EventBridge	이 확장은 구성이 배포될 때 EventBridge 기본 이벤트 버스로 이벤트를 보냅니다.
AWS AppConfig 아마존 심플 알림 서비스 (아마존 SNS) 에 이벤트 배포	이 확장은 구성을 배포할 때 지정한 Amazon SNS 주제로 메시지를 전송합니다.
AWS AppConfig 아마존 심플 큐 서비스 (Amazon SQS) 로의 배포 이벤트	이 확장은 구성이 배포될 때 메시지를 Amazon SQS 대기열에 넣습니다.
통합 확장-Atlassian Jira	이 확장을 사용하면 기능 플래그를 변경할 때마다 AWS AppConfig 이슈를 생성하고 업데이트할 수 있습니다.

아마존 CloudWatch 에비빌리티 확장 프로그램 사용하기

Amazon CloudWatch Evivently를 사용하면 기능을 출시하는 동안 지정된 비율의 사용자에게 새 기능을 제공하여 새 기능을 안전하게 검증할 수 있습니다. 새 기능의 성능을 모니터링해 사용자에게 트래픽을 늘릴 시기를 결정할 수 있습니다. 이를 통해 위험을 줄이고 의도하지 않은 결과를 파악한 후 기능을 완전히 출시할 수 있습니다. 또한 A/B 실험을 수행해 증거와 데이터를 기반으로 기능 설계 결정을 내릴 수 있습니다.

CloudWatch Eviventy의 AWS AppConfig 확장 프로그램을 사용하면 애플리케이션에서 작업을 호출하는 대신 로컬에서 사용자 세션에 변형을 할당할 수 있습니다. EvaluateFeature 로컬 세션은 API 호출로 인한 지연 시간 및 가용성 위험을 줄일 수 있습니다. 확장 프로그램을 구성하고 사용하는 방법에 대한 자세한 내용은 Amazon 사용 CloudWatch 설명서의 CloudWatch Evively를 사용하여 출시 및 A/B 실험 수행을 참조하십시오.

AWS AppConfig deployment events to Amazon EventBridge 확장 작업

AWS AppConfig deployment events to Amazon EventBridge확장 프로그램은 구성 배포 AWS 워크플로를 모니터링하고 조치를 취하는 데 도움이 되는 작성된 확장입니다. AWS AppConfig 확장은 구성이 배포될 때마다 EventBridge 기본 이벤트 버스에 이벤트 알림을 보냅니다. 확장 AWS AppConfig 프로그램을 응용 프로그램, 환경 또는 구성 프로필 중 하나에 연결한 후에는 구성 배포가 시작, 종료 및 롤백될 때마다 이벤트 버스에 이벤트 알림을 AWS AppConfig 보냅니다.

EventBridge 알림을 보내는 작업 지점을 더 세밀하게 제어하려면 사용자 지정 확장을 만들고 URI 필드에 EventBridge 기본 이벤트 버스 Amazon Resource Name (ARN) 을 입력하면 됩니다. 확장 생성에 대한 자세한 내용은 안내: 사용자 지정 확장 만들기 AWS AppConfig 섹션을 참조하세요.

> **Important**
>
> 이 확장은 EventBridge 기본 이벤트 버스만 지원합니다.

확장 사용

AWS AppConfig deployment events to Amazon EventBridge확장을 사용하려면 먼저 확장 연결을 만들어 AWS AppConfig 리소스 중 하나에 확장을 연결합니다. AWS AppConfig 콘솔이나 CreateExtensionAssociationAPI 작업을 사용하여 연결을 생성합니다. 연결을 생성할 때 AWS AppConfig 애플리케이션, 환경 또는 구성 프로필의 ARN을 지정합니다. 확장을 애플리케이션 또는 환경에 연결하는 경우 지정된 애플리케이션 또는 환경에 포함된 모든 구성 프로필에 대한 이벤트 알림이 전송됩니다.

연결을 생성한 후 지정된 AWS AppConfig 리소스의 구성이 배포되면 확장을 AWS AppConfig 호출하고 확장에 지정된 작업 지점에 따라 알림을 보냅니다.

> **Note**
>
> 이 확장은 다음 액션 포인트에서 간접적으로 호출됩니다.
>
> - ON_DEPLOYMENT_START
> - ON_DEPLOYMENT_COMPLETE
> - ON_DEPLOYMENT_ROLLED_BACK

이 확장의 액션 포인트는 사용자 지정할 수 없습니다. 자체 확장을 생성하여 다른 액션 포인트를 간접적으로 호출할 수 있습니다. 자세한 설명은 안내: 사용자 지정 확장 만들기 AWS AppConfig 섹션을 참조하세요.

다음 절차에 따라 AWS Systems Manager 콘솔이나 를 사용하여 AWS AppConfig 확장 연결을 만들 수 있습니다. AWS CLI

확장 연결을 생성하려면(콘솔)

1. https://console.aws.amazon.com/systems-manager/appconfig/ 에서 AWS Systems Manager 콘솔을 엽니다.
2. 탐색 창에서 AWS AppConfig를 선택합니다.
3. 확장 탭에서 리소스에 추가를 선택합니다.
4. 확장 리소스 세부 정보 섹션의 리소스 유형에서 AWS AppConfig 리소스 유형을 선택합니다. 선택한 리소스에 따라 다른 리소스를 AWS AppConfig 선택하라는 메시지가 표시됩니다.
5. 리소스에 연결 만들기를 선택합니다.

다음은 확장 프로그램이 EventBridge 호출될 때 전송되는 샘플 이벤트입니다.

```
{
   "version":"0",
   "id":"c53dbd72-c1a0-2302-9ed6-c076e9128277",
   "detail-type":"On Deployment Complete",
   "source":"aws.appconfig",
   "account":"111122223333",
   "time":"2022-07-09T01:44:15Z",
   "region":"us-east-1",
   "resources":[
      "arn:aws:appconfig:us-east-1:111122223333:extensionassociation/z763ff5"
   ],
   "detail":{
      "InvocationId":"5tfjcig",
       "Parameters":{

      },
      "Type":"OnDeploymentComplete",
      "Application":{
```

```
         "Id":"ba8toh7",
         "Name":"MyApp"
      },
      "Environment":{
         "Id":"pgil2o7",
         "Name":"MyEnv"
      },
      "ConfigurationProfile":{
         "Id":"ga3tqep",
         "Name":"MyConfigProfile"
      },
      "DeploymentNumber":1,
      "ConfigurationVersion":"1"
   }
}
```

AWS AppConfig deployment events to Amazon SNS 확장 작업

AWS AppConfig deployment events to Amazon SNS확장 프로그램은 AWS AppConfig 구성 배포 워크플로를 모니터링하고 조치를 취하는 데 도움이 되는 AWS 작성된 확장입니다. 확장 기능은 구성이 배포될 때마다 Amazon SNS 주제에 메시지를 게시합니다. 확장 AWS AppConfig 프로그램을 응용 프로그램, 환경 또는 구성 프로필 중 하나에 연결하면 구성 배포가 시작, 종료 및 롤백될 때마다 주제에 메시지를 AWS AppConfig 게시합니다.

Amazon SNS 알림을 보내는 액션 포인트를 더 세밀하게 제어하려면 사용자 지정 확장을 만들고 URI 필드에 Amazon SNS 주제 Amazon 리소스 이름(ARN) 을 입력하면 됩니다. 확장 생성에 대한 자세한 내용은 안내: 사용자 지정 확장 만들기 AWS AppConfig 섹션을 참조하세요.

확장 사용

이 섹션에서는 AWS AppConfig deployment events to Amazon SNS 확장 작업 방법에 대해 설명합니다.

1단계: 주제에 메시지를 AWS AppConfig 게시하도록 구성

Amazon SNS 주제에 액세스 제어 정책을 추가하여 AWS AppConfig (appconfig.amazonaws.com) 게시 권한 (sns:Publish) 을 부여합니다. 자세한 내용은 Amazon SNS 액세스 제어의 예제 사례를 참조하십시오.

2단계: 확장 연결 생성

확장 프로그램 연결을 만들어 AWS AppConfig 리소스 중 하나에 확장 프로그램을 연결합니다. AWS AppConfig 콘솔이나 CreateExtensionAssociationAPI 작업을 사용하여 연결을 생성합니다. 연결을 생성할 때 AWS AppConfig 애플리케이션, 환경 또는 구성 프로필의 ARN을 지정합니다. 확장을 애플리케이션 또는 환경에 연결하는 경우 지정된 애플리케이션 또는 환경에 포함된 구성 프로필에 대한 알림이 전송됩니다. 연결을 생성할 때, 사용하려는 Amazon SNS 주제의 ARN이 포함된 `topicArn` 파라미터 값을 입력해야 합니다.

연결을 생성한 후 지정된 AWS AppConfig 리소스의 구성이 배포되면 확장을 AWS AppConfig 호출하고 확장에 지정된 작업 지점에 따라 알림을 보냅니다.

> **Note**
>
> 이 확장은 다음 액션 포인트에서 간접적으로 호출됩니다.
>
> - `ON_DEPLOYMENT_START`
> - `ON_DEPLOYMENT_COMPLETE`
> - `ON_DEPLOYMENT_ROLLED_BACK`
>
> 이 확장의 액션 포인트는 사용자 지정할 수 없습니다. 자체 확장을 생성하여 다른 액션 포인트를 간접적으로 호출할 수 있습니다. 자세한 설명은 안내: 사용자 지정 확장 만들기 AWS AppConfig 섹션을 참조하세요.

다음 절차에 따라 AWS Systems Manager 콘솔이나 를 사용하여 AWS AppConfig 확장 연결을 만들 수 있습니다. AWS CLI

확장 연결을 생성하려면(콘솔)

1. https://console.aws.amazon.com/systems-manager/appconfig/ 에서 AWS Systems Manager 콘솔을 엽니다.
2. 탐색 창에서 AWS AppConfig를 선택합니다.
3. 확장 탭에서 리소스에 추가를 선택합니다.
4. 확장 리소스 세부 정보 섹션의 리소스 유형에서 AWS AppConfig 리소스 유형을 선택합니다. 선택한 리소스에 따라 다른 리소스를 AWS AppConfig 선택하라는 메시지가 표시됩니다.
5. 리소스에 연결 만들기를 선택합니다.

다음은 확장이 간접적으로 호출될 때 Amazon SNS 주제로 전송되는 메시지의 샘플입니다.

```
{
    "Type": "Notification",
    "MessageId": "ae9d702f-9a66-51b3-8586-2b17932a9f28",
    "TopicArn": "arn:aws:sns:us-east-1:111122223333:MySNSTopic",
    "Message": {
        "InvocationId": "7itcaxp",
        "Parameters": {
            "topicArn": "arn:aws:sns:us-east-1:111122223333:MySNSTopic"
        },
        "Application": {
            "Id": "1a2b3c4d",
            "Name": MyApp
        },
        "Environment": {
            "Id": "1a2b3c4d",
            "Name": MyEnv
        },
        "ConfigurationProfile": {
            "Id": "1a2b3c4d",
            "Name": "MyConfigProfile"
        },
        "Description": null,
        "DeploymentNumber": "3",
        "ConfigurationVersion": "1",
        "Type": "OnDeploymentComplete"
    },
    "Timestamp": "2022-06-30T20:26:52.067Z",
    "SignatureVersion": "1",
    "Signature": "<...>",
    "SigningCertURL": "<...>",
    "UnsubscribeURL": "<...>",
    "MessageAttributes": {
        "MessageType": {
            "Type": "String",
            "Value": "OnDeploymentStart"
        }
    }
}
```

AWS AppConfig deployment events to Amazon SQS 확장 작업

확장은 AWS AppConfig 구성 배포 워크플로를 모니터링하고 조치를 취하는 데 도움이 되는 AWS 작성된 AWS AppConfig deployment events to Amazon SQS 확장입니다. 확장은 구성이 배포될 때마다 메시지를 Amazon Simple Queue Service(Amazon SQS) 대기열에 추가합니다. 확장 AWS AppConfig 프로그램을 응용 프로그램, 환경 또는 구성 프로필 중 하나에 연결하면 구성 배포가 시작, AWS AppConfig 종료 및 롤백될 때마다 메시지를 대기열에 넣습니다.

Amazon SQS 알림을 보내는 액션 포인트를 더 세밀하게 제어하려면 사용자 지정 확장을 생성하고 URI 필드에 Amazon SQS 대기열 Amazon 리소스 이름(ARN)을 입력하면 됩니다. 확장 생성에 대한 자세한 내용은 안내: 사용자 지정 확장 만들기 AWS AppConfig 섹션을 참조하세요.

확장 사용

이 섹션에서는 AWS AppConfig deployment events to Amazon SQS 확장 작업 방법에 대해 설명합니다.

1단계: 메시지를 대기열에 넣도록 구성 AWS AppConfig

Amazon SQS 대기열에 메시지 전송 권한(sqs:SendMessage)을 부여하는 AWS AppConfig (appconfig.amazonaws.com) Amazon SQS 정책을 추가합니다. 자세한 내용은 Amazon SQ 정책의 기본 예를 참조하십시오.

2단계: 확장 연결 생성

확장 프로그램 연결을 만들어 AWS AppConfig 리소스 중 하나에 확장 프로그램을 연결합니다. AWS AppConfig 콘솔이나 CreateExtensionAssociationAPI 작업을 사용하여 연결을 생성합니다. 연결을 생성할 때 AWS AppConfig 애플리케이션, 환경 또는 구성 프로필의 ARN을 지정합니다. 확장을 애플리케이션 또는 환경에 연결하는 경우 지정된 애플리케이션 또는 환경에 포함된 구성 프로필에 대한 알림이 전송됩니다. 연결을 생성할 때, 사용하려는 Amazon SQS 대기열의 ARN이 포함된 Here 파라미터를 입력해야 합니다.

연결을 생성한 후 지정된 AWS AppConfig 리소스의 구성이 생성되거나 배포되면 확장을 AWS AppConfig 호출하고 확장에 지정된 작업 지점에 따라 알림을 보냅니다.

> **Note**
>
> 이 확장은 다음 액션 포인트에서 간접적으로 호출됩니다.
>
> - ON_DEPLOYMENT_START
> - ON_DEPLOYMENT_COMPLETE

- ON_DEPLOYMENT_ROLLED_BACK

이 확장의 액션 포인트는 사용자 지정할 수 없습니다. 자체 확장을 생성하여 다른 액션 포인트를 간접적으로 호출할 수 있습니다. 자세한 설명은 안내: 사용자 지정 확장 만들기 AWS AppConfig 섹션을 참조하세요.

다음 절차에 따라 AWS Systems Manager 콘솔이나 를 사용하여 AWS AppConfig 확장 연결을 만들 수 있습니다. AWS CLI

확장 연결을 생성하려면(콘솔)

1. https://console.aws.amazon.com/systems-manager/appconfig/ 에서 AWS Systems Manager 콘솔을 엽니다.
2. 탐색 창에서 AWS AppConfig를 선택합니다.
3. 확장 탭에서 리소스에 추가를 선택합니다.
4. 확장 리소스 세부 정보 섹션의 리소스 유형에서 AWS AppConfig 리소스 유형을 선택합니다. 선택한 리소스에 따라 다른 리소스를 AWS AppConfig 선택하라는 메시지가 표시됩니다.
5. 리소스에 연결 만들기를 선택합니다.

다음은 확장이 간접적으로 호출될 때 Amazon SQS 대기열로 전송되는 메시지의 예입니다.

```
{
   "InvocationId":"7itcaxp",
   "Parameters":{
      "queueArn":"arn:aws:sqs:us-east-1:111122223333:MySQSQueue"
   },
   "Application":{
      "Id":"1a2b3c4d",
      "Name":MyApp
   },
   "Environment":{
      "Id":"1a2b3c4d",
      "Name":MyEnv
   },
   "ConfigurationProfile":{
      "Id":"1a2b3c4d",
      "Name":"MyConfigProfile"
```

```
    },
    "Description":null,
    "DeploymentNumber":"3",
    "ConfigurationVersion":"1",
    "Type":"OnDeploymentComplete"
}
```

에 대한 Atlassian Jira 확장 프로그램을 사용하여 작업하기 AWS AppConfig

Atlassian Jira와 통합하면 지정된 기능에 대한 기능 플래그를 변경할 때마다 Atlassian 콘솔에서 이슈를 생성하고 업데이트할 AWS AppConfig 수 있습니다. AWS 계정 AWS 리전각 Jira 이슈에는 플래그 이름, 애플리케이션 ID, 구성 프로필 ID 및 플래그 값이 포함됩니다. 플래그 변경 사항을 업데이트, 저장 및 배포한 후 Jira는 변경 세부 정보와 함께 기존 이슈를 업데이트합니다.

> **Note**
>
> 기능 플래그를 생성하거나 업데이트할 때마다 Jira에서 이슈를 업데이트합니다. Jira는 상위 수준 플래그에서 하위 수준 플래그 속성을 삭제할 때도 문제를 업데이트합니다. 상위 수준 플래그를 삭제하면 Jira는 정보를 기록하지 않습니다.

통합을 구성하려면 다음을 수행하여야 합니다.

- AWS AppConfig Jira 통합을 위한 권한 구성
- AWS AppConfig Jira 통합 애플리케이션 구성

AWS AppConfig Jira 통합을 위한 권한 구성

AWS AppConfig Jira와의 통합을 구성할 때 사용자의 자격 증명을 지정합니다. 특히 Jira 애플리케이션을 위한AWS AppConfig 에 사용자의 액세스 키 ID와 암호 키를 입력합니다. 이 사용자는 Jira와 통신할 수 있는 권한을 부여합니다. AWS AppConfig AWS AppConfig 이러한 자격 증명을 한 번 사용하여 AWS AppConfig Jira와 Jira 간의 연결을 설정합니다. 자격 증명은 저장되지 않습니다. AWS AppConfig Jira용 애플리케이션을 제거하여 연결을 제거할 수 있습니다.

사용자 계정에는 다음 액션이 포함된 권한 정책이 필요합니다.

- `appconfig:CreateExtensionAssociation`
- `appconfig:GetConfigurationProfile`

- appconfig:ListApplications
- appconfig:ListConfigurationProfiles
- appconfig:ListExtensionAssociations
- sts:GetCallerIdentity

다음 작업을 완료하여 AWS AppConfig 및 Jira의 통합을 위한 IAM 권한 정책과 사용자를 생성하십시오.

작업

- 작업 1: Jira 통합을 위한 AWS AppConfig IAM 권한 정책 생성
- 작업 2: Jira 통합을 AWS AppConfig 위한 사용자 생성

작업 1: Jira 통합을 위한 AWS AppConfig IAM 권한 정책 생성

다음 절차를 사용하여 Atlassian Jira가 통신할 수 있도록 허용하는 IAM 권한 정책을 생성하십시오. AWS AppConfig새 정책을 생성한 후 이 정책을 새 IAM 역할에 연결하는 것이 좋습니다. 기존 IAM 정책 및 역할에 필요한 권한을 추가하는 것은 최소 권한 원칙에 위배되므로 권장되지 않습니다.

및 Jira 통합을 위한 IAM 정책을 만들려면 AWS AppConfig

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 탐색 창에서 정책을 선택한 후 정책 생성을 선택합니다.
3. 정책 생성 페이지에서 JSON 탭을 선택하고, 기본 콘텐츠를 다음 정책으로 바꿉니다. 다음 정책에서 *##*, *##_ID*, *######_ID*, *##_###_ID*를 AWS AppConfig 기능 플래그 환경 정보로 대체하십시오.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                      "appconfig:CreateExtensionAssociation",
                      "appconfig:ListExtensionAssociations",
                      "appconfig:GetConfigurationProfile"
             ],
            "Resource": [
```

```
 "arn:aws:appconfig:Region:account_ID:application/application_ID",

 "arn:aws:appconfig:Region:account_ID:application/application_ID/
configurationprofile/configuration_profile_ID"
             ]
       },
      {
           "Effect": "Allow",
           "Action": [
                     "appconfig:ListApplications"

            ],
           "Resource": [
                     "arn:aws:appconfig:Region:account_ID:*"
            ]
       },
        {
           "Effect": "Allow",
           "Action": [
                     "appconfig:ListConfigurationProfiles"
            ],
           "Resource": [

 "arn:aws:appconfig:Region:account_ID:application/application_ID"
             ]
       },
       {
           "Effect": "Allow",
           "Action": "sts:GetCallerIdentity",
           "Resource": "*"
       }
    ]
}
```

4. 다음: 태그(Next: Tags)를 선택합니다.
5. (선택 사항) 이 정책에 대한 액세스를 구성, 추적 또는 제어할 태그-키 값 페어를 하나 이상 추가한 후 [다음: 검토]를 선택합니다.
6. 정책 검토 페이지에서 이름 상자에 **AppConfigJiraPolicy** 등의 이름을 입력한 다음 설명을 입력합니다(선택 사항).
7. 정책 생성을 선택합니다.

작업 2: Jira 통합을 AWS AppConfig 위한 사용자 생성

다음 절차를 사용하여 Atlassian Jira 통합을 위한 AWS AppConfig 사용자를 생성하십시오. 사용자를 생성한 후 통합 완료 시 지정할 액세스 키 ID와 암호 키를 복사할 수 있습니다.

Jira 통합을 위한 AWS AppConfig 사용자 생성하기

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 탐색 창에서 사용자와 사용자 추가를 차례로 선택합니다.
3. 사용자 이름 필드에 이름(예:**`AppConfigJiraUser`**)을 입력합니다.
4. AWS 자격 증명 유형 선택에서 액세스 키 - 프로그래밍 방식 액세스를 선택합니다.
5. 다음: 권한을 선택합니다.
6. 권한 설정 페이지에서 기존 정책을 직접 연결을 선택합니다. 작업 1: Jira 통합을 위한 AWS AppConfig IAM 권한 정책 생성에서 생성한 정책을 검색하고 확인란을 선택한 다음 다음: 태그를 선택하십시오.
7. 태그 추가 (선택 사항) 페이지에서, 이 사용자에 대한 액세스를 구성, 추적 또는 제어할 태그-키 값 페어를 하나 이상 추가합니다. 다음: 검토를 선택합니다.
8. 검토 페이지에서 사용자 세부 정보를 확인합니다.
9. 사용자 생성을 선택합니다. 시스템은 사용자의 액세스 키 ID와 암호 키를 표시합니다. .csv 파일을 다운로드하거나 이러한 자격 증명을 별도의 위치에 복사하십시오. 통합을 구성할 때 이러한 자격 증명을 지정해야 합니다.

AWS AppConfig Jira 통합 애플리케이션 구성

다음 절차를 사용하여 AWS AppConfig for Jira 애플리케이션에서 필수 옵션을 구성하십시오. 이 절차를 완료하면 Jira는 지정된 기능에 대한 각 기능 플래그에 대해 새 이슈를 AWS 계정 생성합니다. AWS 리전에서 AWS AppConfig기능 플래그를 변경하면 Jira는 기존 이슈에 세부 정보를 기록합니다.

> **Note**
>
> AWS AppConfig 기능 플래그는 여러 하위 수준 플래그 속성을 포함할 수 있습니다. Jira는 각 상위 수준 기능 플래그에 대해 하나의 이슈를 생성합니다. 하위 수준 플래그 속성을 변경하는 경우 상위 수준 플래그에 대한 Jira 이슈에서 해당 변경의 세부 정보를 볼 수 있습니다.

통합을 구성하려면

1. Atlassian 마켓플레이스에 로그인하세요.
2. 검색 필드에 **`AWS AppConfig`**를 입력하고 Enter(입력)을 누릅니다.
3. Jira 인스턴스에 애플리케이션을 설치합니다.
4. Atlassian 콘솔에서 앱 관리를 선택한 다음 Jira용AWS AppConfig 를 선택합니다.
5. 구성을 선택합니다.
6. 구성 세부 정보에서 Jira 프로젝트를 선택한 다음 AWS AppConfig 기능 플래그와 연결할 프로젝트를 선택합니다.
7. AWS 리전을 선택한 다음 AWS AppConfig 기능 플래그가 있는 리전을 선택합니다.
8. 애플리케이션 ID 필드에 기능 플래그가 포함된 AWS AppConfig 애플리케이션의 이름을 입력합니다.
9. 구성 프로필 ID 필드에 기능 플래그의 AWS AppConfig 구성 프로필 이름을 입력합니다.
10. 액세스 키 ID 및 암호 키 필드에 작업 2: Jira 통합을 AWS AppConfig 위한 사용자 생성에 복사한 자격 증명을 입력합니다. 선택적으로 세션 토큰을 지정할 수도 있습니다.
11. 제출을 선택합니다.
12. Atlassian 콘솔에서 프로젝트를 선택한 다음 통합을 위해 선택한 프로젝트를 선택합니다. AWS AppConfig 이슈 페이지에는 지정된 AWS 계정 및 의 각 기능 플래그에 대한 이슈가 표시됩니다. AWS 리전

Jira 애플리케이션 및 데이터용 AWS AppConfig 삭제

AWS AppConfig 기능 플래그와 Jira 통합을 더 이상 사용하지 않으려면 Atlassian 콘솔에서 AWS AppConfig Jira용 애플리케이션을 삭제할 수 있습니다. 통합 애플리케이션을 삭제하면 다음 동작을 수행합니다.

- Jira 인스턴스와 Jira 인스턴스 간의 연결을 삭제합니다. AWS AppConfig
- 에서 Jira 인스턴스 세부 정보를 삭제합니다. AWS AppConfig

AWS AppConfig Jira용 애플리케이션을 삭제하려면

1. Atlassian 콘솔에서 앱 관리를 선택합니다.
2. Jira용AWS AppConfig 를 선택합니다.

3. 제거를 선택합니다.

안내: 사용자 지정 확장 만들기 AWS AppConfig

사용자 지정 AWS AppConfig 확장을 만들려면 다음 작업을 완료하세요. 각 작업에 대해서는 이후 주제에 자세히 설명되어 있습니다.

> **Note**
>
> 다음에서 사용자 지정 AWS AppConfig 확장 프로그램의 샘플을 볼 수 있습니다 GitHub.
>
> - Systems Manager 변경 일정을 `blocked day` 사용하여 유예 달력을 통한 배포를 방지하는 샘플 확장 프로그램
> - git-secrets를 사용하여 비밀이 구성 데이터로 유출되는 것을 방지하는 샘플 확장
> - Amazon Comprehend를 사용하여 개인 식별 정보 (PII) 가 구성 데이터로 유출되는 것을 방지하는 샘플 확장 프로그램

1. 함수 생성 AWS Lambda

   대부분의 사용 사례에서 사용자 지정 확장을 만들려면 확장에 정의된 모든 계산 및 처리를 수행하는 AWS Lambda 함수를 만들어야 합니다. 이 규칙의 예외는 액션 포인트를 추가하거나 제거하기 위해 AWS 작성 알림 확장의 사용자 지정 버전을 만드는 경우입니다. 이 예외에 대한 자세한 정보는 사용자 지정 AWS AppConfig 확장 생성 섹션을 참조하세요.

2. 사용자 지정 확장 권한 구성

   사용자 지정 확장 권한을 구성하려면 다음 중 한 가지 방법을 시도하면 됩니다.

   - 권한이 포함된 AWS Identity and Access Management `InvokeFunction` (IAM) 서비스 역할을 생성합니다.
   - AddPermissionLambda API 작업을 사용하여 리소스 정책을 생성합니다.

   이 연습은 IAM 서비스 역할 생성 방법에 대해서 설명합니다.

3. 확장 생성

   AWS AppConfig 콘솔을 사용하거나 AWS CLI AWS Tools for PowerShell, 또는 SDK에서 CreateExtensionAPI 작업을 호출하여 확장을 생성할 수 있습니다. 이 연습에서는 콘솔을 사용합니다.

4. 확장 연결 생성

   AWS AppConfig 콘솔을 사용하거나 AWS CLI AWS Tools for PowerShell, 또는 SDK에서 CreateExtensionAssociationAPI 작업을 호출하여 확장 연결을 만들 수 있습니다. 이 연습에서는 콘솔을 사용합니다.

5. 확장을 간접적으로 호출하는 액션 수행

   연결을 만든 후 확장에 의해 정의된 작업 지점이 해당 리소스에 대해 발생할 때 확장 프로그램을 AWS AppConfig 호출합니다. 예를 들어 PRE_CREATE_HOSTED_CONFIGURATION_VERSION 액션이 포함된 확장을 연결하면 호스팅된 구성 버전을 새로 만들 때마다 확장이 간접적으로 호출됩니다.

이 섹션의 주제에서는 사용자 지정 AWS AppConfig 확장 생성과 관련된 각 작업을 설명합니다. 각 작업은 고객이 Amazon Simple Storage Service(S3) 버킷에 자동으로 백업하는 확장을 생성하려는 사용 사례의 맥락에서 설명됩니다. 확장은 호스팅된 구성을 생성(PRE_CREATE_HOSTED_CONFIGURATION_VERSION) 하거나 배포 (PRE_START_DEPLOYMENT) 할 때마다 실행됩니다.

주제

- 사용자 지정 확장을 위한 Lambda 함수 생성 AWS AppConfig
- 사용자 지정 확장에 대한 권한 구성 AWS AppConfig
- 사용자 지정 AWS AppConfig 확장 생성
- 사용자 지정 AWS AppConfig 확장 프로그램의 확장 프로그램 연결 만들기
- 사용자 지정 확장을 호출하는 작업 수행 AWS AppConfig

사용자 지정 확장을 위한 Lambda 함수 생성 AWS AppConfig

대부분의 사용 사례에서 사용자 지정 확장을 생성하려면 확장에 정의된 모든 계산 및 처리를 수행하는 AWS Lambda 함수를 생성해야 합니다. 이 섹션에는 사용자 지정 확장을 위한 Lambda 함수 샘플 코드가 포함되어 있습니다. AWS AppConfig 이 섹션에는 페이로드 요청 및 응답 참조 세부 정보도 포함되어 있습니다. Lambda 함수 생성에 대한 자세한 내용은 AWS Lambda 개발자 안내서의 Lambda 시작하기를 참조하세요.

샘플 코드

Lambda 함수의 다음 샘플 코드는 호출 시 구성을 Amazon S3 버킷에 자동으로 AWS AppConfig 백업합니다. 구성은 새 구성이 생성되거나 배포될 때마다 백업됩니다. 샘플은 확장 파라미터를 사용하므로

Lambda 함수에서 버킷 이름을 하드코딩할 필요가 없습니다. 확장 파라미터를 사용하여 사용자는 확장을 여러 애플리케이션에 연결하고 구성을 다른 버킷에 백업할 수 있습니다. 코드 샘플에는 함수를 더 자세히 설명하는 주석이 포함되어 있습니다.

확장 프로그램을 위한 샘플 Lambda 함수 AWS AppConfig

```
from datetime import datetime
import base64
import json

import boto3


def lambda_handler(event, context):
    print(event)

    # Extensions that use the PRE_CREATE_HOSTED_CONFIGURATION_VERSION and
 PRE_START_DEPLOYMENT
    # action points receive the contents of AWS AppConfig configurations in Lambda
 event parameters.
    # Configuration contents are received as a base64-encoded string, which the lambda
 needs to decode
    # in order to get the configuration data as bytes. For other action points, the
 content
    # of the configuration isn't present, so the code below will fail.
    config_data_bytes = base64.b64decode(event["Content"])

    # You can specify parameters for extensions. The CreateExtension API action lets
 you define
    # which parameters an extension supports. You supply the values for those
 parameters when you
    # create an extension association by calling the CreateExtensionAssociation API
 action.
    # The following code uses a parameter called S3_BUCKET to obtain the value
 specified in the
    # extension association. You can specify this parameter when you create the
 extension
    # later in this walkthrough.
    extension_association_params = event.get('Parameters', {})
    bucket_name = extension_association_params['S3_BUCKET']
    write_backup_to_s3(bucket_name, config_data_bytes)
```

```
    # The PRE_CREATE_HOSTED_CONFIGURATION_VERSION and PRE_START_DEPLOYMENT action
 points can
    # modify the contents of a configuration. The following code makes a minor change
    # for the purposes of a demonstration.
    old_config_data_string = config_data_bytes.decode('utf-8')
    new_config_data_string = old_config_data_string.replace('hello', 'hello!')
    new_config_data_bytes = new_config_data_string.encode('utf-8')

    # The lambda initially received the configuration data as a base64-encoded string
    # and must return it in the same format.
    new_config_data_base64string =
 base64.b64encode(new_config_data_bytes).decode('ascii')

    return {
        'statusCode': 200,
        # If you want to modify the contents of the configuration, you must include the
 new contents in the
        # Lambda response. If you don't want to modify the contents, you can omit the
 'Content' field shown here.
        'Content': new_config_data_base64string
    }


def write_backup_to_s3(bucket_name, config_data_bytes):
    s3 = boto3.resource('s3')
    new_object = s3.Object(bucket_name,
 f"config_backup_{datetime.now().isoformat()}.txt")
    new_object.put(Body=config_data_bytes)
```

이 연습에서 이 샘플을 사용하려면 이름 **MyS3ConfigurationBackUpExtension**과 함께 저장하고 함수의 Amazon 리소스 이름(ARN)을 복사합니다. 다음 섹션에서 AWS Identity and Access Management (IAM) 수임 역할을 생성할 때 ARN을 지정합니다. 확장을 생성할 때 ARN과 이름을 지정합니다.

페이로드 참조

이 섹션에는 사용자 지정 확장 작업을 위한 페이로드 요청 및 응답 참조 세부 정보가 포함되어 있습니다. AWS AppConfig

요청 구조

PreCreateHostedConfigurationVersion

```
{
    'InvocationId': 'vlns753', // id for specific invocation
    'Parameters': {
        'ParameterOne': 'ValueOne',
        'ParameterTwo': 'ValueTwo'
    },
    'ContentType': 'text/plain',
    'ContentVersion': '2',
    'Content': 'SGVsbG8gZWFydGgh', // Base64 encoded content
    'Application': {
        'Id': 'abcd123',
        'Name': 'ApplicationName'
    },
    'ConfigurationProfile': {
        'Id': 'ijkl789',
        'Name': 'ConfigurationName'
    },
    'Description': '',
    'Type': 'PreCreateHostedConfigurationVersion',
    'PreviousContent': {
        'ContentType': 'text/plain',
        'ContentVersion': '1',
        'Content': 'SGVsbG8gd29ybGQh'
    }
}
```

PreStartDeployment

```
{
    'InvocationId': '765ahdm',
    'Parameters': {
        'ParameterOne': 'ValueOne',
        'ParameterTwo': 'ValueTwo'
    },
    'ContentType': 'text/plain',
    'ContentVersion': '2',
    'Content': 'SGVsbG8gZWFydGgh',
    'Application': {
        'Id': 'abcd123',
        'Name': 'ApplicationName'
    },
    'Environment': {
        'Id': 'ibpnqlq',
```

```
        'Name': 'EnvironmentName'
    },
    'ConfigurationProfile': {
        'Id': 'ijkl789',
        'Name': 'ConfigurationName'
    },
    'DeploymentNumber': 2,
    'Description': 'Deployment description',
    'Type': 'PreStartDeployment'
}
```

비동기 이벤트

OnStartDeployment, OnDeploymentStep, OnDeployment

```
{
    'InvocationId': 'o2xbtm7',
    'Parameters': {
        'ParameterOne': 'ValueOne',
        'ParameterTwo': 'ValueTwo'
    },
    'Type': 'OnDeploymentStart',
    'Application': {
        'Id': 'abcd123'
    },
    'Environment': {
        'Id': 'efgh456'
    },
    'ConfigurationProfile': {
        'Id': 'ijkl789',
        'Name': 'ConfigurationName'
    },
    'DeploymentNumber': 2,
    'Description': 'Deployment description',
    'ConfigurationVersion': '2'
}
```

응답 구조

다음 예는 사용자 지정 확장 프로그램의 요청에 대한 응답으로 Lambda 함수가 반환하는 내용을 보여 줍니다. AWS AppConfig

동기 이벤트 - 성공적인 응답

콘텐츠를 변환하려면 다음을 사용하세요.

```
"Content": "SomeBase64EncodedByteArray"
```

콘텐츠를 변환하고 싶지 않으면 아무것도 반환하지 않습니다.

비동기 이벤트 - 성공적인 응답

아무것도 반환하지 않습니다.

모든 오류 이벤트

```
{
        "Error": "BadRequestError",
        "Message": "There was malformed stuff in here",
        "Details": [{
            "Type": "Malformed",
            "Name": "S3 pointer",
            "Reason": "S3 bucket did not exist"
        }]
    }
```

사용자 지정 확장에 대한 권한 구성 AWS AppConfig

다음 절차를 사용하여 AWS Identity and Access Management (IAM) 서비스 역할 (또는 역할 수임) 을 만들고 구성합니다. AWS AppConfig 이 역할을 사용하여 Lambda 함수를 호출합니다.

IAM 서비스 역할을 생성하고 해당 역할을 맡도록 허용하는 방법 AWS AppConfig

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 탐색 창에서 역할(Roles)을 선택한 후 역할 생성(Create role)을 선택합니다.
3. 신뢰할 수 있는 엔터티 유형 선택에서 사용자 지정 신뢰 정책을 선택합니다.
4. 사용자 지정 신뢰 정책 필드에 다음의 JSON 정책을 붙여 넣습니다.

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

```
        "Service": "appconfig.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

다음을 선택합니다.

5. 권한 추가 페이지에서 정책 생성을 선택합니다. 정책 생성 페이지가 새 탭에서 열립니다.
6. JSON 탭을 선택한 후 다음과 같은 권한 정책을 편집기에 붙여 넣습니다. `lambda:InvokeFunction` 액션은 `PRE_*` 액션 포인트에 사용됩니다. `lambda:InvokeAsync` 액션은 `ON_*` 액션 포인트에 사용됩니다. *Lambda ARN*을 Lambda의 Amazon 리소스 이름(ARN)으로 바꿉니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
        "lambda:InvokeFunction",
        "lambda:InvokeAsync"
      ],
      "Resource": "Your Lambda ARN"
    }
  ]
}
```

7. 다음: 태그를 선택합니다.
8. 태그 추가 (선택 사항) 페이지에서 하나 이상의 키-값 페어를 추가한 후 다음: 검토를 선택합니다.
9. 정책 검토 페이지에 이름과 설명을 입력한 다음 정책 생성을 선택합니다.
10. 사용자 지정 신뢰 정책의 브라우저 탭에서 새로 고침 아이콘을 선택한 다음 방금 만든 권한 정책을 검색합니다.
11. 권한 정책의 확인란을 선택하고 다음을 선택합니다.
12. 이름, 검토, 생성 페이지에서 역할 이름 상자에 이름을 입력한 후 설명을 입력합니다.
13. 역할 생성을 선택합니다. 그러면 역할 페이지로 돌아갑니다. 배너에서 역할 보기를 선택합니다.
14. ARN을 복사합니다. 확장 생성 시 이 ARN을 지정합니다.

사용자 지정 AWS AppConfig 확장 생성

확장은 AWS AppConfig 워크플로우 중에 수행하는 하나 이상의 작업을 정의합니다. 예를 들어, AWS 작성된 `AWS AppConfig deployment events to Amazon SNS` 확장 프로그램에는 Amazon SNS 주제에 알림을 보내는 작업이 포함됩니다. 각 작업은 사용자와 상호 작용할 때 AWS AppConfig 또는 사용자를 대신하여 프로세스를 수행할 때 AWS AppConfig 호출됩니다. 이를 액션 포인트라고 합니다. AWS AppConfig 확장 프로그램은 다음과 같은 액션 포인트를 지원합니다.

- `PRE_CREATE_HOSTED_CONFIGURATION_VERSION`
- `PRE_START_DEPLOYMENT`
- `ON_DEPLOYMENT_START`
- `ON_DEPLOYMENT_STEP`
- `ON_DEPLOYMENT_BAKING`
- `ON_DEPLOYMENT_COMPLETE`
- `ON_DEPLOYMENT_ROLLED_BACK`

`PRE_*`액션 포인트에 구성된 확장 액션은 요청 유효성 검사 후 적용되지만 액션 포인트 이름에 해당하는 액티비티를 AWS AppConfig 수행하기 전에 적용됩니다. 이러한 액션은 요청과 동시에 처리됩니다. 요청이 두 개 이상 이루어진 경우 액션 간접 호출은 순차적으로 실행됩니다. 또한 `PRE_*` 액션 포인트는 구성 내용을 수신하고 변경할 수 있다는 점에 유의하십시오. `PRE_*` 액션 포인트는 오류에 대응하여 조치가 취해지는 것을 방지할 수도 있습니다.

`ON_*`작업 지점을 사용하여 확장 프로그램을 AWS AppConfig 워크플로와 병렬로 실행할 수도 있습니다. `ON_*`액션 포인트는 비동기적으로 호출됩니다. `ON_*`액션 포인트는 구성의 내용을 수신하지 않습니다. `ON_*` 액션 포인트 중에 확장에 오류가 발생하는 경우 서비스는 오류를 무시하고 워크플로우를 계속합니다.

다음 샘플 확장은 `PRE_CREATE_HOSTED_CONFIGURATION_VERSION` 액션 포인트를 호출하는 하나의 액션을 정의합니다. `Uri` 필드에서, 액션은 이 연습의 앞부분에서 생성한 `MyS3ConfigurationBackUpExtension` Lambda 함수의 Amazon 리소스 이름(ARN)을 지정합니다. 또한 이 작업은 이 안내의 앞부분에서 생성한 AWS Identity and Access Management (IAM) 역할 ARN을 수임하도록 지정합니다.

샘플 확장 AWS AppConfig

```
{
```

```
    "Name": "MySampleExtension",
    "Description": "A sample extension that backs up configurations to an S3 bucket.",
    "Actions": {
        "PRE_CREATE_HOSTED_CONFIGURATION_VERSION": [
            {
                "Name": "PreCreateHostedConfigVersionActionForS3Backup",
                "Uri": "arn:aws:lambda:aws-
region:111122223333:function:MyS3ConfigurationBackUpExtension",
                "RoleArn": "arn:aws:iam::111122223333:role/ExtensionsTestRole"
            }
        ]
    },
    "Parameters" : {
        "S3_BUCKET": {
            "Required": false
        }
    }
}
```

> **Note**
>
> 확장을 만들 때 요청 구문과 필드 설명을 보려면 AWS AppConfig API CreateExtensionReference의 주제를 참조하십시오.

확장을 만들려면 (콘솔)

1. https://console.aws.amazon.com/systems-manager/appconfig/ 에서 AWS Systems Manager 콘솔을 엽니다.
2. 탐색 창에서 AWS AppConfig를 선택합니다.
3. 확장 탭에서 확장 생성을 선택합니다.
4. 확장 이름에 고유한 이름을 입력합니다. 이 연습에서는 **MyS3ConfigurationBackUpExtension**을 입력합니다. 필요한 경우 설명을 입력합니다.
5. 액션 섹션에서 새 액션 추가를 선택합니다.
6. 액션 이름에 고유한 이름을 입력합니다. 이 연습에서는 **PreCreateHostedConfigVersionActionForS3Backup**을 입력합니다. 이 이름은 액션에서 사용하는 액션 포인트와 확장 용도를 설명합니다.
7. 액션 포인트 목록에서 PRE_CREATE_HOSTED_CONFIGURATION_VERSION을 선택합니다.

8. Uri의 경우, Lambda 함수를 선택한 다음 Lambda 함수 목록에서 함수를 선택합니다. 함수가 보이지 않는 경우 함수를 만든 AWS 리전 위치에 있는지 확인하세요.
9. IAM 역할의 경우, 이 연습의 앞부분에서 생성한 역할을 선택하십시오.
10. 확장 파라미터 (선택 사항) 섹션에서 새 파라미터 추가를 선택합니다.
11. 파라미터 이름에 이름을 입력합니다. 이 연습에서는 **S3_BUCKET**을 입력합니다.
12. 5~11단계를 반복하여 PRE_START_DEPLOYMENT 액션 포인트에 사용할 두 번째 액션을 생성합니다.
13. 확장 생성을 선택합니다.

AWS 작성한 알림 확장 프로그램 사용자 지정하기

Lambda 또는 확장을 생성하지 않아도 AWS 작성 알림 확장을 사용할 수 있습니다. 확장 연결을 생성한 다음 지원되는 액션 포인트 중 하나를 호출하는 작업을 수행하기만 하면 됩니다. 기본적으로 AWS 작성된 알림 확장은 다음 액션 포인트를 지원합니다.

- ON_DEPLOYMENT_START
- ON_DEPLOYMENT_COMPLETE
- ON_DEPLOYMENT_ROLLED_BACK

AWS AppConfig deployment events to Amazon SQS 확장 및 AWS AppConfig deployment events to Amazon SNS 확장의 사용자 지정 버전을 생성하는 경우 알림을 받을 액션 포인트를 지정할 수 있습니다.

> **Note**
>
> AWS AppConfig deployment events to EventBridge 확장은 PRE_* 액션 포인트를 지원하지 않습니다. 작성된 버전에 할당된 기본 액션 포인트 중 일부를 제거하려는 경우 사용자 지정 버전을 만들 수 있습니다. AWS

AWS 작성 알림 확장의 사용자 지정 버전을 생성하는 경우 Lambda 함수를 생성할 필요가 없습니다. 새 확장 버전의 Uri 필드에 Amazon 리소스 이름(ARN)을 지정하기만 하면 됩니다.

- 사용자 지정 EventBridge 알림 확장의 경우 필드에 EventBridge 기본 이벤트의 ARN을 입력합니다. Uri

- 사용자 지정 Amazon SNS 알림 확장의 경우 `Uri` 필드에 Amazon SNS 주제의 ARN을 입력합니다.
- 사용자 지정 Amazon SQS 알림 확장의 경우 `Uri` 필드에 Amazon SQS 메시지 대기열의 ARN을 입력합니다.

사용자 지정 AWS AppConfig 확장 프로그램의 확장 프로그램 연결 만들기

확장을 만들거나 AWS 제작된 확장을 구성하려면 특정 AWS AppConfig 리소스가 사용될 때 확장 프로그램을 호출하는 작업 지점을 정의합니다. 예를 들어, 특정 애플리케이션에 대한 구성 배포가 시작될 때마다 `AWS AppConfig deployment events to Amazon SNS` 확장을 실행하고 Amazon SNS 주제에 대한 알림을 수신하도록 선택할 수 있습니다. 특정 AWS AppConfig 리소스에 대한 확장 프로그램을 호출하는 작업 지점을 정의하는 것을 확장 연결이라고 합니다. 확장 연결은 확장 프로그램과 AWS AppConfig 리소스 (예: 응용 프로그램 또는 구성 프로필) 간의 지정된 관계입니다.

단일 AWS AppConfig 응용 프로그램에는 여러 환경 및 구성 프로필이 포함될 수 있습니다. 확장을 응용 프로그램이나 환경에 연결하는 경우 응용 프로그램 또는 환경 리소스와 관련된 모든 워크플로에 대해 확장을 AWS AppConfig 호출합니다 (해당하는 경우).

예를 들어, 라는 구성 프로필이 포함된 AWS AppConfig 응용 프로그램이 MobileApps 있다고 가정해 보겠습니다. AccessList 그리고 MobileApps 애플리케이션에 베타, 통합 및 프로덕션 환경이 포함되어 있다고 가정해 보겠습니다. AWS 작성한 Amazon SNS 알림 확장에 대한 확장 프로그램 연결을 생성하고 확장 프로그램을 애플리케이션에 연결합니다. MobileApps Amazon SNS 알림 확장은 애플리케이션 구성이 세 가지 환경 중 하나에 배포될 때마다 간접적으로 호출됩니다.

콘솔을 사용하여 AWS AppConfig 확장 연결을 생성하려면 다음 절차를 따르십시오. AWS AppConfig

확장 연결을 생성하려면(콘솔)

1. https://console.aws.amazon.com/systems-manager/appconfig/ 에서 AWS Systems Manager 콘솔을 엽니다.
2. 탐색 창에서 AWS AppConfig를 선택합니다.
3. 확장 탭에서 확장의 옵션 버튼을 선택한 다음 리소스에 추가를 선택합니다. 이 안내를 위해 MyS3를 선택하십시오. ConfigurationBackUpExtension
4. 확장 리소스 세부 정보 섹션의 리소스 유형에서 리소스 유형을 선택합니다. AWS AppConfig 선택한 리소스에 따라 다른 리소스를 AWS AppConfig 선택하라는 메시지가 표시됩니다. 이 연습에서는 애플리케이션을 선택합니다.
5. 목록에서 애플리케이션을 선택합니다.

6. 매개변수 섹션에서 S3_BUCKET이 키 필드에 나열되어 있는지 확인합니다. 값 필드에 Lambda 확장의 ARN을 붙여 넣습니다. 예를 들면 `arn:aws:lambda:aws-region:111122223333:function:MyS3ConfigurationBackUpExtension`입니다.
7. 리소스에 연결 만들기를 선택합니다.

사용자 지정 확장을 호출하는 작업 수행 AWS AppConfig

연결을 만든 후 `SourceUri`에 대한 `hosted`를 지정하는 새 구성 프로필을 만들어 `MyS3ConfigurationBackUpExtension` 확장을 간접적으로 호출할 수 있습니다. 새 구성을 만드는 워크플로의 일부로 `PRE_CREATE_HOSTED_CONFIGURATION_VERSION` 작업 AWS AppConfig 지점이 발생합니다. 이 액션 포인트가 발생하면 `MyS3ConfigurationBackUpExtension` 확장이 간접적으로 호출되어 새로 생성된 구성을 확장 연결 `Parameter` 섹션에 지정된 S3 버킷에 자동으로 백업합니다.

AWS AppConfig Atlassian Jira와의 확장 통합

AWS AppConfig 아틀라시안 지라와 통합됩니다. 통합을 통해 AWS AppConfig 지정된 기능 플래그를 변경할 때마다 Atlassian 콘솔에서 이슈를 생성하고 업데이트할 수 있습니다. AWS 계정 AWS 리전각 Jira 이슈에는 플래그 이름, 애플리케이션 ID, 구성 프로필 ID 및 플래그 값이 포함됩니다. 플래그 변경 사항을 업데이트, 저장 및 배포한 후 Jira는 변경 세부 정보와 함께 기존 이슈를 업데이트합니다. 자세한 내용은 에 대한 Atlassian Jira 확장 프로그램을 사용하여 작업하기 AWS AppConfig을(를) 참조하세요.

AWS AppConfig 코드 샘플

이 섹션에는 일반적인 AWS AppConfig 작업을 프로그래밍 방식으로 수행하기 위한 코드 샘플이 포함되어 있습니다. 테스트 환경에서 작업을 수행하려면 이러한 샘플을 Java, Python 및 JavaScriptSDK와 함께 사용하는 것이 좋습니다. 이 섹션에는 완료 후 테스트 환경을 정리하기 위한 코드 샘플이 포함되어 있습니다.

주제

- 호스팅된 구성 저장소에 저장된 자유 형식 구성 생성 또는 업데이트
- Secrets Manager에 저장된 시크릿에 대한 구성 프로파일 생성
- 구성 프로필 배포
- AWS AppConfig에이전트를 사용하여 자유 형식 구성 프로필 읽기
- AWS AppConfig에이전트를 사용하여 특정 기능 플래그 읽기
- GetLatestConfig API 작업을 사용하여 자유 형식 구성 프로필 읽기
- 환경 정리하기

호스팅된 구성 저장소에 저장된 자유 형식 구성 생성 또는 업데이트

다음 각 샘플에는 코드에 의해 수행된 작업에 대한 설명이 포함되어 있습니다. 이 섹션의 샘플은 다음 API를 호출합니다.

- CreateApplication
- CreateConfigurationProfile
- CreateHostedConfigurationVersion

Java

```
public CreateHostedConfigurationVersionResponse createHostedConfigVersion() {
        AppConfigClient appconfig = AppConfigClient.create();

        // Create an application
        CreateApplicationResponse app = appconfig.createApplication(req ->
 req.name("MyDemoApp"));

        // Create a hosted, freeform configuration profile
```

```
        CreateConfigurationProfileResponse configProfile =
 appconfig.createConfigurationProfile(req -> req
            .applicationId(app.id())
            .name("MyConfigProfile")
            .locationUri("hosted")
            .type("AWS.Freeform"));

        // Create a hosted configuration version
        CreateHostedConfigurationVersionResponse hcv =
 appconfig.createHostedConfigurationVersion(req -> req
            .applicationId(app.id())
            .configurationProfileId(configProfile.id())
            .contentType("text/plain; charset=utf-8")
            .content(SdkBytes.fromUtf8String("my config data")));

        return hcv;
    }
```

Python

```
import boto3

appconfig = boto3.client('appconfig')

# create an application
application = appconfig.create_application(Name='MyDemoApp')

# create a hosted, freeform configuration profile
config_profile = appconfig.create_configuration_profile(
    ApplicationId=application['Id'],
    Name='MyConfigProfile',
    LocationUri='hosted',
    Type='AWS.Freeform')

# create a hosted configuration version
hcv = appconfig.create_hosted_configuration_version(
    ApplicationId=application['Id'],
    ConfigurationProfileId=config_profile['Id'],
    Content=b'my config data',
    ContentType='text/plain')
```

JavaScript

```
import {
  AppConfigClient,
  CreateApplicationCommand,
  CreateConfigurationProfileCommand,
  CreateHostedConfigurationVersionCommand,
} from "@aws-sdk/client-appconfig";

const appconfig = new AppConfigClient();

// create an application
const application = await appconfig.send(
  new CreateApplicationCommand({ Name: "MyDemoApp" })
);

// create a hosted, freeform configuration profile
const profile = await appconfig.send(
  new CreateConfigurationProfileCommand({
    ApplicationId: application.Id,
    Name: "MyConfigProfile",
    LocationUri: "hosted",
    Type: "AWS.Freeform",
  })
);

// create a hosted configuration version
await appconfig.send(
  new CreateHostedConfigurationVersionCommand({
    ApplicationId: application.Id,
    ConfigurationProfileId: profile.Id,
    ContentType: "text/plain",
    Content: "my config data",
  })
);
```

Secrets Manager에 저장된 시크릿에 대한 구성 프로파일 생성

다음 각 샘플에는 코드에 의해 수행된 작업에 대한 설명이 포함되어 있습니다. 이 섹션의 샘플은 다음 API를 호출합니다.

- CreateApplication

- CreateConfigurationProfile

Java

```
private void createSecretsManagerConfigProfile() {
        AppConfigClient appconfig = AppConfigClient.create();

        // Create an application
        CreateApplicationResponse app = appconfig.createApplication(req ->
 req.name("MyDemoApp"));

        // Create a configuration profile for Secrets Manager Secret
        CreateConfigurationProfileResponse configProfile =
 appconfig.createConfigurationProfile(req -> req
            .applicationId(app.id())
            .name("MyConfigProfile")
            .locationUri("secretsmanager://MySecret")
            .retrievalRoleArn("arn:aws:iam::000000000000:role/
RoleTrustedByAppConfigThatCanRetrieveSecret")
            .type("AWS.Freeform"));
    }
```

Python

```
import boto3

appconfig = boto3.client('appconfig')

# create an application
application = appconfig.create_application(Name='MyDemoApp')

# create a configuration profile for Secrets Manager Secret
config_profile = appconfig.create_configuration_profile(
    ApplicationId=application['Id'],
    Name='MyConfigProfile',
    LocationUri='secretsmanager://MySecret',
    RetrievalRoleArn='arn:aws:iam::000000000000:role/
RoleTrustedByAppConfigThatCanRetrieveSecret',
    Type='AWS.Freeform')
```

JavaScript

```
import {
  AppConfigClient,
  CreateConfigurationProfileCommand,
} from "@aws-sdk/client-appconfig";

const appconfig = new AppConfigClient();

// create an application
const application = await appconfig.send(
  new CreateApplicationCommand({ Name: "MyDemoApp" })
);

// create a configuration profile for Secrets Manager Secret
await appconfig.send(
  new CreateConfigurationProfileCommand({
    ApplicationId: application.Id,
    Name: "MyConfigProfile",
    LocationUri: "secretsmanager://MySecret",
    RetrievalRoleArn: "arn:aws:iam::000000000000:role/
RoleTrustedByAppConfigThatCanRetrieveSecret",
    Type: "AWS.Freeform",
  })
);
```

구성 프로필 배포

다음 각 샘플에는 코드에서 수행한 작업에 대한 설명이 포함되어 있습니다. 이 섹션의 샘플은 다음 API를 호출합니다.

- CreateApplication
- CreateConfigurationProfile
- CreateHostedConfigurationVersion
- CreateEnvironment
- StartDeployment
- GetDeployment

Java

```
private void createDeployment() throws InterruptedException {
        AppConfigClient appconfig = AppConfigClient.create();

        // Create an application
        CreateApplicationResponse app = appconfig.createApplication(req ->
 req.name("MyDemoApp"));

        // Create a hosted, freeform configuration profile
        CreateConfigurationProfileResponse configProfile =
 appconfig.createConfigurationProfile(req -> req
            .applicationId(app.id())
            .name("MyConfigProfile")
            .locationUri("hosted")
            .type("AWS.Freeform"));

        // Create a hosted configuration version
        CreateHostedConfigurationVersionResponse hcv =
 appconfig.createHostedConfigurationVersion(req -> req
            .applicationId(app.id())
            .configurationProfileId(configProfile.id())
            .contentType("text/plain; charset=utf-8")
            .content(SdkBytes.fromUtf8String("my config data")));


        // Create an environment
        CreateEnvironmentResponse env = appconfig.createEnvironment(req -> req
            .applicationId(app.id())
            .name("Beta")
            // If you have CloudWatch alarms that monitor the health of your
 service, you can add them here and they
            // will trigger a rollback if they fire during an appconfig deployment
            //.monitors(Monitor.builder().alarmArn("arn:aws:cloudwatch:us-
east-1:520900602629:alarm:MyAlarm")
            //
  .alarmRoleArn("arn:aws:iam::520900602629:role/MyAppConfigAlarmRole").build())
        );

        // Start a deployment
        StartDeploymentResponse deploymentResponse = appconfig.startDeployment(req -
> req
            .applicationId(app.id())
            .configurationProfileId(configProfile.id())
```

```
            .environmentId(env.id())
            .configurationVersion(hcv.versionNumber().toString())
            .deploymentStrategyId("AppConfig.Linear50PercentEvery30Seconds")
        );

        // Wait for deployment to complete
        List<DeploymentState> nonFinalDeploymentStates = Arrays.asList(
            DeploymentState.DEPLOYING,
            DeploymentState.BAKING,
            DeploymentState.ROLLING_BACK,
            DeploymentState.VALIDATING);
        GetDeploymentRequest getDeploymentRequest =
 GetDeploymentRequest.builder().applicationId(app.id())

 .environmentId(env.id())

 .deploymentNumber(deploymentResponse.deploymentNumber()).build();
        GetDeploymentResponse deployment =
 appconfig.getDeployment(getDeploymentRequest);
        while (nonFinalDeploymentStates.contains(deployment.state())) {
            System.out.println("Waiting for deployment to complete: " + deployment);
            Thread.sleep(1000L);
            deployment = appconfig.getDeployment(getDeploymentRequest);
        }

        System.out.println("Deployment complete: " + deployment);
    }
```

Python

```
import boto3

appconfig = boto3.client('appconfig')

# create an application
application = appconfig.create_application(Name='MyDemoApp')

# create an environment
environment = appconfig.create_environment(
    ApplicationId=application['Id'],
    Name='MyEnvironment')

# create a configuration profile
```

```
config_profile = appconfig.create_configuration_profile(
    ApplicationId=application['Id'],
    Name='MyConfigProfile',
    LocationUri='hosted',
    Type='AWS.Freeform')

# create a hosted configuration version
hcv = appconfig.create_hosted_configuration_version(
    ApplicationId=application['Id'],
    ConfigurationProfileId=config_profile['Id'],
    Content=b'my config data',
    ContentType='text/plain')

# start a deployment
deployment = appconfig.start_deployment(
    ApplicationId=application['Id'],
    EnvironmentId=environment['Id'],
    ConfigurationProfileId=config_profile['Id'],
    ConfigurationVersion=str(hcv['VersionNumber']),
    DeploymentStrategyId='AppConfig.Linear20PercentEvery6Minutes')
```

JavaScript

```
import {
  AppConfigClient,
  CreateApplicationCommand,
  CreateEnvironmentCommand,
  CreateConfigurationProfileCommand,
  CreateHostedConfigurationVersionCommand,
  StartDeploymentCommand,
} from "@aws-sdk/client-appconfig";

const appconfig = new AppConfigClient();

// create an application
const application = await appconfig.send(
  new CreateApplicationCommand({ Name: "MyDemoApp" })
);

// create an environment
const environment = await appconfig.send(
  new CreateEnvironmentCommand({
    ApplicationId: application.Id,
```

```
    Name: "MyEnvironment",
  })
);

// create a configuration profile
const config_profile = await appconfig.send(
  new CreateConfigurationProfileCommand({
    ApplicationId: application.Id,
    Name: "MyConfigProfile",
    LocationUri: "hosted",
    Type: "AWS.Freeform",
  })
);

// create a hosted configuration version
const hcv = await appconfig.send(
  new CreateHostedConfigurationVersionCommand({
    ApplicationId: application.Id,
    ConfigurationProfileId: config_profile.Id,
    Content: "my config data",
    ContentType: "text/plain",
  })
);

// start a deployment
await appconfig.send(
  new StartDeploymentCommand({
    ApplicationId: application.Id,
    EnvironmentId: environment.Id,
    ConfigurationProfileId: config_profile.Id,
    ConfigurationVersion: hcv.VersionNumber.toString(),
    DeploymentStrategyId: "AppConfig.Linear20PercentEvery6Minutes",
  })
);
```

AWS AppConfig에이전트를 사용하여 자유 형식 구성 프로필 읽기

다음 각 샘플에는 코드에 의해 수행된 작업에 대한 설명이 포함되어 있습니다.

Java

```
public void retrieveConfigFromAgent() throws Exception {
```

```
        /*
        In this sample, we will retrieve configuration data from the AWS AppConfig
 Agent.
        The agent is a sidecar process that handles retrieving configuration data
 from AppConfig
        for you in a way that implements best practices like configuration caching.

        For more information about the agent, see Simplified retrieval methods
        */

        // The agent runs a local HTTP server that serves configuration data
        // Make a GET request to the agent's local server to retrieve the
 configuration data
        URL url = new URL("http://localhost:2772/applications/MyDemoApp/
environments/Beta/configurations/MyConfigProfile");
        HttpURLConnection con = (HttpURLConnection) url.openConnection();
        con.setRequestMethod("GET");
        StringBuilder content;
        try (BufferedReader in = new BufferedReader(new
 InputStreamReader(con.getInputStream()))) {
            content = new StringBuilder();
            int ch;
            while ((ch = in.read()) != -1) {
                content.append((char) ch);
            }
        }
        con.disconnect();
        System.out.println("Configuration from agent via HTTP: " + content);
    }
```

Python

```
# in this sample, we will retrieve configuration data from the AWS AppConfig Agent.
# the agent is a sidecar process that handles retrieving configuration data from AWS
 AppConfig
# for you in a way that implements best practices like configuration caching.
#
# for more information about the agent, see
# Simplified retrieval methods
#

import requests
```

```
application_name = 'MyDemoApp'
environment_name = 'MyEnvironment'
config_profile_name = 'MyConfigProfile'

# the agent runs a local HTTP server that serves configuration data
# make a GET request to the agent's local server to retrieve the configuration data
response = requests.get(f"http://localhost:2772/applications/{application_name}/
environments/{environment_name}/configurations/{config_profile_name}")
config = response.content
```

JavaScript

```
// in this sample, we will retrieve configuration data from the AWS AppConfig Agent.
// the agent is a sidecar process that handles retrieving configuration data from
 AppConfig
// for you in a way that implements best practices like configuration caching.

// for more information about the agent, see
// Simplified retrieval methods

const application_name = "MyDemoApp";
const environment_name = "MyEnvironment";
const config_profile_name = "MyConfigProfile";

// the agent runs a local HTTP server that serves configuration data
// make a GET request to the agent's local server to retrieve the configuration data
const url = `http://localhost:2772/applications/${application_name}/environments/
${environment_name}/configurations/${config_profile_name}`;
const response = await fetch(url);
const config = await response.text(); // (use `await response.json()` if your config
 is json)
```

AWS AppConfig에이전트를 사용하여 특정 기능 플래그 읽기

다음 각 샘플에는 코드에서 수행한 작업에 대한 설명이 포함되어 있습니다.

Java

```
public void retrieveSingleFlagFromAgent() throws Exception {
        /*
          You can retrieve a single flag's data from the agent by providing the
 "flag" query string parameter.
```

```
          Note: the configuration's type must be AWS.AppConfig.FeatureFlags
        */

        URL url = new URL("http://localhost:2772/applications/MyDemoApp/
environments/Beta/configurations/MyFlagsProfile?flag=myFlagKey");
        HttpURLConnection con = (HttpURLConnection) url.openConnection();
        con.setRequestMethod("GET");
        StringBuilder content;
        try (BufferedReader in = new BufferedReader(new
 InputStreamReader(con.getInputStream()))) {
            content = new StringBuilder();
            int ch;
            while ((ch = in.read()) != -1) {
                content.append((char) ch);
            }
        }
        con.disconnect();
        System.out.println("MyFlagName from agent: " + content);
    }
```

Python

```
import requests

application_name = 'MyDemoApp'
environment_name = 'MyEnvironment'
config_profile_name = 'MyConfigProfile'
flag_key = 'MyFlag'

# retrieve a single flag's data by providing the "flag" query string parameter
# note: the configuration's type must be AWS.AppConfig.FeatureFlags
response = requests.get(f"http://localhost:2772/applications/{application_name}/
environments/{environment_name}/configurations/{config_profile_name}?
flag={flag_key}")
config = response.content
```

JavaScript

```
const application_name = "MyDemoApp";
const environment_name = "MyEnvironment";
const config_profile_name = "MyConfigProfile";
const flag_name = "MyFlag";
```

```
// retrieve a single flag's data by providing the "flag" query string parameter
// note: the configuration's type must be AWS.AppConfig.FeatureFlags
const url = `http://localhost:2772/applications/${application_name}/environments/
${environment_name}/configurations/${config_profile_name}?flag=${flag_name}`;
const response = await fetch(url);
const flag = await response.json(); // { "enabled": true/false }
```

GetLatestConfig API 작업을 사용하여 자유 형식 구성 프로필 읽기

다음 각 샘플에는 코드에서 수행한 작업에 대한 설명이 포함되어 있습니다. 이 섹션의 샘플은 다음 API를 호출합니다.

- GetLatestConfiguration
- StartConfigurationSession

Java

```
public void retrieveConfigFromApi() {
        /*
        The example below uses two AppConfigData APIs: StartConfigurationSession and
 GetLatestConfiguration.
        For more information on these APIs, see AWS AppConfig Data         */
        AppConfigDataClient appConfigData = AppConfigDataClient.create();

        /*
        Start a new configuration session using the StartConfigurationSession API.
 This operation does not return configuration data.
        Rather, it returns an initial configuration token that should be passed to
 GetLatestConfiguration.
        IMPORTANT: This operation should only be performed once (per configuration),
 prior to the first GetLatestConfiguration
        call you preform. Each GetLatestConfiguration will return a new
 configuration token that you should then use in the
        next GetLatestConfiguration call.
        */
        StartConfigurationSessionResponse session =
            appConfigData.startConfigurationSession(req -> req
                .applicationIdentifier("MyDemoApp")
                .configurationProfileIdentifier("MyConfigProfile")
                .environmentIdentifier("Beta"));
```

```
        /*
        Retrieve configuration data using the GetLatestConfiguration API. The first
time you call this API your configuration
        data will be returned. You should cache that data (and the configuration
token) and update that cache asynchronously
        by regularly polling the GetLatestConfiguration API in a background thread.
If you already have the latest configuration
        data, subsequent GetLatestConfiguration calls will return an empty response.
If you then deploy updated configuration
        data the next time you call GetLatestConfiguration it will return that
updated data.

        You can also avoid all the complexity around writing this code yourself by
leveraging our agent instead.
        For more information about the agent, see Simplified retrieval methods
        */

        // The first getLatestConfiguration call uses the token from
StartConfigurationSession
        String configurationToken = session.initialConfigurationToken();
        GetLatestConfigurationResponse configuration =

appConfigData.getLatestConfiguration(GetLatestConfigurationRequest.builder().configurationT

        System.out.println("Configuration retrieved via API: " +
configuration.configuration().asUtf8String());

        // You'll want to hold on to the token in the getLatestConfiguration
response because you'll need to use it
        // the next time you call
        configurationToken = configuration.nextPollConfigurationToken();
        configuration =

appConfigData.getLatestConfiguration(GetLatestConfigurationRequest.builder().configurationT

        // Try creating a new deployment at this point to see how the output below
changes.
        if (configuration.configuration().asByteArray().length != 0) {
            System.out.println("Configuration contents have changed
since the last GetLatestConfiguration call, new contents = " +
configuration.configuration().asUtf8String());
        } else {
            System.out.println("GetLatestConfiguration returned an empty response
because we already have the latest configuration");
```

```
        }
    }
```

Python

```
# the example below uses two AppConfigData APIs: StartConfigurationSession and
 GetLatestConfiguration.
#
# for more information on these APIs, see
# AWS AppConfig Data
#

import boto3

application_name = 'MyDemoApp'
environment_name = 'MyEnvironment'
config_profile_name = 'MyConfigProfile'

appconfigdata = boto3.client('appconfigdata')

# start a new configuration session.
# this operation does not return configuration data.
# rather, it returns an initial configuration token that should be passed to
 GetLatestConfiguration.
#
# note: this operation should only be performed once (per configuration).
#   all subsequent calls to AppConfigData should be via GetLatestConfiguration.
scs = appconfigdata.start_configuration_session(
    ApplicationIdentifier=application_name,
    EnvironmentIdentifier=environment_name,
    ConfigurationProfileIdentifier=config_profile_name)
initial_token = scs['InitialConfigurationToken']

# retrieve configuration data from the session.
# this operation returns your configuration data.
# each invocation of this operation returns a unique token that should be passed to
 the subsequent invocation.
#
# note: this operation does not always return configuration data after the first
 invocation.
#   data is only returned if the configuration has changed within AWS AppConfig
 (i.e. a deployment occurred).
```

```
#   therefore, you should cache the data returned by this call so that you can use
 it later.
glc = appconfigdata.get_latest_configuration(ConfigurationToken=initial_token)
config = glc['Configuration'].read()
```

JavaScript

```
// the example below uses two AppConfigData APIs: StartConfigurationSession and
 GetLatestConfiguration.

// for more information on these APIs, see
// AWS AppConfig Data

import {
  AppConfigDataClient,
  GetLatestConfigurationCommand,
  StartConfigurationSessionCommand,
} from "@aws-sdk/client-appconfigdata";

const appconfigdata = new AppConfigDataClient();

const application_name = "MyDemoApp";
const environment_name = "MyEnvironment";
const config_profile_name = "MyConfigProfile";

// start a new configuration session.
// this operation does not return configuration data.
// rather, it returns an initial configuration token that should be passed to
 GetLatestConfiguration.
//
// note: this operation should only be performed once (per configuration).
//   all subsequent calls to AppConfigData should be via GetLatestConfiguration.
const scs = await appconfigdata.send(
  new StartConfigurationSessionCommand({
    ApplicationIdentifier: application_name,
    EnvironmentIdentifier: environment_name,
    ConfigurationProfileIdentifier: config_profile_name,
  })
);
const { InitialConfigurationToken } = scs;

// retrieve configuration data from the session.
// this operation returns your configuration data.
```

```
// each invocation of this operation returns a unique token that should be passed to
 the subsequent invocation.
//
// note: this operation does not always return configuration data after the first
 invocation.
//   data is only returned if the configuration has changed within AWS AppConfig
 (i.e. a deployment occurred).
//   therefore, you should cache the data returned by this call so that you can use
 it later.
const glc = await appconfigdata.send(
  new GetLatestConfigurationCommand({
    ConfigurationToken: InitialConfigurationToken,
  })
);
const config = glc.Configuration.transformToString();
```

환경 정리하기

이 섹션의 코드 샘플 중 하나 이상을 실행한 경우 다음 샘플 중 하나를 사용하여 해당 코드 샘플로 생성된 AWS AppConfig 리소스를 찾아 삭제하는 것이 좋습니다. 이 섹션의 샘플은 다음 API를 호출합니다.

- ListApplications
- DeleteApplication
- ListEnvironments
- DeleteEnvironments
- ListConfigurationProfiles
- DeleteConfigurationProfile
- ListHostedConfigurationVersions
- DeleteHostedConfigurationVersion

Java

```
/*
    This sample provides cleanup code that deletes all the AWS AppConfig resources
 created in the samples above.

    WARNING: this code will permanently delete the given application and all of its
 sub-resources, including
```

```
   configuration profiles, hosted configuration versions, and environments. DO NOT
run this code against
   an application that you may need in the future.
   */

   public void cleanUpDemoResources() {
       AppConfigClient appconfig = AppConfigClient.create();

       // The name of the application to delete
       // IMPORTANT: verify this name corresponds to the application you wish to
delete
       String applicationToDelete = "MyDemoApp";


appconfig.listApplicationsPaginator(ListApplicationsRequest.builder().build()).items().forE
-> {
           if (app.name().equals(applicationToDelete)) {
               System.out.println("Deleting App: " + app);
               appconfig.listConfigurationProfilesPaginator(req ->
req.applicationId(app.id())).items().forEach(cp -> {
                   System.out.println("Deleting Profile: " + cp);
                   appconfig
                       .listHostedConfigurationVersionsPaginator(req -> req
                           .applicationId(app.id())
                           .configurationProfileId(cp.id()))
                       .items()
                       .forEach(hcv -> {
                           System.out.println("Deleting HCV: " + hcv);
                           appconfig.deleteHostedConfigurationVersion(req -> req
                               .applicationId(app.id())
                               .configurationProfileId(cp.id())
                               .versionNumber(hcv.versionNumber()));
                       });
                   appconfig.deleteConfigurationProfile(req -> req
                       .applicationId(app.id())
                       .configurationProfileId(cp.id()));
               });

               appconfig.listEnvironmentsPaginator(req-
>req.applicationId(app.id())).items().forEach(env -> {
                   System.out.println("Deleting Environment: " + env);
                   appconfig.deleteEnvironment(req-
>req.applicationId(app.id()).environmentId(env.id()));
               });
```

```
                appconfig.deleteApplication(req -> req.applicationId(app.id()));
            }
        });
    }
```

Python

```
# this sample provides cleanup code that deletes all the AWS AppConfig resources
 created in the samples above.
#
# WARNING: this code will permanently delete the given application and all of its
 sub-resources, including
#   configuration profiles, hosted configuration versions, and environments. DO NOT
 run this code against
#   an application that you may need in the future.
#

import boto3

# the name of the application to delete
# IMPORTANT: verify this name corresponds to the application you wish to delete
application_name = 'MyDemoApp'

# create and iterate over a list paginator such that we end up with a list of pages,
 which are themselves lists of applications
# e.g. [ [{'Name':'MyApp1',...},{'Name':'MyApp2',...}], [{'Name':'MyApp3',...}] ]
list_of_app_lists = [page['Items'] for page in
 appconfig.get_paginator('list_applications').paginate()]
# retrieve the target application from the list of lists
application = [app for apps in list_of_app_lists for app in apps if app['Name'] ==
 application_name][0]
print(f"deleting application {application['Name']} (id={application['Id']})")

# delete all configuration profiles
list_of_config_lists = [page['Items'] for page in
 appconfig.get_paginator('list_configuration_profiles').paginate(ApplicationId=application['
for config_profile in [config for configs in list_of_config_lists for config in
 configs]:
    print(f"\tdeleting configuration profile {config_profile['Name']}
 (Id={config_profile['Id']})")

    # delete all hosted configuration versions
```

```
    list_of_hcv_lists = [page['Items'] for page in
 appconfig.get_paginator('list_hosted_configuration_versions').paginate(ApplicationId=applic
 ConfigurationProfileId=config_profile['Id'])]
    for hcv in [hcv for hcvs in list_of_hcv_lists for hcv in hcvs]:

 appconfig.delete_hosted_configuration_version(ApplicationId=application['Id'],
 ConfigurationProfileId=config_profile['Id'], VersionNumber=hcv['VersionNumber'])
        print(f"\t\tdeleted hosted configuration version {hcv['VersionNumber']}")

    # delete the config profile itself
    appconfig.delete_configuration_profile(ApplicationId=application['Id'],
 ConfigurationProfileId=config_profile['Id'])
    print(f"\tdeleted configuration profile {config_profile['Name']}
 (Id={config_profile['Id']})")

# delete all environments
list_of_env_lists = [page['Items'] for page in
 appconfig.get_paginator('list_environments').paginate(ApplicationId=application['Id'])]
for environment in [env for envs in list_of_env_lists for env in envs]:
    appconfig.delete_environment(ApplicationId=application['Id'],
 EnvironmentId=environment['Id'])
    print(f"\tdeleted environment {environment['Name']} (Id={environment['Id']})")

# delete the application itself
appconfig.delete_application(ApplicationId=application['Id'])
print(f"deleted application {application['Name']} (id={application['Id']})")
```

JavaScript

```
// this sample provides cleanup code that deletes all the AWS AppConfig resources
 created in the samples above.

// WARNING: this code will permanently delete the given application and all of its
 sub-resources, including
//   configuration profiles, hosted configuration versions, and environments. DO NOT
 run this code against
//   an application that you may need in the future.

import {
  AppConfigClient,
  paginateListApplications,
  DeleteApplicationCommand,
  paginateListConfigurationProfiles,
```

```
  DeleteConfigurationProfileCommand,
  paginateListHostedConfigurationVersions,
  DeleteHostedConfigurationVersionCommand,
  paginateListEnvironments,
  DeleteEnvironmentCommand,
} from "@aws-sdk/client-appconfig";

const client = new AppConfigClient();

// the name of the application to delete
// IMPORTANT: verify this name corresponds to the application you wish to delete
const application_name = "MyDemoApp";

// iterate over all applications, deleting ones that have the name defined above
for await (const app_page of paginateListApplications({ client }, {})) {
  for (const application of app_page.Items) {

    // skip applications that dont have the name thats set
    if (application.Name !== application_name) continue;

    console.log( `deleting application ${application.Name} (id=${application.Id})`);

    // delete all configuration profiles
    for await (const config_page of paginateListConfigurationProfiles({ client },
 { ApplicationId: application.Id })) {
      for (const config_profile of config_page.Items) {
        console.log(`\tdeleting configuration profile ${config_profile.Name} (Id=
${config_profile.Id})`);

        // delete all hosted configuration versions
        for await (const hosted_page of
 paginateListHostedConfigurationVersions({ client },
          { ApplicationId: application.Id, ConfigurationProfileId:
 config_profile.Id }
        )) {
          for (const hosted_config_version of hosted_page.Items) {
            await client.send(
              new DeleteHostedConfigurationVersionCommand({
                ApplicationId: application.Id,
                ConfigurationProfileId: config_profile.Id,
                VersionNumber: hosted_config_version.VersionNumber,
              })
            );
```

```
            console.log(`\t\tdeleted hosted configuration version
 ${hosted_config_version.VersionNumber}`);
          }
        }

        // delete the config profile itself
        await client.send(
          new DeleteConfigurationProfileCommand({
            ApplicationId: application.Id,
            ConfigurationProfileId: config_profile.Id,
          })
        );
        console.log(`\tdeleted configuration profile ${config_profile.Name} (Id=
${config_profile.Id})`)
      }

      // delete all environments
      for await (const env_page of paginateListEnvironments({ client },
 { ApplicationId: application.Id })) {
        for (const environment of env_page.Items) {
          await client.send(
            new DeleteEnvironmentCommand({
              ApplicationId: application.Id,
              EnvironmentId: environment.Id,
            })
          );
          console.log(`\tdeleted environment ${environment.Name} (Id=
${environment.Id})`)
        }
      }
    }

    // delete the application itself
    await client.send(
      new DeleteApplicationCommand({ ApplicationId: application.Id })
    );
    console.log(`deleted application ${application.Name} (id=${application.Id})`)
  }
}
```

AWS AppConfig의 보안

AWS에서는 클라우드 보안을 가장 중요하게 생각합니다. AWS 고객은 보안에 가장 보안에 민감한 조직의 요구 사항에 부합하도록 구축된 데이터 센터 및 네트워크 아키텍처의 혜택을 누릴 수 있습니다.

보안은 AWS와 사용자의 공동 책임입니다. 공동 책임 모델은 이 사항을 클라우드의 보안 및 클라우드 내 보안으로 설명합니다.

- 클라우드의 보안 - AWS는 AWS 클라우드에서 AWS 서비스를 실행하는 인프라를 보호합니다. AWS는 또한 안전하게 사용할 수 있는 서비스를 제공합니다. 서드 파티 감사자는 AWS 규정 준수 프로그램의 일환으로 보안 효과를 정기적으로 테스트하고 검증합니다. AWS Systems Manager에 적용되는 규정 준수 프로그램에 대한 자세한 내용은 규정 준수 프로그램의 범위에 속하는 AWS서비스 를 참조하십시오.
- 클라우드의 보안 – 귀하의 책임은 귀하가 사용하는 AWS 서비스에 의해 결정됩니다. 또한 귀하는 귀사의 데이터의 민감도, 귀사의 요구 사항, 관련 법률 및 규정을 비롯한 기타 요소에 대해서도 책임이 있습니다.

AWS AppConfig는 AWS Systems Manager의 기능입니다. AWS AppConfig 사용 시 책임 분담 모델을 적용하는 방법을 이해하려면 AWS Systems Manager 보안을 참조하십시오. 이 섹션에서는 AWS AppConfig의 보안 및 규정 준수 목표를 충족하도록 Systems Manager를 구성하는 방법을 설명합니다.

최소 권한 액세스 구현

보안 모범 사례로서, 특정 조건에서 특정 리소스에 대한 특정 작업을 수행하는 데 필요한 최소 권한을 ID에 부여하십시오. AWS AppConfig 에이전트는 에이전트가 인스턴스 또는 컨테이너의 파일 시스템에 액세스할 수 있도록 하는 두 가지 기능인 백업과 디스크 쓰기를 제공합니다. 이러한 기능을 활성화하는 경우 AWS AppConfig 에이전트만 파일 시스템의 지정된 구성 파일에 쓸 수 있는 권한을 가지고 있는지 확인하십시오. 또한 이러한 구성 파일을 읽는 데 필요한 프로세스만 읽을 수 있는지 확인하십시오. 최소 권한 액세스를 구현하는 것이 오류 또는 악의적인 의도로 인해 발생할 수 있는 보안 위험과 영향을 최소화할 수 있는 근본적인 방법입니다.

최소 권한 액세스 구현에 대한 자세한 내용은 AWS Well-Architected Tool사용 설명서의 SEC03-BP02 최소 권한 액세스 허용을 참조하십시오. 이 섹션에 언급된 AWS AppConfig 에이전트 기능에 대한 자세한 내용은 을 참조하십시오. 추가 검색 기능

AWS AppConfig의 저장된 데이터 암호화

AWS AppConfig는 AWS 소유 키를 사용하여 저장된 고객 데이터를 보호하기 위해 기본적으로 암호화를 제공합니다.

AWS 소유 키 — AWS AppConfig는 기본적으로 이러한 키를 사용하여 서비스에서 배포하고 AWS AppConfig 데이터 저장소에 호스팅되는 데이터를 자동으로 암호화합니다. 사용자는 AWS 소유 키를 확인, 관리 또는 사용하거나 사용을 감사할 수 없습니다. 하지만 데이터를 암호화하는 키를 보호하기 위해 어떤 액션을 수행하거나 어떤 프로그램을 변경할 필요가 없습니다. 자세한 내용은 AWS Key Management Service 개발자 안내서의 AWS 소유 키을 참조하세요.

이 암호화 계층을 비활성화하거나 다른 암호화 유형을 선택할 수는 없지만 AWS AppConfig 데이터 저장소에 호스팅된 구성 데이터를 저장하고 구성 데이터를 배포할 때 사용할 고객 관리 키를 지정할 수 있습니다.

고객 관리 키 — AWS AppConfig는 직접 생성하고 소유하고 관리하는 대칭형 고객 관리 키를 지원하여 기존 AWS 소유 키에 두 번째 암호화 계층을 추가할 수 있습니다. 이 암호화 계층을 완전히 제어할 수 있으므로 다음과 같은 작업을 수행할 수 있습니다.

- 키 정책 및 권한 부여 수립 및 유지
- IAM 정책 수립 및 유지
- 키 정책 활성화 및 비활성화
- 키 암호화 자료 교체
- 태그 추가
- 키 별칭 생성
- 키 삭제 일정 수립

자세한 내용은 AWS Key Management Service 개발자 안내서의 고객 관리 키를 참조하세요.

AWS AppConfig는 고객 관리형 키를 지원합니다

AWS AppConfig는 구성 데이터에 대한 고객 관리 키 암호화를 지원합니다. AWS AppConfig 호스팅된 데이터 저장소에 저장된 구성 버전의 경우 고객은 해당 구성 프로필에 `KmsKeyIdentifier`를 설정할 수 있습니다. `CreateHostedConfigurationVersion` API 작업을 사용하여 새 버전의 구성 데이터를 만들 때마다 AWS AppConfig는 저장하기 전에 데이터 암호화를 위해 `KmsKeyIdentifier`에서 AWS KMS 데이터 키를 생성합니다. 나중에 `GetHostedConfigurationVersion` 또는

StartDeployment API 작업 중에 데이터에 액세스하면 AWS AppConfig는 생성된 데이터 키에 대한 정보를 사용하여 구성 데이터를 복호화합니다.

AWS AppConfig는 또한 배포된 구성 데이터에 대한 고객 관리형 키 암호화에 대한 지원도 제공합니다. 구성 데이터를 암호화하기 위해 고객은 KmsKeyIdentifier를 배포에 제공할 수 있습니다. AWS AppConfig는 이 KmsKeyIdentifier로 AWS KMS 데이터 키를 생성하여 StartDeployment API 작업에 대한 데이터를 암호화합니다.

AWS AppConfig 암호화 액세스

고객 관리형 키를 생성할 때는 다음 키 정책을 사용하여 키를 사용할 수 있는지 확인하십시오.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
        "Sid": "Allow use of the key",
        "Effect": "Allow",
        "Principal": {
            "AWS": "arn:aws:iam::account_ID:role/role_name"
        },
        "Action": [
            "kms:Decrypt",
            "kms:GenerateDataKey"
            ],
        "Resource": "*"
        }
 ]
```

고객 관리 키를 사용하여 호스팅된 구성 데이터를 암호화하려면 CreateHostedConfigurationVersion을 호출하는 ID에 사용자, 그룹 또는 역할에 할당할 수 있는 다음과 같은 정책 설명문이 필요합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "kms:GenerateDataKey,
            "Resource": "arn:aws:kms:Region:account_ID:key_ID"
        }
    ]
```

```
}
```

Secrets Manager 암호를 사용하거나 고객 관리 키로 암호화된 기타 구성 데이터를 사용하는 경우 `retrievalRoleArn`은 데이터를 복호화하고 검색하기 위해 `kms:Decrypt`가 필요합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "kms:Decrypt",
      "Resource": "arn:aws:kms:Region:account_ID:configuration source/object"
    }
  ]
}
```

AWS AppConfigStartDeploymentAPI 작업을 호출할 때 자격 증명 `StartDeployment` 호출에는 사용자, 그룹 또는 역할에 할당할 수 있는 다음과 같은 IAM 정책이 필요합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:GenerateDataKey*"
      ],
      "Resource": "arn:aws:kms:Region:account_ID:key_ID"
    }
  ]
}
```

AWS AppConfigGetLatestConfigurationAPI 작업을 호출할 때 ID `GetLatestConfiguration` 호출에는 사용자, 그룹 또는 역할에 할당할 수 있는 다음과 같은 정책이 필요합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "kms:Decrypt,
            "Resource": "arn:aws:kms:Region:account_ID:key_ID"
```

```
        }
    ]
}
```

암호화 컨텍스트

암호화 컨텍스트는 데이터에 대한 추가 컨텍스트 정보를 포함하는 선택적 키-값 페어 세트입니다.

AWS KMS는 암호화 컨텍스트를 추가 인증 데이터로 사용하여 인증된 암호화를 지원합니다. 데이터 암호화 요청에 암호화 컨텍스트를 포함하는 경우, AWS KMS는 암호화된 데이터에 암호화 컨텍스트를 바인딩합니다. 데이터 복호화를 위해, 이 요청에 동일한 암호화 컨텍스트를 포함합니다.

AWS AppConfig암호화 컨텍스트: AWS AppConfig는 암호화된 호스팅 구성 데이터 및 배포에 대한 모든 AWS KMS 암호화 작업에서 암호화 컨텍스트를 사용합니다. 컨텍스트에는 데이터 유형에 해당하는 키와 특정 데이터 항목을 식별하는 값이 포함됩니다.

AWS의 암호화 키 모니터링

에서 AWS KMS 고객 관리형 키를 사용하는 경우AWS AppConfig, AWS CloudTrail 또는 Amazon CloudWatch Logs를 사용하여 로 AWS AppConfig 보내는 요청을 추적할 수 AWS KMS 있습니다.

다음은 고객 관리 키로 암호화된 데이터에 `Decrypt` AWS AppConfig 액세스하기 위해 에서 호출한 AWS KMS 작업을 모니터링하기 위한 CloudTrail 이벤트입니다.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "appconfig.amazonaws.com"
    },
    "eventTime": "2023-01-03T02:22:28z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "Region",
    "sourceIPAddress": "172.12.34.56",
    "userAgent": "ExampleDesktop/1.0 (V1; OS)",
    "requestParameters": {
        "encryptionContext": {
            "aws:appconfig:deployment:arn":
 "arn:aws:appconfig:Region:account_ID:application/application_ID/
environment/environment_ID/deployment/deployment_ID"
        },
        "keyId": "arn:aws:kms:Region:account_ID:key/key_ID",
```

```
        "encryptionAlgorithm": "SYMMETRIC_DEFAULT"
    },
    "responseElements": null,
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": true,
    "resources": [
        {
            "accountId": "account_ID",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:Region:account_ID:key_ID"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "eventCategory": "Management",
    "recipientAccountId": "account_ID",
    "sharedEventID": "dc129381-1d94-49bd-b522-f56a3482d088"
}
```

인터페이스 엔드포인트(AWS PrivateLink)를 사용하여 AWS AppConfig에 액세스

AWS PrivateLink(을)를 사용하여 VPC와 Amazon EKS 사이에 프라이빗 연결을 생성할 수 있습니다. 인터넷 게이트웨이, NAT 디바이스, VPN 연결 또는 AWS Direct Connect 연결을 사용하지 않고 VPC에 있는 것처럼 AWS AppConfig에 액세스할 수 있습니다. VPC의 인스턴스에서 AWS AppConfig API에 액세스하는 데는 퍼블릭 IP 주소가 필요하지 않습니다.

AWS PrivateLink에서 제공되는 인터페이스 엔드포인트를 생성하여 이 프라이빗 연결을 설정합니다. 인터페이스 엔드포인트에 대해 사용 설정하는 각 서브넷에서 엔드포인트 네트워크 인터페이스를 생성합니다. 이는 AWS AppConfig(으)로 향하는 트래픽의 진입점 역할을 하는 요청자 관리형 네트워크 인터페이스입니다.

자세한 내용은 AWS PrivateLink 가이드의 AWS PrivateLink를 통해 AWS 서비스에 액세스를 참조하세요.

AWS AppConfig에 대한 고려 사항

AWS AppConfig에 대한 인터페이스 엔드포인트를 설정하려면 먼저 AWS PrivateLink 가이드의 고려 사항를 검토합니다.

AWS AppConfig인터페이스 엔드포인트를 통해 appconfig및 appconfigdata서비스에 대한 호출을 지원합니다.

AWS AppConfig용 인터페이스 엔드포인트 생성

Amazon VPC 콘솔 또는 AWS Command Line Interface(AWS CLI)를 사용하여 AWS AppConfig에 대한 인터페이스 엔드포인트를 생성할 수 있습니다. 자세한 내용은 AWS PrivateLink 가이드의 인터페이스 엔드포인트 생성을 참조하세요.

다음 서비스 이름을 사용하여 AWS AppConfig에 대한 엔드포인트를 생성합니다.

```
com.amazonaws.region.appconfig
```

```
com.amazonaws.region.appconfigdata
```

인터페이스 엔드포인트에 프라이빗 DNS를 사용하도록 설정하는 경우, 리전에 대한 기본 DNS 이름(예: AWS AppConfig)을 사용하여 에 API 요청을 할 수 있습니다. 예: `appconfig.us-east-1.amazonaws.com` 및 `appconfigdata.us-east-1.amazonaws.com`.

인터페이스 엔드포인트에 엔드포인트 정책 생성

엔드포인트 정책은 인터페이스 인터페이스 엔드포인트에 연결할 수 있는 IAM 리소스입니다. 기본 엔드포인트 정책을 사용하면 인터페이스 엔드포인트를 통해 AWS AppConfigI에 대한 전체 액세스를 허용합니다. VPC에서 AWS AppConfig에 허용되는 액세스를 제어하려면 사용자 지정 엔드포인트 정책을 인터페이스 엔드포인트에 연결합니다.

엔드포인트 정책은 다음 정보를 지정합니다.

- 작업을 수행할 수 있는 보안 주체 (AWS 계정, IAM 사용자, IAM 역할)
- 수행할 수 있는 작업.
- 작업을 수행할 수 있는 리소스.

자세한 내용은 AWS PrivateLink 가이드의 엔드포인트 정책을 사용하여 서비스에 대한 액세스 제어를 참조하세요.

예제: AWS AppConfig 작업에 대한 VPC 엔드포인트 정책

다음은 사용자 지정 엔드포인트 정책의 예입니다. 이 정책은 인터페이스 엔드포인트에 연결될 때 모든 리소스의 모든 보안 주체에 대한 액세스 권한을 나열된 AWS AppConfig 작업에 부여합니다.

```
{
   "Statement": [
      {
         "Principal": "*",
         "Effect": "Allow",
         "Action": [
            "appconfig:CreateApplication",
            "appconfig:CreateEnvironment",
            "appconfig:CreateConfigurationProfile",
            "appconfig:StartDeployment",
            "appconfig:GetLatestConfiguration"
            "appconfig:StartConfigurationSession"
         ],
         "Resource":"*"
      }
   ]
}
```

Secrets Manager 키 교체

이 섹션에서는 AWS AppConfig와 Secrets Manager와의 통합에 대한 중요한 보안 정보를 설명합니다. Secrets Manager에 대한 자세한 내용은 AWS Secrets Manager 사용 설명서의 AWS Secrets Manager란 무엇입니까?를 참조하세요.

AWS AppConfig에서 배포한 Secrets Manager 암호의 자동 교체 설정

교체는 Secrets Manager에 저장되어 있는 암호를 주기적으로 업데이트하는 프로세스입니다. 보안 암호를 교체하면 보안 암호 및 데이터베이스 또는 서비스 모두에서 자격 증명이 업데이트됩니다. Secrets Manager에서 암호와 데이터베이스를 업데이트하기 위해 AWS Lambda 함수를 사용하여 자동 암호 교체를 구성할 수 있습니다. 자세한 내용은 AWS Secrets Manager 사용 설명서에서 AWS Secrets Manager 보안 암호 교체를 참조하세요.

AWS AppConfig에서 배포한 Secrets Manager 암호의 키 교체를 활성화하려면 교체 Lambda 함수를 업데이트하고 교체된 암호를 배포하십시오.

> **Note**
>
> 암호가 교체되고 새 버전으로 완전히 업데이트된 후 AWS AppConfig 구성 프로필을 배포하십시오. `VersionStage` 상태가 `AWSPENDING`에서 `AWSCURRENT`로 변경되었으므로 암호

가 교체되었는지 확인할 수 있습니다. 암호 교체 완료는 Secrets Manager Rotation 템플릿 `finish_secret` 함수 내에서 이루어집니다.

다음은 보안 암호가 교체된 후 AWS AppConfig 배포를 시작하는 함수의 예입니다.

```
import time
import boto3
client = boto3.client('appconfig')

def finish_secret(service_client, arn, new_version):
    """Finish the rotation by marking the pending secret as current
    This method finishes the secret rotation by staging the secret staged AWSPENDING
 with the AWSCURRENT stage.
    Args:
        service_client (client): The secrets manager service client
        arn (string): The secret ARN or other identifier
        new_version (string): The new version to be associated with the secret
    """
    # First describe the secret to get the current version
    metadata = service_client.describe_secret(SecretId=arn)
    current_version = None
    for version in metadata["VersionIdsToStages"]:
        if "AWSCURRENT" in metadata["VersionIdsToStages"][version]:
            if version == new_version:
                # The correct version is already marked as current, return
                logger.info("finishSecret: Version %s already marked as AWSCURRENT for
 %s" % (version, arn))
                return
            current_version = version
            break

    # Finalize by staging the secret version current
    service_client.update_secret_version_stage(SecretId=arn, VersionStage="AWSCURRENT",
 MoveToVersionId=new_version, RemoveFromVersionId=current_version)

    # Deploy rotated secret
    response = client.start_deployment(
            ApplicationId='TestApp',
            EnvironmentId='TestEnvironment',
            DeploymentStrategyId='TestStrategy',
            ConfigurationProfileId='ConfigurationProfileId',
            ConfigurationVersion=new_version,
```

```
            KmsKeyIdentifier=key,
            Description='Deploy secret rotated at ' + str(time.time())
        )

    logger.info("finishSecret: Successfully set AWSCURRENT stage to version %s for
secret %s." % (new_version, arn))
```

AWS AppConfig 모니터링

AWS AppConfig 및 다른 AWS 솔루션의 신뢰성, 가용성 및 성능을 유지하려면 모니터링이 중요합니다. AWS는 AWS AppConfig를 모니터링하고, 이상이 있을 때 이를 보고하고, 필요한 경우 자동 조치를 취할 수 있도록 다음과 같은 모니터링 도구를 제공합니다.

- AWS CloudTrail은 사용자의 AWS 계정에서 혹은 이를 대신하여 수행한 API 호출 및 관련 이벤트를 캡처하고 사용자가 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 참조하세요.
- Amazon CloudWatch Logs를 사용하면 Amazon EC2 인스턴스 및 기타 소스에서 로그 파일을 모니터링 CloudTrail, 저장 및 액세스할 수 있습니다. CloudWatch 로그는 로그 파일의 정보를 모니터링하고 특정 임계값이 충족되면 알려줄 수 있습니다. 또한 매우 내구력 있는 스토리지에 로그 데이터를 저장할 수 있습니다. 자세한 내용은 Amazon CloudWatch Logs 사용 설명서를 참조하십시오.

주제

- AWS CloudTrail을 사용하여 AWS AppConfig API 호출 로깅
- AWS AppConfig데이터 플레인 호출에 대한 로깅 지표

AWS CloudTrail을 사용하여 AWS AppConfig API 호출 로깅

AWS AppConfig에서 사용자AWS CloudTrail, 역할 또는 서비스가 수행한 작업의 기록을 제공하는 AWS 서비스와 통합됩니다AWS AppConfig. CloudTrail 모든 API 호출을 AWS AppConfig 이벤트로 캡처합니다. 캡처되는 호출에는 AWS AppConfig 콘솔로부터의 호출과 AWS AppConfig API 작업에 대한 코드 호출이 포함됩니다. 트레일을 생성하면 에 대한 이벤트를 포함하여 Amazon S3 버킷으로 CloudTrail 이벤트를 지속적으로 전송할 수 AWS AppConfig 있습니다. 트레일을 구성하지 않아도 CloudTrail 콘솔의 이벤트 기록에서 가장 최근 이벤트를 계속 볼 수 있습니다. 에서 수집한 CloudTrail 정보를 사용하여 요청을 받은 사람AWS AppConfig, 요청한 IP 주소, 요청한 사람, 요청 시기 및 추가 세부 정보를 확인할 수 있습니다.

자세한 CloudTrail 내용은 AWS CloudTrail사용 설명서를 참조하십시오.

AWS AppConfig에 대한 정보 CloudTrail

CloudTrail 계정을 만들 AWS 계정 때 활성화됩니다. 에서 AWS AppConfig 활동이 발생하면 해당 활동이 CloudTrail 이벤트 기록의 다른 AWS 서비스 이벤트와 함께 이벤트에 기록됩니다. AWS 계정에서 최

신 이벤트를 확인, 검색 및 다운로드할 수 있습니다. 자세한 내용은 이벤트 기록으로 CloudTrail 이벤트 보기를 참조하십시오.

AWS AppConfig에 대한 이벤트를 포함하여 AWS 계정에 이벤트를 지속적으로 기록하려면 추적을 생성합니다. 트레일을 사용하면 CloudTrail Amazon S3 버킷으로 로그 파일을 전송할 수 있습니다. 콘솔에서 추적을 생성하면 기본적으로 모든 AWS 리전에 추적이 적용됩니다. 추적은 AWS 파티션에 있는 모든 영역의 이벤트를 로깅하고 지정된 Amazon S3 버킷으로 로그 파일을 전송합니다. 또한 CloudTrail 로그에서 수집된 이벤트 데이터를 추가로 분석하고 이에 따라 조치를 취하도록 다른 AWS 서비스를 구성할 수 있습니다. 자세한 내용은 다음 자료를 참조하십시오.

- 추적 생성 개요
- CloudTrail 지원되는 서비스 및 통합
- 에 대한 Amazon SNS 알림 구성 CloudTrail
- 여러 지역에서 CloudTrail 로그 파일 수신 및 여러 계정으로부터 CloudTrail 로그 파일 수신

모든 AWS AppConfig 작업은 AWS AppConfigAPI Reference에 의해 CloudTrail 기록되고 문서화됩니다. 예를 들어`CreateApplication`, 에 대한 호출 `GetApplication` 및 `ListApplications` 작업은 CloudTrail 로그 파일에 항목을 생성합니다.

모든 이벤트 및 로그 항목에는 요청을 생성한 사용자에 대한 정보가 들어 있습니다. 보안 인증 정보를 이용하면 다음을 쉽게 판단할 수 있습니다.

- 요청을 루트로 했는지 아니면 AWS Identity and Access Management(IAM) 사용자 보안 인증 정보로 했는지
- 역할 또는 페더레이션 사용자에 대한 임시 보안 인증 정보를 사용하여 요청이 생성되었는지 여부.
- 다른 AWS 서비스에서 요청했는지.

자세한 내용은 CloudTrail userIdentity 요소를 참조하십시오.

AWS AppConfig의 데이터 이벤트 CloudTrail

데이터 이벤트는 리소스에서 또는 리소스에서 수행된 리소스 작업에 대한 정보를 제공합니다 (예: 호출을 통해 배포된 최신 구성 검색 GetLatestConfiguration). 이를 데이터 영역 작업이라고도 합니다. 데이터 이벤트가 대량 활동인 경우도 있습니다. 기본적으로 데이터 이벤트를 기록하지 CloudTrail 않습니다. CloudTrail 이벤트 기록에는 데이터 이벤트가 기록되지 않습니다.

데이터 이벤트에는 추가 요금이 적용됩니다. CloudTrail 요금에 대한 자세한 내용은 AWS CloudTrail요금을 참조하십시오.

CloudTrail 콘솔 또는 CloudTrail API 작업을 사용하여 AWS AppConfig 리소스 유형에 대한 데이터 이벤트를 기록할 수 있습니다. AWS CLI 이 섹션의 표에는 사용 가능한 리소스 유형이 나와 AWS AppConfig 있습니다.

- CloudTrail 콘솔을 사용하여 데이터 이벤트를 기록하려면 트레일 또는 이벤트 데이터 저장소를 생성하여 데이터 이벤트를 기록하거나 기존 트레일 또는 이벤트 데이터 저장소를 업데이트하여 데이터 이벤트를 기록하십시오.

  1. 데이터 이벤트를 선택하여 데이터 이벤트를 기록합니다.
  2. 데이터 이벤트 유형 목록에서 선택합니다 AWS AppConfig.
  3. 사용하려는 로그 선택기 템플릿을 선택합니다. 리소스 유형에 대한 모든 데이터 이벤트를 기록하거나, 모든 이벤트를 기록하거나, 모든 `readOnly` 이벤트를 기록하거나 `readOnlyeventName`, 및 `resources.ARN` 필드를 기준으로 필터링할 사용자 지정 로그 선택기 템플릿을 만들 수 있습니다. `writeOnly`
  4. 선택기 이름에 를 입력합니다. AppConfigDataEvents 데이터 이벤트 추적을 위해 Amazon CloudWatch Logs를 활성화하는 방법에 대한 자세한 내용은 을 참조하십시오AWS AppConfig 데이터 플레인 호출에 대한 로깅 지표.

- 를 사용하여 데이터 이벤트를 기록하려면 필드를 리소스 유형 값과 같게 설정하고 `eventCategory` 필드를 리소스 유형 값과 같게 설정하도록 `--advanced-event-selectors` 파라미터를 구성하십시오 (표 참조). AWS CLI `Data resources.type` 조건을 추가하여 `readOnlyeventName`, 및 `resources.ARN` 필드의 값을 기준으로 필터링할 수 있습니다.
  - 데이터 이벤트를 기록하도록 트레일을 구성하려면 put-event-selectors명령을 실행합니다. 자세한 내용은 를 사용한 트레일의 데이터 이벤트 로깅을 AWS CLI 참조하십시오.
  - 데이터 이벤트를 기록하도록 이벤트 데이터 저장소를 구성하려면 create-event-data-store명령을 실행하여 데이터 이벤트를 기록할 새 이벤트 데이터 저장소를 만들거나 update-event-data-store명령을 실행하여 기존 이벤트 데이터 저장소를 업데이트하십시오. 자세한 내용은 를 사용한 이벤트 데이터 저장소의 데이터 이벤트 로깅을 참조하십시오AWS CLI.

다음 표는 AWS AppConfig 리소스 유형의 목록입니다. 데이터 이벤트 유형 (콘솔) 열에는 CloudTrail 콘솔의 데이터 이벤트 유형 목록에서 선택할 수 있는 값이 표시됩니다. resources.type 값 열에는 또는 `resources.type` API를 사용하여 고급 이벤트 선택기를 구성할 때 지정하는 값이 표시됩니다. AWS CLI CloudTrail 데이터 API 로깅 대상 CloudTrail 열에는 해당 리소스 유형에 대해 로깅된 API 호출이 표시됩니다. CloudTrail

데이터 이벤트 유형(콘솔)	resources.type 값	로깅된 데이터 API* CloudTrail
AWS AppConfig	`AWS::AppConfig::Configuration`	• GetLatestConfiguration • StartConfigurationSession

*고급 이벤트 선택기를 구성하여`eventName`,`readOnly`, `resources.ARN` 필드를 필터링하여 중요한 이벤트만 기록하도록 할 수 있습니다. 필드에 대한 자세한 내용은 AdvancedFieldSelector 섹션을 참조하세요.

AWS AppConfig의 관리 이벤트 CloudTrail

관리 이벤트는 AWS 계정의 리소스에 대해 수행되는 관리 작업에 대한 정보를 제공합니다. 이를 제어 영역 작업이라고도 합니다. 기본적으로 관리 이벤트를 CloudTrail 기록합니다.

AWS AppConfig모든 AWS AppConfig 컨트롤 플레인 작업을 관리 이벤트로 기록합니다. AWS AppConfig로그되는 AWS AppConfig 컨트롤 플레인 작업 목록은 AWS AppConfigAPI 참조를 참조하십시오. CloudTrail

AWS AppConfig 로그 파일 항목 이해

트레일은 지정한 Amazon S3 버킷에 이벤트를 로그 파일로 전송할 수 있는 구성입니다. CloudTrail 로그 파일에는 하나 이상의 로그 항목이 포함되어 있습니다. 이벤트는 모든 소스의 단일 요청을 나타내며 요청된 작업, 작업 날짜 및 시간, 요청 매개 변수 등에 대한 정보를 포함합니다. CloudTrail 로그 파일은 공개 API 호출의 정렬된 스택 트레이스가 아니므로 특정 순서로 표시되지 않습니다.

다음 예제는 StartConfigurationSession작업을 보여주는 CloudTrail 로그 항목을 보여줍니다.

```
{
      "eventVersion": "1.09",
      "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/Administrator",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
          "sessionIssuer": {},
          "attributes": {
            "creationDate": "2024-01-11T14:37:02Z",
            "mfaAuthenticated": "false"
```

```
          }
        }
      },
      "eventTime": "2024-01-11T14:45:15Z",
      "eventSource": "appconfig.amazonaws.com",
      "eventName": "StartConfigurationSession",
      "awsRegion": "us-east-1",
      "sourceIPAddress": "203.0.113.0",
      "userAgent": "Boto3/1.34.11 md/Botocore#1.34.11 ua/2.0 os/macos#22.6.0
 md/arch#x86_64 lang/python#3.11.4 md/pyimpl#CPython cfg/retry-mode#legacy
 Botocore/1.34.11",
      "requestParameters": {
        "applicationIdentifier": "rrfexample",
        "environmentIdentifier": "mexampleqe0",
        "configurationProfileIdentifier": "3eexampleu1"
      },
      "responseElements": null,
      "requestID": "a1b2c3d4-5678-90ab-cdef-aaaaaEXAMPLE",
      "eventID": "a1b2c3d4-5678-90ab-cdef-bbbbbEXAMPLE",
      "readOnly": false,
      "resources": [
        {
          "accountId": "123456789012",
          "type": "AWS::AppConfig::Configuration",
          "ARN": "arn:aws:appconfig:us-east-1:123456789012:application/rrfexample/
environment/mexampleqe0/configuration/3eexampleu1"
        }
      ],
      "eventType": "AwsApiCall",
      "managementEvent": false,
      "recipientAccountId": "123456789012",
      "eventCategory": "Data",
      "tlsDetails": {
        "tlsVersion": "TLSv1.3",
        "cipherSuite": "TLS_AES_128_GCM_SHA256",
        "clientProvidedHostHeader": "appconfigdata.us-east-1.amazonaws.com"
      }
    }
```


AWS AppConfig데이터 플레인 호출에 대한 로깅 지표

AWS AppConfig데이터 이벤트를 AWS CloudTrail 기록하도록 구성한 경우 Amazon CloudWatch Logs를 활성화하여 AWS AppConfig 데이터 플레인 호출에 대한 지표를 기록할 수 있습니다. 그런 다음

하나 이상의 지표 필터를 생성하여 CloudWatch Logs의 로그 데이터를 검색하고 필터링할 수 있습니다. 지표 필터는 CloudWatch Logs로 전송되는 로그 데이터에서 찾을 용어와 패턴을 정의합니다. CloudWatch 로그는 지표 필터를 사용하여 로그 데이터를 수치 CloudWatch 지표로 변환합니다. 지표를 그래프로 표시하거나 경보를 사용하여 구성할 수 있습니다.

시작하기 전 준비 사항

에서 AWS AppConfig 데이터 이벤트 로깅을 AWS CloudTrail 활성화합니다. 다음 절차는 기존 AWS AppConfig 트레일에 대한 메트릭 로깅을 활성화하는 방법을 설명합니다 CloudTrail. AWS AppConfig 데이터 플랜 호출에 대한 CloudTrail 로깅을 활성화하는 방법에 대한 자세한 내용은 을 참조하십시오AWS AppConfig의 데이터 이벤트 CloudTrail.

다음 절차를 사용하여 CloudWatch 로그를 활성화하여 AWS AppConfig 데이터 플레인 호출에 대한 지표를 기록할 수 있습니다.

CloudWatch 로그를 활성화하여 AWS AppConfig 데이터 플레인에 대한 호출에 대한 메트릭을 기록하려면

1. https://console.aws.amazon.com/cloudtrail/ 에서 CloudTrail 콘솔을 엽니다.
2. 대시보드에서 AWS AppConfig 트레일을 선택합니다.
3. CloudWatch 로그 섹션에서 편집을 선택합니다.
4. 활성을 선택합니다.
5. 로그 그룹 이름의 경우 기본 이름을 그대로 두거나 이름을 입력합니다. 이름을 기록해 둡니다. 나중에 로그 콘솔에서 CloudWatch 로그 그룹을 선택합니다.
6. 역할 이름에 이름을 입력합니다.
7. 변경 사항 저장을 선택합니다.

다음 절차에 따라 AWS AppConfig CloudWatch 로그에 대한 지표와 지표 필터를 만들 수 있습니다. 이 절차는 및 에 의한 호출 `operation` 및 (선택 사항) 호출에 대한 지표 필터를 만드는 방법을 설명합니다`Amazon Resource Name (ARN)`. `operation`

AWS AppConfig CloudWatch 로그에서 메트릭 및 메트릭 필터를 만들려면

1. 에서 CloudWatch 콘솔을 엽니다https://console.aws.amazon.com/cloudwatch/.
2. 왼쪽 탐색 창에서 로그(Logs)를 선택한 다음, 로그 그룹(Log groups)을 선택합니다.
3. AWS AppConfig로그 그룹 옆의 확인란을 선택합니다.
4. 작업을 선택한 후 지표 필터 생성을 선택합니다.

5. 필터 이름에 이름을 입력합니다.
6. 필터 패턴에 다음을 입력합니다.

```
{ $.eventSource = "appconfig.amazonaws.com" }
```

7. (선택 사항) 테스트 패턴 섹션의 테스트할 로그 데이터 선택 목록에서 로그 그룹을 선택합니다. 통화를 기록하지 CloudTrail 않은 경우 이 단계를 건너뛰어도 됩니다.
8. 다음을 선택합니다.
9. 메트릭 네임스페이스의 경우 를 입력합니다. **AWS AppConfig**
10. 지표 이름(Metric name)에서 **Calls**을 입력합니다.
11. 지표 값에 **1**를 입력합니다.
12. 기본값 및 단위를 건너뛰십시오.
13. 차원 이름에는 을 입력합니다**operation**.
14. 차원 값에는 을 입력합니다**$.eventName**.

    (선택 사항) 호출하는 Amazon 리소스 이름 (ARN) 이 포함된 두 번째 차원을 입력할 수 있습니다. 두 번째 차원을 추가하려면 차원 이름에 를 입력합니다**resource**. 차원 값에 을 입력합니다**$.resources[0].ARN**.

    다음을 선택합니다.
15. 필터 세부 정보를 검토하고 지표 필터 만들기를 선택합니다.

(선택 사항) 이 절차를 반복하여 다음과 같은 특정 오류 코드에 대한 새 지표 필터를 만들 수 AccessDenied있습니다. 그럴 경우 다음 세부 정보를 입력하십시오.

1. 필터 이름에 이름을 입력합니다.
2. 필터 패턴에 다음을 입력합니다.

```
{ $.errorCode = "codename" }
```

   예

```
{ $.errorCode = "AccessDenied" }
```

3. 메트릭 네임스페이스의 경우 를 입력합니다. **AWS AppConfig**
4. 지표 이름(Metric name)에서 **Errors**을 입력합니다.

5. 지표 값에 **1**를 입력합니다.
6. 기본값에는 0을 입력합니다.
7. 단위, 치수, 알람은 건너뛰세요.

CloudTrail 로그 API 호출 후에는 에서 지표를 볼 수 있습니다. CloudWatch 자세한 내용은 Amazon CloudWatch 사용 설명서의 콘솔에서 지표 및 로그 보기를 참조하십시오. 생성한 지표를 찾는 방법에 대한 자세한 내용은 사용 가능한 지표 검색을 참조하십시오.

> **Note**
>
> 여기에 설명된 대로 차원이 없는 오류 지표를 설정하면 차원이 없는 지표 페이지에서 해당 지표를 볼 수 있습니다.

CloudWatch지표에 대한 경보 생성

지표를 생성한 후 에서 CloudWatch 지표 경보를 생성할 수 있습니다. 예를 들어, 이전 절차에서 만든 AWS AppConfig통화 지표에 대한 경보를 만들 수 있습니다. 특히 임계값을 초과하는 AWS AppConfig `StartConfigurationSession` API 작업에 대한 호출에 대한 경보를 생성할 수 있습니다. 지표에 대한 경보를 생성하는 방법에 대한 자세한 내용은 Amazon CloudWatch User Guide의 정적 임계값 기반 CloudWatch 경보 생성을 참조하십시오. AWS AppConfig데이터 플레인 호출의 기본 한도에 대한 자세한 내용은 의 데이터 플레인 기본 한도를 참조하십시오 Amazon Web Services 일반 참조.

AWS AppConfig 사용자 가이드 문서 기록

다음 표에는 의 마지막 릴리스 이후 설명서의 중요한 변경 사항이 설명되어 있습니다 AWS AppConfig.

현재 API 버전: 2019-10-09

변경 사항	설명	날짜
AWS AppConfig 사용자 지정 확장 샘플	안내: 사용자 지정 AWS AppConfig 확장 프로그램 만들기 항목에 이제 다음 샘플 확장으로 연결되는 링크가 포함됩니다. GitHub • Systems Manager 변경 일정을 blocked day 사용하여 유예 달력을 통한 배포를 방지하는 샘플 확장 프로그램 • git-secrets를 사용하여 비밀이 구성 데이터로 유출되는 것을 방지하는 샘플 확장 • Amazon Comprehend를 사용하여 개인 식별 정보 (PII)가 구성 데이터로 유출되는 것을 방지하는 샘플 확장 프로그램	2024년 2월 28일
새 주제: 를 AWS AppConfig 사용하여 API 호출 로깅 AWS CloudTrail	AWS AppConfig 에서 사용자 AWS CloudTrail, 역할 또는 서비스가 수행한 작업의 기록을 제공하는 AWS 서비스와 AWS AppConfig통합되었습니다. CloudTrail 모든 API 호출을 AWS AppConfig 이벤트로 캡처합니다. 이 새 항목은 AWS Systems Manager 사용 설명서의 해당 콘텐츠로 연결되는	2024년 1월 18일

	대신 AWS AppConfig특정 콘텐츠를 제공합니다. 자세한 내용은 사용한 AWS AppConfig AWS CloudTrail API 호출 로깅을 참조하십시오.	
AWS AppConfig 현재 지원 AWS PrivateLink	를 AWS PrivateLink 사용하여 VPC와 (과) 사이에 프라이빗 연결을 생성할 수 있습니다. AWS AppConfig인터넷 게이트웨이, NAT 디바이스, VPN 연결 또는 연결을 사용하지 않고도 VPC에 있는 AWS AppConfig 것처럼 액세스할 수 있습니다. AWS Direct Connect VPC의 인스턴스에서 AWS AppConfig API에 액세스하는 데는 퍼블릭 IP 주소가 필요하지 않습니다. 자세한 내용은 인터페이스 엔드포인트를 AWS AppConfig 사용한 액세스 ()AWS PrivateLink를 참조하십시오.	2023년 12월 6일

추가 AWS AppConfig 에이전트 검색 기능 및 새로운 로컬 개발 모드	AWS AppConfig Agent는 애플리케이션의 구성을 검색하는 데 도움이 되는 다음과 같은 추가 기능을 제공합니다. 추가 검색 기능 • 다중 계정 검색: 기본 계정 또는 검색에서 AWS AppConfig 에이전트를 사용하여 여러 공급업체 AWS 계정 계정에서 구성 데이터를 검색합니다. • 구성 복사본을 디스크에 쓰기: AWS AppConfig 에이전트를 사용하여 구성 데이터를 디스크에 기록합니다. 이 기능을 사용하면 디스크에서 구성 데이터를 읽는 애플리케이션을 사용하는 고객이 이 기능을 통합하여 사용할 수 AWS AppConfig있습니다. **Note** 디스크에 구성 쓰기는 구성 백업 기능으로 설계되지 않았습니다. AWS AppConfig 에이전트는 디스크에 복사된 구성 파일을 읽지 않습니다. 구성을 디스크에 백업하려면 Amazon EC2에서 에이전트 사용 또는 Amazon ECS	2023년 12월 1일

BACKUP_DIRECTORY 및 Amazon EKS에서 AWS AppConfig AWS AppConfig 에이전트 사용에 대한 및 PRELOAD_BACKUP 환경 변수를 참조하십시오.

로컬 개발 모드

AWS AppConfig 에이전트는 로컬 개발 모드를 지원합니다. 로컬 개발 모드를 활성화하면 에이전트는 디스크의 지정된 디렉터리에서 구성 데이터를 읽습니다. 에서는 구성 데이터를 검색하지 않습니다 AWS AppConfig. 지정된 디렉터리의 파일을 업데이트하여 구성 배포를 시뮬레이션할 수 있습니다. 다음 사용 사례에는 로컬 개발 모드를 사용하는 것이 좋습니다.

- 를 사용하여 AWS AppConfig배포하기 전에 다양한 구성 버전을 테스트하십시오.
- 코드 리포지토리에 변경 사항을 커밋하기 전에 새 기능에 대한 다양한 구성 옵션을 테스트하세요.

	• 다양한 구성 시나리오를 테스트하여 예상대로 작동하는지 확인하세요.	
새 코드 샘플 주제	이 가이드에 새 코드 샘플 주제를 추가했습니다. 이 항목에는 Java, Python으로 작성된 예제와 JavaScript 6가지 일반적인 AWS AppConfig 작업을 프로그래밍 방식으로 수행하는 예제가 포함되어 있습니다.	2023년 11월 17일
AWS AppConfig 워크플로우를 더 잘 반영하도록 목차를 수정했습니다.	이 사용 설명서의 내용은 이제 워크플로우 생성, 배포, 검색 및 확장이라는 제목 아래에 그룹화되어 있습니다. 이 조직은 AWS AppConfig 사용 워크플로우를 더 잘 반영하고 콘텐츠를 더 쉽게 검색할 수 있도록 하는 것을 목표로 합니다.	2023년 11월 7일
페이로드 참조 추가	이제 사용자 지정 AWS AppConfig 확장을 위한 Lambda 함수 생성 주제에 요청 및 응답 페이로드 참조가 포함됩니다.	2023년 11월 7일
AWS 사전 정의된 새 배포 전략	AWS AppConfig 이제 `AppConfig.Linear20PercentEvery6Minutes` 사전 정의된 배포 전략을 제공하고 권장합니다. 자세한 정보는 사전 정의된 배포 전략을 참조하십시오.	2023년 8월 11일

AWS AppConfig 아마존 EC2와의 통합	에이전트를 사용하여 Amazon Elastic Compute Cloud (Amazon EC2) Linux 인스턴스에서 실행되는 AWS AppConfig 애플리케이션과 통합할 수 있습니다. AWS AppConfig 에이전트는 Amazon EC2용 x86_64 및 ARM64 아키텍처를 지원합니다. 자세한 내용은 Amazon EC2와AWS AppConfig 통합을 참조하십시오.	2023년 7월 20일
AWS CloudFormation 새 AWS AppConfig 리소스 지원 및 기능 플래그 예제	AWS CloudFormation 이제 AWS AppConfig 확장 프로그램을 시작하는 데 도움이 되는 AWS::AppConfig::Extension및 AWS::AppConfig::ExtensionAssociation리소스를 지원합니다. AWS::AppConfig::ConfigurationProfile및 AWS::AppConfig::HostedConfigurationVersion 리소스에는 이제 AWS AppConfig 호스팅된 구성 저장소에서 기능 플래그 구성 프로필을 만드는 예제가 포함되어 있습니다.	2023년 8월 12일

AWS AppConfig 통합: AWS Secrets Manager	AWS AppConfig 와 통합됩니다. AWS Secrets Manager Secrets Manager는 데이터베이스와 다른 서비스의 자격 증명을 안전하게 암호화, 저장 및 검색하는 데 효과적입니다. 앱에서 자격 증명을 하드 코딩하는 대신 필요할 때마다 Secrets Manager를 호출하여 자격 증명을 검색할 수 있습니다. Secrets Manager를 사용하면 암호에 대한 액세스를 교체 및 관리할 수 있으므로 IT 리소스 및 데이터에 대한 액세스를 보호할 수 있습니다. 자유 형식 구성 프로필을 만들 때 Secrets Manager를 구성 데이터의 소스로 선택할 수 있습니다. 구성 프로필을 생성하기 전에 Secrets Manager에 온보딩하고 암호를 생성해야 합니다. Secrets Manager에 대한 자세한 내용은 무엇입니까 AWS Secrets Manager? 를 참조하십시오. AWS Secrets Manager 사용 설명서에서 구성 프로필을 만드는 방법에 대한 자세한 내용은 자유 형식 구성 프로필 만들기를 참조하십시오.	2023년 2월 2일

AWS AppConfig 아마존 ECS 및 아마존 EKS와의 통합	에이전트를 AWS AppConfig 사용하여 Amazon Elastic Container Service (Amazon ECS) 및 Amazon Elastic Kubernetes 서비스 (Amazon EKS) 와 통합할 수 있습니다. AWS AppConfig 에이전트는 Amazon ECS 및 Amazon EKS 컨테이너 애플리케이션과 함께 실행되는 사이드카 컨테이너 역할을 합니다. 에이전트는 다음과 같은 방식으로 컨테이너식 애플리케이션 처리 및 관리를 개선합니다. • 에이전트는 AWS Identity and Access Management (IAM) 역할을 사용하고 구성 데이터의 로컬 캐시를 관리하여 사용자를 대신하여 전화를 겁니다 AWS AppConfig . 로컬 캐시에서 구성 데이터를 가져오면 애플리케이션이 구성 데이터를 관리하는 데 필요한 코드 업데이트 횟수가 줄어들고, 구성 데이터를 밀리초 단위로 검색할 수 있으며, 이러한 데이터에 대한 호출을 방해할 수 있는 네트워크 문제의 영향을 받지 않습니다. • 에이전트는 기능 플래그를 검색하고 해결하기 AWS AppConfig 위한 기본 환경을 제공합니다.	2022년 12월 2일

- 에이전트는 기본적으로 캐싱 전략, 폴링 간격, 로컬 구성 데이터 가용성에 대한 모범 사례를 제공하는 동시에 후속 서비스 호출에 필요한 구성 토큰을 추적합니다.
- 에이전트는 백그라운드에서 실행되는 동안 정기적으로 AWS AppConfig 데이터 플레인을 폴링하여 구성 데이터 업데이트를 확인합니다. 컨테이너화된 애플리케이션은 포트 2772(사용자 지정 가능한 기본 포트 값)에서 localhost에 연결하고 HTTP GET을 호출하여 데이터를 검색함으로써 데이터를 검색할 수 있습니다.
- AWS AppConfig 에이전트는 컨테이너를 다시 시작하거나 재활용할 필요 없이 컨테이너의 구성 데이터를 업데이트합니다.

자세한 내용은 Amazon ECS 및 Amazon EKS와AWS AppConfig 통합을 참조하십시오.

새 확장: AWS AppConfig Evidently의 CloudWatch 확장 프로그램	Amazon CloudWatch Evivently를 사용하면 기능을 출시하는 동안 지정된 비율의 사용자에게 새 기능을 제공하여 새 기능을 안전하게 검증할 수 있습니다. 새 기능의 성능을 모니터링해 사용자에게 트래픽을 늘릴 시기를 결정할 수 있습니다. 이를 통해 위험을 줄이고 의도하지 않은 결과를 파악한 후 기능을 완전히 출시할 수 있습니다. 또한 A/B 실험을 수행해 증거와 데이터를 기반으로 기능 설계 결정을 내릴 수 있습니다. CloudWatch Evivality의 AWS AppConfig 확장 프로그램을 사용하면 애플리케이션에서 작업을 호출하는 대신 로컬에서 사용자 세션에 변형을 할당할 수 있습니다. EvaluateFeature 로컬 세션은 API 호출로 인한 지연 시간 및 가용성 위험을 줄일 수 있습니다. 확장 프로그램을 구성하고 사용하는 방법에 대한 자세한 내용은 Amazon 사용 CloudWatch 설명서의 CloudWatch Evively를 사용하여 출시 및 A/B 실험 수행을 참조하십시오.	2022년 9월 13일

GetConfiguration API 작업 지원 중단	2021년 11월 18일에 새로운 데이터 플레인 AWS AppConfig 서비스가 출시되었습니다. 이 서비스는 GetConfiguration API 작업을 사용하여 구성 데이터를 검색하는 이전 프로세스를 대체합니다. 데이터 플레인 서비스는 두 개의 새로운 API 작업, StartConfigurationSession 및 GetLatestConfiguration을 사용합니다. 또한 데이터 영역 서비스는 새 엔드포인트를 사용합니다. 자세한 내용은 AWS AppConfig 데이터 플레인 서비스 정보를 참조하십시오.	2022년 9월 13일
AWS AppConfig 에이전트 Lambda 확장 프로그램의 새 버전	이제 에이전트 AWS AppConfig Lambda 확장 프로그램 버전 2.0.122를 사용할 수 있습니다. 새로운 확장에서는 다양한 Amazon 리소스 이름(ARN)을 사용합니다. 자세한 내용은 AWS AppConfig 에이전트 Lambda 확장 릴리스 정보를 참조하세요.	2022년 8월 23일

확장 프로그램 출시 AWS AppConfig	확장 프로그램은 구성을 만들거나 배포하는 AWS AppConfig 워크플로우 중에 다양한 지점에 로직 또는 동작을 삽입할 수 있는 기능을 강화합니다. AWS-Authored 확장을 사용하거나 직접 만들 수 있습니다. 자세한 내용은 확장 프로그램 사용을 참조하십시오. AWS AppConfig	2022년 7월 12일
AWS AppConfig 에이전트 Lambda 확장 프로그램의 새 버전	이제 에이전트 AWS AppConfig Lambda 확장 프로그램 버전 2.0.58을 사용할 수 있습니다. 새로운 확장에서는 다양한 Amazon 리소스 이름(ARN)을 사용합니다. 자세한 내용은 AWS AppConfig Lambda 확장 프로그램의 사용 가능한 버전을 참조하십시오.	2022년 5월 3일
AWS AppConfig 아틀라시안 지라와의 통합	Atlassian Jira와 통합하면 지정된 기능에 AWS AppConfig 대한 기능 플래그를 변경할 때마다 Atlassian 콘솔에서 이슈를 생성하고 업데이트할 수 있습니다. AWS 계정 AWS 리전 각 Jira 이슈에는 플래그 이름, 애플리케이션 ID, 구성 프로필 ID 및 플래그 값이 포함됩니다. 플래그 변경 사항을 업데이트, 저장 및 배포한 후 Jira는 변경 세부 정보와 함께 기존 이슈를 업데이트합니다. 자세한 내용은 Atlassian Jira와AWS AppConfig 통합을 참조하세요.	2022년 4월 7일

ARM64 (Graviton2) 프로세서에 대한 기능 플래그 및 Lambda 확장 지원 정식 출시	AWS AppConfig 기능 플래그를 사용하면 새 기능을 개발하고 프로덕션 환경에 배포할 수 있으며, 사용자는 이 기능을 숨길 수 있습니다. 먼저 플래그를 구성 AWS AppConfig 데이터로 추가합니다. 기능을 릴리스할 준비가 되면 코드를 배포하지 않고도 플래그 구성 데이터를 업데이트할 수 있습니다. 이 기능을 사용하면 기능을 릴리스하기 위해 새 코드를 배포할 필요가 없으므로 데스 옵스 환경의 안전성이 향상됩니다. 자세한 내용은 기능 플래그 구성 프로필 생성을 참조하십시오. AWS AppConfig 의 기능 플래그 정식 출시에는 다음과 같은 개선 사항이 포함됩니다. • 콘솔에는 플래그를 단기 플래그로 지정하는 옵션이 포함되어 있습니다. 단기 플래그의 플래그 목록을 필터링하고 정렬할 수 있습니다. • AWS Lambda의 기능 플래그를 사용하는 고객의 경우, 새로운 Lambda 확장을 사용하면 HTTP 엔드포인트를 사용하여 개별 기능 플래그를 호출할 수 있습니다. 자세한 내용은 기능 플래그 구성에서 하나 이상의 플래그 검색을 참조하십시오.	2022년 3월 15일

	또한 이 업데이트는 ARM64 (Graviton2) 프로세서용으로 개발된 AWS Lambda 확장에 대한 지원을 제공합니다. 자세한 내용은 AWS AppConfig Lambda 확장 프로그램의 사용 가능한 버전을 참조하십시오.	
GetConfiguration API 작업은 더 이상 사용되지 않습니다.	`GetConfiguration` API 작업은 더 이상 사용되지 않습니다. 구성 데이터를 수신하기 위한 호출은 대신 `StartConfigurationSession` 및 `GetLatestConfiguration` API를 사용해야 합니다. 이러한 API 및 사용 방법에 대한 자세한 내용은 구성 검색을 참조하세요.	2022년 1월 28일
Lambda AWS AppConfig 확장 프로그램을 위한 새로운 리전 ARN	AWS AppConfig Lambda 확장 프로그램은 새로운 아시아 태평양 (오사카) 지역에서 사용할 수 있습니다. 리전에서 Lambda를 생성하려면 Amazon 리소스 이름(ARN) 이 필요합니다. 아시아 태평양 (Osaka) 리전 ARN에 대한 자세한 내용은 Lambda AWS AppConfig 확장 프로그램 추가를 참조하십시오.	2021년 3월 4일

AWS AppConfig 람다 익스텐션	를 AWS AppConfig 사용하여 Lambda 함수의 구성을 관리하는 경우 Lambda 확장 프로그램을 추가하는 것이 좋습니다. AWS AppConfig 이 확장에는 사용을 AWS AppConfig 단순화하는 동시에 비용을 절감하는 모범 사례가 포함되어 있습니다. AWS AppConfig 서비스에 대한 API 호출 횟수가 줄어들어 비용이 절감되고, 이와 별도로 Lambda 함수 처리 시간이 단축되어 비용이 절감되었습니다. 자세한 내용은 Lambda 확장과AWS AppConfig 통합을 참조하십시오.	2020년 10월 8일
새로운 섹션	AWS AppConfig설정 지침을 제공하는 새 섹션이 추가되었습니다. 자세한 내용은 설정 AWS AppConfig을 참조하십시오.	2020년 9월 30일
추가된 명령줄 프로시저	이 사용 설명서의 절차에는 이제 AWS Command Line Interface (AWS CLI) 및 Windows용 도구의 명령줄 단계가 포함됩니다. PowerShell 자세한 내용은 작업을 참조하십시오. AWS AppConfig	2020년 9월 30일

AWS AppConfig 사용 설명서 시작	의 AWS Systems Manager 기능을 사용하여 AWS AppConfig애플리케이션 구성을 생성, 관리 및 신속하게 배포할 수 있습니다. AWS AppConfig 모든 규모의 애플리케이션에 대한 제어된 배포를 지원하며 내장된 검증 검사 및 모니터링 기능을 포함합니다. EC2 인스턴스, 컨테이너 AWS Lambda, 모바일 애플리케이션 또는 IoT 디바이스에서 호스팅되는 애플리케이션과 AWS AppConfig 함께 사용할 수 있습니다.	2020년 7월 31일

AWS 용어집

최신 AWS 용어는 AWS 용어집 참조서의 AWS 용어집을 참조하세요.

기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.