CodeArtifact 사용자 가이드

CodeArtifact

CodeArtifact: CodeArtifact 사용자 가이드

Table of Contents

AWS CodeArtifact란 무엇입니까?

AWS CodeArtifact는 조직이 애플리케이션 개발을 위해 소프트웨어 패키지를 저장하고 공유하는 데 도움이 되는 안전하고 확장성이 뛰어난 관리형 아티팩트 리포지토리 서비스입니다. CodeArtifact는 NuGet CLI, Maven, Gradle, npm, yarn, pip 및 twine과 같은 인기 있는 빌드 도구 및 패키지 관리자와 함께 사용할 수 있습니다. CodeArtifac를 사용하면 자체 아티팩트 스토리지 시스템 관리 및 인프라 확장에 대한 걱정을 줄이는 데 도움이 됩니다. CodeArtifact 리포지토리에 저장할 수 있는 패키지의 수 또는 전체 크기에는 제한이 없습니다.

프라이빗 CodeArtifact 리포지토리와 npmjs.com 또는 Maven Central과 같은 외부 공용 리포지토리 간에 연결을 생성할 수 있습니다. 그러면 CodeArtifact는 패키지 관리자가 요청하면 필요에 따라 공용 리포지토리에서 패키지를 가져와 저장합니다. 이렇게 하면 애플리케이션에서 사용하는 오픈 소스 종속성을 더 편리하게 사용할 수 있고 패키지를 빌드 및 개발에 항상 사용할 수 있습니다. CodeArtifact 리포지토리에 비공개 패키지를 게시할 수도 있습니다. 이를 통해 조직의 여러 애플리케이션과 개발 팀 간에 전용 소프트웨어 구성 요소를 공유할 수 있습니다.

자세한 내용은 AWS CodeArtifact를 참조하세요.

CodeArtifact는 어떻게 작동하나요?

CodeArtifact는 소프트웨어 패키지를 리포지토리에 저장합니다. 리포지토리는 다국어 개체이며, 따라서 지원되는 모든 유형의 패키지가 단일 리포지토리에 포함될 수 있습니다. 모든 CodeArtifact 리포지토리는 단일 CodeArtifact 도메인의 구성원입니다. 리포지토리가 하나 이상 있는 조직의 프로덕션 도메인은 한 개만 사용하는 것이 좋습니다. 예를 들어 각 리포지토리를 다른 개발 팀에서 사용할 수 있습니다. 그런 경우 전체 개발팀에서 리포지토리의 패키지를 검색하고 공유할 수 있습니다.

리포지토리에 패키지를 추가하려면 npm 또는 Maven과 같은 패키지 관리자가 리포지토리 엔드포인트(URL)를 사용하도록 구성합니다. 그런 다음 패키지 관리자를 사용하여 패키지를 리포지토리에 게시할 수 있습니다. npmjs, NuGet Gallery, Maven Central 또는 PyPI와 같은 공용 리포지토리에 대한 외부 연결을 통해 구성하여 오픈 소스 패키지를 리포지토리로 가져올 수도 있습니다. 자세한 내용은 CodeArtifact 저장소를 공용 저장소에 연결하기 단원을 참조하십시오.

패키지를 한 리포지토리에서 동일한 도메인의 다른 리포지토리에서 사용할 수도 있습니다. 이렇게 하려면 한 리포지토리를 다른 리포지토리의 업스트림으로 구성해야 합니다. 업스트림 리포지토리에서 사용할 수 있는 모든 패키지 버전을 다운스트림 리포지토리에서도 사용할 수 있습니다. 또한 공용 저장소에 대한 외부 연결을 통해 업스트림 리포지토리에서 사용할 수 있는 모든 패키지를 다운스트림 리포지토리에서도 사용할 수 있습니다. 자세한 내용은 CodeArtifact에서의 업스트림 리포지토리 작업 단원을 참조하십시오.

CodeArtifact에서 패키지 버전을 게시하거나 사용하려면 서비스에 인증해야 합니다. AWS 자격 증명을 사용하여 권한 부여 토큰을 생성하여 CodeArtifact 서비스에 인증해야 합니다. CodeArtifact 리포지토리의 패키지는 공개적으로 사용할 수 없습니다. CodeArtifact에서 인증 및 액세스에 대한 자세한 내용은 AWS CodeArtifact 인증 및 토큰 섹션을 참조하세요.

AWS CodeArtifact 개념

CodeArtifact를 사용할 때 알아야 할 몇 가지 개념과 용어는 다음과 같습니다.

주제

- 자산
- 도메인
- 리포지토리
- 패키지
- 패키지 그룹
- 패키지 네임스페이스
- 패키지 버전
- 패키지 버전 개정
- 업스트림 리포지토리

자산

자산은 패키지 버전(예: `.tgz` npm 파일, 메이븐 POM 및 JAR 파일)과 관련된 CodeArtifact에 저장된 개별 파일입니다.

도메인

리포지토리는 도메인이라는 상위 수준 엔티티로 통합됩니다. 모든 패키지 자산과 메타데이터는 도메인에 저장되지만 리포지토리를 통해 소비됩니다. Maven JAR 파일과 같은 특정 패키지 자산은 해당 자산이 있는 리포지토리 개수와 관계없이 도메인당 한 번씩 저장됩니다. 도메인의 모든 자산과 메타데이터는 AWS Key Management Service ()에 저장된 것과 동일한 AWS KMS key (KMS 키)로 암호화됩니다AWS KMS.

각 리포지토리는 단일 도메인의 구성원이며 다른 도메인으로 이동할 수 없습니다.

도메인을 사용하면 여러 리포지토리에 조직 정책을 적용할 수 있습니다. 이 방식을 사용하면 도메인의 리포지토리에 액세스할 수 있는 계정과 패키지 소스로 사용할 수 있는 공용 리포지토리를 지정할 수 있습니다.

조직에 여러 도메인이 있을 수 있지만 게시된 아티팩트를 모두 포함하는 단일 프로덕션 도메인이 있는 것이 좋습니다. 이렇게 하면 팀이 조직 전체에서 패키지를 찾고 공유할 수 있습니다.

리포지토리

CodeArtifact 리포지토리에는 패키지 버전 세트가 포함되어 있으며, 각 패키지 버전은 자산 세트에 매핑됩니다. 리포지토리는 다국어 개체이며, 따라서 지원되는 모든 유형의 패키지가 단일 리포지토리에 포함될 수 있습니다. 각 리포지토리는 nuget CLI, npm CLI, Maven CLI(mvn), pip와 같은 도구를 사용하여 패키지를 가져오고 게시하기 위한 엔드포인트를 노출합니다. 도메인당 최대 1,000개의 리포지토리를 생성할 수 있습니다.

패키지

패키지는 종속성을 해결하고 소프트웨어를 설치하는 데 필요한 소프트웨어 및 메타데이터 번들입니다. CodeArtifact에서 패키지는 패키지 이름, `@types/node`의 `@types` 같은 선택적 네임스페이스, 패키지 버전 세트, 패키지 수준 메타데이터(예: npm 태그)로 구성됩니다.

AWS CodeArtifact는 Cargo, generic, Maven, npm, NuGet, PyPI, Ruby, Swift 패키지 형식을 지원합니다.

패키지 그룹

패키지 그룹은 패키지 형식, 패키지 네임스페이스 및 패키지 이름을 사용하여 정의된 패턴과 일치하는 여러 패키지에 구성을 적용하는 데 사용할 수 있습니다. 패키지 그룹을 사용하여 여러 패키지에 대한 패키지 원본 제어를 보다 편리하게 구성할 수 있습니다. 패키지 원본 제어는 새 패키지 버전의 수집 또는 게시를 차단하거나 허용하는 데 사용되며, 이를 통해 의존성 대체 공격이라는 악의적인 행위로부터 사용자를 보호합니다.

패키지 네임스페이스

일부 패키지 형식은 계층적 패키지 이름을 지원하여 패키지를 논리적 그룹으로 구성할 수 있고 이름 충돌을 방지하는 데 도움이 됩니다. 예를 들어 npm은 범위를 지원합니다. 자세한 내용은 npm 범위 설명서를 참조하세요. `@types/node` npm 패키지의 범위는 `@types`, 이름은 `node`입니다. `@types` 범위에는 다른 패키지 이름이 많습니다. CodeArtifact에서 범위('유형')는 패키지 네임스페이스라고 하고 이름('노드')은 패키지 이름이라고 합니다. Maven 패키지의 경우 패키지 네임스페이

스는 Maven groupID에 해당합니다. `org.apache.logging.log4j:log4j` Maven 패키지에는 `org.apache.logging.log4j` groupID(패키지 네임스페이스)와 `log4j` artifactID(패키지 이름)가 있습니다. 일반 패키지의 경우 네임스페이스가 필요합니다. PyPI와 같은 일부 패키지 형식은 npm 범위 또는 Maven GroupId와 유사한 개념의 계층적 이름을 지원하지 않습니다. 패키지 이름을 그룹화하는 방법이 없으면 이름 충돌을 피하기가 더 어려울 수 있습니다.

패키지 버전

패키지 버전은 패키지의 특정 버전(예: `@types/node 12.6.9`)을 식별합니다. 버전 번호 형식과 의미 체계는 패키지 형식에 따라 다릅니다. 예를 들어, npm 패키지 버전은 의미 체계 버전 관리 사양을 준수해야 합니다. CodeArtifact 패키지 버전은 버전 식별자, 패키지 버전 수준 메타데이터 및 자산 세트로 구성됩니다.

패키지 버전 개정

패키지 버전 개정은 패키지 버전의 특정 자산 및 메타데이터 세트를 식별하는 문자열입니다. 패키지 버전이 업데이트될 때마다 새 패키지 버전 개정이 생성됩니다. 예를 들어 Python 패키지 버전의 소스 배포 아카이브(sdist)를 게시하고 나중에 컴파일된 코드가 포함된 Python 휠을 동일한 버전에 추가할 수 있습니다. 휠을 게시하면 새 패키지 버전 개정이 생성됩니다.

업스트림 리포지토리

다운스트림 리포지토리의 리포지토리 엔드포인트에서 해당 리포지토리의 패키지 버전에 액세스할 수 있는 경우 한 리포지토리는 다른 리포지토리의 업스트림입니다. 이 방식은 클라이언트의 관점에서 볼 때 두 리포지토리의 콘텐츠를 효과적으로 병합합니다. CodeArtifact를 사용하면 두 리포지토리 간에 업스트림 관계를 만들 수 있습니다.

CodeArtifact는 어떻게 시작할 수 있나요?

다음 단계를 수행하는 것이 좋습니다.

1. CodeArtifact에 대해 자세히 알아보려면 다음을 읽어보세요. AWS CodeArtifact 개념
2. AWS 계정의 단계에 따라 AWS CLI, 및 IAM 사용자를 설정합니다with AWS CodeArtifact 설정.
3. CodeArtifact 시작하기의 지침에 따라 CodeArtifact를 사용합니다.

with AWS CodeArtifact 설정

Amazon Web Services(AWS)에 이미 가입한 경우 즉시 AWS CodeArtifact 사용을 시작할 수 있습니다. CodeArtifact 콘솔을 열고 도메인 및 리포지토리 생성을 선택한 다음 시작 마법사의 단계에 따라 첫 번째 도메인과 리포지토리를 만들 수 있습니다.

AWS 아직에 가입하지 않았거나 첫 번째 도메인 및 리포지토리를 생성하는 데 도움이 필요한 경우 다음 작업을 완료하여 CodeArtifact를 사용하도록 설정합니다.

주제

- 에 가입 AWS
- 설치 또는 업그레이드 후 구성 AWS CLI
- IAM 사용자 구성하기
- 패키지 관리자 또는 빌드 도구를 설치하기

에 가입 AWS

Amazon Web Services(AWS)에 가입하면 AWS CodeArtifact를 포함하여 사용하는 서비스 및 리소스에 대해서만 요금이 청구됩니다.

가 이미 있는 경우 다음 작업인 로 AWS 계정건너뜁니다설치 또는 업그레이드 후 구성 AWS CLI. 이 없는 경우 AWS 계정다음 절차에 따라 생성합니다.

를 생성하려면 AWS 계정

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

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

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

설치 또는 업그레이드 후 구성 AWS CLI

로컬 개발 시스템의 AWS Command Line Interface (AWS CLI)에서 CodeArtifact 명령을 호출하려면를 설치해야 합니다 AWS CLI.

이전 버전의가 AWS CLI 설치된 경우 CodeArtifact 명령을 사용할 수 있도록 업그레이드해야 합니다. CodeArtifact 명령은 다음 버전의 AWS CLI 에서 사용할 수 있습니다.

1. AWS CLI 1: 1.18.77 이상
2. AWS CLI 2: 2.0.21 이상

`aws --version` 명령을 사용하여 버전을 확인하세요.

를 설치하고 구성하려면 AWS CLI

1. 설치 AWS CLI 의 지침에 따라 를 설치하거나 업그레이드합니다 AWS Command Line Interface.
2. 다음과 같이 configure 명령을 AWS CLI사용하여를 구성합니다.

   ```
   aws configure
   ```

   메시지가 표시되면 CodeArtifact와 함께 사용할 IAM 사용자의 AWS 액세스 키와 AWS 보안 액세스 키를 지정합니다. AWS 리전 기본 위치 이름을 입력하라는 메시지가 표시되면, 파이프라인을 생성할 위치를 지정합니다(예:`us-east-2`). 기본 출력 형식을 묻는 메시지가 표시되면 `json`을 지정합니다.

   > **Important**
   >
   > 를 구성할 때 AWS CLI를 지정하라는 메시지가 표시됩니다 AWS 리전. AWS 일반 참조의 위치 및 엔드포인트에 나열된 지원되는 위치 중 하나를 선택합니다.

   자세한 내용은 IAM 사용자의 액세스 키 AWS Command Line Interface구성 및 관리하기를 참조하세요.
3. 설치 또는 업그레이드를 확인하려면 AWS CLI에서 다음 명령을 직접 호출합니다.

   ```
   aws codeartifact help
   ```

성공적으로 진행하면 사용 가능한 CodeArtifact 명령 목록을 표시합니다.

다음으로 IAM 사용자를 생성하고 해당 사용자에게 CodeArtifact 접근 권한을 부여할 수 있습니다. 자세한 내용은 IAM 사용자 구성하기 단원을 참조하십시오.

IAM 사용자 구성하기

다음 지침에 따라 CodeArtifact를 사용할 수 있도록 IAM 사용자를 준비합니다.

IAM 사용자 구성 방법

1. IAM 사용자를 만들거나 AWS 계정계정과 연결된 사용자를 사용합니다. 자세한 내용은 IAM 사용 설명서의 IAM 사용자 생성 및 IAM 정책 개요를 AWS 참조하세요.
2. IAM 사용자에게 CodeArtifact의 접근 권한을 부여합니다.

   - 옵션 1: 사용자 지정 IAM 정책을 생성합니다. 사용자 지정 IAM 정책을 사용하면 필요한 최소 권한을 제공하고 인증 토큰의 유효 기간을 변경할 수 있습니다. 자세한 내용과 정책 예는 AWS CodeArtifact에 대한 자격 증명 기반 정책 예제 섹션을 참조하세요.
   - 옵션 2: `AWSCodeArtifactAdminAccess` AWS 관리형 정책을 사용합니다. 이 정책의 내용은 다음 코드 조각에 나와 있습니다.

     > **Important**
     >
     > 이 정책은 CodeArtifact API의 모든 접근 권한을 부여합니다. 작업을 완료하는 데 필요한 최소 권한을 항상 사용하는 것이 좋습니다. 자세한 내용은 IAM 사용 설명서의 IAM 모범 사례를 참조하세요.

     JSON

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

```
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "sts:GetServiceBearerToken",
            "Resource": "*",
                "Condition": {
                    "StringEquals": {
                        "sts:AWSServiceName": "codeartifact.amazonaws.com"
                    }
                }
        }
    ]
}
```

> **Note**
>
> `sts:GetServiceBearerToken` 권한은 IAM 사용자 또는 역할 정책에 추가되어야 합니다. CodeArtifact 도메인 또는 리포지토리 리소스 정책에 추가할 수 있지만 해당 권한이 리소스 정책에 영향을 주지는 않습니다.

CodeArtifact `GetAuthorizationToken` API를 직접 호출하려면 `sts:GetServiceBearerToken` 권한이 필요합니다. 이 API는 CodeArtifact같은 패키지 관리자를 사용할 때 또는 CodeArtifact와 함께 npm과 `pip`를 사용할 때 필요한 토큰을 반환합니다.. CodeArtifact 리포지토리와 함께 패키지 관리자를 사용하려면 IAM 사용자 또는 역할이 위의 정책 예제에 표시된 대로 `sts:GetServiceBearerToken`을 허용해야 합니다.

CodeArtifact와 함께 사용할 패키지 관리자 또는 빌드 도구를 설치하지 않은 경우 패키지 관리자 또는 빌드 도구를 설치하기을 참조하세요.

패키지 관리자 또는 빌드 도구를 설치하기

CodeArtifact에서 패키지를 게시하거나 사용하려면 패키지 관리자를 사용해야 합니다. 각 패키지 형식마다 다른 패키지 관리자가 있습니다. 다음 목록에는 CodeArtifact와 함께 사용할 수 있는 몇 가지 패키지 관리자가 포함되어 있습니다. 아직 설치하지 않았다면 사용하려는 패키지 형식에 맞는 패키지 관리자를 설치하세요.

- npm의 경우 npm CLI 또는 pnpm을 사용하세요.

- Maven의 경우 Apache Maven (mvn) 또는 Gradle을 사용하세요.
- Python의 경우 pip를 사용하여 패키지를 설치하고 twine을 사용하여 패키지를 게시합니다.
- NuGet의 경우 비주얼 스튜디오의 Visual Studio용 툴킷이나 nuget 또는 dotnet CLI를 사용하세요.
- 일반 패키지의 경우 AWS CLI또는 SDK를 사용하여 패키지 콘텐츠를 게시하고 다운로드할 수 있습니다.

다음 단계

다음 단계는 CodeArtifact와 함께 사용하는 패키지 유형과 CodeArtifact 리소스 상태에 따라 달라집니다.

본인, 팀 또는 조직에서 CodeArtifact를 처음 시작하는 경우 다음 설명서에서 일반적인 시작 정보와 필요한 리소스를 만드는 데 필요한 도움말을 확인하세요.

- 콘솔을 사용하여 시작하기
- AWS CLI를 사용하여 시작하기

리소스가 이미 생성되어 있고 패키지 관리자를 구성해 CodeArtifact 저장소에 패키지를 불러오거나 저장소에 패키지를 설치할 준비가 되었다면 패키지 형식 또는 패키지 관리자 설명서를 참조하세요.

- CodeArtifact를 npm과 함께 사용
- CodeArtifact를 Python과 함께 사용
- CodeArtifact를 Maven과 함께 사용
- CodeArtifact를 NuGet과 함께 사용
- CodeArtifact를 일반 패키지와 함께 사용

CodeArtifact 시작하기

이 시작하기 자습서에서는 CodeArtifact를 사용하여 다음을 생성합니다.

- `my-domain`이라는 도메인.
- `my-domain`에 포함된 `my-repo`라는 이름의 리포지토리.
- `my-domain`에 포함된 `npm-store`라는 이름의 리포지토리. `npm-store`는 npm 퍼블릭 리포지토리에 대한 외부 연결이 있습니다. 이 연결은 npm 패키지를 `my-repo` 리포지토리로 수집하는 데 사용됩니다.

이 자습서를 시작하기 전에 AWS CodeArtifact 개념 CodeArtifact를 검토하는 것이 좋습니다.

> **Note**
>
> 이 자습서에서는 결과적으로 AWS 계정에 요금이 부과될 수 있는 리소스를 생성해야 합니다. 자세한 내용은 CodeArtifact 요금을 참조하세요.

주제

사전 조건

AWS Management Console 또는 AWS Command Line Interface(AWS CLI)를 사용하여 이 자습서를 완료할 수 있습니다. 이 자습서를 따르려면 다음과 같은 사전 조건부터 먼저 완료해야 합니다.

- with AWS CodeArtifact 설정의 단계를 수행하세요.
- npm CLI를 설치합니다. 자세한 내용은 npm 설명서의 Node.js 및 npm 다운로드 및 설치를 참조합니다.

콘솔을 사용하여 시작하기

AWS Management Console을 사용하여 CodeArtifact를 시작하려면 다음 단계를 실행합니다. 이 안내서는 npm 패키지 관리자를 사용합니다. 다른 패키지 관리자를 사용하는 경우 다음 단계 중 일부를 수정해야 합니다.

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/codesuite/codeartifact/start에서 AWS CodeArtifact 콘솔을 엽니다. 자세한 내용은 with AWS CodeArtifact 설정 섹션을 참조하세요.
2. 리포지토리 생성을 선택합니다.
3. 리포지토리 이름에서 **my-repo**를 입력합니다.
4. (선택 사항) 리포지토리 설명에는 이 리포지토리에 대한 설명(선택 사항)을 입력합니다.
5. 퍼블릭 업스트림 리포지토리에서 npm-store를 선택하여 `my-repo` 리포지토리의 업스트림인 npmjs에 연결된 리포지토리를 생성합니다.

   CodeArtifact는 사용자를 대신하여 이 리포지토리에 이름 `npm-store`를 할당합니다. 업스트림 리포지토리 `npm-store`에서 사용할 수 있는 모든 패키지를 다운스트림 리포지토리 `my-repo`에서도 사용할 수 있습니다.
6. 다음을 선택합니다.
7. AWS 계정에서 이 AWS 계정을 선택합니다.
8. 도메인 이름에 **my-domain**을 입력합니다.
9. 추가 구성을 확장합니다.
10. AWS KMS key(KMS 키)를 사용하여 도메인의 모든 자산을 암호화해야 합니다. 관리하는 AWS 관리형 키 또는 KMS 키를 사용할 수 있습니다.

    - 기본 AWS 관리형 키를 사용하려면 AWS 관리형 키를 선택합니다.
    - 관리하는 KMS 키를 사용하려면 고객 관리형 키를 선택합니다. 관리하는 KMS 키를 사용하려면 고객 관리형 키 ARN에서 KMS 키를 검색하여 선택합니다.

    자세한 내용은 AWS Key Management Service 개발자 안내서의 AWS 관리형 키 및 고객 관리형 키를 참조하세요.
11. 다음을 선택합니다.
12. 검토 및 생성에서 CodeArtifact가 무엇을 생성하고 있는지 검토하세요.

- 패키지 흐름은 `my-domain`, `my-repo` 및 `npm-store`가 어떻게 관련되어 있는지를 보여줍니다.
- 1단계: 리포지토리 생성에는 `my-repo` 및 `npm-store`에 관한 세부 정보가 표시됩니다.
- 2단계: 도메인 선택에서는 `my-domain`에 관한 세부 정보가 표시됩니다.

준비가 되었으면 리포지토리 생성을 선택합니다.

13. my-repo 페이지에서 연결 지침 보기를 선택한 다음 npm을 선택합니다.
14. AWS CLI를 사용하여 이 AWS CLI CodeArtifact 명령을 사용하여 npm 클라이언트 구성 아래에 표시된 `login` 명령을 실행합니다.

```
aws codeartifact login --tool npm --repository my-repo --domain my-domain --domain-
owner 111122223333
```

로그인이 성공했음을 확인하는 출력 화면이 표시될 것입니다.

```
Successfully configured npm to use AWS CodeArtifact repository https://my-
domain-111122223333.d.codeartifact.us-east-2.amazonaws.com/npm/my-repo/
Login expires in 12 hours at 2020-10-08 02:45:33-04:00
```

`Could not connect to the endpoint URL` 오류가 표시되는 경우 AWS CLI가 구성되어 있고 기본 지역 이름이 리포지토리를 생성한 리전과 동일한 리전으로 설정되어 있는지 확인합니다. AWS Command Line Interface 구성을 참조합니다.

자세한 내용은 CodeArtifact로 npm 구성 및 사용을 참조합니다.

15. npm CLI를 사용하여 npm 패키지를 설치합니다. 예를 들어, 인기 있는 npm 패키지 `lodash`를 설치하려면 다음 명령을 실행합니다.

```
npm install lodash
```

16. CodeArtifact 콘솔로 돌아갑니다. my-repo 리포지토리가 열려 있는 경우 페이지 새로 고침을 실행합니다. 아니면 탐색 창에서 리포지토리를 선택한 다음 my-repo를 선택합니다.

설치한 npm 라이브러리 또는 패키지가 패키지 아래로 표시될 것입니다. 패키지 이름을 선택하여 패키지의 버전과 상태를 볼 수 있습니다. 최신 버전을 선택하여 종속성, 자산 등과 같은 패키지 세부 정보를 볼 수 있습니다.

> **Note**
>
> 패키지를 설치하는 시점과 리포지토리에 수집되는 시점 간에 지연이 발생할 수 있습니다.

17. 추가 AWS 요금이 부과되지 않도록 하려면 이 자습서에서 사용한 리소스를 삭제하세요.

> **Note**
>
> 리포지토리가 포함된 도메인은 삭제할 수 없으므로 `my-domain`을 삭제하기 전에 먼저 `my-repo`와 `npm-store`를 삭제해야 합니다.

a. 탐색 창에서 리포지토리를 선택합니다.

b. npm-store를 선택하고 삭제를 선택한 다음, 단계에 따라 리포지토리를 삭제합니다.

c. my-repo를 선택하고 삭제를 선택한 다음, 단계에 따라 리포지토리를 삭제합니다.

d. 탐색 창에서 도메인을 선택합니다.

e. my-domain을 선택하고 삭제를 선택한 다음, 단계에 따라 도메인을 삭제합니다.

AWS CLI를 사용하여 시작하기

AWS Command Line Interface(AWS CLI)를 사용하여 CodeArtifact를 시작하려면 다음 단계를 실행합니다. 자세한 내용은 설치 또는 업그레이드 후 구성 AWS CLI 섹션을 참조하세요. 이 안내서는 npm 패키지 관리자를 사용합니다. 다른 패키지 관리자를 사용하는 경우 다음 단계 중 일부를 수정해야 합니다.

1. AWS CLI를 사용하여 create-domain 명령을 실행합니다.

```
aws codeartifact create-domain --domain my-domain
```

JSON 형식의 데이터는 새 도메인에 관한 세부 정보와 함께 출력에 표시됩니다.

```
{
    "domain": {
        "name": "my-domain",
        "owner": "111122223333",
        "arn": "arn:aws:codeartifact:us-west-2:111122223333:domain/my-domain",
```

```
        "status": "Active",
        "createdTime": "2020-10-07T15:36:35.194000-04:00",
        "encryptionKey": "arn:aws:kms:us-west-2:111122223333:key/your-kms-key",
        "repositoryCount": 0,
        "assetSizeBytes": 0
    }
}
```

`Could not connect to the endpoint URL` 오류가 표시되는 경우 AWS CLI가 구성되어 있고 기본 지역 이름이 리포지토리를 생성한 리전과 동일한 리전으로 설정되어 있는지 확인합니다. AWS Command Line Interface 구성을 참조합니다.

2. create-repository 명령을 사용하여 도메인에 리포지토리를 생성합니다.

```
aws codeartifact create-repository --domain my-domain --domain-owner 111122223333
 --repository my-repo
```

JSON 형식의 데이터는 새 리포지토리에 관한 세부 정보와 함께 출력에 표시됩니다.

```
{
    "repository": {
        "name": "my-repo",
        "administratorAccount": "111122223333",
        "domainName": "my-domain",
        "domainOwner": "111122223333",
        "arn": "arn:aws:codeartifact:us-west-2:111122223333:repository/my-
domain/my-repo",
        "upstreams": [],
        "externalConnections": []
    }
}
```

3. create-repository 명령을 실행하여 `my-repo` 리포지토리의 업스트림 리포지토리를 생성합니다.

```
aws codeartifact create-repository --domain my-domain --domain-owner 111122223333
 --repository npm-store
```

JSON 형식의 데이터는 새 리포지토리에 관한 세부 정보와 함께 출력에 표시됩니다.

```
{
    "repository": {
```

```
        "name": "npm-store",
        "administratorAccount": "111122223333",
        "domainName": "my-domain",
        "domainOwner": "111122223333",
        "arn": "arn:aws:codeartifact:us-west-2:111122223333:repository/my-
domain/npm-store",
        "upstreams": [],
        "externalConnections": []
    }
}
```

4. associate-external-connection 명령을 사용하여 npm 퍼블릭 리포지토리에 대한 외부 연결을 npm-store 리포지토리에 추가합니다.

```
aws codeartifact associate-external-connection --domain my-domain --domain-
owner 111122223333 --repository npm-store --external-connection "public:npmjs"
```

JSON 형식의 데이터는 리포지토리와 그것의 새 외부 연결에 관한 세부 정보와 함께 출력 화면에 표시됩니다.

```
{
    "repository": {
        "name": "npm-store",
        "administratorAccount": "111122223333",
        "domainName": "my-domain",
        "domainOwner": "111122223333",
        "arn": "arn:aws:codeartifact:us-west-2:111122223333:repository/my-
domain/npm-store",
        "upstreams": [],
        "externalConnections": [
            {
                "externalConnectionName": "public:npmjs",
                "packageFormat": "npm",
                "status": "AVAILABLE"
            }
        ]
    }
}
```

자세한 내용은 CodeArtifact 저장소를 공용 저장소에 연결하기 섹션을 참조하세요.

5. update-repository 명령을 사용하여 npm-store 리포지토리를 my-repo 리포지토리에 대한 업스트림 리포지토리로 연결합니다.

```
aws codeartifact update-repository --repository my-repo --domain my-domain --
domain-owner 111122223333 --upstreams repositoryName=npm-store
```

JSON 형식의 데이터는 새 업스트림 리포지토리를 포함해 업데이트된 리포지토리에 관한 세부 정보와 함께 출력 화면에 표시됩니다.

```
{
    "repository": {
        "name": "my-repo",
        "administratorAccount": "111122223333",
        "domainName": "my-domain",
        "domainOwner": "111122223333",
        "arn": "arn:aws:codeartifact:us-west-2:111122223333:repository/my-
domain/my-repo",
        "upstreams": [
            {
                "repositoryName": "npm-store"
            }
        ],
        "externalConnections": []
    }
}
```

자세한 내용은 업스트림 리포지토리 추가 또는 제거(AWS CLI) 섹션을 참조하세요.

6. login 명령을 실행하여 my-repo 리포지토리와 함께 npm 패키지 관리자를 구성합니다.

```
aws codeartifact login --tool npm --repository my-repo --domain my-domain --domain-
owner 111122223333
```

로그인이 성공했음을 확인하는 출력 화면이 표시될 것입니다.

```
Successfully configured npm to use AWS CodeArtifact repository https://my-
domain-111122223333.d.codeartifact.us-east-2.amazonaws.com/npm/my-repo/
Login expires in 12 hours at 2020-10-08 02:45:33-04:00
```

자세한 내용은 CodeArtifact로 npm 구성 및 사용 섹션을 참조하세요.

7. npm CLI를 사용하여 npm 패키지를 설치합니다. 예를 들어, 인기 있는 npm 패키지 `lodash`를 설치하려면 다음 명령을 실행합니다.

```
npm install lodash
```

8. list-packages 명령을 실행하면 `my-repo` 리포지토리에 방금 설치한 패키지를 볼 수 있습니다.

> **Note**
>
> `npm install` 설치 명령이 완료되는 시점과 패키지가 리포지토리에 표시되는 시점 간에 지연이 발생할 수 있습니다. 퍼블릭 리포지토리에서 패키지를 가져올 때 일반적인 지연 시간에 관한 자세한 내용은 외부 연결 지연 시간를 참조하세요.

```
aws codeartifact list-packages --domain my-domain --repository my-repo
```

JSON 형식의 데이터는 설치한 패키지의 형식 및 이름과 함께 출력 화면에 표시됩니다.

```
{
    "packages": [
        {
            "format": "npm",
            "package": "lodash"
        }
    ]
}
```

이제 다음과 같이 세 개의 CodeArtifact 리소스가 있습니다.

- 도메인 `my-domain`.
- `my-domain`에 포함된 `my-repo`라는 이름의 리포지토리. 이 리포지토리에는 npm 패키지가 있습니다.
- `my-domain`에 포함된 `npm-store`라는 이름의 리포지토리. 이 리포지토리는 퍼블릭 npm 리포지토리에 대한 외부 연결이 있으며 업스트림 리포지토리로서 `my-repo` 리포지토리와 연결되어 있습니다.

9. 추가 AWS 요금이 부과되지 않도록 하려면 이 자습서에서 사용한 리소스를 삭제하세요.

> **Note**
>
> 리포지토리가 포함된 도메인은 삭제할 수 없으므로 `my-domain`을 삭제하기 전에 먼저 `my-repo`와 `npm-store`를 삭제해야 합니다.

a. `npm-store` 리포지토리를 삭제하려면 delete-repository 명령을 사용합니다.

```
aws codeartifact delete-repository --domain my-domain --domain-
owner 111122223333 --repository my-repo
```

JSON 형식의 데이터는 삭제된 리포지토리에 관한 세부 정보와 함께 출력 화면에 표시됩니다.

```
{
    "repository": {
        "name": "my-repo",
        "administratorAccount": "111122223333",
        "domainName": "my-domain",
        "domainOwner": "111122223333",
        "arn": "arn:aws:codeartifact:us-west-2:111122223333:repository/my-
domain/my-repo",
        "upstreams": [
            {
                "repositoryName": "npm-store"
            }
        ],
        "externalConnections": []
    }
}
```

b. `npm-store` 리포지토리를 삭제하려면 delete-repository 명령을 사용합니다.

```
aws codeartifact delete-repository --domain my-domain --domain-
owner 111122223333 --repository npm-store
```

JSON 형식의 데이터는 삭제된 리포지토리에 관한 세부 정보와 함께 출력 화면에 표시됩니다.

```
{
    "repository": {
        "name": "npm-store",
```

```
        "administratorAccount": "111122223333",
        "domainName": "my-domain",
        "domainOwner": "111122223333",
        "arn": "arn:aws:codeartifact:us-west-2:111122223333:repository/my-
domain/npm-store",
        "upstreams": [],
        "externalConnections": [
            {
                "externalConnectionName": "public:npmjs",
                "packageFormat": "npm",
                "status": "AVAILABLE"
            }
        ]
    }
}
```

c. my-domain 리포지토리를 삭제하려면 delete-domain 명령을 사용합니다.

```
aws codeartifact delete-domain --domain my-domain --domain-owner 111122223333
```

JSON 형식의 데이터는 삭제된 도메인에 관한 세부 정보와 함께 출력 화면에 표시됩니다.

```
{
    "domain": {
        "name": "my-domain",
        "owner": "111122223333",
        "arn": "arn:aws:codeartifact:us-west-2:111122223333:domain/my-domain",
        "status": "Deleted",
        "createdTime": "2020-10-07T15:36:35.194000-04:00",
        "encryptionKey": "arn:aws:kms:us-west-2:111122223333:key/your-kms-key",
        "repositoryCount": 0,
        "assetSizeBytes": 0
    }
}
```

CodeArtifact에서의 리포지토리 작업

이 주제에서는 CodeArtifact 콘솔 및 CodeArtifact APIs를 사용하여 리포지토리를 생성 AWS CLI, 나열, 업데이트 및 삭제하는 방법을 보여줍니다.

주제

리포지토리 생성

CodeArtifact에서 모든 패키지는 리포지토리에 저장되므로, CodeArtifact를 사용하려면 리포지토리를 만들어야 합니다. CodeArtifact 콘솔, AWS Command Line Interface (AWS CLI) 또는를 사용하여 리포지토리를 생성할 수 있습니다 CloudFormation. 각 리포지토리는 생성할 때 사용하는 AWS 계정과 연결됩니다. 여러 리포지토리를 가질 수 있으며, 이러한 리포지토리는 도메인에서 생성되고 그룹화됩니다. 생성되는 리포지토리에는 패키지가 포함되지 않습니다. 리포지토리는 다국어 개체이며, 따라서 지원되는 모든 유형의 패키지가 단일 리포지토리에 포함될 수 있습니다.

단일 도메인에 허용되는 리포지토리 최대 수 같은 CodeArtifact 서비스 제한에 대한 자세한 내용은 in AWS CodeArtifact 할당량 섹션을 참조하세요. 허용된 리포지토리 최대 수에 도달한 경우, 리포지토리를 삭제하여 추가 공간을 확보할 수 있습니다.

리포지토리에는 하나 이상의 CodeArtifact 리포지토리가 업스트림 리포지토리로 연결될 수 있습니다. 이렇게 하면 패키지 관리자 클라이언트가 단일 URL 엔드포인트를 사용하여 둘 이상의 리포지토리에 포함된 패키지에 액세스할 수 있습니다. 자세한 내용은 CodeArtifact에서의 업스트림 리포지토리 작업 단원을 참조하십시오.

CloudFormation으로 CodeArtifact 리포지토리를 관리하는 방법에 대한 자세한 내용은 AWS CloudFormation을 사용하여 CodeArtifact 리소스 생성 섹션을 참조하십시오.

> **Note**
>
> 리포지토리를 생성한 후에는 리포지토리 이름, 관련 AWS 계정 또는 도메인을 변경할 수 없습니다.

주제

리포지토리 생성(콘솔)

1. https://console.aws.amazon.com/codesuite/codeartifact/home AWS CodeArtifact 콘솔을 엽니다.
2. 탐색 창에서 리포지토리를 선택한 다음 리포지토리 생성을 선택합니다.
3. 리포지토리 이름에 리포지토리의 고유한 이름을 입력합니다.
4. (선택 사항) 리포지토리 설명에 리포지토리에 대한 설명을 입력합니다.
5. (선택 사항) 업스트림 리포지토리 게시에 Maven Central이나 npmjs.com 같은, 리포지토리를 패키지 기관에 연결하는 중간 리포지토리를 추가합니다.
6. 다음을 선택합니다.
7. AWS 계정에서, 도메인을 소유한 계정으로 로그인한 경우 이 AWS 계정을 선택합니다. 다른 AWS 계정이 도메인을 소유하고 있는 경우 다른 AWS 계정을 선택합니다.
8. 도메인에서 리포지토리를 생성할 도메인을 선택합니다.

   계정에 도메인이 없는 경우 도메인을 만들어야 합니다. 도메인 이름에 새 도메인의 이름을 입력합니다.

   추가 구성을 확장합니다.

   도메인의 모든 자산을 암호화하려면 AWS KMS key (KMS 키)를 사용해야 합니다. AWS 관리형 키 또는 관리하는 KMS 키를 사용할 수 있습니다.

> **Important**
>
> CodeArtifact는 대칭 KMS 키만 지원합니다. 비대칭 KMS 키를 사용하여 CodeArtifact 도메인을 암호화할 수 없습니다. KMS 키가 대칭인지 비대칭인지 확인하는 방법은 대칭 및 비대칭 KMS키 식별를 참조하세요.

- 기본 AWS 관리형 키를 사용하려면 AWS 관리형 키를 선택합니다.
- 관리하는 KMS 키를 사용하려면 고객 관리형 키를 선택합니다. 관리하는 KMS 키를 사용하려면 고객 관리형 키 ARN에서 KMS 키를 검색하여 선택합니다.

자세한 내용은 AWS Key Management Service 개발자 안내서의 AWS 관리형 키 및 고객 관리형 키를 참조하세요.

9. 다음을 선택합니다.
10. 검토 및 생성에서 CodeArtifact가 무엇을 생성하고 있는지 검토하세요.

- 패키지 흐름에서는 도메인과 리포지토리 연결 상태를 확인할 수 있습니다.
- 1단계: 리포지토리 생성에서는 생성될 리포지토리 및 선택적 업스트림 리포지토리 관련 세부 정보가 표시됩니다.
- 2단계: 도메인 선택에서는 `my_domain`에 관한 세부 정보가 표시됩니다.

준비가 되었으면 리포지토리 생성을 선택합니다.

리포지토리 생성(AWS CLI)

`create-repository` 명령을 사용하여 도메인에 리포지토리를 생성합니다.

```
aws codeartifact create-repository --domain my_domain --domain-owner 111122223333 --
repository my_repo --description "My new repository"
```

출력 예시:

```
{
    "repository": {
        "name": "my_repo",
```

```
        "administratorAccount": "123456789012",
        "domainName": "my_domain",
        "domainOwner": "111122223333",
        "arn": "arn:aws:codeartifact:region-
id:111122223333:repository/my_domain/my_repo",
        "description": "My new repository",
        "upstreams": "[]",
        "externalConnections"" "[]"
    }
}
```

새 리포지토리에는 패키지가 포함되어 있지 않습니다. 각 리포지토리는 리포지토리가 생성될 때 인증된 AWS 계정과 연결됩니다.

태그를 사용하여 리포지토리 생성

태그가 있는 리포지토리를 만들려면 `create-domain` 명령에 `--tags` 파라미터를 추가합니다.

```
aws codeartifact create-repository --domain my_domain --domain-owner 111122223333 --
repository my_repo --tags key=k1,value=v1 key=k2,value=v2
```

업스트림 리포지토리를 사용하여 리포지토리 생성

리포지토리를 생성할 때 업스트림 리포지토리를 하나 이상 지정할 수 있습니다.

```
aws codeartifact create-repository --domain my_domain --domain-owner 111122223333 --
repository my_repo \
  --upstreams repositoryName=my-upstream-repo --repository-description "My new
 repository"
```

출력 예시:

```
{
    "repository": {
        "name": "my_repo",
        "administratorAccount": "123456789012",
        "domainName": "my_domain",
        "domainOwner": "111122223333",
        "arn": "arn:aws:codeartifact:region-
id:111122223333:repository/my_domain/my_repo",
        "description": "My new repository",
```

```
        "upstreams": [
            {
                "repositoryName": "my-upstream-repo"
            }
        ],
        "externalConnections"" "[]"
    }
}
```

> **Note**
>
> 업스트림을 사용하여 리포지토리를 만들려면 업스트림 리포지토리에서 `AssociateWithDownstreamRepository` 작업을 수행하는 데 필요한 권한이 있어야 합니다.

생성된 리포지토리에 업스트림을 추가하려면 업스트림 리포지토리 추가 또는 제거(콘솔) 및업스트림 리포지토리 추가 또는 제거(AWS CLI) 섹션을 참조하세요.

리포지토리에 연결

AWS 계정을 인증하도록 프로필과 보안 인증을 구성한 후에는, CodeArtifact에서 사용할 리포지토리를 결정합니다. 다음과 같은 옵션이 있습니다.

- 리포지토리를 생성합니다. 자세한 내용은 리포지토리 생성을 참조하세요.
- 계정에 이미 존재하는 리포지토리를 사용합니다. `list-repositories` 명령을 사용하면 AWS 계정에 생성된 리포지토리를 찾을 수 있습니다. 자세한 내용은 리포지토리 나열 단원을 참조하십시오.
- 다른 AWS 계정의 리포지토리를 사용합니다. 자세한 내용은 리포지토리 정책을 참조하세요.

패키지 관리자 클라이언트 사용

사용할 리포지토리를 확인한 후에는 다음 주제 중 하나를 참조하세요.

- CodeArtifact를 Maven과 함께 사용
- CodeArtifact를 npm과 함께 사용
- CodeArtifact를 NuGet과 함께 사용
- CodeArtifact를 Python과 함께 사용

리포지토리 삭제

CodeArtifact 콘솔 또는 AWS CLI를 사용하여 리포지토리를 삭제할 수 있습니다. 리포지토리를 삭제한 후에는 패키지를 리포지토리로 푸시하거나 리포지토리에서 패키지를 가져올 수 없습니다. 리포지토리에 있는 모든 패키지는 영구적으로 사용 불가능하며 복원할 수 없게 됩니다. 이름이 같은 리포지토리를 만들 수 있지만 이러한 리포지토리에는 아무 내용도 들어 있지 않습니다.

> **Important**
>
> 리포지토리 삭제는 실행 취소할 수 없습니다. 리포지토리를 삭제한 후에는 더 이상 복구 및 복원할 수 없습니다.

주제

리포지토리 삭제(콘솔)

1. https://console.aws.amazon.com/codesuite/codeartifact/home AWS CodeArtifact 콘솔을 엽니다.
2. 탐색 창에서 리포지토리를 선택한 다음 삭제할 리포지토리를 선택합니다.
3. 삭제를 선택한 다음, 단계에 따라 도메인을 삭제합니다.

리포지토리 삭제(AWS CLI)

리포지토리를 삭제하려면 `delete-repository` 명령을 사용합니다.

```
aws codeartifact delete-repository --domain my_domain --domain-owner 111122223333 --
repository my_repo
```

출력 예시:

```
{
    "repository": {
        "name": "my_repo",
```

```
        "administratorAccount": "123456789012",
        "domainName": "my_domain",
        "domainOwner": "123456789012",
        "arn": "arn:aws:codeartifact:region-
id:123456789012:repository/my_domain/my_repo",
        "description": "My new repository",
        "upstreams": [],
        "externalConnections": []
    }
}
```

리포지토리가 삭제되지 않도록 보호

다음과 유사한 도메인 정책을 포함하면 리포지토리가 실수로 삭제되는 것을 방지할 수 있습니다.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DenyRepositoryDeletion",
            "Action": [
                "codeartifact:DeleteRepository"
            ],
            "Effect": "Deny",
            "Resource": "*",
            "Principal": "*"
        }
    ]
}
```

이 정책은 모든 보안 주체가 리포지토리를 삭제하는 것을 방지하지만 나중에 리포지토리를 삭제해야 한다고 결정하는 경우 다음 단계에 따라 삭제할 수 있습니다.

1. 도메인 정책에서 정책을 다음으로 업데이트합니다.

   JSON

   ```
   {
   ```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DenyRepositoryDeletion",
            "Action": [
                "codeartifact:DeleteRepository"
            ],
            "Effect": "Deny",
            "NotResource": "arn:aws:iam::*:role/Service*",
            "Principal": "*"
        }
    ]
}
```

*repository-arn*을 삭제하려는 리포지토리의 ARN으로 바꿉니다.

2. AWS CodeArtifact 콘솔에서 리포지토리를 선택하고 선택한 리포지토리를 삭제합니다.
3. 리포지토리를 삭제한 후 정책을 다시 변경하여 실수로 삭제하지 않도록 할 수 있습니다.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DenyRepositoryDeletion",
            "Action": [
                "codeartifact:DeleteRepository"
            ],
            "Effect": "Deny",
            "Resource": "*",
            "Principal": "*"
        }
    ]
}
```

또는 리포지토리 정책에 동일한 거부 명령문을 포함할 수 있습니다. 이를 통해 높은 가치의 리포지토리가 삭제되는 것을 방지할 수 있는 유연성을 높일 수 있습니다.

리포지토리 나열

이 주제의 명령을 사용하여 AWS 계정 또는 도메인의 리포지토리를 나열합니다.

AWS 계정의 리포지토리 나열

이 명령을 사용하여 AWS 계정의 모든 리포지토리를 나열합니다.

```
aws codeartifact list-repositories
```

샘플 출력:

```
{
    "repositories": [
        {
            "name": "repo1",
            "administratorAccount": "123456789012",
            "domainName": "my_domain",
            "domainOwner": "123456789012",
            "arn": "arn:aws:codeartifact:region-
id:123456789012:repository/my_domain/repo1",
            "description": "Description of repo1"
        },
        {
            "name": "repo2",
            "administratorAccount": "123456789012",
            "domainName": "my_domain",
            "domainOwner": "123456789012",
            "arn": "arn:aws:codeartifact:region-
id:123456789012:repository/my_domain/repo2",
            "description": "Description of repo2"

        },
        {
            "name": "repo3",
            "administratorAccount": "123456789012",
            "domainName": "my_domain2",
            "domainOwner": "123456789012",
            "arn": "arn:aws:codeartifact:region-
id:123456789012:repository/my_domain2/repo3",
            "description": "Description of repo3"
        }
```

```
    ]
}
```

--max-results 및 --next-token 파라미터를 사용하여 list-repositories의 응답에 페이지를 매길 수 있습니다. --max-results의 경우 1에서 1000 사이의 정수를 지정하여 한 페이지에 반환되는 결과 수를 지정합니다. 기본값은 50입니다. 후속 페이지를 반환하려면 list-repositories를 다시 실행하고 이전 명령 출력에서 받은 nextToken 값을 --next-token에 전달하세요. --next-token 옵션을 사용하지 않으면 항상 결과의 첫 페이지가 반환됩니다.

도메인에 있는 리포지토리 나열

도메인에 있는 모든 리포지토리 목록을 얻으려면 list-repositories-in-domain 명령을 사용합니다.

```
aws codeartifact list-repositories-in-domain --domain my_domain --domain-
owner 123456789012 --max-results 3
```

출력을 보면 일부 리포지토리가 서로 다른 AWS 계정에서 관리된다는 사실을 알 수 있습니다.

```
{
    "repositories": [
        {
            "name": "repo1",
            "administratorAccount": "123456789012",
            "domainName": "my_domain",
            "domainOwner": "111122223333",
            "arn": "arn:aws:codeartifact:region-
id:111122223333:repository/my_domain/repo1",
            "description": "Description of repo1"
        },
        {
            "name": "repo2",
            "administratorAccount": "444455556666",
            "domainName": "my_domain",
            "domainOwner": "111122223333",
            "arn": "arn:aws:codeartifact:region-
id:111122223333:repository/my_domain/repo2",
            "description": "Description of repo2"
        },
        {
            "name": "repo3",
```

```
            "administratorAccount": "444455556666",
            "domainName": "my_domain",
            "domainOwner": "111122223333",
            "arn": "arn:aws:codeartifact:region-
id:111122223333:repository/my_domain/repo3",
            "description": "Description of repo3"
        }
    ]
}
```

--max-results 및 --next-token 파라미터를 사용하여 list-repositories-in-domain의 응답에 페이지를 매길 수 있습니다. --max-results의 경우 1에서 1000 사이의 정수를 지정하여 한 페이지에 반환되는 결과 수를 지정합니다. 기본값은 50입니다. 후속 페이지를 반환하려면 list-repositories-in-domain를 다시 실행하고 이전 명령 출력에서 받은 nextToken 값을 --next-token에 전달하세요. --next-token 옵션을 사용하지 않으면 항상 결과의 첫 페이지가 반환됩니다.

리포지토리 이름을 보다 간결한 목록으로 출력하려면 다음 명령을 사용해 보세요.

```
aws codeartifact list-repositories-in-domain --domain my_domain --domain-
owner 111122223333 \
  --query 'repositories[*].[name]' --output text
```

샘플 출력:

```
repo1
repo2
repo3
```

다음 예제 출력에서는 리포지토리 이름과 계정 ID가 함께 출력됩니다.

```
aws codeartifact list-repositories-in-domain --domain my_domain --domain-
owner 111122223333  \
  --query 'repositories[*].[name,administratorAccount]' --output text
```

샘플 출력:

```
repo1 710221105108
repo2 710221105108
repo3 532996949307
```

`--query` 파라미터에 대해 자세히 알아보려면 CodeArtifact API 참조의 ListRepositories를 참조하세요.

리포지토리 구성 보기 또는 수정

CodeArtifact 콘솔 또는 AWS Command Line Interface (AWS CLI)를 사용하여 리포지토리 세부 정보를 보고 업데이트할 수 있습니다.

> **Note**
>
> 리포지토리를 생성한 후에는 리포지토리 이름, 관련 AWS 계정 또는 도메인을 변경할 수 없습니다.

주제

리포지토리 구성 보기 또는 수정(콘솔)

CodeArtifact 콘솔을 사용하여 리포지토리 세부 정보를 보고 리포지토리를 업데이트할 수 있습니다.

1. https://console.aws.amazon.com/codesuite/codeartifact/home AWS CodeArtifact 콘솔을 엽니다.
2. 탐색 창에서 리포지토리를 선택한 다음 보거나 수정할 리포지토리를 선택합니다.
3. 세부 정보를 확장해 다음 정보를 확인합니다.

   - 리포지토리의 도메인. 도메인 이름을 선택하면 자세한 정보를 확인할 수 있습니다.
   - 리포지토리의 리소스 정책. 리포지토리 정책 적용을 선택하여 정책을 추가합니다.
   - 리포지토리의 Amazon 리소스 이름(ARN).
   - 리포지토리에 외부 연결이 있는 경우 연결을 선택하면 자세한 정보를 확인할 수 있습니다. 리포지토리는 외부 연결을 하나만 가질 수 있습니다. 자세한 내용은 CodeArtifact 저장소를 공용 저장소에 연결하기 단원을 참조하십시오.
   - 리포지토리에 업스트림 리포지토리가 있는 경우 하나를 선택하면 세부 정보를 볼 수 있습니다. 단일 리포지토리에는 최대 10개의 직접 업스트림 리포지토리가 존재할 수 있습니다. 자세한 내용은 CodeArtifact에서의 업스트림 리포지토리 작업 단원을 참조하십시오.

> **Note**
>
> 리포지토리에는 외부 연결 또는 업스트림 리포지토리가 존재할 수 있지만 두 항목이 모두 존재할 수는 없습니다.

4. 패키지에서는 이 리포지토리에 사용할 수 있는 모든 패키지를 볼 수 있습니다. 패키지를 선택하면 자세한 내용을 확인할 수 있습니다.
5. 연결 지침 보기를 선택한 다음 패키지 관리자를 선택하여 CodeArtifact를 이용해 구성하는 방법을 알아보세요.
6. 리포지토리 정책 적용을 선택하여 리소스 정책을 업데이트하거나 리포지토리에 추가합니다. 자세한 내용은 리포지토리 정책 단원을 참조하십시오.
7. 편집을 선택하여 다음을 추가하거나 업데이트합니다.

   - 리포지토리 설명.
   - 리포지토리에 연결된 태그.
   - 리포지토리에 외부 연결이 있는 경우, 리포지토리가 연결되는 퍼블릭 리포지토리를 변경할 수 있습니다. 외부 연결이 없다 하나 이상의 기존 리포지토리를 업스트림 리포지토리로 추가할 수 있습니다. 패키지가 요청되면, CodeArtifact에서 우선 순위를 지정하려는 순서대로 정렬합니다. 자세한 내용은 업스트림 리포지토리 우선순위 순서 단원을 참조하십시오.

리포지토리 구성 보기 또는 수정(AWS CLI)

CodeArtifact에서 리포지토리의 현재 구성을 보려면 `describe-repository` 명령을 사용합니다.

```
aws codeartifact describe-repository --domain my_domain --domain-owner 111122223333 --
repository my_repo
```

출력 예시:

```
{
    "repository": {
        "name": "my_repo",
        "administratorAccount": "123456789012",
        "domainName": "my_domain",
        "domainOwner": "111122223333",
```

```
        "arn": "arn:aws:codeartifact:region-
id:111122223333:repository/my_domain/my_repo"
        "upstreams": [],
        "externalConnections": []
    }
}
```

리포지토리 업스트림 구성 수정

업스트림 리포지토리를 사용하면 패키지 관리자 클라이언트가 단일 URL 엔드포인트를 사용하여 둘 이상의 리포지토리에 포함된 패키지에 액세스할 수 있습니다. 리포지토리의 업스트림 관계를 추가하거나 변경하려면 `update-repository` 명령을 사용합니다.

```
aws codeartifact update-repository --domain my_domain --domain-owner 111122223333 --
repository my_repo \
       --upstreams repositoryName=my-upstream-repo
```

출력 예시:

```
{
    "repository": {
        "name": "my_repo",
        "administratorAccount": "123456789012,
        "domainName": "my_domain",
        "domainOwner": "111122223333",
        "arn": "arn:aws:codeartifact:region-
id:111122223333:repository/my_domain/my_repo"
        "upstreams": [
            {
                "repositoryName": "my-upstream-repo"
            }
        ],
        "externalConnections": []
    }
}
```

> **Note**
>
> 업스트림 리포지토리를 추가하려면 업스트림 리포지토리에서 `AssociateWithDownstreamRepository` 작업을 수행하는 데 필요한 권한이 있어야 합니다.

리포지토리의 업스트림 관계를 제거하려면 빈 목록을 `--upstreams` 옵션의 인수로 사용해야 합니다.

```
aws codeartifact update-repository --domain my_domain --domain-owner 111122223333 --
repository my_repo --upstreams []
```

출력 예시:

```
{
    "repository": {
        "name": "my_repo",
        "administratorAccount": "123456789012,
        "domainName": "my_domain",
        "domainOwner": "111122223333",
        "arn": "arn:aws:codeartifact:region-
id:111122223333:repository/my_domain/my_repo"
        "upstreams": [],
        "externalConnections": []
    }
}
```

리포지토리 정책

CodeArtifact는 리소스 기반 권한을 사용하여 액세스를 제어합니다. 리소스 기반 권한을 사용하면 리포지토리에 액세스할 수 있는 사용자 및 이러한 사용자가 리포지토리에서 수행할 수 있는 작업을 지정할 수 있습니다. 기본적으로 리포지토리 소유자만 리포지토리에 액세스할 수 있습니다. 사용자는 다른 IAM 보안 주체가 자신의 리포지토리에 액세스하도록 허용하는 정책 문서를 적용할 수 있습니다.

자세한 내용은 리소스 기반 정책과 자격 증명 기반 정책 및 리소스 기반 정책을 참조하세요.

리소스 정책을 생성하여 읽기 액세스 권한 부여

리소스 정책은 JSON 형식의 텍스트 파일입니다. 파일은 보안 주체(액터), 하나 이상의 작업 및 효과(`Allow` 또는 `Deny`)를 지정해야 합니다. 예를 들어 다음 리소스 정책은 리포지토리에서 패키지를 다운로드할 수 있는 `123456789012` 권한을 계정에 부여합니다.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "codeartifact:ReadFromRepository"
            ],
            "Effect": "Allow",
            "Principal": {
                 "AWS": "arn:aws:iam::123456789012:root"
            },
            "Resource": "*"
        }
    ]
}
```

정책은 정책이 연결된 리포지토리 대상 작업에 대해서만 평가되므로, 리소스를 지정하지 않아도 됩니다. 리소스는 암시적이기 때문에 `Resource`를 `*`로 설정할 수 있습니다. 패키지 관리자가 이 리포지토리에서 패키지를 다운로드하려면 교차 계정 액세스를 위한 도메인 정책도 만들어야 합니다. 도메인 정책은 보안 주체에 최소한 `codeartifact:GetAuthorizationToken` 권한을 부여해야 합니다. 교차 계정 액세스 권한을 부여하는 전체 도메인 정책 예제는 도메인 정책 예제 섹션을 참조하세요.

> **Note**
>
> `codeartifact:ReadFromRepository` 작업은 리포지토리 리소스에서만 사용할 수 있습니다. `codeartifact:ReadFromRepository`가 있는 리소스인 패키지의 Amazon 리소스 이름(ARN)은 리포지토리에 있는 패키지의 하위 집합에 대한 읽기 액세스를 허용하는 작업으로 넣을 수 없습니다. 지정된 보안 주체는 리포지토리의 모든 패키지를 읽을 수도 있고 아무 패키지도 읽지 않을 수 있습니다.

리포지토리에서 지정된 작업은 ReadFromRepository뿐이므로, 계정 1234567890의 사용자 및 역할은 리포지토리에서 패키지를 다운로드할 수 있습니다. 하지만 다른 작업(예: 패키지 이름 및 버전 나열)은 수행할 수 없습니다. 리포지토리에서 패키지를 다운로드하는 사용자는 다른 방법으로도 패키지와 상호 작용해야 하니, 대부분의 경우 사용자는 ReadFromRepository와 다음 정책에서 권한을 부여합니다.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "codeartifact:DescribePackageVersion",
                "codeartifact:DescribeRepository",
                "codeartifact:GetPackageVersionReadme",
                "codeartifact:GetRepositoryEndpoint",
                "codeartifact:ListPackages",
                "codeartifact:ListPackageVersions",
                "codeartifact:ListPackageVersionAssets",
                "codeartifact:ListPackageVersionDependencies",
                "codeartifact:ReadFromRepository"
            ],
            "Effect": "Allow",
            "Principal": {
                 "AWS": "arn:aws:iam::123456789012:root"
            },
            "Resource": "*"
        }
    ]
}
```

정책 설정

정책 문서를 만든 후에는 put-repository-permissions-policy 명령을 사용하여 정책 문서를 리포지토리에 첨부합니다.

```
aws codeartifact put-repository-permissions-policy --domain my_domain --domain-
owner 111122223333 \
          --repository my_repo --policy-document file:///PATH/TO/policy.json
```

put-repository-permissions-policy를 호출하면 권한을 평가할 때 리포지토리의 리소스 정책이 무시됩니다. 이렇게 하면 도메인 소유자가 자기 자신을 리포지토리 외부에 고립되게 해 리소스 정책을 업데이트하지 못하는 일을 방지할 수 있습니다.

> **Note**
>
> put-repository-permissions-policy를 호출할 때 리소스 정책이 무시되므로 리소스 정책을 사용하여 리포지토리의 리소스 정책을 업데이트할 수 있는 권한을 다른 AWS 계정에 부여할 수 없습니다.

샘플 출력:

```
{
    "policy": {
        "resourceArn": "arn:aws:codeartifact:region-
id:111122223333:repository/my_domain/my_repo",
        "document": "{ ...policy document content...}",
        "revision": "MQlyyTQRASRU3HB58gBtSDHXG7Q3hvxxxxxxx="
    }
}
```

명령 결과에는 리포지토리 리소스의 Amazon 리소스 이름(ARN), 정책 문서의 전체 내용과 개정 식별자가 포함됩니다. --policy-revision 옵션을 사용하여 개정 식별자를 put-repository-permissions-policy에 전달할 수 있습니다. 이렇게 하면 문서의 알려진 개정은 덮어쓰고, 다른 작성자가 설정한 최신 버전은 덮어쓰지 않게 됩니다.

정책 읽기

get-repository-permissions-policy 명령을 사용하면 정책 문서의 기존 버전을 읽을 수 있습니다. 읽기 쉬운 출력 형식을 지정하려면 --output 및 --query policy.document를 Python json.tool 모듈과 함께 사용하세요.

```
aws codeartifact get-repository-permissions-policy --domain my_domain --domain-
owner 111122223333 \
          --repository my_repo --output text --query policy.document | python -m
 json.tool
```

샘플 출력:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:root"
            },
            "Action": [
                "codeartifact:DescribePackageVersion",
                "codeartifact:DescribeRepository",
                "codeartifact:GetPackageVersionReadme",
                "codeartifact:GetRepositoryEndpoint",
                "codeartifact:ListPackages",
                "codeartifact:ListPackageVersions",
                "codeartifact:ListPackageVersionAssets",
                "codeartifact:ListPackageVersionDependencies",
                "codeartifact:ReadFromRepository"
            ],
            "Resource": "*"
        }
    ]
}
```

정책 삭제

`delete-repository-permissions-policy` 명령을 사용하여 리포지토리에서 정책을 삭제합니다.

```
aws codeartifact delete-repository-permissions-policy --domain my_domain --domain-
owner 111122223333 \
          --repository my_repo
```

출력의 형식은 `get-repository-permissions-policy` 명령의 형식과 동일합니다.

보안 주체에게 읽기 액세스 권한 부여

계정의 루트 사용자를 정책 문서의 보안 주체로 지정하면, 계정의 모든 사용자와 역할에 액세스 권한을 부여하게 됩니다. 선택한 사용자 또는 역할에 대한 액세스를 제한하려면 정책의 `Principal` 섹션에서 관련 ARN을 사용하세요. 예를 들어 다음을 사용하여 계정 123456789012의 IAM 사용자 bob에게 읽기 권한을 부여할 수 있습니다.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "codeartifact:ReadFromRepository"
            ],
            "Effect": "Allow",
            "Principal": {
                 "AWS": "arn:aws:iam::123456789012:user/bob"
            },
            "Resource": "*"
        }
    ]
}
```

패키지에 쓰기 액세스 권한 부여

`codeartifact:PublishPackageVersion` 작업은 패키지의 새 버전을 게시할 권한을 제어하는 용도로 사용합니다. 이 작업에 사용하는 리소스는 반드시 패키지여야 합니다. CodeArtifact 패키지 ARN의 형식은 다음과 같습니다.

```
arn:aws:codeartifact:region-id:111122223333:package/my_domain/my_repo/package-format/package-namespace/package-name
```

다음 예제에서는 도메인 `my_domain`의 `my_repo` 리포지토리에 있는, 범위가 `@parity`고 이름이 `ui`인 npm 패키지의 ARN을 확인할 수 있습니다.

```
arn:aws:codeartifact:region-id:111122223333:package/my_domain/my_repo/npm/parity/ui
```

범위가 없는 npm 패키지의 ARN은 네임스페이스 필드에 빈 문자열이 있습니다. 예를 들어 `my_domain` 도메인의 `my_repo` 리포지토리에 있는, 범위가 없고 이름이 `react`인 패키지의 ARN은 다음과 같습니다.

```
arn:aws:codeartifact:region-id:111122223333:package/my_domain/my_repo/npm//react
```

다음 정책은 `my_repo` 리포지토리에 `@parity/ui`의 버전을 게시할 수 있는 123456789012 권한을 계정에 부여합니다.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "codeartifact:PublishPackageVersion"
            ],
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:root"
            },
            "Resource": "arn:aws:codeartifact:us-
east-1:111122223333:package/my_domain/my_repo/npm/parity/ui"
        }
    ]
}
```

Important

Maven 및 NuGet 패키지 버전을 게시할 수 있는 권한을 부여하려면 `codeartifact:PublishPackageVersion`에 더해 다음 권한을 추가하세요.

1. NuGet: `codeartifact:ReadFromRepository` 및 리포지토리 리소스 지정
2. Maven: `codeartifact:PutPackageMetadata`

이 정책은 도메인과 리포지토리를 리소스의 일부로 지정하기 때문에, 해당 리포지토리에 연결된 경우에만 게시를 허용합니다.

리포지토리에 대한 쓰기 액세스 권한 부여

와일드카드를 사용하여 리포지토리에 있는 모든 패키지에 쓰기 권한을 부여할 수 있습니다. 예를 들어 다음 정책을 사용하면 `my_repo` 리포지토리에 있는 모든 패키지에 대한 쓰기 권한을 계정에 부여할 수 있습니다.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "codeartifact:PublishPackageVersion"
            ],
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:root"
            },
            "Resource": "arn:aws:codeartifact:us-
east-1:111122223333:package/my_domain/my_repo/*"
        }
    ]
}
```

리포지토리와 도메인 정책 간의 상호 작용

CodeArtifact는 도메인 및 리포지토리에 대한 리소스 정책을 지원합니다. 리소스 정책은 선택 사항입니다. 각 도메인에는 하나의 정책이 있을 수 있으며 도메인의 각 리포지토리에는 자체 리포지토리 정책이 있을 수 있습니다. 도메인 정책과 리포지토리 정책이 모두 있는 경우 CodeArtifact 리포지토리에 대한 요청이 허용 또는 거부되는지 여부를 결정할 때 둘 모두 평가됩니다. 도메인 및 리포지토리 정책은 다음 규칙을 사용하여 평가됩니다.

- ListDomains 또는 ListRepositories와 같은 계정 수준 작업을 수행할 때는 리소스 정책이 평가되지 않습니다.
- DescribeDomain 또는 ListRepositoriesInDomain과 같은 도메인 수준 작업을 수행할 때는 리포지토리 정책이 평가되지 않습니다.

- PutDomainPermissionsPolicy를 수행할 때는 도메인 정책이 평가되지 않습니다. 이 규칙은 잠금을 방지합니다.
- 도메인 정책은 PutRepositoryPermissionsPolicy를 수행할 때 평가되지만 리포지토리 정책은 평가되지 않습니다.
- 정책의 명시적 거부는 다른 정책의 허용을 무시합니다.
- 명시적 허용은 하나의 리소스 정책에서만 필요합니다. 리포지토리 정책에서 작업을 생략해도 도메인 정책에서 작업을 허용하는 경우 암시적 거부가 발생하지 않습니다.
- 작업을 허용하는 리소스 정책이 없는 경우 직접 호출하는 보안 주체의 계정이 도메인 소유자 또는 리포지토리 관리자 계정이고 자격 증명 기반 정책이 작업을 허용하지 않는 한 결과는 암시적 거부입니다.

리소스 정책은 리포지토리에 액세스하는 데 사용되는 호출자 계정이 도메인 소유자 및 리포지토리 관리자 계정과 동일한 단일 계정 시나리오에서 액세스 권한을 부여하는 데 사용되는 경우 선택 사항입니다. 호출자의 계정이 도메인 소유자 또는 리포지토리 관리자 계정과 동일하지 않은 교차 계정 시나리오에서 액세스 권한을 부여하려면 리소스 정책이 필요합니다. CodeArtifact의 교차 계정 액세스는 IAM 사용 설명서의 교차 계정 요청 허용 여부 결정에 설명된 교차 계정 액세스에 대한 일반 IAM 규칙을 따릅니다.

- 도메인 소유자 계정의 보안 주체에게는 자격 증명 기반 정책을 통해 도메인의 모든 리포지토리에 대한 액세스 권한이 부여될 수 있습니다. 이 경우 도메인 또는 리포지토리 정책에는 명시적인 허용이 필요하지 않습니다.
- 도메인 소유자 계정의 보안 주체에게는 도메인 또는 리포지토리 정책을 통해 모든 리포지토리에 대한 액세스 권한이 부여될 수 있습니다. 이 경우 자격 증명 기반 정책에는 명시적인 허용이 필요하지 않습니다.
- 리포지토리 관리자 계정의 보안 주체에게는 자격 증명 기반 정책을 통해 리포지토리에 대한 액세스 권한이 부여될 수 있습니다. 이 경우 도메인 또는 리포지토리 정책에는 명시적인 허용이 필요하지 않습니다.
- 다른 계정의 보안 주체는 작업을 명시적으로 거부하는 정책 없이 하나 이상의 리소스 정책 및 하나 이상의 자격 증명 기반 정책에서 허용하는 경우에만 액세스 권한이 부여됩니다.

CodeArtifact에 있는 리포지토리에 태그 추가

태그는 AWS 리소스와 연결된 키-값 페어입니다. CodeArtifact에 있는 리포지토리에 태그를 적용할 수 있습니다. CodePipeline 리소스 태그 추가, 사용 사례, 태그 키 및 값 제약, 지원되는 리소스 유형에 대한 자세한 내용은 리소스에 태그 지정 섹션을 참조하세요.

리포지토리를 만들 때 CLI를 사용하여 태그를 지정할 수 있습니다. 리포지토리에서 콘솔 또는 CLI를 사용하여 태그를 추가 또는 제거하고 태그 값을 업데이트할 수 있습니다. 각 리포지토리에 최대 50개의 태그를 추가할 수 있습니다.

주제

리포지토리 태그 추가(CLI)

CLI를 사용하여 리포지토리 태그를 관리할 수 있습니다.

주제

리포지토리에 태그 추가(CLI)

콘솔 또는를 사용하여 리포지토리 AWS CLI 에 태그를 지정할 수 있습니다.

리포지토리를 생성할 때 태그를 추가하려면 리포지토리 생성 단원을 참조하세요.

이 단계에서는 사용자가 이미 AWS CLI 최신 버전을 설치했거나 현재 버전으로 업데이트했다고 가정합니다. 자세한 정보는 AWS Command Line Interface설치 섹션을 참조하세요.

터미널이나 명령줄에서 tag-resource 명령을 실행하여, 태그를 추가할 리포지토리의 Amazon 리소스 이름(ARN)과 추가할 태그의 키와 값을 지정합니다.

> **Note**
>
> 리포지토리의 ARN을 가져오려면 `describe-repository` 명령을 실행합니다.
>
> ```
> aws codeartifact describe-repository --domain my_domain --repository my_repo --
> query repository.arn
> ```

리포지토리에 두 개 이상의 태그를 추가할 수 있습니다. 예를 들어 *my_domain* 도메인에 있는 *my_repo* 리포지토리에 태그 2개, 즉 태그 키가 *key1*이고 태그 값이 *value1*인 태그와 태그 키가 *key2*이고 태그 값이 *value2*인 태그를 추가하는 방법은 다음과 같습니다.

```
aws codeartifact tag-resource --resource-arn arn:aws:codeartifact:us-
west-2:111122223333:repository/my_domain/my_repo --tags key=key1,value=value1
 key=key2,value=value2
```

성공하면 이 명령에는 출력이 표시되지 않습니다.

리포지토리에 대한 태그 보기(CLI)

다음 단계에 따라 AWS CLI 를 사용하여 리포지토리의 AWS 태그를 확인합니다. 태그가 추가되지 않은 경우 반환되는 목록은 비어 있습니다.

터미널 또는 명령줄에서 list-tags-for-resource 명령을 실행합니다.

> **Note**
>
> 리포지토리의 ARN을 가져오려면 `describe-repository` 명령을 실행합니다.
>
> ```
> aws codeartifact describe-repository --domain my_domain --repository my_repo --
> query repository.arn
> ```

예를 들어 `arn:aws:codeartifact:us-west-2:111122223333:repository/my_domain/my_repo` ARN 값이 있는 *my_domain* 도메인의 *my_repo* 리포지토리에 대한 태그 키 및 태그 값 목록을 보는 방법은 다음과 같습니다.

```
aws codeartifact list-tags-for-resource --resource-arn arn:aws:codeartifact:us-
west-2:111122223333:repository/my_domain/my_repo
```

이 명령이 제대로 실행되면 다음과 비슷한 정보를 반환합니다.

```
{
    "tags": {
        "key1": "value1",
        "key2": "value2"
    }
```

```
}
```

리포지토리에 대한 태그 편집(CLI)

다음 단계에 따라 AWS CLI 를 사용하여 리포지토리의 태그를 편집합니다. 기존 키의 값을 변경하거나 다른 키를 추가할 수 있습니다.

터미널이나 명령줄에서 tag-resource 명령을 실행하여, 태그를 업데이트하고 태그 키 및 태그 값을 지정할 리포지토리의 ARN을 지정합니다.

> **Note**
>
> 리포지토리의 ARN을 가져오려면 `describe-repository` 명령을 실행합니다.
>
> ```
> aws codeartifact describe-repository --domain my_domain --repository my_repo --
> query repository.arn
> ```

```
aws codeartifact tag-resource --resource-arn arn:aws:codeartifact:us-
west-2:111122223333:repository/my_domain/my_repo --tags key=key1,value=newvalue1
```

이 명령이 제대로 실행되면 출력이 표시되지 않습니다.

리포지토리에서 태그 제거(CLI)

다음 단계에 따라 AWS CLI 를 사용하여 리포지토리에서 태그를 제거합니다.

> **Note**
>
> 리포지토리를 삭제하면 삭제된 리포지토리에서 모든 태그 연결이 제거됩니다. 리포지토리를 삭제하기 전에 태그를 제거하지 않아도 됩니다.

터미널이나 명령줄에서 untag-resource 명령을 실행하여, 태그를 제거할 리포지토리의 ARN과 제거할 태그의 태그 키를 지정합니다.

> **Note**
>
> 리포지토리의 ARN을 가져오려면 `describe-repository` 명령을 실행합니다.

```
aws codeartifact describe-repository --domain my_domain --repository my_repo --
query repository.arn
```

예를 들어 태그 키 태그 키 *key1* 및 *key2*가 있는 *my_domain* 도메인의 *my_repo* 리포지토리에서 여러 태그를 제거하는 방법은 다음과 같습니다.

```
aws codeartifact untag-resource --resource-arn arn:aws:codeartifact:us-
west-2:111122223333:repository/my_domain/my_repo --tag-keys key1 key2
```

이 명령이 제대로 실행되면 출력이 표시되지 않습니다. 태그를 제거한 후에는 `list-tags-for-resource` 명령을 사용하여 리포지토리에 남아 있는 태그를 볼 수 있습니다.

리포지토리 태그 추가(콘솔)

콘솔 또는 CLI를 사용하여 리소스에 태그를 지정할 수 있습니다.

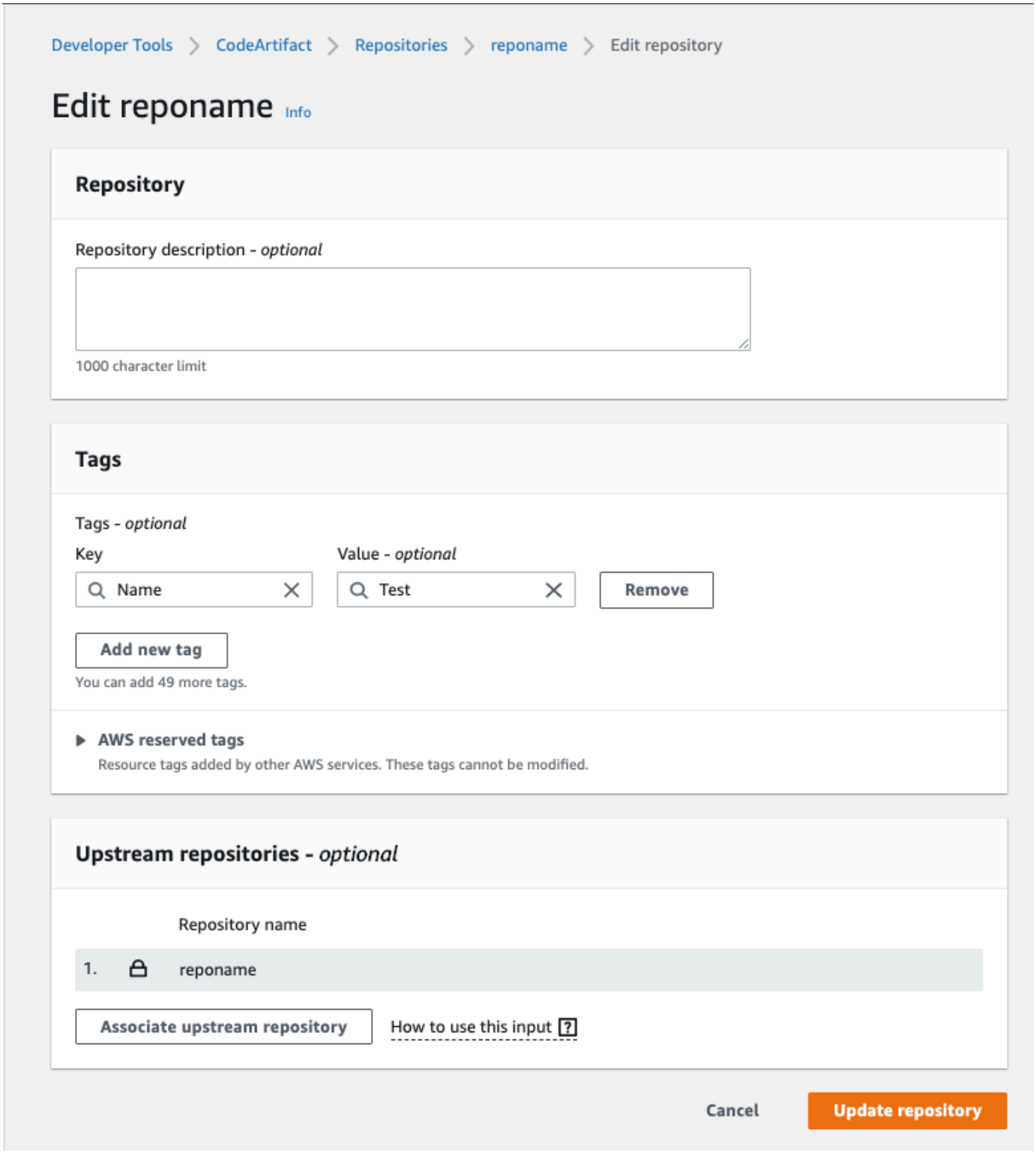
주제

- 리포지토리에 태그 추가(콘솔)
- 리포지토리에 대한 태그 보기(콘솔)
- 리포지토리 태그 편집(콘솔)
- 리포지토리에서 태그 제거(콘솔)

리포지토리에 태그 추가(콘솔)

콘솔을 사용하여 기존 리포지토리에 태그를 추가할 수 있습니다.

1. https://console.aws.amazon.com/codesuite/codeartifact/home AWS CodeArtifact 콘솔을 엽니다.
2. 리포지토리 페이지에서 태그를 추가할 리포지토리를 선택합니다.
3. 세부 정보 섹션을 확장합니다.
4. 리포지토리 태그에서, 리포지토리에 태그가 없는 경우 리포지토리 태그 추가를 선택합니다. 리포지토리에 태그가 있다면 리포지토리 태그 보기 및 편집을 선택합니다.
5. 새 태그 추가를 선택합니다.
6. 키 및 값 필드에, 추가할 각 태그의 텍스트를 입력합니다. (값 필드는 선택 사항입니다.) 예를 들어 키에 **Name**을 입력합니다. 값에는 **Test**를 입력합니다.



7. (선택 사항) 행을 추가하고 태그를 더 입력하려면 태그 추가를 선택합니다.
8. 리포지토리 업데이트를 선택합니다.

리포지토리에 대한 태그 보기(콘솔)

콘솔을 사용하여 기존 리포지토리에 대한 태그를 나열할 수 있습니다.

1. https://console.aws.amazon.com/codesuite/codeartifact/home AWS CodeArtifact 콘솔을 엽니다.
2. 리포지토리 페이지에서 태그를 확인할 리포지토리를 선택합니다.
3. 세부 정보 섹션을 확장합니다.
4. 리포지토리 태그에서 리포지토리 태그 보기 및 편집을 선택합니다.

> **Note**
>
> 이 리포지토리에 추가된 태그가 없는 경우 콘솔에 리포지토리 태그 추가가 표시됩니다.

리포지토리 태그 편집(콘솔)

리포지토리에 추가된 태그를 콘솔을 사용하여 편집할 수 있습니다.

1. https://console.aws.amazon.com/codesuite/codeartifact/home AWS CodeArtifact 콘솔을 엽니다.
2. 리포지토리 페이지에서 태그를 업데이트할 리포지토리를 선택합니다.
3. 세부 정보 섹션을 확장합니다.
4. 리포지토리 태그에서 리포지토리 태그 보기 및 편집을 선택합니다.

> **Note**
>
> 이 리포지토리에 추가된 태그가 없는 경우 콘솔에 리포지토리 태그 추가가 표시됩니다.

5. 키 및 값 필드에서 필요에 따라 각 필드의 값을 업데이트합니다. 예를 들어 **`Name`** 키의 값에서 **`Test`**를 **`Prod`**로 변경합니다.
6. 리포지토리 업데이트를 선택합니다.

리포지토리에서 태그 제거(콘솔)

콘솔을 사용하여 리포지토리에서 태그를 삭제할 수 있습니다.

1. https://console.aws.amazon.com/codesuite/codeartifact/home AWS CodeArtifact 콘솔을 엽니다.
2. 리포지토리 페이지에서 태그를 제거할 리포지토리를 선택합니다.

3. 세부 정보 섹션을 확장합니다.
4. 리포지토리 태그에서 리포지토리 태그 보기 및 편집을 선택합니다.

> **Note**
>
> 이 리포지토리에 추가된 태그가 없는 경우 콘솔에 리포지토리 태그 추가가 표시됩니다.

5. 삭제할 각 태그의 키와 값 옆에 있는 제거를 선택합니다.
6. 리포지토리 업데이트를 선택합니다.

CodeArtifact에서의 업스트림 리포지토리 작업

리포지토리에는 다른 AWS CodeArtifact 리포지토리가 업스트림 리포지토리로 있을 수 있습니다. 이를 통해 패키지 관리자 클라이언트는 단일 리포지토리 엔드포인트를 사용하여 둘 이상의 리포지토리에 포함된 패키지에 액세스할 수 있습니다.

AWS Management Console AWS CLI, 또는 SDK를 사용하여 a AWS CodeArtifact 리포지토리에 하나 이상의 업스트림 리포지토리를 추가할 수 있습니다. 리포지토리를 업스트림 리포지토리와 연결하려면 업스트림 리포지토리에서 `AssociateWithDownstreamRepository` 작업을 수행하는 데 필요한 권한이 있어야 합니다. 자세한 내용은 업스트림 리포지토리를 사용하여 리포지토리 생성 및 업스트림 리포지토리 추가 또는 제거 섹션을 참조하세요.

업스트림 리포지토리가 공용 저장소에 대한 외부 연결을 가지고 있는 경우 업스트림 리포지토리의 다운스트림에 있는 리포지토리는 해당 공용 저장소에서 패키지를 가져올 수 있습니다. 예를 들어 `my_repo` 리포지토리에 `upstream` 이름이 지정된 업스트림 리포지토리가 있고 `upstream`에 공용 npm 리포지토리에 대한 외부 연결이 있다고 가정해 보겠습니다. 이 경우 `my_repo`에 연결된 패키지 관리자는 npm 공용 저장소에서 패키지를 가져올 수 있습니다. 업스트림 리포지토리 또는 외부 연결에서 패키지를 요청하는 방법에 대한 자세한 내용은 업스트림 리포지토리가 포함된 패키지 버전 요청 또는 외부 연결을 통한 패키지 요청하기를 참조하세요.

주제

업스트림 리포지토리와 외부 연결의 차이점은 무엇인가요?

CodeArtifact에서 업스트림 리포지토리와 외부 연결은 거의 동일하게 동작하지만 몇 가지 중요한 차이점이 있습니다.

1. CodeArtifact 리포지토리에 업스트림 리포지토리를 10개까지 추가할 수 있습니다. 외부 연결은 하나만 추가할 수 있습니다.

2. 업스트림 리포지토리 또는 외부 연결을 추가하기 위한 별도의 API 호출이 있습니다.
3. 업스트림 리포지토리에서 요청된 패키지는 해당 리포지토리에 유지되므로 패키지 보존 동작은 약간 다릅니다. 자세한 내용은 중간 리포지토리에 패키지 보존 단원을 참조하십시오.

업스트림 리포지토리 추가 또는 제거

다음 섹션의 단계에 따라 CodeArtifact 리포지토리에 업스트림 리포지토리를 추가하거나 CodeArtifact 리포지토리에서 업스트림 리포지토리를 제거합니다. 업스트림 리포지토리에 대한 자세한 내용은 CodeArtifact에서의 업스트림 리포지토리 작업 섹션을 참조하세요.

이 가이드에는 다른 CodeArtifact 리포지토리를 업스트림 리포지토리로 구성하는 방법에 대한 정보가 포함되어 있습니다. npmjs.com, Nuget Gallery, Maven Central 또는 PyPI와 같은 공용 리포지토리에 대한 외부 연결을 구성하는 방법에 대한 자세한 내용은 외부 연결 추가를 참조하세요.

업스트림 리포지토리 추가 또는 제거(콘솔)

CodeArtifact 콘솔을 사용하여 리포지토리를 업스트림 리포지토리로 추가하려면 다음 절차의 단계를 수행합니다. 를 사용하여 업스트림 리포지토리를 추가하는 방법에 대한 자세한 내용은 섹션을 AWS CLI참조하세요업스트림 리포지토리 추가 또는 제거(AWS CLI).

CodeArtifact 콘솔을 사용하여 업스트림 리포지토리를 추가하려면

1. https://console.aws.amazon.com/codesuite/codeartifact/home AWS CodeArtifact 콘솔을 엽니다.
2. 탐색 창에서 도메인을 선택한 후 저장소가 포함된 도메인 이름을 선택합니다.
3. 저장소의 이름을 선택합니다.
4. 편집을 선택합니다.
5. 업스트림 리포지토리에서 업스트림 리포지토리 연결을 선택하고 업스트림 리포지토리로 추가하려는 리포지토리를 추가합니다. 업스트림 리포지토리와 동일한 도메인에만 리포지토리를 추가할 수 있습니다.
6. 리포지토리 업데이트를 선택합니다.

CodeArtifact 콘솔을 사용하여 업스트림 리포지토리를 제거하려면

1. https://console.aws.amazon.com/codesuite/codeartifact/home AWS CodeArtifact 콘솔을 엽니다.
2. 탐색 창에서 도메인을 선택한 후 저장소가 포함된 도메인 이름을 선택합니다.

3. 저장소의 이름을 선택합니다.
4. 편집을 선택합니다.
5. 업스트림 리포지토리에서 제거하려는 업스트림 리포지토리의 목록 항목을 찾은 다음 연결 해제를 선택합니다.

> **Important**
>
> CodeArtifact 리포지토리에서 업스트림 리포지토리를 제거하면 패키지 관리자는 업스트림 리포지토리 또는 업스트림 리포지토리의 패키지에 액세스할 수 없습니다.

6. 리포지토리 업데이트를 선택합니다.

업스트림 리포지토리 추가 또는 제거(AWS CLI)

AWS Command Line Interface (AWS CLI)를 사용하여 CodeArtifact 리포지토리의 업스트림 리포지토리를 추가하거나 제거할 수 있습니다. 이렇게 하려면 `update-repository` 명령을 사용하고 `--upstreams` 파라미터를 사용하여 업스트림 리포지토리를 지정합니다.

업스트림 리포지토리와 동일한 도메인의 리포지토리만 추가할 수 있습니다.

업스트림 리포지토리를 추가하려면(AWS CLI)

1. 그렇지 않은 경우의 단계에 따라 CodeArtifactwith AWS CodeArtifact 설정로 AWS CLI 를 설정하고 구성합니다.
2. `aws codeartifact update-repository` 명령을 `--upstreams` 플래그와 함께 사용하여 업스트림 리포지토리를 추가합니다.

> **Note**
>
> `update-repository` 명령을 호출하면 기존에 구성된 업스트림 리포지토리가 `--upstreams` 플래그와 함께 제공된 리포지토리 목록으로 바뀝니다. 업스트림 리포지토리를 추가하고 기존 업스트림 리포지토리를 유지하려면 호출에 기존 업스트림 리포지토리를 포함해야 합니다.

다음 예제 명령은 `my_domain` 이름이 지정된 도메인에 있는 `my_repo` 이름이 지정된 리포지토리에 두 개의 업스트림 리포지토리를 추가합니다. CodeArtifact가 `my_repo` 리포지토리의 패키지를

요청할 때 검색 우선순위는 `--upstreams` 파라미터의 업스트림 리포지토리 순서에 따라 결정됩니다. 자세한 내용은 업스트림 리포지토리 우선순위 순서 단원을 참조하십시오.

npmjs.com 또는 Maven Central과 같은 퍼블릭 외부 리포지토리에 연결하는 방법에 대한 자세한 내용은 CodeArtifact 저장소를 공용 저장소에 연결하기 섹션을 참조하세요.

```
aws codeartifact update-repository \
   --repository my_repo \
   --domain my_domain \
   --domain-owner 111122223333 \
   --upstreams repositoryName=upstream-1 repositoryName=upstream-2
```

출력에는 다음과 같은 업스트림 리포지토리가 포함됩니다.

```
{
       "repository": {
           "name": "my_repo",
           "administratorAccount": "123456789012",
           "domainName": "my_domain",
           "domainOwner": "111122223333",
           "arn": "arn:aws:codeartifact:us-
east-2:111122223333:repository/my_domain/my_repo",
           "upstreams": [
               {
                   "repositoryName": "upstream-1"
               },
               {
                   "repositoryName": "upstream-2"
               }
           ],
           "externalConnections": []
       }
   }
```

업스트림 리포지토리를 제거하려면(AWS CLI)

1. 그렇지 않은 경우의 단계에 따라 CodeArtifactwith AWS CodeArtifact 설정로 AWS CLI 를 설정하고 구성합니다.
2. CodeArtifact 리포지토리에서 업스트림 리포지토리를 제거하려면`--upstreams` 플래그와 함께 `update-repository` 명령을 사용합니다. 명령에 제공된 리포지토리 목록은 CodeArtifact 리포

지토리의 새 업스트림 리포지토리 세트가 됩니다. 유지하려는 기존 업스트림 리포지토리는 포함하고 제거하려는 업스트림 리포지토리는 생략합니다.

리포지토리에서 업스트림 리포지토리를 모두 제거하려면 `update-repository` 명령을 사용하고 인수 없이 `--upstreams`를 포함합니다. 다음은 `my_domain` 이름이 지정된 도메인에 포함된 `my_repo` 이름이 지정된 리포지토리에서 업스트림 리포지토리를 제거합니다.

```
aws codeartifact update-repository \
   --repository my_repo \
   --domain my_domain \
   --domain-owner 111122223333 \
   --upstreams
```

출력에는 `upstreams` 목록이 비어 있는 것으로 표시됩니다.

```
{
       "repository": {
           "name": "my_repo",
           "administratorAccount": "123456789012",
           "domainName": "my_domain",
           "domainOwner": "111122223333",
           "arn": "arn:aws:codeartifact:us-
east-2:111122223333:repository/my_domain/my_repo",
           "upstreams": [],
           "externalConnections": []
       }
   }
```

CodeArtifact 저장소를 공용 저장소에 연결하기

CodeArtifact 저장소와 외부 공용 저장소(예: https://npmjs.com 또는 Maven Central 저장소) 사이에 외부 연결을 추가할 수 있습니다. 그러면 아직 저장소에 없는 패키지를 CodeArtifact 저장소에서 요청하면 외부 연결을 통해 해당 패키지를 가져올 수 있습니다. 이렇게 하면 애플리케이션에서 사용하는 오픈 소스 종속성을 사용할 수 있습니다.

CodeArtifact에서 외부 연결을 사용하는 정확한 방법은 지정된 공용 저장소에 외부 연결로 연결된 도메인 당 하나의 저장소를 갖는 것입니다. 예로, npmjs.com에 연결하려면 도메인에 있는 하나의 저장소를 npmjs.com에 외부 연결로 구성하고, 다른 모든 저장소를 이 저장소의 업스트림으로 구성하는 것입니

다. 이렇게 하면 모든 저장소에서 패키지를 다시 가져와 저장하지 않고 npmjs.com에서 미리 가져온 패키지를 사용할 수 있습니다.

주제

외부 저장소(콘솔)에 연결하기

콘솔을 사용하여 외부 저장소에 연결을 추가하면 다음과 같은 과정을 거치게 됩니다.

1. 외부 저장소가 아직 없는 경우 CodeArtifact 도메인에 외부 저장소용 저장소인 `-store`가 생성됩니다. 이런 `-store` 저장소는 저장소와 외부 저장소 사이의 중간 저장소 역할을 하며, 둘 이상의 외부 저장소에 연결할 수 있습니다.
2. 적절한 `-store` 저장소가 저장소의 업스트림으로 추가됩니다.

다음 목록에는 CodeArtifact의 각 `-store` 리포지토리와 해당 리포지토리가 연결되는 각 외부 리포지토리가 포함되어 있습니다.

1. `cargo-store`는 crates.io에 연결되어 있습니다.
2. `clojars-store`는 Clojars Repository에 연결되어 있습니다.
3. `commonsware-store`는 CommonsWare Android Repository에 연결되어 있습니다.
4. `google-android-store`는 Google Android에 연결되어 있습니다.
5. `gradle-plugins-store`는 Gradle 플러그인에 연결되어 있습니다.
6. `maven-central-store`는 Maven Central Repository에 연결되어 있습니다.
7. `npm-store`는 npmjs.com에 연결되어 있습니다.
8. `nuget-store`는 nuget.org에 연결되어 있습니다.
9. `pypi-store`는 Python Packaging Authority에 연결되어 있습니다.
10. `rubygems-store`는 RubyGems.org에 연결되어 있습니다.

외부 리포지토리(콘솔)에 연결하기

1. https://console.aws.amazon.com/codesuite/codeartifact/home AWS CodeArtifact 콘솔을 엽니다.
2. 탐색 창에서 도메인을 선택한 후 저장소가 포함된 도메인 이름을 선택합니다.
3. 저장소의 이름을 선택합니다.
4. 편집을 선택합니다.
5. 업스트림 저장소에서 업스트림 저장소 연결을 선택하고 업스트림으로 연결된 적절한 `-store` 저장소를 추가합니다.
6. 저장소 업데이트를 선택합니다.

`-store` 저장소가 업스트림 저장소로 추가되면 CodeArtifact 저장소에 연결된 패키지 관리자로 각 외부 저장소에서 패키지를 가져올 수 있습니다.

외부 저장소(CLI)에 연결하기

를 사용하여 리포지토리 AWS CLI 에 직접 외부 연결을 추가하여 CodeArtifact 리포지토리를 외부 리포지토리에 연결할 수 있습니다. 이렇게 하면 CodeArtifact 저장소 또는 그 다운스트림 저장소에 연결된 사용자가 구성된 외부 저장소에서 패키지를 가져올 수 있습니다. 각 CodeArtifact 저장소는 단 하나만 외부로 연결할 수 있습니다.

특정 공개 저장소에 외부 연결된 도메인당 하나의 저장소를 사용하는 것이 좋습니다. 다른 저장소를 공용 저장소에 연결하려면 외부로 연결된 저장소를 해당 저장소의 업스트림으로 추가합니다. 본인 또는 도메인의 다른 사람이 이미 콘솔에서 외부 연결을 구성한 경우, 도메인에 연결하려는 공용 저장소에 외부로 연결된 `-store` 저장소가 이미 있을 가능성이 높습니다. `-store` 저장소 및 콘솔 연결에 관한 자세한 내용은 외부 저장소(콘솔)에 연결하기를 참조하십시오.

CodeArtifact 저장소(CLI)에 외부 연결 추가하기

- `associate-external-connection`로 외부 연결을 추가합니다. 다음 예시는 저장소를 npm 공용 레지스트리인 npmjs.com에 연결한 것입니다. 지원하는 저장소 목록은 지원하는 외부 연결 저장소를 참조하십시오.

```
aws codeartifact associate-external-connection --external-connection public:npmjs \
    --domain my_domain --domain-owner 111122223333 --repository my_repo
```

출력 예시:

```
{
```

```
    "repository": {
        "name": my_repo
        "administratorAccount": "123456789012",
        "domainName": "my_domain",
        "domainOwner": "111122223333",
        "arn": "arn:aws:codeartifact:us-
west-2:111122223333:repository/my_domain/my_repo",
        "description": "A description of my_repo",
        "upstreams": [],
        "externalConnections": [
            {
                "externalConnectionName": "public:npmjs",
                "packageFormat": "npm",
                "status": "AVAILABLE"
            }
        ]
    }
}
```

외부 연결을 추가한 뒤에는 외부 연결을 통한 패키지 요청하기에서 외부 연결을 통해 외부 저장소에서 패키지를 요청하는 방법을 참조하십시오.

지원하는 외부 연결 저장소

CodeArtifact는 다음 공용 저장소의 외부 연결을 지원합니다. CodeArtifact CLI를 사용하여 외부 연결을 지정하려면 `associate-external-connection` 명령을 실행할 때 매개변수의 이름 열에 있는 `--external-connection` 값을 사용하십시오.

리포지토리 유형	설명	이름
Maven	Clojars 리포지토리	`public:maven-clojars`
Maven	CommonsWare Android 저장소	`public:maven-commonsware`
Maven	Google Android Repository	`public:maven-googleandroid`

리포지토리 유형	설명	이름
Maven	Gradle 플러그인 저장소	`public:maven-gradleplugins`
Maven	Maven Central	`public:maven-central`
npm	npm 공용 레지스트리	`public:npmjs`
NuGet	NuGet Gallery	`public:nuget-org`
Python	Python Package Index	`public:pypi`
Ruby	RubyGems.org	`public:ruby-gems-org`
Rust	Crates.io	`public:crates-io`

외부 연결 제거하기(CLI)

에서 `associate-external-connection` 명령을 사용하여 추가된 외부 연결을 제거하려면를 AWS CLI사용합니다`disassociate-external-connection`.

```
aws codeartifact disassociate-external-connection --external-connection public:npmjs \
    --domain my_domain --domain-owner 111122223333 --repository my_repo
```

출력 예시:

```
{
    "repository": {
        "name": my_repo
        "administratorAccount": "123456789012",
        "domainName": "my_domain",
        "domainOwner": "111122223333",
        "arn": "arn:aws:codeartifact:us-
west-2:111122223333:repository/my_domain/my_repo",
        "description": "A description of my_repo",
        "upstreams": [],
        "externalConnections": []
```

```
    }
}
```

업스트림 리포지토리가 포함된 패키지 버전 요청

클라이언트(예: npm)가 여러 업스트림 리포지토리가 있는 `my_repo` 이름이 지정된 CodeArtifact 리포지토리에서 패키지 버전을 요청하면 다음과 같은 상황이 발생할 수 있습니다.

- 요청된 패키지 버전이 `my_repo`에 포함되어 있는 경우 클라이언트에 반환됩니다.
- 요청된 패키지 버전이 `my_repo`에 포함되어 있지 않은 경우 CodeArtifact는 `my_repo` 업스트림 리포지토리에서 해당 버전을 찾습니다. 패키지 버전을 찾으면 해당 패키지에 대한 참조가 `my_repo`에 복사되고 패키지 버전이 클라이언트에 반환됩니다.
- `my_repo` 및 해당 업스트림 리포지토리 모두에 패키지 버전이 포함되어 있지 않은 경우 HTTP 404 `Not Found` 응답이 클라이언트에 반환됩니다.

`create-repository` 또는 `update-repository` 명령을 사용하여 업스트림 리포지토리를 추가할 경우 패키지 버전 요청 시 우선 순위는 `--upstreams` 파라미터에 전달되는 순서에 따라 결정됩니다. `--upstreams`를 사용하여 패키지 버전이 요청될 때 CodeArtifact가 사용할 순서대로 업스트림 리포지토리를 지정합니다. 자세한 내용은 업스트림 리포지토리 우선순위 순서 단원을 참조하십시오.

하나의 리포지토리에 허용되는 최대 직접 업스트림 리포지토리 수는 10개입니다. 직접 업스트림 리포지토리에도 자체 업스트림 리포지토리가 있을 수 있으므로 CodeArtifact는 10개 이상의 리포지토리에서 패키지 버전을 검색할 수 있습니다. 패키지 버전이 요청될 때 CodeArtifact가 확인하는 최대 리포지토리 수는 25개입니다.

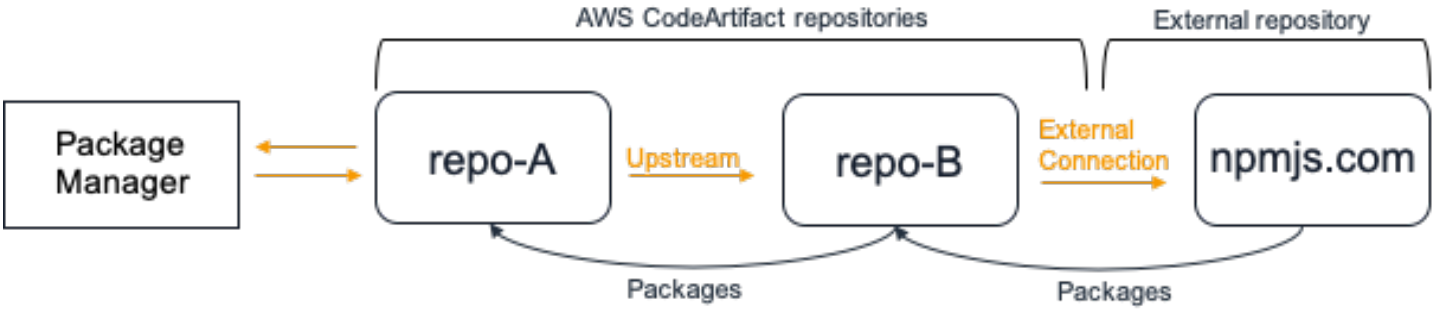
업스트림 리포지토리의 패키지 보존

요청된 패키지 버전이 업스트림 저장소에서 발견되면 해당 버전에 대한 참조가 유지되며 다운스트림 저장소에서 항상 사용할 수 있습니다. 유지되는 패키지 버전은 다음 사항에 영향을 받지 않습니다.

- 업스트림 리포지토리 삭제
- 업스트림 리포지토리와 다운스트림 리포지토리 간 연결 해제
- 업스트림 리포지토리에서 패키지 버전 삭제
- 업스트림 리포지토리의 패키지 버전 편집(예: 새 자산 추가)

업스트림 관계를 통해 패키지 가져오기

CodeArtifact 리포지토리가 외부 연결이 있는 리포지토리와 업스트림 관계에 있는 경우 업스트림 리포지토리에 없는 패키지에 대한 요청은 외부 리포지토리에서 복사됩니다. 예를 들어, 다음과 같은 구성을 살펴보겠습니다. `repo-A` 이름이 지정된 리포지토리에는 `repo-B` 이름이 지정된 업스트림 저장소가 있고 `repo-B`는 https://npmjs.com에 대한 외부 연결이 있습니다.



npm이 `repo-A` 리포지토리를 사용하도록 구성된 경우 `npm install`을 실행하면 https://npmjs.com에서 `repo-B`로 패키지 복사가 트리거됩니다. 설치된 버전도 함께 `repo-A`로 가져옵니다. 다음 예제에서는 `lodash`를 설치합니다.

```
$ npm config get registry
https://my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/npm/my-
downstream-repo/
$ npm install lodash
+ lodash@4.17.20
added 1 package from 2 contributors in 6.933s
```

`npm install`을 실행한 후에는 `repo-A`에는 최신 버전(`lodash 4.17.20`)만 포함됩니다. 해당 버전이 `repo-A`에서 npm를 통해 가져온 버전이기 때문입니다.

```
aws codeartifact list-package-versions --repository repo-A --domain my_domain \
            --domain-owner 111122223333 --format npm --package lodash
```

출력 예시:

```
{
    "package": "lodash",
    "format": "npm",
    "versions": [
        {
```

```
            "version": "4.17.15",
            "revision": "REVISION-1-SAMPLE-6C81EFF7DA55CC",
            "status": "Published"
        }
    ]
}
```

repo-B에 https://npmjs.com에 대한 외부 연결이 있기 때문에 https://npmjs.com에서 가져온 모든 패키지 버전이 repo-B에 저장됩니다. 이러한 패키지 버전은 repo-B와 업스트림 관계가 있는 모든 다운스트림 리포지토리에서 가져올 수 있었을 것입니다.

repo-B의 내용은 시간이 지남에 따라 https://npmjs.com에서 가져온 모든 패키지 및 패키지 버전을 볼 수 있는 방법을 제공합니다. 예를 들어, 시간이 지남에 따라 가져온 lodash 패키지의 모든 버전을 보려면 다음과 같이 list-package-versions를 사용할 수 있습니다.

```
aws codeartifact list-package-versions --repository repo-B --domain my_domain \
            --domain-owner 111122223333 --format npm --package lodash --max-results 5
```

출력 예시:

```
{
    "package": "lodash",
    "format": "npm",
    "versions": [
        {
            "version": "0.10.0",
            "revision": "REVISION-1-SAMPLE-6C81EFF7DA55CC",
            "status": "Published"
        },
        {
            "version": "0.2.2",
            "revision": "REVISION-2-SAMPLE-6C81EFF7DA55CC",
            "status": "Published"
        },
        {
            "version": "0.2.0",
            "revision": "REVISION-3-SAMPLE-6C81EFF7DA55CC",
            "status": "Published"
        },
        {
            "version": "0.2.1",
            "revision": "REVISION-4-SAMPLE-6C81EFF7DA55CC",
```

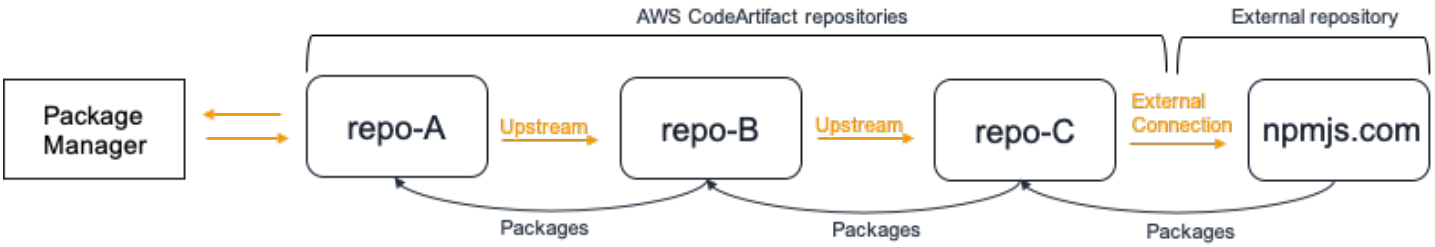
```
            "status": "Published"
        },
        {
            "version": "0.1.0",
            "revision": "REVISION-5-SAMPLE-6C81EFF7DA55CC",
            "status": "Published"
        }
    ],
    "nextToken": "eyJsaXN0UGFja2FnZVZlcnNpb25zVG9rZW4iOiIwLjIuMiJ9"
}
```

중간 리포지토리에 패키지 보존

CodeArtifact를 사용하면 업스트림 리포지토리를 체인화하여 연결할 수 있습니다. 예를 들어 repo-A는 repo-B의 업스트림이 되고 repo-B는 repo-C의 업스트림이 됩니다. 이 구성을 통해 repo-A에서 repo-B 및 repo-C 패키지 버전을 사용할 수 있습니다.

패키지 관리자가 repo-A 리포지토리에 연결하고 repo-C 리포지토리에서 패키지 버전을 가져오면 해당 패키지 버전은 repo-B 리포지토리에 유지되지 않습니다. 이 예시에서는 패키지 버전은 최종 repo-A 다운스트림 리포지토리에만 유지됩니다. 중간 리포지토리에는 유지되지 않습니다. 더 긴 체인의 경우에도 마찬가지입니다. 예를 들어, 네 개의 리포지토리 repo-A, repo-B, repo-C, repo-D 및 패키지 관리자가 repo-A에 연결되어 있고 repo-D에서 패키지 버전을 가져온 경우, 해당 패키지 버전은 repo-A에는 유지되지만 repo-B 또는 repo-C에는 유지되지 않습니다.

패키지 보존 동작은 외부 저장소에서 패키지 버전을 가져올 때와 비슷합니다. 단, 패키지 버전이 외부 연결이 연결된 리포지토리에 항상 유지된다는 점이 다릅니다. 예를 들어 repo-A는 repo-B의 업스트림입니다. repo-B는 repo-C의 업스트림이고 repo-C는 npmjs.com 이 외부 연결로 구성되어 있습니다. 다음 다이어그램을 참조하세요.



repo-A에 연결된 패키지 관리자가 패키지 버전(예: lodash 4.17.20)을 요청했으나 해당 패키지 버전이 세 리포지토리 중 어디에도 없는 경우 npmjs.com에서 해당 패키지 버전을 가져옵니다. lodash 4.17.20을 가져오면 이는 최종 다운스트림 리포지토리인 repo-A와 npmjs.com에 대한 외부 연결이 되어 있는 repo-C에 유지됩니다. lodash 4.17.20은 중간 저장소인 repo-B에는 유지되지 않습니다.

외부 연결을 통한 패키지 요청하기

다음 글에서는 외부 연결을 통해 패키지를 요청하는 방법과 패키지 요청 시 사용할 수 있는 CodeArtifact 동작을 설명합니다.

주제

외부 연결에서 패키지 불러오기

외부 연결을 통해 패키지를 불러오려면 CodeArtifact 저장소를 공용 저장소에 연결하기에 설명한 대로 CodeArtifact 저장소에 패키지를 추가한 뒤 패키지 관리자를 구성해 패키지를 설치합니다.

> **Note**
>
> 다음 지침에서는 npm을 사용하며, 다른 패키지 형식 구성과 사용에 관한 지침을 확인하려면 CodeArtifact를 Maven과 함께 사용나 CodeArtifact를 NuGet과 함께 사용, CodeArtifact를 Python과 함께 사용을 참조하십시오.

외부 연결을 통해 패키지를 불러오려면 다음과 같이 하십시오.

1. CodeArtifact 저장소에서 패키지 관리자를 구성 후 인증 절차를 거치십시오. npm의 경우 다음 aws codeartifact login 명령을 사용합니다.

```
aws codeartifact login --tool npm --domain my_domain --domain-owner 111122223333 --
repository my_repo
```

2. 공용 저장소에서 패키지를 요청합니다. npm의 경우 다음 `npm install` 명령을 사용해 *lodash*를 설치하려는 패키지로 바꿉니다.

```
npm install lodash
```

3. 패키지를 CodeArtifact 저장소에 불러온 후에 `list-packages` 및 `list-package-versions` 명령을 사용하여 패키지를 볼 수 있습니다.

```
aws codeartifact list-packages --domain my_domain --domain-owner 111122223333 --
repository my_repo
```

출력 예시:

```
{
    "packages": [
        {
            "format": "npm",
            "package": "lodash"
        }
    ]
}
```

`list-package-versions` 명령은 CodeArtifact 저장소에 복사한 패키지의 모든 버전을 나열합니다.

```
aws codeartifact list-package-versions --domain my_domain --domain-
owner 111122223333 --repository my_repo --format npm --package lodash
```

출력 예시:

```
{
    "defaultDisplayVersion: "1.2.5"
    "format": "npm",
    "package": "lodash",
    "namespace": null,
    "versions": [
        {
            "version": "1.2.5",
            "revision": "REVISION-1-SAMPLE-6C81EFF7DA55CC",
            "status": "Published"
```

```
        }
    ]
}
```

외부 연결 지연 시간

외부 연결을 통해 공용 저장소에서 패키지를 불러올 때, 패키지를 공용 저장소에서 불러오는 시점과 CodeArtifact 저장소에 저장되는 시점 사이에 지연이 발생합니다. 예를 들어, 외부 연결에서 패키지 불러오기에서 설명한 대로 npm 패키지 "lodash" 버전 1.2.5를 설치했다고 가정해 보겠습니다. `npm install lodash` lodash 명령 작업은 성공적으로 완료했으나, 패키지 버전이 아직 CodeArtifact 저장소에 나타나지 않을 수 있습니다. 패키지 버전이 저장소에 표시되는 데 보통 3분 가량 소요되지만, 경우에 따라 더 오래 걸릴 수도 있습니다.

이러한 지연 시간으로 인해 패키지 버전을 성공적으로 불러왔지만, 아직 저장소 버전을 CodeArtifact 콘솔에서 보거나 ListPackages 및 ListPackageVersions API 작업을 불러올 때 버전을 확인하지 못할 수 있습니다. CodeArtifact가 패키지 버전을 비동기화하면 콘솔과 API 요청을 통해 해당 버전을 볼 수 있습니다.

외부 저장소를 사용할 수 없을 때의 CodeArtifact 동작

때때로 외부 저장소가 중단되어 CodeArtifact가 해당 저장소에서 패키지를 가져올 수 없거나 패키지를 가져오는 시간이 평소에 비해 훨씬 느려질 수 있습니다. 이러한 경우 외부 저장소(예: npmjs.com)에서 미리 가져온 패키지나 CodeArtifact 저장소에 저장한 패키지 버전은 CodeArtifact에서 계속 다운로드할 수 있습니다. 하지만 해당 저장소와 외부 연결을 구성한 경우에도 CodeArtifact에 아직 저장하지 않은 패키지는 사용하지 못할 수 있습니다. 예를 들면, 지금까지 애플리케이션에서 사용했던 버전이 npm 패키지 `lodash 4.17.19` 버전이기 때문에 CodeArtifact 저장소에 npm 패키지 버전이 포함되어 있을 수 있습니다. `4.17.20`로 업그레이드하려는 경우, CodeArtifact는 npmjs.com에서 새 버전을 가져와 CodeArtifact 저장소에 저장합니다. 그러나 npmjs.com이 작동을 중단하는 경우 이 새 버전을 사용할 수 없습니다. 이 경우 유일한 해결 방법은 차후 npmjs.com이 복구된 뒤 다시 시도하는 것입니다.

외부 저장소 작동 중단은 CodeArtifact에 새 패키지 버전을 게시하는 데에도 영향을 줄 수 있습니다. 외부 연결이 구성된 저장소에서는 이미 외부 저장소에 존재하는 패키지 버전을 게시할 수 없습니다. 자세한 내용은 패키지 개요 단원을 참조하십시오. 그러나 드물게 외부 저장소가 중단되면 CodeArtifact에 외부 저장소 내 패키지 및 패키지 버전에 관한 최신 정보 등이 존재하지 않을 수 있습니다. 이 경우 CodeArtifact가 보통 거부하는 패키지 버전을 게시하도록 허용할 수 있습니다.

새 패키지 버전 사용 가능 여부

npmjs.com과 같은 공용 저장소에 있는 패키지 버전을 CodeArtifact 저장소를 통해 사용하려면 먼저 위치 패키지 메타데이터 캐시에 추가해야 합니다. 이 캐시는 각 AWS 리전의 CodeArtifact에서 유지 관리하며 지원되는 퍼블릭 리포지토리의 내용을 설명하는 메타데이터를 포함합니다. 해당 캐시로 인해 새 패키지 버전이 공용 저장소에 게시되는 시점과 CodeArtifact에서 사용할 수 있는 시점 사이에 지연이 발생하는 것입니다. 이 지연은 패키지 형식에 따라 다릅니다.

npm과 Python 및 Nuget 패키지의 경우, 새 패키지 버전이 npmjs.com, pypi.org 또는 nuget.org에 게시된 이후 CodeArtifact 저장소에 설치할 수 있는 시점까지 최대 30분 가량 지연될 수 있습니다. CodeArtifact는 두 저장소의 메타데이터를 자동으로 동기화하여 캐시를 최신 상태로 유지합니다.

Maven 패키지의 경우, 새 패키지 버전이 공용 저장소에 게시된 후 CodeArtifact 저장소에 설치할 수 있을 때까지 최대 3시간 가량 지연될 수 있습니다. CodeArtifact는 최대 3시간에 한 번씩 패키지의 새 버전을 확인합니다. 3시간의 캐시 수명이 만료되고 지정된 패키지 이름을 처음 요청하면 해당 패키지의 새 버전을 모두 지역 캐시로 가져옵니다.

일반적으로 사용하는 Maven 패키지의 경우, 요청이 많이 전송되어 캐시 수명이 만료되는 즉시 캐시가 업데이트되는 경우가 많아 3시간마다 새 버전을 가져옵니다. 자주 사용하지 않는 패키지의 경우 CodeArtifact 저장소에서 패키지 버전을 요청할 때까지 캐시에 최신 버전을 저장하지 않습니다. 첫 요청에서는 이전에 불러온 버전만 CodeArtifact에서 사용할 수 있지만, 이 요청을 하면 캐시가 업데이트됩니다. 후속 요청 시 새 패키지 버전이 캐시에 추가되어 다운로드할 수 있습니다.

둘 이상의 에셋을 포함한 패키지 버전 가져오기

Maven과 Python 패키지, 또는 두 패키지 모두 버전 당 여러 개의 에셋을 포함할 수 있습니다. 따라서 이러한 패키지 형식을 가져오려면 패키지 버전당 에셋이 하나뿐인 npm이나 NuGet 패키지보다 더 복잡한 절차를 걸쳐야 합니다. 이러한 패키지 형식에서 불러올 수 있는 에셋과 새로 추가된 에셋을 사용하는 방법은 업스트림 및 외부 연결에서 Python 패키지 요청하기과 업스트림 및 외부 연결에서 Maven 패키지 요청를 참조하십시오.

업스트림 리포지토리 우선순위 순서

업스트림 리포지토리가 하나 이상 있는 리포지토리에서 패키지 버전을 요청하는 경우 패키지 버전의 우선 순위는 `create-repository` 또는 `update-repository` 명령을 호출할 때 나열된 순서와 일치합니다. 요청된 패키지 버전을 찾으면 업스트림 저장소를 모두 검색하지 않았더라도 검색이 중지됩니다. 자세한 내용은 업스트림 리포지토리 추가 또는 제거(AWS CLI) 단원을 참조하십시오.

describe-repository 명령을 사용하면 우선순위 순서를 확인할 수 있습니다.

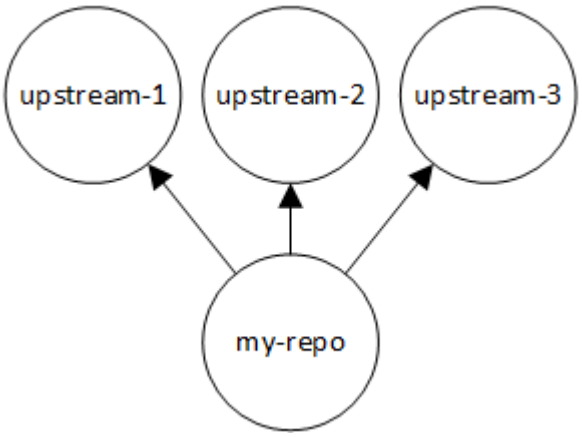
```
aws codeartifact describe-repository --repository my_repo --domain my_domain --domain-
owner 111122223333
```

결과는 다음과 같을 수 있습니다. 업스트림 리포지토리 우선 순위가 upstream-1 첫 번째, upstream-2 두 번째, upstream-3 세 번째임을 알 수 있습니다.

```
{
    "repository": {
        "name": "my_repo",
        "administratorAccount": "123456789012",
        "domainName": "my_domain",
        "domainOwner": "111122223333",
        "arn": "arn:aws:codeartifact:us-
east-1:111122223333:repository/my_domain/my_repo",
        "description": "My new repository",
        "upstreams": [
            {
                "repositoryName": "upstream-1"
            },
            {
                "repositoryName": "upstream-2"
            },
            {
                "repositoryName": "upstream-3"
            }
        ],
        "externalConnections": []
    }
}
```

간단한 우선순위 순서 예제

다음 다이어그램에서 my_repo 리포지토리에는 업스트림 리포지토리가 세 개 있습니다. 업스트림 리포지토리의 우선 순위는 upstream-1, upstream-2, upstream-3입니다.



my_repo에서 패키지 버전을 요청하면 해당 버전을 찾거나 HTTP 404 Not Found 응답이 클라이언트에 반환될 때까지 다음 순서로 리포지토리를 검색합니다.

1. my_repo
2. upstream-1
3. upstream-2
4. upstream-3

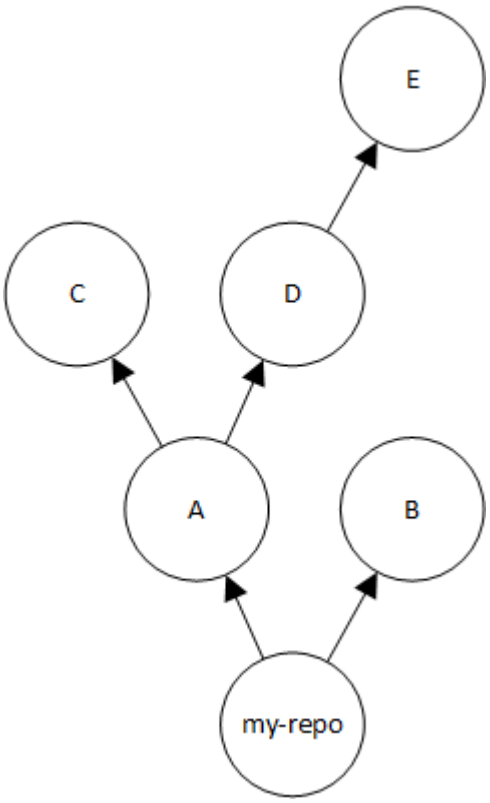
패키지 버전을 찾으면 업스트림 저장소를 모두 검색하지 않았더라도 검색이 중지됩니다. 예를 들어 패키지 버전이 upstream-1에 있는 경우 검색이 중지되고 CodeArtifact는 upstream-2 또는 upstream-3를 검색하지 않습니다.

AWS CLI 명령을 사용하여에서 패키지 버전을 list-package-versions 나열하면 에서만 볼 my_repo수 있습니다my_repo. 업스트림 리포지토리의 패키지 버전은 나열되지 않습니다.

복잡한 우선순위 순서 예제

업스트림 리포지토리에 자체 업스트림 리포지토리가 있는 경우 다음 업스트림 리포지토리로 이동하기 전에 동일한 로직을 사용하여 패키지 버전을 찾습니다. 예를 들어 my_repo 리포지토리에 업스트림 리포지토리 두 개와 A 및 B가 있다고 가정해 보겠습니다. A 리포지토리에 업스트림 리포지토리가 있는 경우 my_repo에서 패키지 버전에 대한 요청을 하면 가장 먼저 my_repo, 두 번째로는 A, 그 다음엔 A의 업스트림 리포지토리를 찾는 식입니다.

다음 다이어그램에서 my_repo 리포지토리는 업스트림 리포지토리를 포함합니다. A 업스트림 리포지토리에는 업스트림 리포지토리가 두 개 있고 D 업스트림 리포지토리에는 한 개 있습니다. 다이어그램에서 동일한 수준의 업스트림 리포지토리는 왼쪽에서 오른쪽으로 우선 순위가 표시됩니다. A 리포지토리는 B 리포지토리보다 우선 순위가 높고 C 리포지토리는 D 리포지토리보다 우선 순위가 높습니다.



이 예제에서는 `my_repo`에서 패키지 버전을 요청하면 해당 버전을 찾거나 패키지 관리자가 HTTP `404 Not Found` 응답을 클라이언트에 반환할 때까지 리포지토리를 다음 순서로 검색합니다.

1. `my_repo`
2. `A`
3. `C`
4. `D`
5. `E`
6. `B`

업스트림 리포지토리에서의 API 동작

업스트림 리포지토리에 연결된 리포지토리에서 특정 CodeArtifact API를 호출하면 패키지 또는 패키지 버전이 대상 리포지토리에 저장되어 있는지 업스트림 리포지토리에 저장되는지에 따라 동작이 달라질 수 있습니다. 이러한 API의 동작은 여기에 설명되어 있습니다.

CodeArtifact API에 대한 자세한 내용은 CodeArtifact API 참조를 참조하세요.

패키지 또는 패키지 버전을 참조하는 대부분의 API는 지정된 패키지 버전이 대상 리포지토리에 없는 경우 `ResourceNotFound` 오류를 반환합니다. 이는 패키지 또는 패키지 버전이 업스트림 저장소에 있는 경우에도 마찬가지입니다. 사실상 이러한 API를 호출할 때 업스트림 리포지토리는 무시됩니다. 이러한 API는 다음과 같습니다.

- DeletePackageVersions
- DescribePackageVersion
- GetPackageVersionAsset
- GetPackageVersionReadme
- ListPackages
- ListPackageVersionAssets
- ListPackageVersionDependencies
- ListPackageVersions
- UpdatePackageVersionsStatus

이 동작을 보여주기 위한 `target-repo`, `upstream-repo` 두 개의 저장소가 있습니다. `target-repo`는 비어 있으며 업스트림 리포지토리로 `upstream-repo`가 구성되었습니다. `upstream-repo`에는 `lodash` npm 패키지가 있습니다.

`upstream-repo`에서 `lodash` 패키지가 포함된 `DescribePackageVersion` API를 호출하면 다음과 같은 결과가 출력됩니다.


```
{
    "packageVersion": {
        "format": "npm",
        "packageName": "lodash",
        "displayName": "lodash",
        "version": "4.17.20",
        "summary": "Lodash modular utilities.",
        "homePage": "https://lodash.com/",
        "sourceCodeRepository": "https://github.com/lodash/lodash.git",
        "publishedTime": "2020-10-14T11:06:10.370000-04:00",
        "licenses": [
            {
                "name": "MIT"
            }
        ],
        "revision": "Ciqe5/9yicvkJT13b5/LdLpCyE6fqA7poa9qp+FilPs=",
```

```
        "status": "Published"
    }
```

target-repo에서 비어 있지만 업스트림으로 upstream-repo가 구성된 동일한 API를 호출하면 다음과 같은 결과가 출력됩니다.

```
An error occurred (ResourceNotFoundException) when calling the DescribePackageVersion
 operation:
Package not found in repository. RepoId: repo-id, Package =
 PackageCoordinate{packageType=npm, packageName=lodash},
```

CopyPackageVersions API는 다르게 동작합니다. 기본적으로 CopyPackageVersions API는 대상 리포지토리에 저장된 패키지 버전만 복사합니다. 패키지 버전이 업스트림 리포지토리에는 저장되지만 대상 리포지토리에는 저장되지 않는 경우 복사되지 않습니다. 업스트림 리포지토리에만 저장되는 패키지의 패키지 버전을 포함하려면 API 요청에서 includeFromUpstream 값을 true로 설정해야 합니다.

CopyPackageVersions API에 대한 자세한 내용은 리포지토리 간 패키지 복사를 참조하세요.

CodeArtifact에서의 패키지 작업

다음 주제에서는 CodeArtifact CLI 및 API를 사용하여 패키지에 대한 작업을 수행하는 방법을 보여줍니다.

주제

패키지 개요

패키지는 종속성을 해결하고 소프트웨어를 설치하는 데 필요한 소프트웨어 및 메타데이터 번들입니다. CodeArtifact에서 패키지는 패키지 이름, 선택적 네임스페이스(예: `@types/node`에 있는 `@types`), 패키지 버전 세트, 패키지 수준 메타데이터(예: npm 태그)로 구성됩니다.

목차

- 패키지 버전 상태
- 패키지 이름, 패키지 버전 및 자산 이름 표준화

지원되는 패키지 형식

AWS CodeArtifact는 Cargo, generic, Maven, npm, NuGet, PyPI, Ruby, Swift 패키지 형식을 지원합니다.

패키지 게시

npm,twine,Maven, Gradle, nuget, dotnet 등의 도구를 사용하여 지원되는 모든 패키지 형식의 새 버전을 CodeArtifact 리포지토리에 게시할 수 있습니다.

게시 권한

AWS Identity and Access Management (IAM) 사용자 또는 역할에는 대상 리포지토리에 게시할 수 있는 권한이 있어야 합니다. 패키지를 게시하려면 다음 권한이 필요합니다.

- Cargo: `codeartifact:PublishPackageVersion`
- 일반: `codeartifact:PublishPackageVersion`
- Maven: `codeartifact:PublishPackageVersion`, `codeartifact:PutPackageMetadata`
- npm: `codeartifact:PublishPackageVersion`
- NuGet: `codeartifact:PublishPackageVersion`, `codeartifact:ReadFromRepository`
- Python: `codeartifact:PublishPackageVersion`
- Ruby: `codeartifact:PublishPackageVersion`
- Swift: `codeartifact:PublishPackageVersion`

위의 권한 목록에서 IAM 정책은 `codeartifact:PublishPackageVersion` 및 `codeartifact:PutPackageMetadata` 권한에 대한 `package` 리소스를 지정해야 합니다. 또한 `codeartifact:ReadFromRepository` 권한에 사용할 `repository` 리소스도 지정해야 합니다.

CodeArtifact에서의 권한에 대한 자세한 내용은 AWS CodeArtifact 권한 참조 섹션을 참조하세요.

패키지 자산 덮어쓰기

이미 존재하며 콘텐츠가 다른 패키지 자산은 다시 게시할 수 없습니다. 예를 들어 Maven 패키지를 JAR 자산 `mypackage-1.0.jar`과 함께 이미 게시했다고 가정해 보겠습니다. 이 자산은 이전 자산과

새 자산의 체크섬이 동일한 경우에만 다시 게시할 수 있습니다. 새 콘텐츠가 있는 동일한 자산을 다시 게시하려면 먼저 delete-package-versions 명령을 이용해 패키지 버전을 삭제하세요. 콘텐츠가 다른 동일한 자산 이름을 다시 게시하려고 하면 HTTP 409 충돌 오류가 발생합니다.

여러 자산을 지원하는 패키지 형식(일반, PyPI 및 Maven)의 경우, 필요한 권한이 있다면 기존 패키지 버전에 이름이 다른 새 자산을 추가할 수 있습니다. 일반 패키지의 경우 패키지 버전이 `Unfinished` 상태에 있는 한 새 자산을 추가할 수 있습니다. npm은 패키지 버전당 자산 하나만 지원하므로, 게시된 패키지 버전을 어떤 식으로든 수정하려면 먼저 delete-package-versions를 사용하여 패키지 버전을 삭제해야 합니다.

이미 존재하는 자산(예:`mypackage-1.0.jar`)을 다시 게시하려고 할 때 게시된 자산과 새 자산의 내용이 동일하면, 작업은 멱등성이기 때문에 성공하게 됩니다.

프라이빗 패키지 및 퍼블릭 리포지토리

CodeArtifact는 CodeArtifact 리포지토리에 저장된 패키지를 npmjs.com 또는 Maven Central 같은 퍼블릭 리포지토리에 게시하지 않습니다. CodeArtifact는 퍼블릭 리포지토리에서 CodeArtifact 리포지토리로 패키지를 가져오지만 패키지를 반대 방향으로 옮기지는 않습니다. CodeArtifact 리포지토리에 게시하는 패키지는 비공개로 유지되며 액세스 권한을 부여한 AWS 계정, 역할 및 사용자만 사용할 수 있습니다.

패치가 적용된 패키지 버전 게시

수정된 패키지 버전을 게시하고 싶지만 이러한 버전이 퍼블릭 리포지토리에서 사용 가능한 버전일 수도 있습니다. 예를 들어 `mydep 1.1`이라는 중요한 애플리케이션 종속성에서 버그를 발견한 경우, 패키지 공급업체가 변경 사항을 검토하고 수락하기 전에 먼저 이를 수정해야 할 수 있습니다. 앞서 설명한 것처럼 CodeArtifact에서는 업스트림 리포지토리와 외부 연결을 통해 CodeArtifact 리포지토리에서 퍼블릭 리포지토리에 접근할 수 있는 경우 CodeArtifact 리포지토리에서 `mydep 1.1`을 게시할 수 없습니다.

이 문제를 해결하려면 퍼블릭 리포지토리에서는 접근할 수 없는 다른 CodeArtifact 리포지토리에 패키지 버전을 게시하세요. 그런 다음 `copy-package-versions` API를 사용하여 `mydep 1.1`의 패치된 버전을 사용할 CodeArtifact 리포지토리에 복사합니다.

게시를 위한 자산 크기 제한

게시할 수 있는 패키지 자산의 최대 크기는 in AWS CodeArtifact 할당량에 표시된 자산 파일 크기 최대 할당량에 따라 제한됩니다. 예를 들어 현재 자산 파일 크기 최대 할당량보다 큰 Maven JAR 또는 Python 휠은 게시할 수 없습니다. CodeArtifact에 더 큰 자산을 저장해야 한다면 할당량 증가를 요청하세요.

자산 파일 크기 최대 할당량을 제외하고, npm 패키지 게시 요청의 최대 크기는 2GB입니다. 이 제한은 자산 파일 크기 최대 할당량과 무관하며 할당량을 늘려도 이 제한이 늘어나지는 않습니다. npm 게시 요청(HTTP PUT)에서는 패키지 메타데이터와 npm 패키지 tar 아카이브의 콘텐츠가 하나로 묶입니다. 따라서 게시할 수 있는 npm 패키지의 실제 최대 크기는 포함된 메타데이터의 크기에 따라 달라집니다.

> **Note**
>
> 게시된 npm 패키지는 최대 크기가 2GB 미만으로 제한됩니다.

게시 지연 시간

CodeArtifact 리포지토리에 게시된 패키지 버전은 대부분 1초 이내에 다운로드할 수 있습니다. 예를 들어 npm 패키지 버전을 `npm publish`를 이용해 CodeArtifact에 게시하는 경우, `npm install` 명령을 이용하면 1초 이내에 해당 버전을 사용할 수 있습니다. 하지만 게시가 일관적이지 않으며 시간이 더 오래 걸릴 수도 있습니다. 게시 후 즉시 패키지 버전을 사용해야 한다면, 재시도를 통해 다운로드가 안정적인지 확인하세요. 예를 들어 패키지 버전을 게시한 후, 방금 게시한 패키지 버전을 최초 다운로드 시도에서 사용할 수 없다면 다운로드를 최대 3번 반복하세요.

> **Note**
>
> 일반적으로 퍼블릭 리포지토리에서 패키지 버전을 가져오는 작업은 게시보다 시간이 더 오래 걸립니다. 자세한 내용은 외부 연결 지연 시간 단원을 참조하십시오.

패키지 버전 상태

CodeArtifact의 각 패키지 버전에는 패키지 버전의 현재 상태 및 가용성을 설명하는 상태가 적용됩니다. AWS CLI 및 SDK에서 패키지 버전 상태를 변경할 수 있습니다. 자세한 내용은 패키지 버전 상태 업데이트 단원을 참조하십시오.

다음은 패키지 버전 상태에 적용되는 값입니다.

- 게시됨 - 패키지 버전이 성공적으로 게시되었으며 패키지 관리자를 이용해 요청할 수 있습니다. 패키지 버전은 패키지 관리자에 반환되는 패키지 버전 목록(예: `npm view <package-name> versions` 출력)에 포함됩니다. 패키지 버전의 모든 자산은 리포지토리에서 사용할 수 있습니다.
- 미완료 - 클라이언트가 패키지 버전용 자산을 하나 이상 업로드했지만 `Published` 상태로 전환하여 완료하지는 않았습니다. 현재는 일반 및 Maven 패키지 버전만이 `Unfinished` 상태가 될 수 있습니다. Maven 패키지의 경우, 클라이언트가 패키지 버전용 자산을 하나 이상 업로드하지만 해당 버전이

포함된 패키지의 maven-metadata.xml 파일을 게시하지 않으면 이 문제가 발생할 수 있습니다. Maven 패키지 버전이 미완료이면 mvn 또는 gradle 같은 클라이언트에게 반환되는 버전 목록에 패키지 버전이 포함되지 않으며, 따라서 빌드의 일부로 사용할 수 없습니다. PublishPackageVersion API를 호출할 때 unfinished 플래그를 제공하면 일반 패키지를 의도적으로 Unfinished 상태로 유지할 수 있습니다. unfinished 플래그를 생략하거나 UpdatePackageVersionsStatus API를 호출하면 일반 패키지를 Published 상태로 변경할 수 있습니다.

- 미등록 - 패키지 버전의 자산을 리포지토리에서 다운로드할 수 있지만, 패키지 버전이 패키지 관리자에게 반환되는 버전 목록에 포함되지 않습니다. 예를 들어 npm 패키지의 경우 npm view <package-name> versions 출력에 패키지 버전이 포함되지 않습니다. 즉, 사용 가능한 버전 목록에 버전이 표시되지 않기 때문에 npm의 종속성 확인 로직은 패키지 버전을 선택하지 않습니다. 하지만 미등록 패키지 버전이 이미 npm package-lock.json 파일에서 참조된 경우, 예를 들어 npm ci를 실행 중일 때 해당 버전을 다운로드하여 설치할 수 있습니다.
- 보관됨 - 패키지 버전의 자산을 더 이상 다운로드할 수 없습니다. 패키지 버전은 패키지 관리자에 반환되는 버전 목록에 포함됩니다. 자산을 사용할 수 없으므로 클라이언트의 패키지 버전 사용이 차단됩니다. 애플리케이션 빌드가 보관됨으로 업데이트된 버전을 사용하는 경우, 패키지 버전이 로컬에 캐시되지 않으면 빌드가 중단됩니다. 보관됨 패키지 버전은 아직 리포지토리에 있기 때문에 패키지 관리자나 빌드 도구를 사용하여 다시 게시할 수 없지만, UpdatePackageVersionsStatus API를 사용하여 패키지 버전의 상태를 미등록 또는 게시됨으로 다시 변경할 수 있습니다.
- 폐기됨 - 패키지 버전이 목록에 표시되지 않으며 리포지토리에서 자산을 다운로드할 수 없습니다. 폐기됨과 보관됨의 주된 차이점은 폐기됨 상태인 경우 CodeArtifact가 패키지 버전의 자산을 영구적으로 삭제한다는 것입니다. 따라서 패키지 버전을 폐기됨에서 보관됨, 미등록 또는 게시됨으로 전환할 수 없습니다. 자산이 삭제되었으므로 패키지 버전을 더 이상 사용할 수 없습니다. 패키지 버전이 폐기됨으로 표시되면 패키지 자산 보관 요금이 청구되지 않습니다.

--status 파라미터 없이 list-package-versions를 호출하면, 기본적으로 모든 상태의 패키지 버전이 반환됩니다.

위에 나열된 상태 외에 DeletePackageVersions API를 사용하여 패키지 버전을 삭제할 수도 있습니다. 삭제된 패키지 버전은 리포지토리에 더 이상 존재하지 않으며 패키지 관리자 또는 빌드 도구를 사용하여 해당 패키지 버전을 자유롭게 다시 게시할 수 있습니다. 패키지 버전이 삭제되면 패키지 버전의 자산 보관 요금이 청구되지 않습니다.

패키지 이름, 패키지 버전 및 자산 이름 표준화

CodeArtifact는 패키지 이름, 패키지 버전과 자산 이름을 저장하기 전에 먼저 표준화합니다. 즉, CodeArtifact의 이름 또는 버전은 패키지가 게시될 때 제공된 것과 다를 수 있습니다. CodeArtifact에서 각 패키지 유형의 이름 및 버전을 표준화하는 방법에 대한 자세한 내용은 다음 설명서를 참조하세요.

- Python 패키지 이름 정규화
- NuGet 패키지 이름, 버전 및 자산 이름 표준화

CodeArtifact는 다른 패키지 형식에서는 표준화를 수행하지 않습니다.

패키지 이름 나열

CodeArtifact에서 `list-packages` 명령을 사용하여 리포지토리에 있는 모든 패키지 이름 목록을 가져옵니다. 이 명령은 버전이 아닌 패키지 이름만 반환합니다.

```
aws codeartifact list-packages --domain my_domain --domain-owner 111122223333 --
repository my_repo
```

샘플 출력:

```
{
    "nextToken": "eyJidWNrZXRJZCI6I...",
    "packages": [
        {
            "package": "acorn",
            "format": "npm",
            "originConfiguration": {
                "restrictions": {
                    "publish": "BLOCK",
                    "upstream": "ALLOW"
            }
        },
        {
            "package": "acorn-dynamic-import",
            "format": "npm",
            "originConfiguration": {
                "restrictions": {
                    "publish": "BLOCK",
                    "upstream": "ALLOW"
            }
        },
        {
            "package": "ajv",
            "format": "npm",
            "originConfiguration": {
```

```
                "restrictions": {
                    "publish": "BLOCK",
                    "upstream": "ALLOW"
            }
        },
        {
            "package": "ajv-keywords",
            "format": "npm",
            "originConfiguration": {
                "restrictions": {
                    "publish": "BLOCK",
                    "upstream": "ALLOW"
            }
        },
        {
            "package": "anymatch",
            "format": "npm",
            "originConfiguration": {
                "restrictions": {
                    "publish": "BLOCK",
                    "upstream": "ALLOW"
            }
        },
        {
            "package": "ast",
            "namespace": "webassemblyjs",
            "format": "npm",
            "originConfiguration": {
                "restrictions": {
                    "publish": "BLOCK",
                    "upstream": "ALLOW"
            }
        }
    ]
}
```

npm 패키지 이름 나열

npm 패키지 이름만 나열하려면 `--format` 옵션 값을 npm으로 설정합니다.

```
aws codeartifact list-packages --domain my_domain --domain-owner 111122223333 --
repository my_repo  \
    --format npm
```

네임스페이스(npm 범위)에 있는 npm 패키지를 나열하려면 `--namespace` 및 `--format` 옵션을 사용하세요.

> **Important**
>
> `--namespace` 옵션 값에는 앞에 `@`이 올 수 없습니다. `@types` 네임스페이스를 검색하려면 값을 *##*으로 설정하세요.

> **Note**
>
> `--namespace` 옵션은 네임스페이스 접두사를 기준으로 필터링합니다. `--namespace` 옵션에 전달된 값으로 시작하는 범위를 가진 모든 npm 패키지가 `list-packages` 응답에서 반환됩니다.

```
aws codeartifact list-packages --domain my_domain --domain-owner 111122223333 --
repository my_repo  \
    --format npm --namespace types
```

샘플 출력:

```
{
    "nextToken": "eyJidWNrZXRJZ...",
    "packages": [
        {
            "package": "3d-bin-packing",
            "namespace": "types",
            "format": "npm"

        },
        {
            "package": "a-big-triangle",
            "namespace": "types",
            "format": "npm"

        },
        {
            "package": "a11y-dialog",
            "namespace": "types",
```

```
            "format": "npm"

        }
    ]
}
```

Maven 패키지 이름 나열

Maven 패키지 이름만 나열하려면 `--format` 옵션 값을 maven으로 설정합니다. 또한 `--namespace` 옵션에서 Maven 그룹 ID를 지정해야 합니다.

> **Note**
>
> `--namespace` 옵션은 네임스페이스 접두사를 기준으로 필터링합니다. `--namespace` 옵션에 전달된 값으로 시작하는 범위를 가진 모든 npm 패키지가 `list-packages` 응답에서 반환됩니다.

```
aws codeartifact list-packages --domain my_domain --domain-owner 111122223333 --
repository my_repo  \
    --format maven --namespace org.apache.commons
```

샘플 출력:

```
{
    "nextToken": "eyJidWNrZXRJZ...",
    "packages": [
        {
            "package": "commons-lang3",
            "namespace": "org.apache.commons",
            "format": "maven"

        },
        {
            "package": "commons-collections4",
            "namespace": "org.apache.commons",
            "format": "maven"

        },
        {
            "package": "commons-compress",
```

```
            "namespace": "org.apache.commons",
            "format": "maven"

        }
    ]
}
```

Python 패키지 이름 나열

Python 패키지 이름만 나열하려면 `--format` 옵션 값을 `pypi`로 설정합니다.

```
aws codeartifact list-packages --domain my_domain --domain-owner 111122223333 --
repository my_repo  \
    --format pypi
```

패키지 이름 접두사 기준으로 필터링

지정된 문자열로 시작하는 패키지를 반환하려면 `--package-prefix` 옵션을 사용합니다.

```
aws codeartifact list-packages --domain my_domain --domain-owner 111122223333 --
repository my_repo  \
    --format npm --package-prefix pat
```

샘플 출력:

```
{
    "nextToken": "eyJidWNrZXRJZ...",
    "packages": [
        {
            "package": "path",
            "format": "npm"

        },
        {
            "package": "pat-test",
            "format": "npm"

        },
        {
            "package": "patch-math3",
            "format": "npm"
```

```
        }
    ]
}
```

지원되는 검색 옵션 조합

`--format`, `--namespace` 및 `--package-prefix` 옵션은 마음껏 조합해서 사용할 수 있으며, `--namespace`는 단독으로 사용할 수 없습니다. 범위가 `@types`로 시작하는 모든 npm 패키지를 검색하려면 `--format` 옵션을 지정해야 합니다. `--namespace`를 단독으로 사용하면 오류가 발생합니다.

`list-packages`에서는 세 가지 옵션 중 어느 것도 사용하지 않을 수 있으며, 리포지토리에 있는 모든 형식의 모든 패키지가 반환됩니다.

출력 형식

모든 AWS CLI 명령에 사용할 수 있는 파라미터를 사용하여 `list-packages` 응답을 압축하고 더 읽기 쉽게 만들 수 있습니다. `--query` 파라미터를 사용하여, 반환되는 각 패키지 버전의 형식을 지정합니다. `--output` 파라미터를 사용하여 응답의 형식을 일반 텍스트로 지정합니다.

```
aws codeartifact list-packages --domain my_domain --domain-owner 111122223333 --
repository my_repo \
  --output text --query 'packages[*].[package]'
```

샘플 출력:

```
accepts
array-flatten
body-parser
bytes
content-disposition
content-type
cookie
cookie-signature
```

자세한 내용은 AWS Command Line Interface 사용 설명서의 AWS CLI의 명령 출력 제어를 참조하세요.

기본값 및 기타 옵션

기본적으로 `list-packages`는 결과를 100개까지 반환합니다. `--max-results` 옵션을 사용하면 이 결과 제한을 변경할 수 있습니다.

```
aws codeartifact list-packages --domain my_domain --domain-owner 111122223333 --
repository my_repo --max-results 20
```

--max-results에 허용되는 최대값은 1,000입니다. 1,000개 이상의 패키지가 있는 리포지토리의 패키지를 나열할 수 있도록, list-packages에서는 응답의 nextToken 필드를 사용한 페이지 매김을 지원합니다. 리포지토리에 있는 패키지 수가 --max-results 값보다 많다면, nextToken 값을 list-packages의 다른 호출에 전달하여 다음 결과 페이지를 가져올 수 있습니다.

```
aws codeartifact list-packages --domain my_domain --domain-owner 111122223333 --
repository my_repo \
  --next-token r00ABXNyAEdjb...
```

패키지 버전 나열

in AWS CodeArtifact list-package-versions 명령을 사용하여 리포지토리에 있는 패키지 이름의 모든 버전 목록을 가져옵니다.

```
aws codeartifact list-package-versions --package kind-of \
--domain my_domain --domain-owner 111122223333 \
--repository my_repository --format npm
```

샘플 출력:

```
{
  "defaultDisplayVersion": "1.0.1",
  "format": "npm",
  "package": "kind-of",
  "versions": [
      {
          "version": "1.0.1",
          "revision": "REVISION-SAMPLE-1-C7F4S5E9B772FC",
          "status": "Published",
          "origin": {
                "domainEntryPoint": {
                    "externalConnectionName": "public:npmjs"
                },
                "originType": "EXTERNAL"
            }
      },
      {
```

```
            "version": "1.0.0",
            "revision": "REVISION-SAMPLE-2-C752BEEF6D2CFC",
            "status": "Published",
            "origin": {
                  "domainEntryPoint": {
                      "externalConnectionName": "public:npmjs"
                  },
                  "originType": "EXTERNAL"
              }
        },
        {
            "version": "0.1.2",
            "revision": "REVISION-SAMPLE-3-654S65A5C5E1FC",
            "status": "Published",
            "origin": {
                  "domainEntryPoint": {
                      "externalConnectionName": "public:npmjs"
                  },
                  "originType": "EXTERNAL"
              }
        },
        {
            "version": "0.1.1",
            "revision": "REVISION-SAMPLE-1-C7F4S5E9B772FC"",
            "status": "Published",
            "origin": {
                  "domainEntryPoint": {
                      "externalConnectionName": "public:npmjs"
                  },
                  "originType": "EXTERNAL"
              }
        },
        {
            "version": "0.1.0",
            "revision": "REVISION-SAMPLE-4-AF669139B772FC",
            "status": "Published",
            "origin": {
                  "domainEntryPoint": {
                      "externalConnectionName": "public:npmjs"
                  },
                  "originType": "EXTERNAL"
              }
        }
    ]
```

```
}
```

list-package-versions 호출에 --status 파라미터를 추가하면 패키지 버전 상태를 기준으로 결과를 필터링할 수 있습니다. 패키지 버전 상태에 대한 자세한 내용은 패키지 버전 상태 섹션을 참조하세요.

--max-results 및 --next-token 파라미터를 사용하여 list-package-versions의 응답에 페이지를 매길 수 있습니다. --max-results의 경우 1에서 1000 사이의 정수를 지정하여 한 페이지에 반환되는 결과 수를 지정합니다. 기본값은 50입니다. 후속 페이지를 반환하려면 list-package-versions를 다시 실행하고 이전 명령 출력에서 받은 nextToken 값을 --next-token에 전달하세요. --next-token 옵션을 사용하지 않으면 결과의 첫 페이지가 항상 반환됩니다.

list-package-versions 명령은 업스트림 리포지토리의 패키지 버전은 나열하지 않습니다. 하지만 패키지 버전 요청 중에 리포지토리로 복사된 업스트림 리포지토리의 패키지 버전에 대한 참조는 나열됩니다. 자세한 내용은 CodeArtifact에서의 업스트림 리포지토리 작업 단원을 참조하십시오.

npm 패키지 버전 나열

npm 패키지의 모든 패키지 버전을 나열하려면 --format 옵션 값을 npm으로 설정합니다.

```
aws codeartifact list-package-versions --package my_package --domain my_domain \
 --domain-owner 111122223333 --repository my_repo --format npm
```

특정 네임스페이스(npm 범위)에 있는 npm 패키지 버전을 나열하려면 --namespace 옵션을 사용하세요. --namespace 옵션 값에는 앞에 @이 올 수 없습니다. @types 네임스페이스를 검색하려면 값을 ##으로 설정하세요.

```
aws codeartifact list-package-versions --package my_package --domain my_domain \
 --domain-owner 111122223333 --repository my_repo --format npm \
 --namespace types
```

Maven 패키지 버전 나열

Maven 패키지의 모든 패키지 버전을 나열하려면 --format 옵션 값을 maven으로 설정합니다. 그리고 --namespace 옵션에서 Maven 그룹 ID를 지정해야 합니다.

```
aws codeartifact list-package-versions --package my_package --domain my_domain \
 --domain-owner 111122223333 --repository my_repo --format maven \
 --namespace org.apache.commons
```

버전 정렬

`list-package-versions`는 게시 시간을 기준으로 내림차순으로 정렬된 버전을 출력할 수 있습니다(가장 최근에 게시된 버전이 먼저 나열됨). 다음과 같이 값이 `PUBLISHED_TIME`인 `--sort-by` 파라미터를 사용합니다.

```
aws codeartifact list-package-versions --domain my_domain --domain-owner 111122223333
 --repository my_repository  \
--format npm --package webpack --max-results 5 --sort-by PUBLISHED_TIME
```

샘플 출력:

```
{

  "defaultDisplayVersion": "4.41.2",
  "format": "npm",
  "package": "webpack",
  "versions": [
      {
        "version": "5.0.0-beta.7",
        "revision": "REVISION-SAMPLE-1-C7F4S5E9B772FC",
        "status": "Published"
      },
      {
        "version": "5.0.0-beta.6",
        "revision": "REVISION-SAMPLE-2-C752BEEF6D2CFC",
        "status": "Published"
      },
      {
        "version": "5.0.0-beta.5",
        "revision": "REVISION-SAMPLE-3-654S65A5C5E1FC",
        "status": "Published"
      },
      {
        "version": "5.0.0-beta.4",
        "revision": "REVISION-SAMPLE-4-AF669139B772FC",
        "status": "Published"
      },
      {
        "version": "5.0.0-beta.3",
        "revision": "REVISION-SAMPLE-5-C752BEE9B772FC",
        "status": "Published"
```

```
        }
    ],
    "nextToken": "eyJsaXN0UGF...."
}
```

기본 표시 버전

`defaultDisplayVersion` 반환 값은 패키지 형식에 따라 달라집니다.

- 일반, Maven 및 PyPI 패키지의 경우 반환 값은 가장 최근에 게시된 패키지 버전입니다.
- npm 패키지의 경우 반환 값은 `latest` 태그가 참조하는 버전입니다. `latest` 태그가 설정되지 않은 경우 반환 값은 가장 최근에 게시된 패키지 버전입니다.

출력 형식

모든 AWS CLI 명령에 사용할 수 있는 파라미터를 사용하여 `list-package-versions` 응답을 압축하고 더 읽기 쉽게 만들 수 있습니다. `--query` 파라미터를 사용하여, 반환되는 각 패키지 버전의 형식을 지정합니다. `--output` 파라미터를 사용하여 응답의 형식을 일반 텍스트로 지정합니다.

```
aws codeartifact list-package-versions --package my-package-name --domain my_domain --
domain-owner 111122223333 \
--repository my_repo --format npm --output text --query 'versions[*].[version]'
```

샘플 출력:

```
0.1.1
0.1.2
0.1.0
3.0.0
```

자세한 내용은 AWS Command Line Interface 사용 설명서의 AWS CLI의 명령 출력 제어를 참조하세요.

패키지 버전 자산 나열

자산은 패키지 버전과 연결된 CodeArtifact에 저장된 개별 파일(예: npm `.tgz` 파일 또는 Maven POM 또는 JAR 파일)입니다. `list-package-version-assets` 명령을 사용하여 각 패키지 버전의 자산을 나열할 수 있습니다.

`list-package-version-assets` 명령을 실행하여 AWS 계정 및 현재 AWS 리전의 각 자산에 대한 다음 정보를 반환합니다.

- 자산 이름.
- 크기(단위: 바이트).
- 체크섬 검증에 사용한 해시 값 세트.

예를 들어 다음 명령을 사용하여 Python 패키지 `flatten-json`, 버전 `0.1.7`의 자산을 나열합니다.

```
aws codeartifact list-package-version-assets --domain my_domain --domain-
owner 111122223333 \
 --repository my_repo --format pypi --package flatten-json \
 --package-version 0.1.7
```

다음은 출력값을 보여줍니다.

```
{
    "format": "pypi",
    "package": "flatten-json",
    "version": "0.1.7",
    "versionRevision": "REVISION-SAMPLE-1-C7F4S5E9B772FC",
    "assets": [
        {
            "name": "flatten_json-0.1.7-py3-none-any.whl",
            "size": 31520,
            "hashes": {
                "MD5": "41bba98d5b9219c43089eEXAMPLE-MD5",
                "SHA-1": "69b215c25dd4cda1d997a786ec6EXAMPLE-SHA-1",
                "SHA-256": "43f24850b7b7b7d79c5fa652418518fbdf427e602b1edabe6EXAMPLE-
SHA-256",
                "SHA-512":
 "3947382ac2c180ee3f2aba4f8788241527c8db9dfe9f4b039abe9fc560aaf5a1fced7bd1e80a0dca9ce320d95f086
SHA-512"
            }
        },
        {
            "name": "flatten_json-0.1.7.tar.gz",
            "size": 2865,
            "hashes": {
                "MD5": "41bba98d5b9219c43089eEXAMPLE-MD5",
                "SHA-1": "69b215c25dd4cda1d997a786ec6EXAMPLE-SHA-1",
```

```
                "SHA-256": "43f24850b7b7b7d79c5fa652418518fbdf427e602b1edabe6EXAMPLE-
SHA-256",
                "SHA-512":
 "3947382ac2c180ee3f2aba4f8788241527c8db9dfe9f4b039abe9fc560aaf5a1fced7bd1e80a0dca9ce320d95f086
SHA-512"
            }
        }
    ]
}
```

npm 패키지의 자산 나열

npm 패키지에는 항상 이름이 `package.tgz`인 단일 자산이 있습니다. 범위가 지정된 npm 패키지의 자산을 나열하려면 `--namespace` 옵션에 범위를 포함하세요.

```
aws codeartifact list-package-version-assets --domain my_domain --domain-
owner 111122223333 \
 --repository my_repo --format npm --package webpack \
 --namespace types --package-version 4.9.2
```

Maven 패키지의 자산 나열

Maven 패키지의 자산을 나열하려면 `--namespace` 옵션에 패키지 네임스페이스를 포함하세요. Maven 패키지 `commons-cli:commons-cli`의 자산을 나열하는 방법은 다음과 같습니다.

```
aws codeartifact list-package-version-assets --domain my_domain --domain-
owner 111122223333 \
 --repository my_repo --format maven --package commons-cli \
 --namespace commons-cli --package-version 1.0
```

패키지 버전 자산 다운로드

자산은 패키지 버전과 연결된 CodeArtifact에 저장된 개별 파일(예: npm `.tgz` 파일 또는 Maven POM 또는 JAR 파일)입니다. `get-package-version-assets command`를 사용하여 패키지 자산을 다운로드할 수 있습니다. 이렇게 하면 npm이나 `pip` 같은 패키지 관리자 클라이언트를 사용하지 않고도 자산을 검색할 수 있습니다. 자산을 다운로드하려면 `list-package-version-assets` 명령을 사용하여 얻을 수 있는 자산 이름을 제공해야 합니다. 자세한 내용은 패키지 버전 자산 나열 섹션을 참조하세요. 자산은 사용자가 지정한 파일 이름으로 로컬 스토리지에 다운로드됩니다.

다음 예제에서는 버전이 *27.1-jre*인 Maven 패키지 *com.google.guava:guava*에서 *guava-27.1-jre.jar* 자산을 다운로드합니다.

```
aws codeartifact get-package-version-asset --domain my_domain --domain-
owner 111122223333 --repository my_repo \
    --format maven --namespace com.google.guava --package guava --package-version 27.1-
jre \
    --asset guava-27.1-jre.jar \
    guava-27.1-jre.jar
```

이 예제에서는 이전 명령의 마지막 인수를 사용하여 파일 이름을 *guava-27.1-jre.jar*로 지정했으므로, 다운로드한 자산의 이름은 *guava-27.1-jre.jar*로 지정됩니다.

명령의 출력은 다음과 같습니다.

```
{
    "assetName": "guava-27.1-jre.jar",
    "packageVersion": "27.1-jre",
    "packageVersionRevision": "YGp9ck2tmy03PGSxioclfYzQ0BfTLR9zzhQJtERv62I="
}
```

> **Note**
>
> 범위가 지정된 npm 패키지에서 자산을 다운로드하려면 `--namespace` 옵션에 범위를 포함하세요. `--namespace`를 사용할 때는 `@` 기호를 생략해야 합니다. 예를 들어 범위가 `@types`라면 `--namespace types`를 사용합니다.

`get-package-version-asset`을 사용하여 자산을 다운로드하려면 패키지 리소스에 대한 `codeartifact:GetPackageVersionAsset` 권한이 필요합니다. 리소스 기반 정책에 대한 자세한 내용은 AWS Identity and Access Management 사용 설명서의 리소스 기반 정책을 참조하세요.

리포지토리 간 패키지 복사

CodeArtifact에서는 특정 리포지토리의 패키지 버전을 다른 리포지토리로 복사할 수 있습니다. 이 기능은 패키지 프로모션 워크플로우나 팀 또는 프로젝트 간 패키지 버전 공유 같은 시나리오에 유용합니다. 소스 리포지토리와 대상 리포지토리는 패키지 버전을 복사하는 도메인과 동일한 도메인에 있어야 합니다.

패키지 복사에 필요한 IAM 권한

CodeArtifact에서 패키지 버전을 복사하려면, 호출하는 사용자에게 필수 IAM 권한이 있어야 하고 소스 및 대상 리포지토리에 연결된 리소스 기반 정책에도 필수 권한이 있어야 합니다. 리소스 기반 권한 정책 및 CodeArtifact 리포지토리에 대한 자세한 내용은 리포지토리 정책 섹션을 참조하세요.

`copy-package-versions`를 호출하는 사용자에게는 소스 리포지토리에 대한 `ReadFromRepository` 권한과 대상 리포지토리에 대한 `CopyPackageVersions` 권한이 있어야 합니다.

소스 리포지토리에는 `ReadFromRepository` 권한이 있어야 하고 대상 리포지토리에는 IAM 계정 또는 사용자 복사 패키지에 할당된 `CopyPackageVersions` 권한이 있어야 합니다. 다음 정책은 `put-repository-permissions-policy` 명령을 사용하여 소스 리포지토리 또는 대상 리포지토리에 추가할 예제 리포지토리 정책입니다. *111122223333*을 계정 호출 `copy-package-versions`의 ID로 바꿉니다.

> **Note**
>
> 현재 리포지토리 정책이 있는 경우 `put-repository-permissions-policy`를 호출하면 현재 정책이 대체됩니다. `get-repository-permissions-policy` 명령을 사용하면 정책이 존재하는지 확인할 수 있습니다. 자세한 내용은 정책 읽기 섹션을 참조하세요. 정책이 존재한다면, 정책을 바꾸는 대신 이러한 권한을 추가하는 것이 좋습니다.

소스 리포지토리 권한 정책 예시

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "codeartifact:ReadFromRepository"
            ],
            "Effect": "Allow",
            "Principal": {
                 "AWS": "arn:aws:iam::111122223333:root"
            },
            "Resource": "*"
```

```
        }
    ]
}
```

대상 리포지토리 권한 정책 예제

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "codeartifact:CopyPackageVersions"
            ],
            "Effect": "Allow",
            "Principal": {
                 "AWS": "arn:aws:iam::111122223333:root"
            },
            "Resource": "*"
        }
    ]
}
```

패키지 버전 복사

CodeArtifact에서 `copy-package-versions` 명령을 사용하여 소스 리포지토리에 있는 하나 이상의 패키지 버전을 동일한 도메인에 있는 대상 리포지토리로 복사합니다. 다음 예제는 `my_repo` 리포지토리에 있는 `my-package`라는 npm 패키지의 버전 6.0.2 및 4.0.0을 `repo-2` 리포지토리로 복사합니다.

```
aws codeartifact copy-package-versions --domain my_domain --domain-owner 111122223333
 --source-repository my_repo \
 --destination-repository repo-2 --package my-package --format npm \
 --versions 6.0.2 4.0.0
```

단일 작업으로 패키지 이름이 같은 여러 버전을 복사할 수 있습니다. 패키지 이름이 다른 버전을 복사하려면 버전별로 `copy-package-versions`를 호출해야 합니다.

두 버전을 모두 성공적으로 복사할 수 있다면, 앞의 명령은 다음과 같은 출력을 생성합니다.

```
{
   "successfulVersions": {
        "6.0.2": {
            "revision": "REVISION-1-SAMPLE-6C81EFF7DA55CC",
            "status": "Published"
        },
        "4.0.0": {
            "revision": "REVISION-1-SAMPLE-6C81EFF7DA55CC",
            "status": "Published"
        }
    },
    "failedVersions": {}
}
```

업스트림 리포지토리에서 패키지 복사

일반적으로 copy-package-versions는 복사할 버전을 --source-repository 옵션에서 지정한 리포지토리에서만 찾습니다. 하지만 --include-from-upstream 옵션을 사용하면 소스 리포지토리와 업스트림 리포지토리 모두에서 버전을 복사할 수 있습니다. CodeArtifact SDK를 사용하는 경우 includeFromUpstream 파라미터를 true로 설정하여 CopyPackageVersions API를 호출하세요. 자세한 내용은 CodeArtifact에서의 업스트림 리포지토리 작업 단원을 참조하십시오.

범위가 지정된 npm 패키지 복사

범위 내의 npm 패키지 버전을 복사하려면 --namespace 옵션을 사용하여 범위를 지정하세요. 예를 들어 @types/react 패키지를 복사하려면 --namespace types를 사용합니다. --namespace를 사용할 때는 @ 기호를 생략해야 합니다.

```
aws codeartifact copy-package-versions  --domain my_domain --domain-owner 111122223333
 --source-repository repo-1 \
 --destination-repository repo-2 --format npm --namespace types \
 --package react --versions 0.12.2
```

Maven 패키지 버전 복사

리포지토리 간에 Maven 패키지 버전을 복사하려면 --namespace 옵션을 적용해 Maven 그룹 ID를 전달하고 --name 옵션을 적용해 Maven artifactID를 전달하여 복사할 패키지를 지정하세요. 예를 들어 com.google.guava:guava의 단일 버전을 복사하는 방법은 다음과 같습니다.

```
 aws codeartifact copy-package-versions --domain my_domain --domain-owner 111122223333
 \
 --source-repository my_repo --destination-repository repo-2 --format maven --
namespace com.google.guava \
 --package guava --versions 27.1-jre
```

패키지 버전이 성공적으로 복사되면 다음과 비슷한 출력이 표시됩니다.

```
{
    "successfulVersions": {
        "27.1-jre": {
            "revision": "REVISION-1-SAMPLE-6C81EFF7DA55CC",
            "status": "Published"
        }
    },
    "failedVersions": {}
}
```

소스 리포지토리에 없는 버전

소스 리포지토리에 없는 버전을 지정하면 복사가 되지 않습니다. 소스 리포지토리에 있는 버전도 있고 없는 버전도 있다면, 모든 버전이 복사되지 않습니다. 다음 예제에서 `array-unique` npm 패키지의 버전 0.2.0은 소스 리포지토리에 있지만 버전 5.6.7은 그렇지 않습니다.

```
aws codeartifact copy-package-versions --domain my_domain --domain-owner 111122223333 \
        --source-repository my_repo --destination-repository repo-2 --format npm \
        --package array-unique --versions 0.2.0 5.6.7
```

이 시나리오에서는 다음과 비슷한 출력이 표시됩니다.

```
{
    "successfulVersions": {},
    "failedVersions": {
        "0.2.0": {
            "errorCode": "SKIPPED",
            "errorMessage": "Version 0.2.0 was skipped"
        },
        "5.6.7": {
            "errorCode": "NOT_FOUND",
            "errorMessage": "Could not find version 5.6.7"
        }
```

```
    }
}
```

SKIPPED 오류 코드는 다른 버전을 복사할 수 없어 해당 버전이 대상 리포지토리에 복사되지 않았음을 나타내는 데 사용합니다.

대상 리포지토리에 이미 존재하는 버전

패키지 버전이 이미 존재하는 리포지토리에 복사되면, CodeArtifact는 두 리포지토리의 패키지 자산과 패키지 버전 수준 메타데이터를 비교합니다.

소스 및 대상 리포지토리의 패키지 버전 자산과 메타데이터가 동일한 경우 복사가 수행되지 않지만 작업은 성공한 것으로 간주됩니다. 즉, `copy-package-versions`는 멱등성입니다. 이 경우 소스 리포지토리와 대상 리포지토리에 이미 존재하는 버전은 `copy-package-versions`의 출력에 나열되지 않습니다.

다음 예제에서는 npm 패키지 `array-unique`의 두 버전이 소스 리포지토리 `repo-1`에 존재합니다. 버전 0.2.1도 대상 리포지토리 `dest-repo`에 있지만 버전 0.2.0은 없습니다.

```
aws codeartifact copy-package-versions --domain my_domain --domain-owner 111122223333 \
         --source-repository my_repo --destination-repository repo-2 --format npm --
package array-unique \
         --versions 0.2.1 0.2.0
```

이 시나리오에서는 다음과 비슷한 출력이 표시됩니다.

```
{
    "successfulVersions": {
        "0.2.0": {
            "revision": "Yad+B1QcBq2kdEVrx1E1vSfHJVh8Pr61hBUkoWPGWX0=",
            "status": "Published"
        }
    },
    "failedVersions": {}
}
```

버전 0.2.0은 소스에서 대상 리포지토리로 성공적으로 복사되었기 때문에 `successfulVersions`에 나열됩니다. 버전 0.2.1은 대상 리포지토리에 이미 존재하므로 출력에 표시되지 않습니다.

패키지 버전 자산 또는 메타데이터가 소스 및 대상 리포지토리에서 서로 다르다면 복사 작업이 실패하게 됩니다. `--allow-overwrite` 파라미터를 사용하여 강제로 덮어쓸 수 있습니다.

대상 리포지토리에 있는 버전도 있고 없는 버전도 있다면, 모든 버전이 복사되지 않습니다. 다음 예제에서 array-unique npm 패키지의 버전 0.3.2는 소스 리포지토리와 대상 리포지토리에 모두 존재하지만 패키지 버전의 콘텐츠가 다릅니다. 버전 0.2.1은 소스 리포지토리에는 있지만 대상 리포지토리에는 없습니다.

```
aws codeartifact copy-package-versions --domain my_domain --domain-owner 111122223333 \
        --source-repository my_repo --destination-repository repo-2 --format npm --
package array-unique \
        --versions 0.3.2 0.2.1
```

이 시나리오에서는 다음과 비슷한 출력이 표시됩니다.

```
{
    "successfulVersions": {},
    "failedVersions": {
        "0.2.1": {
            "errorCode": "SKIPPED",
            "errorMessage": "Version 0.2.1 was skipped"
        },
        "0.3.2": {
            "errorCode": "ALREADY_EXISTS",
            "errorMessage": "Version 0.3.2 already exists"
        }
    }
}
```

버전 0.2.1은 대상 리포지토리에 복사되지 않았기 때문에 SKIPPED로 표시됩니다. 버전 0.3.2는 대상 리포지토리에 존재했지만 소스 및 대상 리포지토리에서 동일하지 않아 복사에 실패했습니다.

패키지 버전 개정 지정

패키지 버전 개정은 패키지 버전의 특정 자산 및 메타데이터 세트를 지정하는 문자열입니다. 패키지 버전 개정을 지정하여 특정 상태의 패키지 버전을 복사할 수 있습니다. 패키지 버전 개정을 지정하려면 --version-revisions 파라미터를 사용하여, 쉼표로 구분된 패키지 버전 하나 이상과 패키지 버전 개정 쌍을 copy-package-versions 명령에 전달합니다.

> **Note**
>
> copy-package-versions가 있는 --versions 또는 --version-revisions 파라미터를 지정해야 합니다. 둘 다 지정할 수는 없습니다.

다음 예제에서는 패키지 버전 개정 REVISION-1-SAMPLE-6C81EFF7DA55CC와 함께 소스 리포지토리에 존재하는 패키지 my-package의 버전 0.3.2만 복사합니다.

```
aws codeartifact copy-package-versions --domain my_domain --domain-owner 111122223333
 --source-repository repo-1 \
 --destination-repository repo-2 --format npm --namespace my-namespace \
 --package my-package --version-revisions 0.3.2=REVISION-1-SAMPLE-6C81EFF7DA55CC
```

다음 예제에서는 패키지 my-package의 두 버전(0.3.2 및 0.3.13)을 복사합니다. my-package의 소스 리포지토리 버전 0.3.2에 개정 REVISION-1-SAMPLE-6C81EFF7DA55CC가 있고 버전 0.3.13에 개정 REVISION-2-SAMPLE-55C752BEE772FC가 있는 경우에만 복사가 성공합니다.

```
aws codeartifact copy-package-versions --domain my_domain --domain-owner 111122223333
 --source-repository repo-1 \
 --destination-repository repo-2 --format npm --namespace my-namespace \
 --package my-package --version-revisions 0.3.2=REVISION-1-
SAMPLE-6C81EFF7DA55CC,0.3.13=REVISION-2-SAMPLE-55C752BEE772FC
```

패키지 버전의 개정을 찾으려면 describe-package-version 또는 list-package-versions 명령을 사용합니다.

자세한 내용은 CodeArtifact API 참조의 패키지 버전 개정 및 CopyPackageVersion을 참조하세요.

npm 패키지 복사

npm 패키지의 copy-package-versions 동작에 대한 자세한 내용은 npm 태그 및 CopyPackageVersions API를 참조하세요.

패키지 또는 패키지 버전 삭제

delete-package-versions 명령을 사용하여 한 번에 하나 이상의 패키지 버전을 삭제할 수 있습니다. 리포지토리에서 관련된 모든 버전 및 구성을 포함하여 패키지를 완전히 제거하려면 delete-package 명령을 사용합니다. 패키지는 패키지 버전 없이 리포지토리에 존재할 수 있습니다. delete-package-versions 명령을 사용하여 모든 버전을 삭제하거나, put-package-origin-configuration API 작업을 사용하여 버전 없이 패키지를 만들면 이렇게 될 수 있습니다(패키지 원본 제어 편집 참조).

주제

패키지 삭제(AWS CLI)

`delete-package` 명령을 사용하면 패키지를 모든 패키지 버전 및 구성과 함께 삭제할 수 있습니다. 다음 예제에서는 `my_domain` 도메인의 `my_repo` 리포지토리에 있는 `my-package`라는 PyPI 패키지를 삭제합니다.

```
aws codeartifact delete-package --domain my_domain --domain-owner 111122223333 \
--repository my_repo --format pypi \
--package my-package
```

샘플 출력:

```
{
   "deletedPackage": {
      "format": "pypi",
      "originConfiguration": {
         "restrictions": {
            "publish": "ALLOW",
            "upstream": "BLOCK"
         }
      },
      "package": "my-package"
   }
}
```

동일한 패키지 이름에 대해 `describe-package`를 실행하면 패키지가 삭제되었는지 확인할 수 있습니다.

```
aws codeartifact describe-package --domain my_domain --domain-owner 111122223333 \
```

```
--repository my_repo --format pypi --package my-package
```

패키지 삭제(콘솔)

1. https://console.aws.amazon.com/codesuite/codeartifact/home AWS CodeArtifact 콘솔을 엽니다.
2. 탐색 창에서 리포지토리를 선택합니다.
3. 패키지를 삭제할 리포지토리를 선택합니다.
4. 삭제할 패키지를 선택합니다.
5. 패키지 삭제를 선택합니다.

패키지 버전 삭제(AWS CLI)

`delete-package-versions` 명령을 사용하여 한 번에 하나 이상의 패키지 버전을 삭제할 수 있습니다. 다음 예제는 `my_domain` 도메인의 `my_repo`에 있는 `my-package`라는 PyPI 패키지의 버전 `4.0.0`, `4.0.1` 및 `5.0.0`을 삭제합니다.

```
aws codeartifact delete-package-versions --domain my_domain --domain-owner 111122223333
 \
--repository my_repo --format pypi \
--package my-package --versions 4.0.0 4.0.1 5.0.0
```

샘플 출력:

```
{
   "successfulVersions": {
      "4.0.0": {
         "revision": "oxwwYC9dDeuBoCt6+PDSwL6OMZ7rXeiXy44BM32Iawo=",
            "status": "Deleted"
      },
      "4.0.1": {
         "revision": "byaaQR748wrsdBaT+PDSwL6OMZ7rXeiBKM0551aqWmo=",
            "status": "Deleted"
      },
      "5.0.0": {
         "revision": "yubm34QWeST345ts+ASeioPI354rXeiSWr734PotwRw=",
            "status": "Deleted"
      }
   },
```

```
    "failedVersions": {}
}
```

동일한 패키지 이름에 대해 `list-package-versions`를 실행하면 버전이 삭제되었는지 확인할 수 있습니다.

```
aws codeartifact list-package-versions --domain my_domain --domain-owner 111122223333 \
--repository my_repo --format pypi --package my-package
```

패키지 버전 삭제(콘솔)

1. https://console.aws.amazon.com/codesuite/codeartifact/home AWS CodeArtifact 콘솔을 엽니다.
2. 탐색 창에서 리포지토리를 선택합니다.
3. 패키지 버전을 삭제할 리포지토리를 선택합니다.
4. 버전을 삭제할 패키지를 선택합니다.
5. 삭제할 패키지 버전을 선택합니다.
6. 삭제를 선택합니다.

> **Note**
>
> 콘솔에서는 패키지 버전을 한 번에 하나만 삭제할 수 있습니다. 한 번에 두 개 이상을 삭제하려면 CLI를 사용하세요.

npm 패키지 또는 패키지 버전 삭제

npm 패키지 또는 개별 패키지 버전을 삭제하려면 `--format` 옵션을 npm으로 설정하세요. 범위 내의 npm 패키지의 패키지 버전을 삭제하려면 `--namespace` 옵션을 사용하여 범위를 지정하세요. 예를 들어 `@types/react` 패키지를 복사하려면 `--namespace types`를 사용합니다. `--namespace`를 사용할 때는 `@` 기호를 생략하세요.

```
aws codeartifact delete-package-versions --domain my_domain --domain-owner 111122223333
 \
--repository my_repo --format npm --namespace types \
--package react --versions 0.12.2
```

@types/react 패키지를 모든 관련 버전과 함께 삭제하는 방법은 다음과 같습니다.

```
aws codeartifact delete-package --domain my_domain --domain-owner 111122223333 \
--repository my_repo --format npm --namespace types \
--package react
```

Maven 패키지 또는 패키지 버전 삭제

Maven 패키지 또는 개별 패키지 버전을 삭제하려면, --format 옵션을 maven으로 설정하고 --namespace 옵션과 함께 Maven 그룹 ID를 전달하고 --name 옵션과 함께 Maven artifactID를 전달하여 삭제할 패키지를 지정합니다. 예를 들어 아래에서는 com.google.guava:guava의 단일 버전을 삭제하는 방법을 확인할 수 있습니다.

```
 aws codeartifact delete-package-versions --domain my_domain --domain-
owner 111122223333 \
--repository my_repo --format maven --namespace com.google.guava \
--package guava --versions 27.1-jre
```

다음 예제에서는 com.google.guava:guava 패키지를 모든 관련 버전과 함께 삭제하는 방법을 보여줍니다.

```
 aws codeartifact delete-package --domain my_domain --domain-owner 111122223333 \
--repository my_repo --format maven --namespace com.google.guava \
--package guava
```

패키지 또는 패키지 버전 삭제에 대한 모범 사례

패키지 버전을 삭제해야 하는 경우 삭제하려는 패키지 버전의 백업 복사본을 저장할 리포지토리를 생성하는 것이 좋습니다. 먼저 백업 리포지토리에 copy-package-versions를 직접 호출하여 이 작업을 수행할 수 있습니다.

```
aws codeartifact copy-package-versions --domain my_domain --domain-owner 111122223333
 --source-repository my_repo \
 --destination-repository repo-2 --package my-package --format npm \
 --versions 6.0.2 4.0.0
```

패키지 버전을 복사한 후에는 삭제하려는 패키지 또는 패키지 버전에서 delete-package-versions를 호출할 수 있습니다.

```
aws codeartifact delete-package-versions --domain my_domain --domain-owner 111122223333
 \
--repository my_repo --format pypi \
--package my-package --versions 4.0.0 4.0.1 5.0.0
```

패키지 버전 세부 정보 및 종속성 보기 및 업데이트

CodeArtifact에서는 종속성을 포함한 패키지 버전 관련 정보를 확인할 수 있습니다. 패키지 버전의 상태를 업데이트할 수도 있습니다. 패키지 버전 상태에 대한 자세한 내용은 패키지 버전 상태 섹션을 참조하세요.

패키지 버전 세부 정보 보기

`describe-package-version` 명령을 사용하여 패키지 버전 관련 세부 정보를 봅니다. 패키지 버전 세부 정보는 CodeArtifact에 게시될 때 패키지에서 추출됩니다. 각 패키지의 세부 정보는 패키지 형식과 작성자가 패키지에 추가한 정보의 양에 따라 달라집니다.

`describe-package-version` 명령 출력에 포함되는 대부분의 정보는 패키지 형식에 따라 달라집니다. 예를 들어 `describe-package-version`은 `package.json` 파일에서 npm 패키지의 정보를 추출합니다. 개정은 CodeArtifact에서 생성합니다. 자세한 내용은 패키지 버전 개정 지정 단원을 참조하십시오.

이름이 같은 두 패키지 버전은 서로 다른 네임스페이스에 있다면 동일한 리포지토리에 존재할 수 있습니다. 선택 사항인 `--namespace` 파라미터를 사용하여 네임스페이스를 지정하세요. 자세한 내용은 npm 패키지 버전 세부 정보 보기 또는 Maven 패키지 버전 세부 정보 보기을 참조하세요.

다음 예제는 `my_repo` 리포지토리에 있는 `pyhamcrest`라는 Python 패키지의 `1.9.0` 버전 관련 세부 정보를 반환합니다.

```
aws codeartifact describe-package-version --domain my_domain --domain-
owner 111122223333 --repository my_repo \
--format pypi --package pyhamcrest --package-version 1.9.0
```

출력은 다음과 같을 수 있습니다.

```
{
  "format": "pypi",
  "package": "PyHamcrest",
  "displayName": "PyHamcrest",
```

```
  "version": "1.9.0",
  "summary": "Hamcrest framework for matcher objects",
  "homePage": "https://github.com/hamcrest/PyHamcrest",
  "publishedTime": 1566002944.273,
  "licenses": [
    {
      "id": "license-id",
      "name": "license-name"
    }
  ],
  "revision": "REVISION-SAMPLE-55C752BEE9B772FC"
}
```

> **Note**
>
> CodeArtifact는 패키지 작성자가 제공한 메타데이터에서 패키지 홈 페이지 또는 패키지 라이선스 정보와 같은 패키지 버전 세부 정보를 가져옵니다. 이 정보 중 하나라도 DynamoDB 항목 크기 제한인 400KB를 초과하는 경우 CodeArtifact는 이러한 데이터를 처리할 수 없으며 콘솔 또는 `describe-package-version`의 응답에서 이 정보를 볼 수 없습니다. 예를 들어 https://pypi.org/project/rapyd-sdk/ 같은 Python 패키지에는 매우 큰 라이선스 필드가 있으므로 CodeArtifact에서 이 정보를 처리하지 않습니다.

npm 패키지 버전 세부 정보 보기

npm 패키지 버전의 세부 정보를 보려면 `--format` 옵션 값을 **npm**으로 설정하세요. 원한다면 `--namespace` 옵션에 패키지 버전 네임스페이스(npm 범위)를 포함해도 됩니다. `--namespace` 옵션 값에는 앞에 `@`이 올 수 없습니다. `@types` 네임스페이스를 검색하려면 값을 `##`으로 설정하세요.

다음 예제에서는 `@types` 범위에 있는 `webpack`이라는 npm 패키지의 `4.41.5` 버전 관련 세부 정보를 반환합니다.

```
aws codeartifact describe-package-version --domain my_domain --domain-
owner 111122223333 --repository my_repo \
--format npm --package webpack --namespace types --package-version 4.41.5
```

출력은 다음과 같을 수 있습니다.

```
{
  "format": "npm",
```

```
  "namespace": "types",
  "package": "webpack",
  "displayName": "webpack",
  "version": "4.41.5",
  "summary": "Packs CommonJs/AMD modules for the browser. Allows ... further output
 omitted for brevity",
  "homePage": "https://github.com/webpack/webpack",
  "sourceCodeRepository": "https://github.com/webpack/webpack.git",
  "publishedTime": 1577481261.09,
  "licenses": [
    {
      "id": "license-id",
      "name": "license-name"
    }
  ],
  "revision": "REVISION-SAMPLE-55C752BEE9B772FC",
  "status": "Published",
  "origin": {
            "domainEntryPoint": {
                "externalConnectionName": "public:npmjs"
            },
            "originType": "EXTERNAL"
  }
}
```

Maven 패키지 버전 세부 정보 보기

Maven 패키지 버전 세부 정보를 보려면 `--format` 옵션의 값을 `maven`으로 설정하고 `--namespace` 옵션에 패키지 버전 네임스페이스를 포함하세요.

다음 예제는 `org.apache.commons` 네임스페이스와 `my_repo` 리포지토리에 있는 `commons-rng-client-api`라는 Maven 패키지의 `1.2` 버전 관련 세부 정보를 반환합니다.

```
aws codeartifact describe-package-version --domain my_domain --domain-
owner 111122223333 --repository my_repo \
--format maven --namespace org.apache.commons --package commons-rng-client-api --
package-version 1.2
```

출력은 다음과 같을 수 있습니다.

```
{
  "format": "maven",
```

```
  "namespace": "org.apache.commons",
  "package": "commons-rng-client-api",
  "displayName": "Apache Commons RNG Client API",
  "version": "1.2",
  "summary": "API for client code that uses random numbers generators.",
  "publishedTime": 1567920624.849,
  "licenses": [],
  "revision": "REVISION-SAMPLE-55C752BEE9B772FC"
}
```

> **Note**
>
> CodeArtifact는 상위 POM 파일에서 패키지 버전 세부 정보를 추출하지 않습니다. 지정된 패키지 버전의 메타데이터에는 정확한 패키지 버전에 대한 정보만 POM에 포함되며, 상위 POM이나 POM `parent` 태그를 사용하여 전이적으로 참조하는 다른 POM에 대한 정보는 포함되지 않습니다. 즉, `parent` 참조를 사용하여 이 메타데이터를 포함하는 Maven 패키지 버전의 경우 `describe-package-version`의 출력에서 메타데이터(예: 라이선스 정보)가 생략됩니다.

패키지 버전 종속성 보기

`list-package-version-dependencies` 명령을 사용하여 패키지 버전의 종속성 목록을 가져옵니다. 다음 명령은 `my_domain` 도메인의 `my_repo` 리포지토리에 있는 `my-package`라는 npm 패키지 버전 `4.41.5`의 종속성을 나열합니다.

```
aws codeartifact list-package-version-dependencies --domain my_domain --domain-
owner 111122223333 --repository my_repo \
--format npm --package my-package --package-version 4.41.5
```

출력은 다음과 같을 수 있습니다.

```
{
  "dependencies": [
    {
      "namespace": "webassemblyjs",
      "package": "ast",
      "dependencyType": "regular",
      "versionRequirement": "1.8.5"
    },
    {
```

```
      "namespace": "webassemblyjs",
      "package": "helper-module-context",
      "dependencyType": "regular",
      "versionRequirement": "1.8.5"
    },
    {
      "namespace": "webassemblyjs",
      "package": "wasm-edit",
      "dependencyType": "regular",
      "versionRequirement": "1.8.5"
    }
  ],
  "versionRevision": "REVISION-SAMPLE-55C752BEE9B772FC"
}
```

dependencyType 필드에 지원되는 값의 범위는 CodeArtifact API의 PackageDependency 데이터 유형을 참조하세요.

패키지 버전 readme 파일 보기

npm과 같은 일부 패키지 형식에는 README 파일이 포함되어 있습니다. `get-package-version-readme`를 사용하여 패키지 버전의 README 파일을 가져옵니다. 다음 명령은 `my_domain` 도메인의 `my_repo` 리포지토리에 있는 `my-package`라는 npm 패키지 버전 `4.41.5`의 README 파일을 반환합니다.

> **Note**
>
> CodeArtifact는 일반 또는 Maven 패키지의 readme 파일을 표시하는 기능을 지원하지 않습니다.

```
aws codeartifact get-package-version-readme --domain my_domain --domain-
owner 111122223333 --repository my_repo \
--format npm --package my-package --package-version 4.41.5
```

출력은 다음과 같을 수 있습니다.

```
{
  "format": "npm",
  "package": "my-package",
```

```
  "version": "4.41.5"
  "readme": "<div align=\"center\">\n   <a href=\https://github.com/webpack/webpack
\"> ... more content ... \n",
  "versionRevision": "REVISION-SAMPLE-55C752BEE9B772FC"
}
```

패키지 버전 상태 업데이트

CodeArtifact의 각 패키지 버전에는 패키지 버전의 현재 상태 및 가용성을 설명하는 상태가 적용됩니다. AWS CLI 및 콘솔을 모두 사용하여 패키지 버전 상태를 변경할 수 있습니다.

> **Note**
>
> 사용 가능한 상태 목록을 포함한 패키지 버전 상태에 대한 자세한 내용은 패키지 버전 상태 섹션을 참조하세요.

패키지 버전 상태 업데이트

패키지 버전의 상태를 설정하면 리포지토리에서 패키지 버전을 완전히 삭제하지 않고도 패키지 버전 사용 방법을 제어할 수 있습니다. 예를 들어 패키지 버전의 상태가 `Unlisted`라면 정상적으로 다운로드할 수는 있지만, `npm view` 같은 명령으로 반환되는 패키지 버전 목록에 표시되지는 않습니다. UpdatePackageVersionsStatus API를 사용하면 단일 API 직접 호출로 동일한 패키지의 여러 버전에 대한 패키지 버전 상태를 설정할 수 있습니다. 다양한 상태에 대한 설명은 패키지 개요 섹션을 참조하세요.

`update-package-versions-status` 명령을 사용하여 패키지 버전의 상태를 `Published`, `Unlisted` 또는 `Archived`로 변경합니다. 명령을 사용하는 데 필요한 IAM 권한을 보려면 패키지 버전 상태를 업데이트하는 데 필요한 IAM 권한 섹션을 참조하세요. 다음 예제는 npm 패키지 `chalk` 버전 4.1.0의 상태를 `Archived`로 설정합니다.

```
aws codeartifact update-package-versions-status --domain my_domain
 --domain-owner 111122223333 --repository my_repo --format npm --package chalk
--versions 4.1.0 --target-status Archived
```

샘플 출력:

```
{
```

```
    "successfulVersions": {
        "4.1.0": {
            "revision": "+Oz8skWbwY3k8M6SrNIqNj6bVH/ax+CxvkJx+No5j8I=",
            "status": "Archived"
        }
    },
    "failedVersions": {}
}
```

이 예제에서는 npm 패키지를 사용하지만 명령은 다른 형식에서도 동일하게 작동합니다. 단일 명령을 사용하여 여러 버전을 동일한 대상 상태로 전환할 수 있습니다. 다음 예제를 참조하세요.

```
aws codeartifact update-package-versions-status --domain my_domain
 --domain-owner 111122223333 --repository my_repo --format npm --package chalk
--versions 4.1.0 4.1.1 --target-status Archived
```

샘플 출력:

```
{
    "successfulVersions": {
        "4.1.0": {
            "revision": "25/UjBleHs1DZewk+zozoeqH/R80Rc9gL1P8vbzVMJ4=",
            "status": "Archived"
        },
        "4.1.1": {
            "revision": "+Oz8skWbwY3k8M6SrNIqNj6bVH/ax+CxvkJx+No5j8I=",
            "status": "Archived"
        }
    },
    "failedVersions": {}
}
```

게시한 패키지 버전은 `Unfinished` 상태로 되돌릴 수 없으며, 따라서 이 상태는 `--target-status` 파라미터 값으로 사용할 수 없습니다. 패키지 버전을 `Disposed` 상태로 전환하려면 아래 설명처럼 `dispose-package-versions` 명령을 대신 사용하세요.

패키지 버전 상태를 업데이트하는 데 필요한 IAM 권한

패키지에 대한 `update-package-versions-status`를 요청하려면 패키지 리소스에 대한 `codeartifact:UpdatePackageVersionsStatus` 권한이 있어야 합니다. 즉, 패키지별로

update-package-versions-status를 호출하는 권한을 부여할 수 있습니다. 예를 들어 npm 패키지 *chalk*에서 update-package-versions-status를 호출할 수 있는 권한을 부여하는 IAM 정책에는 다음과 같은 설명이 포함됩니다.

```
{
  "Action": [
    "codeartifact:UpdatePackageVersionsStatus"
  ],
  "Effect": "Allow",
  "Resource": "arn:aws:codeartifact:us-east-1:111122223333:package/my_domain/my_repo/
npm//chalk"
}
```

범위가 지정된 npm 패키지의 상태 업데이트

범위가 지정된 npm 패키지 버전의 패키지 버전 상태를 업데이트하려면 --namespace 파라미터를 사용하세요. 예를 들어 @nestjs/core의 버전 8.0.0을 나열하지 않으려면 다음 명령을 사용합니다.

```
aws codeartifact update-package-versions-status --domain my_domain
 --domain-owner 111122223333 --repository my_repo --format npm --namespace nestjs
--package core --versions 8.0.0 --target-status Unlisted
```

Maven 패키지의 상태 업데이트

Maven 패키지에는 항상 그룹 ID가 있으며, CodeArtifact에서는 이를 네임스페이스라고 합니다. update-package-versions-status를 호출할 때 --namespace 파라미터를 사용하여 Maven 그룹 ID를 지정합니다. 예를 들어 Maven 패키지 org.apache.logging.log4j:log4j의 버전 2.13.1을 보관하려면 다음 명령을 사용합니다.

```
aws codeartifact update-package-versions-status --domain my_domain
 --domain-owner 111122223333 --repository my_repo --format maven
--namespace org.apache.logging.log4j --package log4j
--versions 2.13.1 --target-status Archived
```

패키지 버전 개정 지정

패키지 버전 개정은 패키지 버전의 특정 자산 및 메타데이터 세트를 지정하는 문자열입니다. 패키지 버전 개정을 지정하여 특정 상태에 있는 패키지 버전의 상태를 업데이트할 수 있습니다. 패키지 버전 개

정을 지정하려면 --version-revisions 파라미터를 사용하여, 쉼표로 구분된 패키지 버전 하나 이상과 패키지 버전 개정 쌍을 명령에 전달해야 합니다. 패키지 버전의 상태는 패키지 버전의 현재 개정이 지정된 값과 일치하는 경우에만 업데이트됩니다.

> **Note**
>
> --version-revisions 파라미터를 사용할 때는 –-versions 파라미터도 정의해야 합니다.

```
aws codeartifact update-package-versions-status --domain my_domain
   --domain-owner 111122223333 --repository my_repo --format npm --package chalk
   --version-revisions "4.1.0=25/UjBleHs1DZewk+zozoeqH/R80Rc9gL1P8bzVMJ4="
   --versions 4.1.0 --target-status Archived
```

단일 명령으로 여러 버전을 업데이트하려면 쉼표로 구분된 버전 및 버전 개정 쌍 목록을 --version-revisions 옵션에 전달해야 합니다. 다음 예제 명령은 서로 다른 두 가지 패키지 버전과 패키지 버전 개정 쌍을 정의합니다.

```
aws codeartifact update-package-versions-status --domain my_domain
 --domain-owner 111122223333 --repository my_repo --format npm
 --package chalk
 --version-revisions "4.1.0=25/UjBleHs1DZewk+zozoeqH/
R80Rc9gL1P8vbzVMJ4=,4.0.0=E3lhBp0RObRTut4pkjV5c1AQGkgSA7Oxtil6hMMzelc="
 --versions 4.1.0 4.0.0 --target-status Published
```

샘플 출력:

```
{
    "successfulVersions": {
        "4.0.0": {
            "revision": "E3lhBp0RObRTut4pkjV5c1AQGkgSA7Oxtil6hMMzelc=",
            "status": "Published"
        },
        "4.1.0": {
            "revision": "25/UjBleHs1DZewk+zozoeqH/R80Rc9gL1P8vbzVMJ4=",
            "status": "Published"
        }
    },
    "failedVersions": {}
```

```
}
```

여러 패키지 버전을 업데이트하는 경우 `--version-revisions`에 전달된 버전이 `--versions`에 전달된 버전과 동일해야 합니다. 개정을 잘못 지정하면 해당 버전의 상태가 업데이트되지 않습니다.

예상 상태 파라미터 사용

`update-package-versions-status` 명령은 패키지 버전의 예상 현재 상태 지정을 지원하는 `--expected-status` 파라미터를 제공합니다. 현재 상태가 `--expected-status`에 전달된 값과 일치하지 않으면 해당 패키지 버전의 상태가 업데이트되지 않습니다.

예를 들어 *my_repo*의 경우 npm 패키지 `chalk`의 버전 4.0.0 및 4.1.0은 현재 상태가 `Published`입니다. 예상 상태를 `Unlisted`로 지정하는 `update-package-versions-status`를 호출하면, 상태 불일치 때문에 두 패키지 버전 모두가 업데이트되지 않습니다.

```
aws codeartifact update-package-versions-status --domain my_domain
--domain-owner 111122223333 --repository my_repo --format npm --package chalk
--versions 4.1.0 4.0.0 --target-status Archived --expected-status Unlisted
```

샘플 출력:

```
{
    "successfulVersions": {},
    "failedVersions": {
        "4.0.0": {
            "errorCode": "MISMATCHED_STATUS",
            "errorMessage": "current status: Published, expected status: Unlisted"
        },
        "4.1.0": {
            "errorCode": "MISMATCHED_STATUS",
            "errorMessage": "current status: Published, expected status: Unlisted"
        }
    }
}
```

개별 패키지 버전 관련 오류

`update-package-versions-status` 호출 시 패키지 버전 상태가 업데이트되지 않는 데는 다양한 이유가 있습니다. 대표적인 이유는 패키지 버전 개정이 잘못 지정되었거나 예상 상태가 현재 상태와 일치하지 않는 것입니다. 이 경우 버전은 API 응답의 `failedVersions` 맵에 포함됩니다. 한 버전

에 오류가 발생하면 update-package-versions-status에 대한 동일한 호출에 지정된 다른 버전을 건너뛰며 상태가 업데이트되지 않을 수 있습니다. 이러한 버전도 SKIPPED errorCode가 있는 failedVersions 맵에 포함됩니다.

현재 update-package-versions-status 구현에서는 하나 이상의 버전이 상태를 변경할 수 없는 경우 다른 버전은 모두 건너뜁니다. 즉, 모든 버전이 성공적으로 업데이트되거나 어떤 버전도 업데이트되지 않게 됩니다. 이 동작은 API 계약에서 보장되지 않습니다. 향후에는 update-package-versions-status에 대한 단일 호출에서 일부 버전은 성공하지만 다른 버전은 실패하게 될 것입니다.

다음 예제 명령에는 패키지 버전 개정 불일치 때문에 발생한 버전 상태 업데이트 실패가 포함되어 있습니다. 이 업데이트 실패 때문에 다른 버전 상태 업데이트 호출은 건너뛰게 됩니다.

```
aws codeartifact update-package-versions-status --domain my_domain
  --domain-owner 111122223333 --repository my_repo
  --format npm --package chalk
  --version-revisions "4.1.0=25/UjBleHs1DZewk+zozoeqH/
R80Rc9gL1P8vbzVMJ=,4.0.0=E3lhBp0RObRTut4pkjV5c1AQGkgSA7Oxtil6hMMzelc="
  --versions 4.1.0 4.0.0 --target-status Archived
```

샘플 출력:

```
{
    "successfulVersions": {},
    "failedVersions": {
        "4.0.0": {
            "errorCode": "SKIPPED",
            "errorMessage": "version 4.0.0 is skipped"
        },
        "4.1.0": {
            "errorCode": "MISMATCHED_REVISION",
            "errorMessage": "current revision: 25/UjBleHs1DZewk+zozoeqH/
R80Rc9gL1P8vbzVMJ4=, expected revision: 25/UjBleHs1DZewk+zozoeqH/R80Rc9gL1P8vbzVMJ="
        }
    }
}
```

패키지 버전 폐기

Disposed 패키지 상태는 Archived에 대한 동작과 비슷하지만, CodeArtifact에서 패키지 자산을 영구적으로 삭제하기 때문에 도메인 소유자의 계정에 자산 스토리에 대한 요금이 더 이상 청구되지 않

는다는 점이 다릅니다. 각 패키지 버전 상태에 대한 자세한 내용은 패키지 버전 상태 섹션을 참조하십시오. 패키지 버전의 상태를 Disposed로 변경하려면 dispose-package-versions 명령을 사용합니다. 이 기능은 update-package-versions-status와는 별개인데, 패키지 버전 폐기는 되돌릴 수 없기 때문입니다. 패키지 자산은 삭제되기 때문에 버전 상태를 Archived, Unlisted 또는 Published로 되돌릴 수 없습니다. 폐기된 패키지 버전에 대해 취할 수 있는 유일한 조치는 delete-package-versions 명령을 사용하여 삭제하는 것입니다.

dispose-package-versions를 성공적으로 호출하려면, 호출하는 IAM 보안 주체에 패키지 리소스에 대한 codeartifact:DisposePackageVersions 권한이 있어야 합니다.

dispose-package-versions 명령의 동작은 update-package-versions-status와 비슷하며, 버전 개정 및 예상 상태 섹션에서 설명하는 --version-revisions 및 --expected-status 옵션의 동작도 마찬가지입니다. 예를 들어 다음 명령은 패키지 버전 폐기를 시도하지만 예상 상태가 일치하지 않아 실패하게 됩니다.

```
aws codeartifact dispose-package-versions –domain my_domain --domain-
owner 111122223333
--repository my_repo --format npm --package chalk --versions 4.0.0
--expected-status Unlisted
```

샘플 출력:

```
{
    "successfulVersions": {},
    "failedVersions": {
        "4.0.0": {
            "errorCode": "MISMATCHED_STATUS",
            "errorMessage": "current status: Published, expected status: Unlisted"
        }
    }
}
```

--expected-status를 Published로 하여 동일한 명령을 다시 실행하면 폐기에 성공하게 됩니다.

```
aws codeartifact dispose-package-versions –domain my_domain --domain-
owner 111122223333
--repository my_repo --format npm --package chalk --versions 4.0.0
--expected-status Published
```

샘플 출력:

```
{
    "successfulVersions": {
        "4.0.0": {
            "revision": "E3lhBp0RObRTut4pkjV5c1AQGkgSA7Oxtil6hMMzelc=",
            "status": "Disposed"
        }
    },
    "failedVersions": {}
}
```

패키지 원본 제어 편집

In AWS CodeArtifact에서는 패키지 버전을 리포지토리에 직접 게시하거나, 업스트림 리포지토리에서 가져오거나, 외부 퍼블릭 리포지토리에서 수집하여 패키지 버전을 리포지토리에 추가할 수 있습니다. 직접 게시와 퍼블릭 리포지토리에서 수집하는 방법 모두를 통해 패키지의 패키지 버전을 추가할 수 있기 때문에, 종속성 대체 공격에 취약해집니다. 자세한 내용은 종속성 대체 공격 단원을 참조하십시오. 종속성 대체 공격을 방어하는 대표적인 방법은 리포지토리의 패키지에서 패키지 원본 제어를 구성하여 패키지 버전을 리포지토리에 추가하는 방법을 제한하는 것입니다.

직접 게시 같은 내부 소스와 퍼블릭 리포지토리 같은 외부 소스 모두에서 서로 다른 패키지의 새 버전을 가져오고 싶은 팀은 패키지 원본 제어 구성을 고려해야 합니다. 기본적으로 패키지 원본 제어는 패키지의 첫 번째 버전이 리포지토리에 추가되는 방식을 기반으로 구성됩니다. 패키지 원본 컨트롤 설정 및 기본값에 대한 자세한 내용은 패키지 원본 제어 설정 섹션을 참조하세요.

`put-package-origin-configuration` API 작업을 사용한 후 패키지 기록을 제거하려면 `delete-package`를 사용하세요(패키지 또는 패키지 버전 삭제 참조).

일반적인 패키지 액세스 제어 시나리오

이 섹션에는 CodeArtifact 리포지토리에 패키지 버전이 추가되는 일반적인 시나리오가 포함되어 있습니다. 새 패키지에 대한 패키지 원본 제어 설정은 첫 번째 패키지 버전이 추가되는 방식에 따라 설정됩니다.

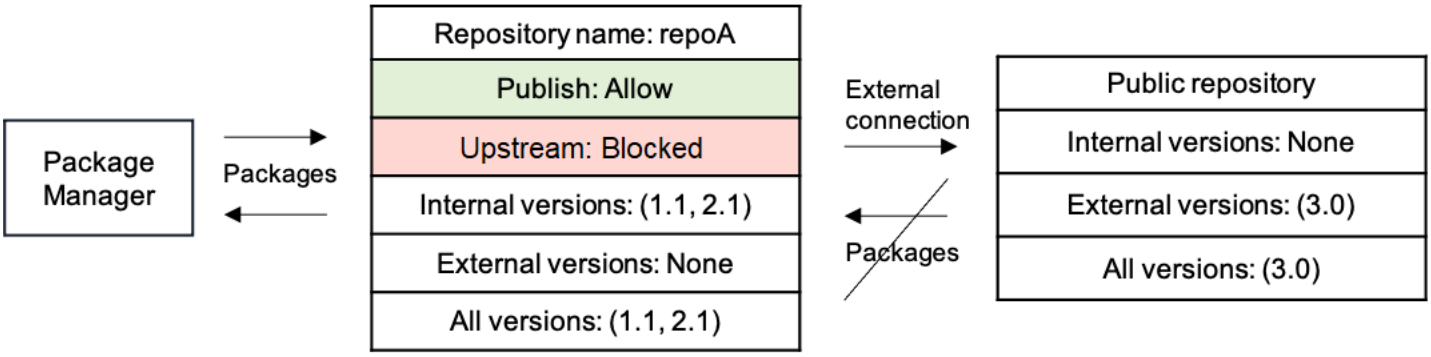
다음 시나리오에서 내부 패키지는 패키지 관리자가 리포지토리에 직접 게시하는 패키지(예: 사용자 또는 사용자 팀이 작성하고 유지 관리하는 패키지)입니다. 외부 패키지는 외부 연결을 통해 리포지토리에 수집될 수 있는 퍼블릭 리포지토리에 있는 패키지입니다.

외부 패키지 버전은 기존 내부 패키지를 대상으로 게시됩니다.

이 시나리오에서는 내부 패키지인 packageA를 가정합니다. 팀은 packageA의 첫 번째 패키지 버전을 CodeArtifact 리포지토리에 게시합니다. 이 패키지의 첫 번째 패키지 버전이므로, 패키지 원본 제어 설정은 게시: 허용 및 업스트림: 차단으로 자동 설정됩니다. 패키지가 리포지토리에 존재하면, CodeArtifact 리포지토리에 연결된 퍼블릭 리포지토리에 같은 이름의 패키지가 게시됩니다. 이것은 내부 패키지에 대한 종속성 대체 공격 시도일 수도 있고 단순한 우연의 일치일 수도 있습니다. 하지만 패키지 원본 제어는 새로운 외부 버전의 수집을 차단하여 잠재적 공격을 방어하도록 구성되어 있습니다.

다음 이미지에서 repoA는 퍼블릭 리포지토리에 대한 외부 연결이 있는 CodeArtifact 리포지토리입니다. 리포지토리에 packageA 버전 1.1 및 2.1이 있지만 버전 3.0은 퍼블릭 리포지토리에 게시됩니다. 일반적으로 repoA는 패키지 관리자가 패키지를 요청한 후 버전 3.0을 수집합니다. 패키지 수집이 차단으

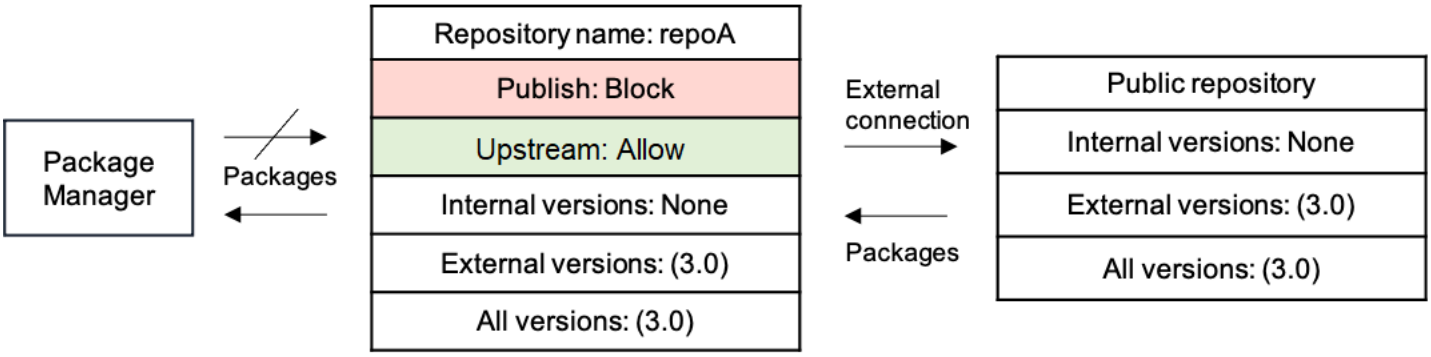
로 설정되어 있기 때문에 버전 3.0은 CodeArtifact 리포지토리에 수집되지 않으며 이 리포지토리에 연결된 패키지 관리자가 사용할 수 없습니다.



내부 패키지 버전은 기존 외부 패키지를 대상으로 게시됩니다.

이 시나리오에서는 패키지인 packageB가 리포지토리에 연결된 퍼블릭 리포지토리의 외부에 존재합니다. 리포지토리에 연결된 패키지 관리자가 packageB를 요청하면 해당 패키지 버전이 퍼블릭 리포지토리에서 사용자의 리포지토리로 수집됩니다. 이것은 리포지토리에 추가된 packageB의 첫 번째 패키지 버전이므로, 패키지 원본 설정은 게시: 차단 및 업스트림: 허용으로 구성됩니다. 나중에 패키지 이름이 동일한 버전을 리포지토리에 게시합니다. 퍼블릭 패키지를 모르는 상태에서 관련 없는 패키지를 같은 이름으로 게시하려고 하거나, 패치가 적용된 버전을 게시하려고 하거나, 이미 외부에 있는 패키지 버전을 직접 게시하게 됩니다. CodeArtifact는 사용자가 게시하려는 버전을 거부하지만, 사용자가 거부를 명시적으로 무시하고 필요한 경우에만 버전을 게시할 수 있습니다.

다음 이미지에서 repoA는 퍼블릭 리포지토리에 대한 외부 연결이 있는 CodeArtifact 리포지토리입니다. 리포지토리에는 퍼블릭 리포지토리에서 수집된 버전 3.0이 포함되어 있습니다. 버전 1.1을 리포지토리에 게시하려 합니다. 일반적으로 버전 1.2를 repoA에 게시할 수 있지만 게시가 차단으로 설정되어 있기 때문에 버전 1.2는 게시할 수 없습니다.



기존 외부 패키지의 패치가 적용된 패키지 버전 게시

이 시나리오에서는 패키지인 packageB가 리포지토리에 연결된 퍼블릭 리포지토리의 외부에 존재합니다. 리포지토리에 연결된 패키지 관리자가 packageB를 요청하면 해당 패키지 버전이 퍼블릭 리포지토

리에서 사용자의 리포지토리로 수집됩니다. 이것은 사용자의 리포지토리에 추가된 packageB의 첫 번째 패키지 버전이므로, 패키지 원본 설정은 게시: 차단 및 업스트림: 허용으로 구성됩니다. 팀에서 이 패키지의 패치가 적용된 패키지 버전을 리포지토리에 게시하기로 합니다. 패키지 버전을 직접 게시하기 위해 팀은 패키지 원본 제어 설정을 게시: 허용 및 업스트림: 차단으로 변경합니다. 이제 이 패키지의 버전을 리포지토리에 직접 게시하고 퍼블릭 리포지토리에서 수집할 수 있습니다. 팀에서 패치된 패키지 버전을 게시하면 패키지 원본 설정을 게시: 차단 및 업스트림: 허용으로 되돌립니다.

패키지 원본 제어 설정

패키지 원본 제어를 사용하여 패키지 버전을 리포지토리에 추가하는 방법을 구성할 수 있습니다. 다음 목록에는 사용 가능한 패키지 원본 제어 설정 및 값이 포함되어 있습니다.

> **Note**
>
> 패키지 그룹에서 원본 제어를 구성할 때는 사용 가능한 설정과 값이 다릅니다. 자세한 내용은 패키지 그룹 원본 제어 단원을 참조하십시오.

게시

이 설정은 패키지 관리자 또는 이와 유사한 도구를 사용하여 패키지 버전을 리포지토리에 직접 게시할 수 있는지를 구성합니다.

- 허용: 패키지 버전을 직접 게시할 수 있습니다.
- 차단: 패키지 버전을 직접 게시할 수 없습니다.

업스트림

이 설정은 패키지 관리자가 요청하는 경우 패키지 버전을 외부 또는 퍼블릭 리포지토리에서 수집하거나 업스트림 리포지토리에서 유지할 수 있는지를 구성합니다.

- 허용: 모든 패키지 버전은 업스트림 리포지토리로 구성된 다른 CodeArtifact 리포지토리에서 유지하거나 외부 연결을 통해 공개 소스에서 수집할 수 있습니다.
- 차단: 패키지 버전은 업스트림 리포지토리로 구성된 다른 CodeArtifact 리포지토리에서 유지하거나 외부 연결을 통해 공개 소스에서 수집할 수 없습니다.

기본 패키지 원본 제어 설정

기본 패키지 원본 제어 설정은 패키지의 연결된 패키지 그룹 원본 제어 설정을 기반으로 구성됩니다. 패키지 그룹 및 패키지 그룹 원본 제어에 대한 자세한 내용은 CodeArtifact에서의 패키지 그룹 작업 및 패키지 그룹 원본 제어 섹션을 참조하세요.

패키지가 모든 제한 유형에 대해 제한 설정이 ALLOW인 패키지 그룹과 연결된 경우 패키지에 대한 기본 패키지 원본 제어는 해당 패키지의 첫 번째 버전이 리포지토리에 추가되는 방법을 기반으로 합니다.

- 패키지 관리자가 첫 번째 패키지 버전을 직접 게시하는 경우 설정은 게시: 허용 및 업스트림: 차단이 됩니다.
- 첫 번째 패키지 버전을 공개 소스에서 수집하는 경우 설정은 게시: 차단 및 업스트림: 허용이 됩니다.

> **Note**
>
> 2022년 5월 이전에 CodeArtifact 리포지토리에 존재했던 패키지는 기본 패키지 원본 제어가 게시: 허용 및 업스트림: 허용으로 설정됩니다. 이러한 패키지에서는 패키지 원본 제어를 수동으로 설정해야 합니다. 현재 기본값은 2022년 5월 이후로 새 패키지에 설정되었고, 2022년 7월 14일 기능이 출시되면서 실행되기 시작했습니다. 패키지 원본 제어 설정에 대한 자세한 내용은 패키지 원본 제어 편집 섹션을 참조하세요.

그렇지 않으면 패키지가 BLOCK 또는 ALLOW_SPECIFIC_REPOSITORIES의 제한 설정이 하나 이상 있는 패키지 그룹과 연결된 경우 해당 패키지의 기본 원본 제어 설정이 게시: 허용 및 업스트림: 허용으로 설정됩니다.

패키지 원본 제어가 패키지 그룹 원본 제어와 상호 작용하는 방법

패키지에 원본 제어 설정이 있고 연결된 패키지 그룹에도 원본 제어 설정이 있으므로 이러한 서로 다른 두 설정이 서로 어떻게 상호 작용하는지 이해하는 것이 중요합니다.

두 설정 간의 상호 작용은 BLOCK의 설정이 항상 ALLOW의 설정보다 우선한다는 것입니다. 다음 표에는 몇 가지 예제 구성과 해당 유효 원본 제어 설정이 나와 있습니다.

패키지 원본 제어 설정	패키지 그룹 원본 제어 설정	유효 원본 제어 설정
게시: 허용	게시: 허용	게시: 허용

패키지 원본 제어 설정	패키지 그룹 원본 제어 설정	유효 원본 제어 설정
업스트림: 허용	업스트림: 허용	업스트림: 허용
게시: 차단 업스트림: 허용	게시: 허용 업스트림: 허용	게시: 차단 업스트림: 허용
게시: 허용	게시: 허용	게시: 허용
업스트림: 허용	업스트림: 차단	업스트림: 차단

즉, 원본 설정이 게시: 허용 및 업스트림: 허용인 패키지는 사실상 연결된 패키지 그룹의 원본 제어 설정을 따르고 있음을 의미합니다.

패키지 원본 제어 편집

패키지 원본 제어는 패키지 첫 번째 패키지 버전이 리포지토리에 추가되는 방식에 따라 자동으로 구성됩니다. 자세한 내용은 기본 패키지 원본 제어 설정 섹션을 참조하세요. CodeArtifact 리포지토리에서 패키지의 패키지 원본 제어를 추가하거나 편집하려면 다음 절차의 단계를 수행하세요.

패키지 원본 제어 추가 또는 편집(콘솔)

1. https://console.aws.amazon.com/codesuite/codeartifact/home AWS CodeArtifact 콘솔을 엽니다.
2. 탐색 창에서 리포지토리를 선택한 다음 수정할 패키지가 포함된 리포지토리를 선택합니다.
3. 패키지 테이블에서 편집할 패키지를 검색하여 선택합니다.
4. 패키지 요약 페이지의 원본 제어에서 편집을 선택합니다.
5. 원본 제어 편집에서 이 패키지에 설정할 패키지 원본 제어를 선택합니다. 패키지 원본 제어 설정인 게시와 업스트림을 동시에 설정해야 합니다.
   - 패키지 버전을 직접 게시할 수 있도록 허용하려면 게시에서 허용을 선택합니다. 패키지 버전 게시를 차단하려면 차단을 선택합니다.
   - 외부 리포지토리에서 패키지를 수집하고 업스트림 리포지토리에서 패키지를 가져오도록 허용하려면 업스트림 소스에서 허용을 선택합니다. 외부 및 업스트림 리포지토리에서의 패키지 버전 수집과 가져오기를 모두 차단하려면 차단을 선택합니다.

패키지 원본 제어 추가 또는 편집(AWS CLI)

1. 그렇지 않은 경우의 단계에 AWS CLI 따라를 구성합니다with AWS CodeArtifact 설정.
2. `put-package-origin-configuration` 명령을 사용하여 패키지 원본 제어를 추가하거나 편집합니다. 다음 필드를 교체합니다.

   - *my_domain*을 업데이트할 패키지가 들어 있는 CodeArtifact 도메인으로 바꿉니다.
   - *my_repo*를 업데이트할 패키지가 들어 있는 CodeArtifact 리포지토리로 바꿉니다.
   - *npm*을 업데이트할 패키지의 패키지 형식으로 바꿉니다.
   - *my_package*를 업데이트할 패키지의 이름으로 바꿉니다.
   - *##* 및 *##*을 원하는 패키지 원본 제어 설정으로 바꿉니다.

```
aws codeartifact put-package-origin-configuration --domain my_domain \
--repository my_repo --format npm --package my_package \
--restrictions publish=ALLOW,upstream=BLOCK
```

리포지토리 게시 및 업스트림

CodeArtifact에서는 연결 가능한 업스트림 리포지토리 또는 퍼블릭 리포지토리에 있는 패키지 버전을 게시할 수 없습니다. 예를 들어 Maven 패키지 `com.mycompany.mypackage:1.0`을 리포지토리 `myrepo`에 게시하려고 하는데 `myrepo`에 Maven Central과의 외부 연결이 있는 업스트림 리포지토리가 있다고 가정해 보겠습니다. 다음 시나리오를 고려해 보세요.

1. `com.mycompany.mypackage`에서의 패키지 원본 제어 설정은 게시: 허용과 업스트림: 허용입니다. `com.mycompany.mypackage:1.0`이 업스트림 리포지토리 또는 Maven Central에 있다면, CodeArtifact는 `myrepo`에서 이에 게시하려는 모든 시도를 거부하며 409 충돌 오류가 발생합니다. `com.mycompany.mypackage:1.1` 같은 다른 버전은 게시할 수 있습니다.
2. `com.mycompany.mypackage`에서의 패키지 원본 제어 설정은 게시: 허용과 업스트림: 차단입니다. `com.mycompany.mypackage`의 모든 버전은 아직 존재하지 않는 리포지토리에 게시할 수 있습니다. 패키지 버전에 연결할 수 없기 때문입니다.
3. `com.mycompany.mypackage`에서의 패키지 원본 제어 설정은 게시: 차단과 업스트림: 하용입니다. 패키지 버전은 리포지토리에 직접 게시할 수 없습니다.

CodeArtifact에서의 패키지 그룹 작업

패키지 그룹은 패키지 형식, 패키지 네임스페이스 및 패키지 이름을 사용하여 정의된 패턴과 일치하는 여러 패키지에 구성을 적용하는 데 사용할 수 있습니다. 패키지 그룹을 사용하여 여러 패키지에 대한 패키지 원본 제어를 보다 편리하게 구성할 수 있습니다. 패키지 원본 제어는 새 패키지 버전의 수집 또는 게시를 차단하거나 허용하는 데 사용되며, 이를 통해 의존성 대체 공격이라는 악의적인 행위로부터 사용자를 보호합니다.

CodeArtifact의 모든 도메인에는 루트 패키지 그룹이 자동으로 포함됩니다. 이 루트 패키지 그룹 `/*`에는 모든 패키지가 포함되며 기본적으로 패키지 버전은 모든 원본 유형의 도메인에 리포지토리를 입력할 수 있습니다. 루트 패키지 그룹은 수정할 수 있지만 삭제할 수는 없습니다.

패키지 그룹 구성 기능은 새 패키지 그룹을 생성하거나 기존 패키지 그룹을 삭제할 때 최종적으로 일관된 방식으로 작동합니다. 즉, 패키지 그룹을 생성하거나 삭제할 때 원본 제어가 예상 연결 패키지에 적용되지만 일관된 최종 동작으로 인해 다소 지연됩니다. 최종 일관성에 도달하는 시간은 도메인의 패키지 그룹 수와 도메인의 패키지 수에 따라 달라집니다. 패키지 그룹 생성 또는 삭제 후 연결된 패키지에 원본 제어가 즉시 반영되지 않는 짧은 기간이 있을 수 있습니다.

또한 패키지 그룹 원본 제어에 대한 업데이트는 거의 즉시 적용됩니다. 패키지 그룹의 생성 또는 삭제와 달리 기존 패키지 그룹의 원본 제어에 대한 변경 사항은 동일한 지연 없이 연결된 패키지에 반영됩니다.

이 주제에는 AWS CodeArtifact의 패키지 그룹에 대한 정보가 포함되어 있습니다.

주제

패키지 그룹 생성

CodeArtifact 콘솔, AWS Command Line Interface(AWS CLI) 또는 CloudFormation을 사용하여 패키지 그룹을 만들 수 있습니다. CloudFormation으로 CodeArtifact 패키지 그룹을 관리하는 방법에 대한 자세한 내용은 AWS CloudFormation을 사용하여 CodeArtifact 리소스 생성 섹션을 참조하세요.

패키지 그룹 생성(콘솔)

1. https://console.aws.amazon.com/codesuite/codeartifact/home에서 AWS CodeArtifact 콘솔을 엽니다.
2. 탐색 창에서 도메인을 선택한 다음, 패키지 그룹을 생성할 도메인을 선택합니다.
3. 패키지 그룹을 선택하고 패키지 그룹 생성을 선택합니다.
4. 패키지 그룹 정의에서 패키지 그룹에 대한 패키지 그룹 정의를 입력합니다. 패키지 그룹 정의에 따라 해당 그룹과 연관된 패키지가 결정됩니다. 패키지 그룹 정의를 텍스트로 직접 입력할 수 있으며, 시각적 모드를 사용하여 선택하면 패키지 그룹 정의가 자동으로 생성됩니다.
5. 시각적 모드를 사용하여 패키지 그룹 정의를 생성하려면 다음을 수행합니다.

   a. 시각적을 선택하여 시각적 모드로 전환합니다.

   b. 패키지 형식에서 이 그룹과 연결할 패키지의 형식을 선택합니다.

   c. 네임스페이스(범위)에서 일치시킬 네임스페이스 기준을 선택합니다.

      - 같음: 지정된 네임스페이스와 정확히 일치합니다. 선택한 경우 일치시킬 네임스페이스를 입력합니다.
      - 비어 있음: 네임스페이스가 없는 패키지를 일치시킵니다.
      - 단어로 시작: 지정된 단어로 시작하는 네임스페이스를 일치시킵니다. 선택한 경우 일치시킬 접두사 단어를 입력합니다. 단어 및 단어 경계에 대한 자세한 내용은 단어, 단어 경계 및 접두사 일치 섹션을 참조하세요.
      - 모두: 모든 네임스페이스의 패키지를 일치시킵니다.

   d. 같음, 비어 있음 또는 단어로 시작을 선택한 경우 패키지 이름에서 일치시킬 패키지 이름 기준을 선택합니다.

      - 정확히 같음: 지정된 패키지 이름과 정확하게 일치합니다. 선택한 경우 일치시킬 패키지 이름을 입력합니다.
      - 접두사로 시작: 지정된 접두사로 시작하는 패키지를 일치시킵니다.

- 단어로 시작: 지정된 단어로 시작하는 패키지를 일치시킵니다. 선택한 경우 일치시킬 접두사 단어를 입력합니다. 단어 및 단어 경계에 대한 자세한 내용은 단어, 단어 경계 및 접두사 일치 섹션을 참조하세요.
- 모두: 모든 패키지를 일치시킵니다.

e. 다음을 선택하여 정의를 검토합니다.

6. 패키지 그룹 정의를 텍스트로 입력하려면 다음을 수행합니다.

   a. 텍스트를 선택하여 텍스트 모드로 전환합니다.

   b. 패키지 그룹 정의에서 패키지 그룹 정의를 입력합니다. 패키지 그룹 정의 구문에 대한 자세한 내용은 패키지 그룹 정의 구문 및 매칭 동작 섹션을 참조하세요.

   c. 다음을 선택하여 정의를 검토합니다.

7. 정의 검토에서 이전에 제공된 정의를 기반으로 새 패키지 그룹에 포함될 패키지를 검토합니다. 검토한 후 다음을 선택합니다.

8. 패키지 그룹 정보에서 선택적으로 패키지 그룹에 대한 설명과 연락처 이메일을 추가합니다. 다음을 선택합니다.

9. 패키지 원본 제어에서 그룹의 패키지에 적용할 원본 제어를 구성합니다. 패키지 그룹 원본 제어에 대한 자세한 내용은 패키지 그룹 원본 제어 섹션을 참조하세요.

10. 패키지 그룹 생성을 선택합니다.

패키지 그룹 생성(AWS CLI)

`create-package-group` 명령을 사용하여 도메인에 패키지 그룹을 생성합니다. `--package-group` 옵션에 그룹과 연결되는 패키지를 결정하는 패키지 그룹 정의를 입력합니다. 패키지 그룹 정의 구문에 대한 자세한 내용은 패키지 그룹 정의 구문 및 매칭 동작 섹션을 참조하세요.

아직 구성하지 않았다면 with AWS CodeArtifact 설정에 나오는 단계에 따라 AWS CLI를 구성하세요.

```
aws codeartifact create-package-group \
        --domain my_domain \
        --package-group '/nuget/*' \
        --domain-owner 111122223333 \
        --contact-info contact@email.com \
        --description "a new package group" \
        --tags key=key1,value=value1
```

패키지 그룹 보기 또는 편집

CodeArtifact 콘솔 또는 AWS Command Line Interface(AWS CLI)를 사용하여 모든 패키지 그룹 목록을 보거나, 특정 패키지 그룹의 세부 정보를 보거나, 패키지 그룹의 세부 정보 또는 구성을 편집할 수 있습니다.

패키지 그룹 보기 또는 편집(콘솔)

1. https://console.aws.amazon.com/codesuite/codeartifact/home에서 AWS CodeArtifact 콘솔을 엽니다.
2. 탐색 창에서 도메인을 선택한 다음, 보거나 편집할 패키지 그룹이 포함된 도메인을 선택합니다.
3. 패키지 그룹을 선택하고 보거나 편집할 패키지 그룹을 선택합니다.
4. 세부 정보에서 상위 그룹, 설명, ARN, 연락처 이메일 및 패키지 원본 제어를 포함한 패키지 그룹에 대한 정보를 확인합니다.
5. 하위 그룹에서 이 그룹을 상위 그룹으로 포함하는 패키지 그룹 목록을 확인합니다. 이 목록의 패키지 그룹은 이 패키지 그룹의 설정을 상속할 수 있습니다. 자세한 내용은 패키지 그룹 계층 구조 및 패턴 특이도 섹션을 참조하세요.
6. 패키지에서 패키지 그룹 정의를 기반으로 이 패키지 그룹에 속하는 패키지를 확인합니다. 강도 열에서 패키지 연결의 강도를 확인할 수 있습니다. 자세한 내용은 패키지 그룹 계층 구조 및 패턴 특이도 섹션을 참조하세요.
7. 패키지 그룹 정보를 편집하려면 패키지 그룹 편집을 선택합니다.

    a. 정보에서 패키지 그룹의 설명 또는 연락처 정보를 업데이트합니다. 패키지 그룹의 정의를 편집할 수는 없습니다.

    b. 패키지 그룹 원본 제어에서 패키지 그룹의 원본 제어 설정을 업데이트하여 관련 패키지가 도메인의 리포지토리에 들어갈 수 있는 방법을 결정합니다. 자세한 내용은 패키지 그룹 원본 제어 섹션을 참조하세요.

패키지 그룹 보기 또는 편집(AWS CLI)

다음 명령을 사용하여 AWS CLI로 패키지 그룹을 보거나 편집합니다. 아직 구성하지 않았다면 with AWS CodeArtifact 설정에 나오는 단계에 따라 AWS CLI를 구성하세요.

도메인의 모든 패키지 그룹을 보려면 `list-package-groups` 명령을 사용합니다.

```
aws codeartifact list-package-groups \
        --domain my_domain \
```

```
        --domain-owner 111122223333
```

패키지 그룹에 대한 세부 정보를 보려면 `describe-package-group` 명령을 사용합니다. 패키지 그룹 정의에 대한 자세한 내용은 패키지 그룹 정의 구문 및 예제 섹션을 참조하세요.

```
aws codeartifact describe-package-group \
        --domain my_domain \
        --domain-owner 111122223333 \
        --package-group '/nuget/*'
```

패키지 그룹의 하위 패키지 그룹을 보려면 `list-sub-package-groups` 명령을 사용합니다.

```
aws codeartifact list-sub-package-groups \
        --domain my_domain \
        --domain-owner 111122223333 \
        --package-group '/nuget/*' \
```

패키지에 연결된 패키지 그룹을 보려면 `get-associated-package-group` 명령을 사용합니다. NuGet, Python 및 Swift 패키지 형식에는 정규화된 패키지 이름과 네임스페이스를 사용해야 합니다. 패키지 이름과 네임스페이스를 정규화하는 방법에 대한 자세한 내용은 NuGet, Python 및 Swift 이름 정규화 설명서를 참조하세요.

```
aws codeartifact get-associated-package-group \
        --domain my_domain \
        --domain-owner 111122223333 \
        --format npm \
        --package packageName \
        --namespace scope
```

패키지 그룹을 편집하려면 `update-package-group` 명령을 사용합니다. 이 명령은 패키지 그룹의 연락처 정보 또는 설명을 업데이트하는 데 사용됩니다. 패키지 그룹 원본 제어 설정 및 추가 또는 편집에 대한 자세한 내용은 패키지 그룹 원본 제어 섹션을 참조하세요. 패키지 그룹 정의에 대한 자세한 내용은 패키지 그룹 정의 구문 및 예제 섹션을 참조하세요.

```
aws codeartifact update-package-group \
        --domain my_domain \
        --package-group '/nuget/*' \
        --domain-owner 111122223333 \
        --contact-info contact@email.com \
        --description "updated package group description"
```

패키지 그룹 삭제

CodeArtifact 콘솔 또는 AWS Command Line Interface(AWS CLI)를 사용하여 패키지 그룹을 삭제할 수 있습니다.

패키지 그룹을 삭제할 때는 다음 동작에 유의하세요.

- 루트 패키지 그룹 /*는 삭제할 수 없습니다.
- 해당 패키지 그룹과 연결된 패키지 및 패키지 버전은 삭제되지 않습니다.
- 패키지 그룹이 삭제되면 직접 하위 패키지 그룹은 패키지 그룹의 직접 상위 패키지 그룹의 하위 패키지 그룹이 됩니다. 따라서 하위 그룹 중 하나가 상위에서 설정을 상속 받는 경우 해당 설정이 변경될 수 있습니다.

패키지 그룹 삭제(콘솔)

1. https://console.aws.amazon.com/codesuite/codeartifact/home에서 AWS CodeArtifact 콘솔을 엽니다.
2. 탐색 창에서 도메인을 선택한 다음, 보거나 편집할 패키지 그룹이 포함된 도메인을 선택합니다.
3. 패키지 그룹을 선택합니다.
4. 삭제하려는 패키지 그룹을 선택하고 삭제를 선택합니다.
5. 필드에 delete를 입력하고 삭제를 선택합니다.

패키지 그룹 삭제(AWS CLI)

패키지 그룹을 삭제하려면 `delete-package-group` 명령을 사용합니다.

```
aws codeartifact delete-package-group \
         --domain my_domain \
         --domain-owner 111122223333 \
         --package-group '/nuget/*'
```

패키지 그룹 원본 제어

패키지 원본 제어는 패키지 버전이 도메인에 들어갈 수 있는 방법을 구성하는 데 사용됩니다. 패키지 그룹에 원본 제어를 설정하여 패키지 그룹과 연결된 모든 패키지의 버전이 도메인에 지정된 리포지토리에 들어갈 수 있는 방법을 구성할 수 있습니다.

패키지 그룹 원본 제어 설정은 다음으로 구성됩니다.

- 제한 설정: 이러한 설정은 게시, 내부 업스트림 또는 외부 퍼블릭 리포지토리에서 패키지가 CodeArtifact의 리포지토리에 들어갈 수 있는지 여부를 정의합니다.
- 허용된 리포지토리 목록: 각 제한 설정은 특정 리포지토리를 허용하도록 설정할 수 있습니다. 제한 설정이 특정 리포지토리를 허용하도록 설정된 경우 해당 제한에는 해당하는 허용 리포지토리 목록이 있습니다.

> **Note**
>
> 패키지 그룹의 원본 제어 설정은 개별 패키지의 원본 제어 설정과 약간 다릅니다. 패키지의 원본 제어 설정에 대한 자세한 내용은 패키지 원본 제어 설정 섹션을 참조하세요.

제한 설정

패키지 그룹의 원본 제어 설정에 대한 제한 설정은 해당 그룹과 연결된 패키지가 도메인의 리포지토리에 들어갈 수 있는 방법을 결정합니다.

PUBLISH

`PUBLISH` 설정은 패키지 관리자 또는 이와 유사한 도구를 사용하여 패키지 버전을 도메인의 리포지토리에 직접 게시할 수 있는지 구성합니다.

- ALLOW: 패키지 버전을 모든 리포지토리에 직접 게시할 수 있습니다.
- BLOCK: 패키지 버전을 리포지토리에 직접 게시할 수 없습니다.
- ALLOW_SPECIFIC_REPOSITORIES: 게시를 위해 허용된 리포지토리 목록에 지정된 리포지토리에만 패키지 버전을 직접 게시할 수 있습니다.
- INHERIT: `PUBLISH` 설정이 `INHERIT`가 아닌 설정을 가진 첫 번째 상위 패키지 그룹에서 상속됩니다.

EXTERNAL_UPSTREAM

`EXTERNAL_UPSTREAM` 설정은 패키지 관리자가 요청하는 경우 패키지 버전을 외부 퍼블릭 리포지토리에서 수집할 수 있는지 구성합니다. 지원되는 외부 리포지토리 목록은 지원하는 외부 연결 저장소 섹션을 참조하세요.

- ALLOW: 모든 패키지 버전을 외부 연결이 있는 퍼블릭 소스의 모든 리포지토리에 수집할 수 있습니다.
- BLOCK: 외부 연결이 있는 퍼블릭 소스의 리포지토리에 패키지 버전을 수집할 수 없습니다.
- ALLOW_SPECIFIC_REPOSITORIES: 퍼블릭 소스에서 외부 업스트림에 대해 허용된 리포지토리 목록에 지정된 리포지토리로만 패키지 버전을 수집할 수 있습니다.
- INHERIT: EXTERNAL_UPSTREAM 설정이 INHERIT가 아닌 설정을 가진 첫 번째 상위 패키지 그룹에서 상속됩니다.

INTERNAL_UPSTREAM

INTERNAL_UPSTREAM 설정은 패키지 관리자가 요청하는 경우 동일한 CodeArtifact 도메인의 내부 업스트림 리포지토리에서 패키지 버전을 유지할 수 있는지 여부를 구성합니다.

- ALLOW: 모든 패키지 버전은 업스트림 리포지토리로 구성된 다른 CodeArtifact 리포지토리에서 유지할 수 있습니다.
- BLOCK: 패키지 버전은 업스트림 리포지토리로 구성된 다른 CodeArtifact 리포지토리에서 유지할 수 없습니다.
- ALLOW_SPECIFIC_REPOSITORIES: 패키지 버전은 업스트림 리포지토리로 구성된 다른 CodeArtifact 리포지토리에서만 내부 업스트림에 대해 허용된 리포지토리 목록에 지정된 리포지토리로 유지할 수 있습니다.
- INHERIT: INTERNAL_UPSTREAM 설정이 INHERIT가 아닌 설정을 가진 첫 번째 상위 패키지 그룹에서 상속됩니다.

허용된 리포지토리 목록

제한 설정이 ALLOW_SPECIFIC_REPOSITORIES로 구성된 경우 패키지 그룹에는 해당 제한 설정에 허용되는 리포지토리 목록이 포함된 허용되는 리포지토리 목록이 함께 포함됩니다. 따라서 패키지 그룹에는 ALLOW_SPECIFIC_REPOSITORIES로 구성된 각 설정에 대해 하나씩 0~3개의 허용된 리포지토리 목록이 포함됩니다.

패키지 그룹의 허용된 리포지토리 목록에 리포지토리를 추가할 경우 리포지토리를 추가할 허용된 리포지토리 목록을 지정해야 합니다.

허용되는 리포지토리 목록은 다음과 같습니다.

- EXTERNAL_UPSTREAM: 추가된 리포지토리의 외부 리포지토리에서 패키지 버전 수집을 허용 또는 차단합니다.

- INTERNAL_UPSTREAM: 추가된 리포지토리의 다른 CodeArtifact 리포지토리에서 패키지 버전 가져오기를 허용 또는 차단합니다.
- PUBLISH: 패키지 관리자에서 추가된 리포지토리로의 패키지 버전 직접 게시를 허용 또는 차단합니다.

패키지 그룹 원본 제어 설정 편집

패키지 그룹에 대한 원본 제어를 추가하거나 편집하려면 다음 절차의 단계를 수행하세요. 패키지 그룹 원본 제어 설정에 대한 자세한 내용은 제한 설정 및 허용된 리포지토리 목록 섹션을 참조하세요.

패키지 그룹 원본 제어 추가 또는 편집(CLI)

1. 아직 구성하지 않았다면 with AWS CodeArtifact 설정에 나오는 단계에 따라 AWS CLI를 구성하세요.
2. update-package-group-origin-configuration 명령을 사용하여 패키지 원본 제어를 추가하거나 편집합니다.

   - --domain에 업데이트하려는 패키지 그룹이 포함된 CodeArtifact 도메인을 입력합니다.
   - --domain-owner에 도메인 소유자의 계정 번호를 입력합니다.
   - --package-group에 업데이트하려는 패키지 그룹을 입력합니다.
   - --restrictions에 원본 제어 제한을 나타내는 키-값 페어를 입력합니다.
   - --add-allowed-repositories에 제한 유형과 리포지토리 이름이 포함된 JSON 객체를 입력하여 제한에 대해 허용되는 해당 리포지토리 목록에 추가합니다.
   - --remove-allowed-repositories에 제한 유형과 리포지토리 이름이 포함된 JSON 객체를 입력하여 제한에 대해 허용되는 해당 리포지토리 목록에서 제거합니다.

```
aws codeartifact update-package-group-origin-configuration \
   --domain my_domain \
   --domain-owner 111122223333 \
   --package-group '/nuget/*' \
   --restrictions INTERNAL_UPSTREAM=ALLOW_SPECIFIC_REPOSITORIES \
   --add-allowed-repositories
 originRestrictionType=INTERNAL_UPSTREAM,repositoryName=my_repo \
   --remove-allowed-repositories
 originRestrictionType=INTERNAL_UPSTREAM,repositoryName=my_repo2
```

다음 예시에서는 하나의 명령에 여러 제한 및 여러 리포지토리를 추가합니다.

```
aws codeartifact update-package-group-origin-configuration \
   --domain my_domain \
   --domain-owner 111122223333 \
   --package-group '/nuget/*' \
   --
restrictions PUBLISH=BLOCK,EXTERNAL_UPSTREAM=ALLOW_SPECIFIC_REPOSITORIES,INTERNAL_UPSTREAM=
 \
   --add-allowed-repositories
 originRestrictionType=INTERNAL_UPSTREAM,repositoryName=my_repo
 originRestrictionType=INTERNAL_UPSTREAM,repositoryName=my_repo2 \
   --remove-allowed-repositories
 originRestrictionType=INTERNAL_UPSTREAM,repositoryName=my_repo2
```

패키지 그룹 원본 제어 구성 예제

다음 예제에서는 일반적인 패키지 관리 시나리오에 대한 패키지 원본 제어 구성을 보여줍니다.

프라이빗 이름이 있는 패키지를 게시할 수 있지만 수집할 수는 없음

이 시나리오는 패키지 관리의 일반적인 시나리오일 수 있습니다.

- 패키지 관리자가 도메인의 리포지토리에 프라이빗 이름을 가진 패키지를 게시하도록 허용하고 외부 퍼블릭 리포지토리에서 도메인의 리포지토리로 수집되지 않도록 차단합니다.
- 외부 퍼블릭 리포지토리에서 도메인의 리포지토리에 다른 모든 패키지를 수집하도록 허용하고 패키지 관리자에서 도메인의 리포지토리에 게시되는 것을 차단합니다.

이렇게 하려면 PUBLISH: ALLOW, EXTERNAL_UPSTREAM: BLOCK 및 INTERNAL_UPSTREAM: ALLOW의 프라이빗 이름 및 원본 설정을 포함하는 패턴으로 패키지 그룹을 구성해야 합니다. 이렇게 하면 프라이빗 이름을 가진 패키지를 직접 게시할 수 있지만 외부 리포지토리에서 수집할 수는 없습니다.

다음 AWS CLI 명령은 원하는 동작과 일치하는 원본 제한 설정을 사용하여 패키지 그룹을 생성하고 구성합니다.

패키지 그룹을 생성하려면

```
aws codeartifact create-package-group \
```

```
    --domain my_domain \
    --package-group /npm/space/anycompany~ \
    --domain-owner 111122223333 \
    --contact-info contact@email.com | URL \
    --description "my package group"
```

패키지 그룹의 원본 구성을 업데이트하려면

```
aws codeartifact update-package-group-origin-configuration \
    --domain my_domain \
    --domain-owner 111122223333 \
    --package-group '/npm/space/anycompany~' \
    --restrictions PUBLISH=ALLOW,EXTERNAL_UPSTREAM=BLOCK,INTERNAL_UPSTREAM=ALLOW
```

하나의 리포지토리를 통해 외부 리포지토리에서 수집 허용

이 시나리오에서 도메인에는 여러 개의 리포지토리가 있습니다. 이러한 리포지토리 중 repoA에는 다음과 같이 퍼블릭 리포지토리 npmjs.com에 대한 외부 연결을 가진 repoB에 대한 업스트림 연결이 있습니다.

```
repoA --> repoB --> npmjs.com
```

특정 패키지 그룹 /npm/space/anycompany~의 패키지를 npmjs.com에서 repoA로, repoB를 통해서만 수집할 수 있도록 허용하려고 합니다. 또한 패키지 그룹과 연결된 패키지를 도메인의 다른 리포지토리로 수집하는 것을 차단하고 패키지 관리자를 통해 패키지를 직접 게시하는 것을 차단하려고 합니다. 이를 위해 다음과 같이 패키지 그룹을 생성하고 구성합니다.

PUBLISH: BLOCK, EXTERNAL_UPSTREAM: ALLOW_SPECIFIC_REPOSITORIES 및 INTERNAL_UPSTREAM: ALLOW_SPECIFIC_REPOSITORIES의 원본 제한 설정.

repoA 및 repoB가 허용되는 적절한 리포지토리 목록에 추가되었습니다.

- repoA는 내부 업스트림인 repoB에서 패키지를 가져오므로 INTERNAL_UPSTREAM 목록에 추가되어야 합니다.
- repoB는 외부 리포지토리인 npmjs.com에서 패키지를 가져오므로 EXTERNAL_UPSTREAM 목록에 추가해야 합니다.

다음 AWS CLI 명령은 원하는 동작과 일치하는 원본 제한 설정을 사용하여 패키지 그룹을 생성하고 구성합니다.

패키지 그룹을 생성하려면

```
aws codeartifact create-package-group \
   --domain my_domain \
   --package-group /npm/space/anycompany~ \
   --domain-owner 111122223333 \
   --contact-info contact@email.com | URL \
   --description "my package group"
```

패키지 그룹의 원본 구성을 업데이트하려면

```
aws codeartifact update-package-group-origin-configuration \
   --domain my_domain \
   --domain-owner 111122223333 \
   --package-group /npm/space/anycompany~ \
   --
restrictions PUBLISH=BLOCK,EXTERNAL_UPSTREAM=ALLOW_SPECIFIC_REPOSITORIES,INTERNAL_UPSTREAM=ALLO
 \
   --add-allowed-repositories
 originRestrictionType=INTERNAL_UPSTREAM,repositoryName=repoA
 originRestrictionType=EXTERNAL_UPSTREAM,repositoryName=repoB
```

패키지 그룹 원본 제어 설정이 패키지 원본 제어 설정과 상호 작용하는 방법

패키지에 원본 제어 설정이 있고 연결된 패키지 그룹에도 원본 제어 설정이 있으므로 이러한 서로 다른 두 설정이 서로 어떻게 상호 작용하는지 이해하는 것이 중요합니다. 설정 간의 상호 작용에 대한 자세한 내용은 패키지 원본 제어가 패키지 그룹 원본 제어와 상호 작용하는 방법 섹션을 참조하세요.

패키지 그룹 정의 구문 및 매칭 동작

이 주제에는 패키지 그룹 정의, 패턴 일치 동작, 패키지 연결 강도 및 패키지 그룹 계층 구조에 대한 정보가 포함되어 있습니다.

목차

- 대소문자 구분
- 강력한 일치와 약한 일치
- 추가 변형

패키지 그룹 정의 구문 및 예제

패키지 그룹을 정의하기 위한 패턴 구문은 패키지 경로의 형식을 밀접하게 따릅니다. 패키지 경로는 시작 부분에 슬래시를 추가하고 각 구성 요소를 슬래시로 구분하여 패키지의 좌표 구성 요소(형식, 네임스페이스 및 이름)에서 생성됩니다. 예를 들어 네임스페이스 space에서 anycompany-ui-components라는 npm 패키지의 패키지 경로는 /npm/space/anycompany-ui-components입니다.

패키지 그룹 패턴은 패키지 경로와 동일한 구조를 따르며, 그룹 정의의 일부로 지정되지 않은 구성 요소는 생략되고 패턴은 접미사로 종료됩니다. 포함된 접미사는 다음과 같이 패턴의 일치 동작을 결정합니다.

- $ 접미사는 전체 패키지 좌표와 일치합니다.
- ~ 접미사는 접두사와 일치합니다.
- * 접미사는 이전에 정의된 구성 요소의 모든 값과 일치합니다.

다음은 허용되는 각 조합에 대한 예제 패턴입니다.

1. 모든 패키지 형식: `/*`
2. 특정 패키지 형식: `/npm/*`
3. 패키지 형식 및 네임스페이스 접두사: `/maven/com.anycompany~`
4. 패키지 형식 및 네임스페이스: `/npm/space/*`
5. 패키지 형식, 네임스페이스 및 이름 접두사: `/npm/space/anycompany-ui~`
6. 패키지 형식, 네임스페이스 및 이름: `/maven/org.apache.logging.log4j/log4j-core$`

위 예제와 같이 ~ 접미사는 네임스페이스 또는 이름 끝에 추가되어 접두사 일치를 나타내며, *는 경로의 다음 구성 요소(모든 형식, 모든 네임스페이스 또는 모든 이름)에 대한 모든 값과 일치시키는 데 사용되는 경우 슬래시 뒤에 옵니다.

패키지 그룹 정의 및 정규화

CodeArtifact는 NuGet, Python 및 Swift 패키지 이름을 정규화하고 저장하기 전에 Swift 패키지 네임스페이스를 정규화합니다. CodeArtifact는 패키지를 패키지 그룹 정의와 일치시킬 때 이러한 정규화된 이

름을 사용합니다. 따라서 이러한 형식의 네임스페이스 또는 이름이 포함된 패키지 그룹은 정규화된 네임스페이스와 이름을 사용해야 합니다. 패키지 이름과 네임스페이스를 정규화하는 방법에 대한 자세한 내용은 NuGet, Python 및 Swift 이름 정규화 설명서를 참조하세요.

패키지 그룹 정의의 네임스페이스

네임스페이스(Python 및 NuGet)가 없는 패키지 또는 패키지 형식의 경우 패키지 그룹에 네임스페이스가 포함되어서는 안 됩니다. 이러한 패키지 그룹에 대한 패키지 그룹 정의에는 빈 네임스페이스 섹션이 포함되어 있습니다. 예를 들어 requests라는 Python 패키지의 경로는 /python//requests입니다.

네임스페이스(Maven, 일반 및 Swift)가 있는 패키지 또는 패키지 형식의 경우 패키지 이름이 포함되면 네임스페이스를 포함해야 합니다. Swift 패키지 형식의 경우 정규화된 패키지 네임스페이스가 사용됩니다. Swift 패키지 네임스페이스를 정규화하는 방법에 대한 자세한 내용은 Swift 패키지 이름 및 네임스페이스 정규화 섹션을 참조하세요.

패키지 그룹 계층 구조 및 패턴 특이도

패키지 그룹에 "있거나" 패키지 그룹과 "연결된" 패키지는 그룹 패턴과 일치하지만 더 구체적인 그룹 패턴과는 일치하지 않는 패키지 경로가 있는 패키지입니다. 예를 들어 패키지 그룹 /npm/* 및 /npm/space/*를 고려할 때 패키지 경로 /npm//react는 첫 번째 그룹(/npm/*)과 연결되고 /npm/space/aui.components 및 /npm/space/amplify-ui-core는 두 번째 그룹(/npm/space/*)과 연결됩니다. 패키지가 여러 그룹과 일치할 수 있지만 각 패키지는 가장 구체적으로 일치하는 단일 그룹과만 연결되며 한 그룹의 구성만 패키지에 적용됩니다.

패키지 경로가 여러 패턴과 일치하면 "보다 구체적인" 패턴을 가장 긴 일치 패턴으로 생각할 수 있습니다. 또는 보다 구체적인 패턴은 덜 구체적인 패턴과 일치하는 패키지의 적절한 하위 집합과 일치하는 패턴입니다. 이전 예제에서 /npm/space/*와 일치하는 모든 패키지는 /npm/*와도 일치하지만 그 반대는 true가 아니며, /npm/*의 적절한 하위 집합이므로 /npm/space/*을 보다 구체적인 패턴으로 만듭니다. 한 그룹은 다른 그룹의 하위 집합이므로 /npm/space/*이 상위 그룹인 /npm/*의 하위 집합인 계층 구조를 생성합니다.

패키지에는 가장 구체적인 패키지 그룹의 구성만 적용되지만 상위 그룹의 구성에서 상속되도록 해당 그룹을 구성할 수 있습니다.

단어, 단어 경계 및 접두사 일치

접두사 일치에 대해 논의하기 전에 몇 가지 주요 용어를 정의해 보겠습니다.

- 단어는 뒤에 0개 이상의 문자, 숫자 또는 기호 문자(예: 악센트, 움라우트 등)가 붙는 문자 또는 숫자입니다.

- 단어 경계는 단어가 아닌 문자에 도달했을 때 단어 끝에 있습니다. 단어가 아닌 문자는 `.`, `-` 및 `_` 등과 같은 구두점 문자입니다.

특히 단어의 정규 표현식 패턴은 `[\p{L}\p{N}][\p{L}\p{N}\p{M}]*`이며 다음과 같이 분류할 수 있습니다.

- `\p{L}`은 모든 문자를 나타냅니다.
- `\p{N}`은 모든 숫자를 나타냅니다.
- `\p{M}`은 악센트, 움라우트 등과 같은 모든 기호 문자를 나타냅니다.

따라서 `[\p{L}\p{N}]`은 숫자 또는 문자를 나타내고 `[\p{L}\p{N}\p{M}]*`은 0개 이상의 문자, 숫자 또는 기호 문자를 나타내며, 단어 경계는 이 정규 표현식 패턴의 각 일치 끝에 있습니다.

> **Note**
>
> 단어 경계 일치는 이 "단어" 정의를 기반으로 합니다. 사전 또는 CameCase에 정의된 단어를 기반으로 하지 않습니다. 예를 들어 `oneword` 또는 `OneWord`에는 단어 경계가 없습니다.

이제 단어 및 단어 경계가 정의되었으므로 이를 사용하여 CodeArtifact에서 접두사 일치를 설명하겠습니다. 단어 경계에서 접두사 일치를 나타내기 위해 단어 뒤에 일치 문자(~)가 사용됩니다. 예를 들어 패턴 `/npm/space/foo~`는 패키지 경로 `/npm/space/foo` 및 `/npm/space/foo-bar`와 일치하지만 `/npm/space/food` 또는 `/npm/space/foot`와는 일치하지 않습니다.

패턴 `/npm/*`에서와 같이 단어가 아닌 문자 뒤에 나오는 경우 ~ 대신 와일드카드(*)를 사용해야 합니다.

대소문자 구분

패키지 그룹 정의는 대소문자를 구분하므로 대소문자만 다른 패턴이 별도의 패키지 그룹으로 존재할 수 있습니다. 예를 들어 사용자는 npm 퍼블릭 레지스트리에 있으며 대소문자만 다른 세 개의 개별 패키지인 AsyncStorage, asyncStorage, asyncstorage에 대해 `/npm//AsyncStorage$`, `/npm//asyncStorage$` 및 `/npm//asyncstorage$` 패턴을 사용하여 별도의 패키지 그룹을 생성할 수 있습니다.

대소문자가 중요하지만 패키지에 대소문자별로 다른 패턴의 변형이 있는 경우 CodeArtifact는 여전히 패키지를 패키지 그룹에 연결합니다. 사용자가 위에 표시된 다른 두 그룹을 생성하지 않고

/npm//AsyncStorage$ 패키지 그룹을 생성하면 asyncStorage 및 asyncstorage를 포함하여 AsyncStorage라는 이름의 모든 대소문자 변형이 패키지 그룹과 연결됩니다. 그러나 다음 섹션인 강력한 일치와 약한 일치에 설명된 대로 이러한 변형은 패턴과 정확히 일치하는 AsyncStorage와 다르게 처리됩니다.

강력한 일치와 약한 일치

이전 섹션 대소문자 구분의 정보는 패키지 그룹이 대소문자를 구분한다고 설명한 다음, 대소문자를 구분하지 않는다고 설명합니다. 이는 CodeArtifact의 패키지 그룹 정의에 강력한 일치(또는 정확한 일치)와 약한 일치(또는 변형 일치)라는 개념이 있기 때문입니다. 강력한 일치는 패키지가 변형 없이 정확하게 패턴과 일치할 때를 말합니다. 약한 일치는 패키지가 다른 대소문자와 같은 패턴 변형과 일치하는 경우를 말합니다. 약한 일치 동작은 패키지 그룹 패턴의 변형인 패키지가 보다 일반적인 패키지 그룹으로 롤업되는 것을 방지합니다. 패키지가 가장 구체적인 일치 그룹 패턴의 변형(약한 일치)인 경우, 패키지는 그룹과 연결되지만 그룹의 원본 제어 구성을 적용하는 대신 패키지가 차단되어 패키지의 새 버전을 업스트림에서 가져오거나 게시되는 것을 방지합니다. 이 동작은 이름이 거의 동일한 패키지의 종속성 혼동으로 인한 공급망 공격 위험을 줄입니다.

약한 일치 동작을 설명하기 위해 패키지 그룹 /npm/*이 수집을 허용하고 게시를 차단한다고 가정합니다. 보다 구체적인 패키지 그룹인 /npm//anycompany-spicy-client$는 수집을 차단하고 게시를 허용하도록 구성됩니다. anycompany-spicy-client라는 패키지는 패키지 그룹과 강력하게 일치하므로 패키지 버전을 게시하고 패키지 버전의 수집을 차단할 수 있습니다. 패키지 이름에서 게시가 허용되는 유일한 대소문자 구분은 anycompany-spicy-client인데, 이는 패키지 정의 패턴과 강력하게 일치하기 때문입니다. AnyCompany-spicy-client와 같은 다른 대소문자 변형은 약한 일치이므로 게시가 차단됩니다. 더 중요한 것은 패키지 그룹이 패턴에 사용되는 소문자 이름뿐만 아니라 모든 대소문자 변형의 수집을 차단하여 종속성 혼동 공격의 위험을 줄이는 것입니다.

추가 변형

대소문자 차이 외에도 약한 일치는 대시(-), 점(.), 밑줄(_) 및 혼동할 수 있는 문자(예: 개별 알파벳과 유사한 모양의 문자) 시퀀스의 차이를 무시합니다. 약한 일치에 사용되는 정규화 중에 CodeArtifact는 케이스폴딩(소문자로 변환하는 것과 유사)을 수행하고 대시, 점 및 밑줄 문자 시퀀스를 단일 점으로 바꾸고 혼동할 수 있는 문자를 정규화합니다.

약한 일치는 대시, 점 및 밑줄을 동등한 것으로 취급하지만 완전히 무시하지는 않습니다. 즉, foo-bar, foo.bar, foo..bar 및 foo_bar는 모두 약한 일치 항목이지만 foobar는 그렇지 않습니다. 여러 퍼블릭 리포지토리는 이러한 유형의 변형을 방지하는 단계를 구현하지만 퍼블릭 리포지토리에서 제공하는 보호 기능으로 인해 패키지 그룹의 이 기능이 불필요해지는 것은 아닙니다. 예를 들어 npm Public Registry 레지스트리와 같은 퍼블릭 리포지토리는 my-package가 이미 게시된 경우에만 my-package라는 패키

지의 새로운 변형을 방지합니다. my-package가 내부 패키지이고 게시를 허용하고 수집을 차단하는 패키지 그룹 `/npm//my-package$`가 생성된 경우 my.package와 같은 변형이 허용되지 않도록 하기 위해 my-package를 npm Public Registry에 게시하고 싶지 않을 수 있습니다.

Maven과 같은 일부 패키지 형식은 이러한 문자를 다르게 취급하지만(Maven은 `.`를 네임스페이스 계층 구분자로 취급하지만 `-` 또는 `_`는 아님), com.act-on과 같은 형식은 여전히 com.act.on과 혼동될 수 있습니다.

> **Note**
>
> 패키지 그룹과 여러 변형이 연결될 때마다 관리자는 특정 변형에 대해 새 패키지 그룹을 생성하여 해당 변형에 대해 다른 동작을 구성할 수 있습니다.

CodeArtifact에서 패키지 그룹에 태그 지정

태그는 AWS 리소스와 연결된 키-값 페어입니다. CodeArtifact에 있는 패키지 그룹에 태그를 적용할 수 있습니다. CodeArtifact 리소스 태그 지정, 사용 사례, 태그 키 및 값 제약 조건, 지원되는 리소스 유형에 대한 자세한 내용은 리소스에 태그 지정 섹션을 참조하세요.

패키지 그룹을 생성하거나 기존 패키지 그룹의 태그 값을 추가, 제거 또는 업데이트할 때 CLI를 사용하여 태그를 지정할 수 있습니다.

패키지 그룹에 태그 지정(CLI)

CLI를 사용하여 패키지 그룹 태그를 관리할 수 있습니다.

아직 구성하지 않았다면 with AWS CodeArtifact 설정에 나오는 단계에 따라 AWS CLI를 구성하세요.

> **Tip**
>
> 태그를 추가하려면 패키지 그룹의 Amazon 리소스 이름(ARN)을 제공해야 합니다. 패키지 그룹의 ARN을 가져오려면 `describe-package-group` 명령을 실행합니다.

```
aws codeartifact describe-package-group \
   --domain my_domain \
   --package-group /npm/scope/anycompany~ \
   --query packageGroup.arn
```

주제

패키지 그룹에 태그 추가(CLI)

패키지 그룹이 생성될 때 패키지 그룹에 태그를 추가하거나 기존 패키지 그룹에 태그를 추가할 수 있습니다. 패키지 그룹을 생성할 때 패키지 그룹에 태그를 추가하는 방법에 대한 자세한 내용은 패키지 그룹 생성 섹션을 참조하세요.

AWS CLI를 사용하여 기존 패키지 그룹에 태그를 추가하려면 터미널이나 명령줄에서 tag-resource 명령을 실행하여 태그를 추가할 패키지 그룹의 Amazon 리소스 이름(ARN) 및 추가하려는 태그의 키와 값을 지정합니다. 패키지 그룹 ARN에 대한 자세한 내용은 패키지 그룹 ARN 섹션을 참조하세요.

하나의 패키지 그룹에 둘 이상의 태그를 추가할 수 있습니다. 예를 들어 */npm/scope/anycompany~* 패키지 그룹에 2개의 태그, 즉 태그 키가 *key1*이고 태그 값이 *value1*인 태그와 태그 키가 *key2*이고 태그 값이 *value2*인 태그를 추가하는 방법은 다음과 같습니다.

```
aws codeartifact tag-resource \
   --resource-arn arn:aws:codeartifact:us-west-2:123456789012:package-
group/my_domain/npm/scope/anycompany~ \
   --tags key=key1,value=value1 key=key2,value=value2
```

이 명령이 제대로 실행되면 출력이 표시되지 않습니다.

패키지 그룹의 태그 보기(CLI)

AWS CLI를 사용하여 패키지 그룹에 대한 AWS 태그를 보려면 다음 단계를 수행합니다. 태그가 추가되지 않은 경우 반환되는 목록은 비어 있습니다.

터미널 또는 명령줄에서 패키지 그룹의 Amazon 리소스 이름(ARN)을 사용하여 list-tags-for-resource 명령을 실행합니다. 패키지 그룹 ARN에 대한 자세한 내용은 패키지 그룹 ARN 섹션을 참조하세요.

예를 들어 패키지 그룹의 태그 키 및 태그 값 목록을 보려면 ARN 값 `arn:aws:codeartifact:`*us-west-2*`:`*123456789012*`:package-group/`*my_domain*`/`*npm/scope/anycompany~*로 명명된 */npm/scope/anycompany~*를 사용합니다.

```
aws codeartifact list-tags-for-resource \
   --resource-arn arn:aws:codeartifact:us-west-2:123456789012:package-
group/my_domain/npm/scope/anycompany~
```

이 명령이 제대로 실행되면 다음과 비슷한 정보를 반환합니다.

```
{
    "tags": {
        "key1": "value1",
        "key2": "value2"
    }
}
```

패키지 그룹의 태그 편집(CLI)

AWS CLI를 사용하여 패키지 그룹의 태그를 편집하려면 다음 단계를 수행합니다. 기존 키의 값을 변경하거나 다른 키를 추가할 수 있습니다. 다음 섹션에서 설명하는 것처럼 패키지 그룹에서 태그를 제거할 수도 있습니다.

터미널이나 명령줄에서 tag-resource 명령을 실행하여, 태그를 업데이트하고 태그 키 및 태그 값을 지정할 패키지 그룹의 ARN을 지정합니다. 패키지 그룹 ARN에 대한 자세한 내용은 패키지 그룹 ARN 섹션을 참조하세요.

```
aws codeartifact tag-resource \
   --resource-arn arn:aws:codeartifact:us-west-2:123456789012:package-
group/my_domain/npm/scope/anycompany~ \
   --tags key=key1,value=newvalue1
```

이 명령이 제대로 실행되면 출력이 표시되지 않습니다.

패키지 그룹에서 태그 제거(CLI)

AWS CLI를 사용하여 패키지 그룹에서 태그를 제거하려면 다음 단계를 수행합니다.

> **Note**
>
> 패키지 그룹을 삭제하면 삭제된 패키지 그룹에서 모든 태그 연결이 제거됩니다. 패키지 그룹을 삭제하기 전에 태그를 제거할 필요는 없습니다.

터미널이나 명령줄에서 untag-resource 명령을 실행하여, 태그를 제거할 패키지 그룹의 ARN 및 제거할 태그의 태그 키를 지정합니다. 패키지 그룹 ARN에 대한 자세한 내용은 패키지 그룹 ARN 섹션을 참조하세요.

예를 들어 태그 키 *key1* 및 *key2*로 패키지 그룹 */npm/scope/anycompany~*에서 여러 태그를 제거하려면 다음을 수행합니다.

```
aws codeartifact untag-resource \
   --resource-arn arn:aws:codeartifact:us-west-2:123456789012:package-
group/my_domain/npm/scope/anycompany~ \
   --tag-keys key1 key2
```

이 명령이 제대로 실행되면 출력이 표시되지 않습니다. 태그를 제거한 후에는 `list-tags-for-resource` 명령을 사용하여 패키지 그룹에 남아 있는 태그를 볼 수 있습니다.

CodeArtifact에서의 도메인 작업

CodeArtifact 도메인을 사용하면 조직 전체에서 여러 리포지토리를 쉽게 관리할 수 있습니다. 도메인을 사용하여 서로 다른 AWS 계정이 소유한 여러 리포지토리에 권한을 적용할 수 있습니다. 자산은 도메인에 한 번만 저장되며, 자산을 여러 리포지토리에서 사용할 수 있는 경우도 마찬가지입니다.

도메인은 여러 개 가질 수 있지만, 개발팀이 패키지를 찾고 공유할 수 있도록 게시된 아티팩트를 모두 포함하는 단일 프로덕션 도메인을 사용하는 것이 좋습니다. 두 번째 프리프로덕션 도메인을 사용하여 프로덕션 도메인 구성의 변경 사항을 테스트할 수 있습니다.

이 주제에서는 CodeArtifact 콘솔, AWS CLI및를 사용하여 CodeArtifact 도메인을 CloudFormation 생성하거나 구성하는 방법을 설명합니다.

주제

도메인 개요

CodeArtifact로 작업할 때 도메인은 다음 작업에 유용합니다.

- 중복 스토리지: 리포지토리 1개 또는 1,000개에서 사용 가능한 도메인 한 번만 저장하면 됩니다. 즉, 스토리지 비용은 한 번만 지불하면 됩니다.
- 빠른 복사: 업스트림 CodeArtifact 리포지토리에서 다운스트림으로 패키지를 가져오거나 CopyPackageVersions API를 사용하는 경우에는 메타데이터 레코드만 업데이트해야 합니다. 자산은 복사되지 않습니다. 이렇게 하면 새 리포지토리의 스테이징이나 테스트를 빠르게 설정할 수 있습니다. 자세한 내용은 CodeArtifact에서의 업스트림 리포지토리 작업 단원을 참조하십시오.
- 리포지토리 및 팀 간 간편한 공유: 도메인의 모든 자산 및 메타데이터는 단일 AWS KMS key (KMS 키)로 암호화됩니다. 각 리포지토리의 키를 관리하거나 단일 키에 대한 액세스 권한을 여러 계정에 부여하지 않아도 됩니다.
- 여러 리포지토리에 정책 적용: 도메인 관리자는 도메인 전체에 정책을 적용할 수 있습니다. 대표적인 예는 도메인 내 리포지토리에 액세스할 수 있는 계정과, 패키지 소스로 사용할 퍼블릭 리포지토리에

대한 연결을 구성할 수 있는 사용자를 제한하는 것입니다. 자세한 내용은 도메인 정책 섹션을 참조하세요.

- 고유한 리포지토리 이름: 도메인은 리포지토리를 위한 네임스페이스를 제공합니다. 리포지토리 이름은 도메인 내에서만 고유해야 합니다. 이해하기 쉽고 의미 있는 이름을 사용해야 합니다.

도메인 이름은 계정 내에서 고유해야 합니다.

도메인이 없으면 리포지토리를 만들 수 없습니다. CreateRepository API를 사용하여 리포지토리를 생성할 때는 도메인 이름을 지정해야 합니다. 한 도메인의 리포지토리를 다른 도메인으로 옮길 수는 없습니다.

리포지토리는 도메인을 소유한 동일한 AWS 계정 또는 다른 계정이 소유할 수 있습니다. 소유 계정이 다른 경우 리포지토리 소유 계정이 도메인 리소스에 대한 `CreateRepository` 권한을 부여받아야 합니다. PutDomainPermissionsPolicy 명령을 사용하여 도메인에 리소스 정책을 추가하면 이 작업을 수행할 수 있습니다.

조직에 여러 도메인이 있을 수 있지만, 개발팀이 조직 전체에서 패키지를 찾고 공유할 수 있도록 게시된 모든 아티팩트를 포함하는 단일 프로덕션 도메인을 사용하는 것이 좋습니다. 두 번째 프리프로덕션 도메인은 프로덕션 도메인 구성의 변경 사항을 테스트할 때 유용합니다.

크로스 계정 도메인

도메인 이름은 계정 내에서만 고유해야 합니다. 즉, 지역 내에 이름이 같은 도메인이 여러 개 존재할 수 있습니다. 따라서 인증되지 않은 계정이 소유한 도메인에 액세스하려면 CLI와 콘솔 모두에서 도메인 이름과 도메인 소유자 ID를 제공해야 합니다. 다음 CLI 예시를 참조하세요.

인증한 계정에서 소유한 도메인에 액세스하는 방법:

인증한 계정에서 도메인에 액세스할 때는 도메인 이름을 지정하기만 하면 됩니다. 다음 예제에서는 사용자 계정이 소유한 *my_domain* 도메인의 *my_repo* 리포지토리에 있는 패키지를 나열합니다.

```
aws codeartifact list-packages --domain my_domain --repository my_repo
```

인증하지 않은 계정에서 소유한 도메인에 액세스하는 방법:

인증하지 않은 계정이 소유한 도메인에 액세스할 때는 도메인 소유자와 도메인 이름을 지정해야 합니다. 다음 예제에서는 인증하지 않은 계정이 소유한 *other_domain* 도메인의 *other_repo* 리포지토리에 있는 패키지를 나열합니다. `--domain-owner` 파라미터가 추가되었다는 사실에 주목하세요.

```
aws codeartifact list-packages --domain other-domain --domain-owner 111122223333 --
repository other-repo
```

CodeArtifact에서 지원되는 AWS KMS 키 유형

CodeArtifact는 대칭 KMS 키만 지원합니다. 비대칭 KMS 키를 사용하여 CodeArtifact 도메인을 암호화할 수 없습니다. 자세한 내용은 대칭 및 비대칭 KMS 키 식별을 참조하세요. 새 고객 관리형 키를 생성하려면 AWS Key Management Service 개발자 안내서의 대칭 암호화 KMS 키 생성을 참조하세요.

CodeArtifact는 AWS KMS 외부 키 스토어(XKS)를 지원합니다. CodeArtifact의 가용성, 내구성 및 지연 시간에 영향을 줄 수 있는 XKS 키 작업의 가용성, 내구성 및 지연 시간은 사용자가 책임집니다. CodeArtifact와 XKS 키를 함께 사용할 때 발생하는 대표적인 효과는 다음과 같습니다.

- 요청된 패키지의 모든 자산과 모든 관련 종속성에는 복호화 지연 시간이 적용되므로, XKS 작업 지연 시간이 증가하면 빌드 지연 시간이 크게 늘어날 수 있습니다.
- CodeArtifact에서는 모든 자산이 암호화되므로, XKS 키 자료가 손실되면 XKS 키를 사용하는 도메인과 관련된 모든 자산이 손실됩니다.

XKS 키 관리에 대한 자세한 내용은 AWS Key Management Service 개발자 안내서의 외부 키 저장소를 참조하세요.

도메인 생성

CodeArtifact 콘솔, AWS Command Line Interface (AWS CLI) 또는를 사용하여 도메인을 생성할 수 있습니다 CloudFormation. 생성되는 도메인에는 리포지토리가 포함되지 않습니다. 자세한 내용은 리포지토리 생성 단원을 참조하십시오. CloudFormation으로 CodeArtifact 도메인을 관리하는 방법에 대한 자세한 내용은 AWS CloudFormation을 사용하여 CodeArtifact 리소스 생성 섹션을 참조하십시오.

주제

- 도메인 생성(콘솔)
- 도메인 생성(AWS CLI)
- AWS KMS 키 정책 예제

도메인 생성(콘솔)

1. https://console.aws.amazon.com/codesuite/codeartifact/home AWS CodeArtifact 콘솔을 엽니다.

2. 탐색 창에서 도메인을 선택하고 도메인 생성을 선택합니다.
3. 이름에 도메인의 이름을 입력합니다.
4. 추가 구성을 확장합니다.
5. AWS KMS key (KMS 키)를 사용하여 도메인의 모든 자산을 암호화합니다. AWS 관리형 KMS 키나 자신이 관리하는 KMS 키를 사용할 수 있습니다. CodeArtifact에서 지원되는 KMS 키 유형에 대한 자세한 내용은 CodeArtifact에서 지원되는 AWS KMS 키 유형 섹션을 참조하세요.

   - 기본 AWS 관리형 키를 사용하려면 AWS 관리형 키를 선택합니다.
   - 관리하는 KMS 키를 사용하려면 고객 관리형 키를 선택합니다. 관리하는 KMS 키를 사용하려면 고객 관리형 키 ARN에서 KMS 키를 검색하여 선택합니다.

   자세한 내용은 AWS Key Management Service 개발자 안내서의 AWS 관리형 키 및 고객 관리형 키를 참조하세요.
6. 도메인 생성을 선택합니다.

도메인 생성(AWS CLI)

를 사용하여 도메인을 생성하려면 `create-domain` 명령을 AWS CLI사용합니다. 도메인의 모든 자산을 암호화하려면 AWS KMS key (KMS 키)를 사용해야 합니다. AWS 관리형 KMS 키 또는 관리하는 KMS 키를 사용할 수 있습니다. AWS 관리형 KMS 키를 사용하는 경우 `--encryption-key` 파라미터를 사용하지 마십시오.

CodeArtifact에서 지원되는 KMS 키 유형에 대한 자세한 내용은 CodeArtifact에서 지원되는 AWS KMS 키 유형 섹션을 참조하세요. KMS 키에 대한 자세한 내용은 AWS Key Management Service 개발자 안내서의 AWS 관리형 키 및 고객 관리형 키를 참조하세요.

```
aws codeartifact create-domain --domain my_domain
```

JSON 형식의 데이터는 새 도메인에 관한 세부 정보와 함께 출력에 표시됩니다.

```
{
    "domain": {
        "name": "my_domain",
        "owner": "111122223333",
        "arn": "arn:aws:codeartifact:us-west-2:111122223333:domain/my_domain",
        "status": "Active",
```

```
        "encryptionKey": "arn:aws:kms:us-west-2:111122223333:key/your-kms-key",
        "repositoryCount": 0,
        "assetSizeBytes": 0,
        "createdTime": "2020-10-12T16:51:18.039000-04:00"
    }
}
```

자신이 관리하는 KMS 키를 사용하는 경우 Amazon 리소스 이름(ARN)을 `--encryption-key` 파라미터와 함께 포함해야 합니다.

```
aws codeartifact create-domain --domain my_domain --encryption-key arn:aws:kms:us-
west-2:111122223333:key/your-kms-key
```

JSON 형식의 데이터는 새 도메인 관련 세부 정보와 함께 출력에 표시됩니다.

```
{
    "domain": {
        "name": "my_domain",
        "owner": "111122223333",
        "arn": "arn:aws:codeartifact:us-west-2:111122223333:domain/my_domain",
        "status": "Active",
        "encryptionKey": "arn:aws:kms:us-west-2:111122223333:key/your-kms-key",
        "repositoryCount": 0,
        "assetSizeBytes": 0,
        "createdTime": "2020-10-12T16:51:18.039000-04:00"
    }
}
```

태그가 있는 도메인 생성

태그가 있는 도메인을 만들려면 `create-domain` 명령에 `--tags` 파라미터를 추가합니다.

```
aws codeartifact create-domain --domain my_domain --tags key=k1,value=v1
 key=k2,value=v2
```

AWS KMS 키 정책 예제

CodeArtifact에서 도메인을 생성할 때 KMS 키를 사용하여 도메인의 모든 자산을 암호화할 수 있습니다. AWS 관리형 KMS 키 또는 사용자가 관리하는 고객 관리형 키를 선택할 수 있습니다. KMS 키에 대한 자세한 내용은 AWS Key Management Service 개발자 가이드를 참조하세요.

고객 관리형 키를 사용하려면 KMS 키에 CodeArtifact에 대한 액세스 권한을 부여하는 키 정책이 있어야 합니다. 키 정책은 AWS KMS 키에 대한 리소스 정책이며 KMS 키에 대한 액세스를 제어하는 기본 방법입니다. 모든 KMS 키에는 단일 키 정책이 요구되고, 키 정책에 따라 KMS 키의 사용 권한 보유자와 사용 방법이 결정됩니다.

다음 예제 키 정책 문은 AWS CodeArtifact가 권한 부여를 생성하고 권한 있는 사용자를 대신하여 키 세부 정보를 볼 수 있도록 허용합니다. 이 정책 문은 `kms:ViaService` 및 `kms:CallerAccount` 조건 키를 사용하여 지정된 계정 ID를 대신하여 작동하는 CodeArtifact에 대한 권한을 제한합니다. 또한 IAM 루트 사용자에게 모든 AWS KMS 권한을 부여하므로 키가 생성된 후 관리할 수 있습니다.

JSON

```
{
    "Version":"2012-10-17",
    "Id": "key-consolepolicy-3",
    "Statement": [
        {
            "Sid": "Allow access through AWS CodeArtifact for all principals in
 the account that are authorized to use CodeArtifact",
            "Effect": "Allow",
            "Principal": {
                "AWS": "*"
            },
            "Action": [
                "kms:CreateGrant",
                "kms:DescribeKey"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "kms:CallerAccount": "111122223333",
                    "kms:ViaService": "codeartifact.us-west-2.amazonaws.com"
                }
            }
        },
        {
            "Sid": "Enable IAM User Permissions",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:root"
            },
            "Action": "kms:*",
```

```
            "Resource": "*"
        }
    ]
}
```

도메인 삭제

CodeArtifact 콘솔 또는 AWS Command Line Interface ()를 사용하여 도메인을 삭제할 수 있습니다 AWS CLI.

주제

- 도메인 삭제 제한
- 도메인 삭제(콘솔)
- 도메인 삭제(AWS CLI)

도메인 삭제 제한

일반적으로 리포지토리가 포함된 도메인은 삭제할 수 없습니다. 도메인을 삭제하려면 먼저 도메인의 리포지토리를 삭제해야 합니다. 자세한 내용은 리포지토리 삭제 단원을 참조하십시오.

하지만 CodeArtifact에서 도메인의 KMS 키에 액세스할 수 없다면, 아직 리포지토리가 포함되어 있는 도메인도 삭제할 수 있습니다. 이 상황은 도메인의 KMS 키를 삭제하거나 CodeArtifact가 키에 액세스하는 데 사용하는 KMS 승인을 취소하는 경우 발생합니다. 이 상태에서는 도메인 내 리포지토리나 도메인에 저장된 패키지에 액세스할 수 없습니다. CodeArtifact가 도메인의 KMS 키에 액세스할 수 없다면 리포지토리를 나열하고 삭제할 수도 없습니다. 따라서 도메인의 KMS 키에 액세스할 수 없는 경우 도메인 삭제 시 도메인에 리포지토리가 포함되어 있는지를 확인하지 않습니다.

> **Note**
>
> 리포지토리가 포함되어 있는 도메인을 삭제하면 CodeArtifact는 15분 이내에 리포지토리를 비동기적으로 삭제합니다. 도메인이 삭제된 후에도 자동 리포지토리 정리가 수행되기 전에는 CodeArtifact 콘솔 및 `list-repositories` 명령의 출력에 리포지토리가 계속 표시됩니다.

도메인 삭제(콘솔)

1. https://console.aws.amazon.com/codesuite/codeartifact/home AWS CodeArtifact 콘솔을 엽니다.

2. 탐색 창에서 도메인을 선택하고 삭제할 도메인을 선택합니다.
3. 삭제를 선택합니다.

도메인 삭제(AWS CLI)

delete-domain 명령을 사용하여 도메인을 삭제합니다.

```
aws codeartifact delete-domain --domain my_domain --domain-owner 111122223333
```

JSON 형식의 데이터는 삭제된 도메인에 관한 세부 정보와 함께 출력 화면에 표시됩니다.

```
{
    "domain": {
        "name": "my_domain",
        "owner": "111122223333",
        "arn": "arn:aws:codeartifact:us-west-2:111122223333:domain/my_domain",
        "status": "Active",
        "encryptionKey": "arn:aws:kms:us-west-2:111122223333:key/your-kms-key",
        "repositoryCount": 0,
        "assetSizeBytes": 0,
        "createdTime": "2020-10-12T16:51:18.039000-04:00"
    }
}
```

도메인 정책

CodeArtifact는 리소스 기반 권한을 사용한 액세스 제어를 지원합니다. 리소스 기반 권한을 사용하면 리소스에 액세스할 수 있는 사람과 이러한 사람이 리소스를 대상으로 수행할 수 있는 작업을 지정할 수 있습니다. 기본적으로는 도메인을 소유하는 AWS 계정만 도메인에서 리포지토리를 생성하고 리포지토리에 액세스할 수 있습니다. 사용자는 다른 IAM 보안 주체의 도메인 액세스를 허용하는 정책 문서를 도메인에 적용할 수 있습니다.

자세한 내용은 정책 및 권한과 자격 증명 기반 정책 및 리소스 기반 정책을 참조하세요.

주제

- 도메인에 대한 크로스 계정 액세스 활성화
- 도메인 정책 예제
- 를 사용한 도메인 정책 예제 AWS Organizations

- 도메인 정책 설정
- 도메인 정책 읽기
- 도메인 정책 삭제

도메인에 대한 크로스 계정 액세스 활성화

리소스 정책은 JSON 형식의 텍스트 파일입니다. 파일은 보안 주체(액터), 하나 이상의 작업과 단일 효과(`Allow`또는`Deny`)를 지정해야 합니다. 다른 계정이 소유한 도메인에 리포지토리를 만들려면, 보안 주체는 도메인 리소스에 대한 `CreateRepository` 권한을 부여받아야 합니다.

예를 들어 다음 리소스 정책은 도메인에 리포지토리를 만들 수 있는 권한을 `123456789012` 계정에 부여합니다.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "codeartifact:CreateRepository"
            ],
            "Effect": "Allow",
            "Principal": {
                 "AWS": "arn:aws:iam::123456789012:root"
            },
            "Resource": "*"
        }
    ]
}
```

태그가 있는 리포지토리를 만들 수 있게 하려면 `codeartifact:TagResource` 권한을 포함해야 합니다. 이렇게 하면 도메인과 도메인에 있는 모든 리포지토리에 태그를 추가할 수 있는 액세스 권한도 계정에 부여됩니다.

도메인 정책은 도메인에 대한 모든 작업과 도메인 내의 모든 리소스에 대해 평가됩니다. 즉, 도메인 정책을 사용하여 도메인의 리포지토리 및 패키지에 권한을 적용할 수 있습니다. `Resource` 요소가 `*`로 설정되면 해당 명령문이 도메인의 모든 리소스에 적용됩니다. 예를 들어 위의 정책이 허용된 IAM 작업

목록에 codeartifact:DescribeRepository를 포함한 경우 정책은 도메인의 모든 리포지토리에서 DescribeRepository 호출을 허용합니다. 도메인 정책은 Resource 요소의 특정 리소스 ARN을 사용하여 도메인의 특정 리소스에 권한을 적용하는 데 사용할 수 있습니다.

> **Note**
>
> 도메인 및 리포지토리 정책을 모두 사용하여 권한을 구성할 수 있습니다. 두 정책이 모두 있으면 두 정책이 모두 평가되며 두 정책에서 허용하는 경우 작업이 허용됩니다. 자세한 내용은 리포지토리와 도메인 정책 간의 상호 작용 단원을 참조하십시오.

다른 계정이 소유한 도메인에 있는 패키지에 액세스하려면, 보안 주체는 도메인 리소스에 대한 GetAuthorizationToken 권한을 부여받아야 합니다. 이렇게 하면 도메인 소유자는 도메인 내 리포지토리의 콘텐츠를 읽을 수 있는 계정을 제어할 수 있습니다.

예를 들어 다음 리소스 정책은 도메인 내 모든 리포지토리에 대한 인증 토큰을 검색할 수 있는 권한을 123456789012 계정에 부여합니다.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "codeartifact:GetAuthorizationToken"
            ],
            "Effect": "Allow",
            "Principal": {
                 "AWS": "arn:aws:iam::123456789012:root"
            },
            "Resource": "*"
        }
    ]
}
```

> **Note**
>
> 리포지토리 엔드포인트에서 패키지를 가져오려는 보안 주체는 도메인에 대한 `ReadFromRepository` 권한 외에 리포지토리 리소스에 대한 `GetAuthorizationToken` 권한도 부여받아야 합니다. 마찬가지로, 리포지토리 엔드포인트에 패키지를 게시하려는 보안 주체는 `GetAuthorizationToken`에 더해 `PublishPackageVersion` 권한도 부여받아야 합니다.
>
> `ReadFromRepository` 및 `PublishPackageVersion` 권한에 대한 자세한 내용은 리포지토리 정책을 참조하세요.

도메인 정책 예제

여러 계정이 도메인을 사용하는 경우, 도메인을 완전히 사용할 수 있도록 계정에 기본 권한 모음을 부여해야 합니다. 다음 리소스 정책에는 도메인의 완전한 사용을 허용하는 권한 모음이 나열되어 있습니다.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "BasicDomainPolicy",
            "Action": [
                "codeartifact:GetDomainPermissionsPolicy",
                "codeartifact:ListRepositoriesInDomain",
                "codeartifact:GetAuthorizationToken",
                "codeartifact:DescribeDomain",
                "codeartifact:CreateRepository"
            ],
            "Effect": "Allow",
            "Resource": "*",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:root"
            }
        }
    ]
}
```

> **Note**
>
> 도메인과 도메인의 모든 리포지토리를 단일 계정에서 소유하며 해당 계정에서만 사용해야 한다면 도메인 정책을 만들지 않아도 됩니다.

를 사용한 도메인 정책 예제 AWS Organizations

다음과 같이 `aws:PrincipalOrgID` 조건 키를 사용하여 조직 내 모든 계정에서 CodeArtifact 도메인에 대한 액세스 권한을 부여할 수 있습니다.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Sid": "DomainPolicyForOrganization",
        "Effect": "Allow",
        "Principal": "*",
        "Action": [
             "codeartifact:GetDomainPermissionsPolicy",
             "codeartifact:ListRepositoriesInDomain",
             "codeartifact:GetAuthorizationToken",
             "codeartifact:DescribeDomain",
             "codeartifact:CreateRepository"
        ],
        "Resource": "*",
        "Condition": {
            "StringEquals": { "aws:PrincipalOrgID":["o-xxxxxxxxxxx"]}
        }
    }
}
```

`aws:PrincipalOrgID` 조건 키 사용에 대한 자세한 내용은 IAM 사용 설명서의 AWS 글로벌 조건 컨텍스트 키를 참조하세요.

도메인 정책 설정

`put-domain-permissions-policy` 명령을 사용하여 정책을 도메인에 연결합니다.

```
aws codeartifact put-domain-permissions-policy --domain my_domain --domain-
owner 111122223333 \
 --policy-document file://</PATH/TO/policy.json>
```

put-domains-permissions-policy를 호출하면 권한을 평가할 때 도메인의 리소스 정책이 무시됩니다. 이렇게 하면 도메인 소유자가 자기 자신을 도메인 외부에 고립되게 해 리소스 정책을 업데이트하지 못하는 일을 방지할 수 있습니다.

> **Note**
>
> put-domain-permissions-policy를 호출할 때 리소스 정책이 무시되므로 리소스 정책을 사용하여 도메인의 리소스 정책을 업데이트할 수 있는 권한을 다른 AWS 계정에 부여할 수 없습니다.

샘플 출력:

```
{
    "policy": {
        "resourceArn": "arn:aws:codeartifact:region-id:111122223333:domain/my_domain",
        "document": "{ ...policy document content...}",
        "revision": "MQlyyTQRASRU3HB58gBtSDHXG7Q3hvxxxxxxx="
    }
}
```

명령 결과에는 도메인 리소스의 Amazon 리소스 이름(ARN), 정책 문서의 전체 내용과 개정 식별자가 포함됩니다. --policy-revision 옵션을 사용하면 개정 식별자를 put-domain-permissions-policy에 전달할 수 있습니다. 이렇게 하면 문서의 알려진 개정은 덮어쓰고, 다른 작성자가 설정한 최신 버전은 덮어쓰지 않게 됩니다.

도메인 정책 읽기

정책 문서의 기존 버전을 읽으려면 get-domain-permissions-policy 명령을 사용합니다. 읽기 쉬운 출력 형식을 지정하려면 다음과 같이 --output 및 --query policy.document 모두를 Python json.tool 모듈과 함께 사용하세요.

```
aws codeartifact get-domain-permissions-policy --domain my_domain --domain-
owner 111122223333 \
   --output text --query policy.document | python -m json.tool
```

샘플 출력:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "BasicDomainPolicy",
            "Action": [
                "codeartifact:GetDomainPermissionsPolicy",
                "codeartifact:ListRepositoriesInDomain",
                "codeartifact:GetAuthorizationToken",
                "codeartifact:CreateRepository"
            ],
            "Effect": "Allow",
            "Resource": "*",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:root"
            }
        }
    ]
}
```

도메인 정책 삭제

`delete-domain-permissions-policy` 명령을 사용하여 도메인에서 정책을 삭제합니다.

```
aws codeartifact delete-domain-permissions-policy --domain my_domain --domain-
owner 111122223333
```

출력의 형식은 `get-domain-permissions-policy` 및 `delete-domain-permissions-policy` 명령의 형식과 동일합니다.

CodeArtifact에서 도메인에 태그 추가

태그는 AWS 리소스와 연결된 키-값 페어입니다. CodeArtifact에 있는 도메인에 태그를 적용할 수 있습니다. CodePipeline 리소스 태그 추가, 사용 사례, 태그 키 및 값 제약, 지원되는 리소스 유형에 대한 자세한 내용은 리소스에 태그 지정 섹션을 참조하세요.

도메인을 만들 때 CLI를 사용하여 태그를 추가할 수 있습니다. 콘솔 또는 CLI를 사용하여 도메인에서 태그를 추가 또는 제거하고 태그 값을 업데이트할 수 있습니다. 각 도메인에 최대 50개의 태그를 추가할 수 있습니다.

주제

도메인에 태그 추가(CLI)

CLI를 사용하여 태그를 관리할 수 있습니다.

주제

도메인에 태그 추가(CLI)

콘솔 또는를 사용하여 도메인 AWS CLI 에 태그를 지정할 수 있습니다.

도메인을 생성할 때 태그를 추가하려면 리포지토리 생성 섹션을 참조하세요.

이 단계에서는 사용자가 이미 AWS CLI 의 최신 버전을 설치했거나 현재 버전으로 업데이트했다고 가정합니다. 자세한 정보는 AWS Command Line Interface설치 섹션을 참조하세요.

터미널이나 명령줄에서 tag-resource 명령을 실행하여, 태그를 추가할 도메인의 Amazon 리소스 이름(ARN)과 추가할 태그의 키와 값을 지정합니다.

> **Note**
>
> 도메인의 ARN을 가져오려면 `describe-domain` 명령을 실행합니다.
>
> ```
> aws codeartifact describe-domain --domain my_domain --query domain.arn
> ```

도메인에 두 개 이상의 태그를 추가할 수 있습니다. 예를 들어 *my_domain* 도메인에 태그 2개, 즉 태그 키가 *key1*이고 태그 값이 *value1*인 태그와 태그 키가 *key2*이고 태그 값이 *value2*인 태그를 추가하는 방법은 다음과 같습니다.

```
aws codeartifact tag-resource --resource-arn arn:aws:codeartifact:us-
west-2:123456789012:domain/my_domain --tags key=key1,value=value1 key=key2,value=value2
```

성공하면 이 명령에는 출력이 표시되지 않습니다.

도메인의 태그 보기(CLI)

다음 단계에 따라를 사용하여 도메인의 AWS 태그를 AWS CLI 확인합니다. 태그가 추가되지 않은 경우 반환되는 목록은 비어 있습니다.

터미널 또는 명령줄에서 도메인의 Amazon 리소스 이름(ARN)을 사용하여 list-tags-for-resource 명령을 실행합니다.

> **Note**
>
> 도메인의 ARN을 가져오려면 `describe-domain` 명령을 실행합니다.
>
> ```
> aws codeartifact describe-domain --domain my_domain --query domain.arn
> ```

예를 들어 `arn:aws:codeartifact:us-west-2:123456789012:domain/my_domain` ARN 값이 있는 *my_repo* 도메인의 태그 키 및 태그 값 목록을 보는 방법은 다음과 같습니다.

```
aws codeartifact list-tags-for-resource --resource-arn arn:aws:codeartifact:us-
west-2:123456789012:domain/my_domain
```

이 명령이 제대로 실행되면 다음과 비슷한 정보를 반환합니다.

```
{
    "tags": {
        "key1": "value1",
        "key2": "value2"
    }
}
```

도메인의 태그 편집(CLI)

다음 단계에 따라 AWS CLI 를 사용하여 도메인의 태그를 편집합니다. 기존 키의 값을 변경하거나 다른 키를 추가할 수 있습니다. 다음 단원에서 설명하는 것처럼 도메인에서 태그를 제거할 수도 있습니다.

터미널이나 명령줄에서 tag-resource 명령을 실행하여, 태그를 업데이트하고 태그 키 및 태그 값을 지정할 도메인의 ARN을 지정합니다.

> **Note**
>
> 도메인의 ARN을 가져오려면 `describe-domain` 명령을 실행합니다.
>
> ```
> aws codeartifact describe-domain --domain my_domain --query domain.arn
> ```

```
aws codeartifact tag-resource --resource-arn arn:aws:codeartifact:us-
west-2:123456789012:domain/my_domain --tags key=key1,value=newvalue1
```

이 명령이 제대로 실행되면 출력이 표시되지 않습니다.

도메인에서 태그 제거(CLI)

다음 단계에 따라 AWS CLI 를 사용하여 도메인에서 태그를 제거합니다.

> **Note**
>
> 도메인을 삭제하면 삭제된 도메인에서 모든 태그 연결이 제거됩니다. 도메인을 삭제하기 전에 태그를 제거할 필요가 없습니다.

터미널이나 명령줄에서 untag-resource 명령을 실행하여, 태그를 제거할 도메인의 ARN과 제거할 태그의 태그 키를 지정합니다.

> **Note**
>
> 도메인의 ARN을 가져오려면 `describe-domain` 명령을 실행합니다.
>
> ```
> aws codeartifact describe-domain --domain my_domain --query domain.arn
> ```

예를 들어 태그 키 *key1* 및 *key2*가 있는 *mydomain* 도메인에서 여러 태그를 제거하는 방법은 다음과 같습니다.

```
aws codeartifact untag-resource --resource-arn arn:aws:codeartifact:us-
west-2:123456789012:domain/my_domain --tag-keys key1 key2
```

이 명령이 제대로 실행되면 출력이 표시되지 않습니다. 태그를 제거한 후에는 `list-tags-for-resource` 명령을 사용하여 도메인에 남아 있는 태그를 볼 수 있습니다.

도메인 태그 지정(콘솔)

콘솔 또는 CLI를 사용하여 리소스에 태그를 지정할 수 있습니다.

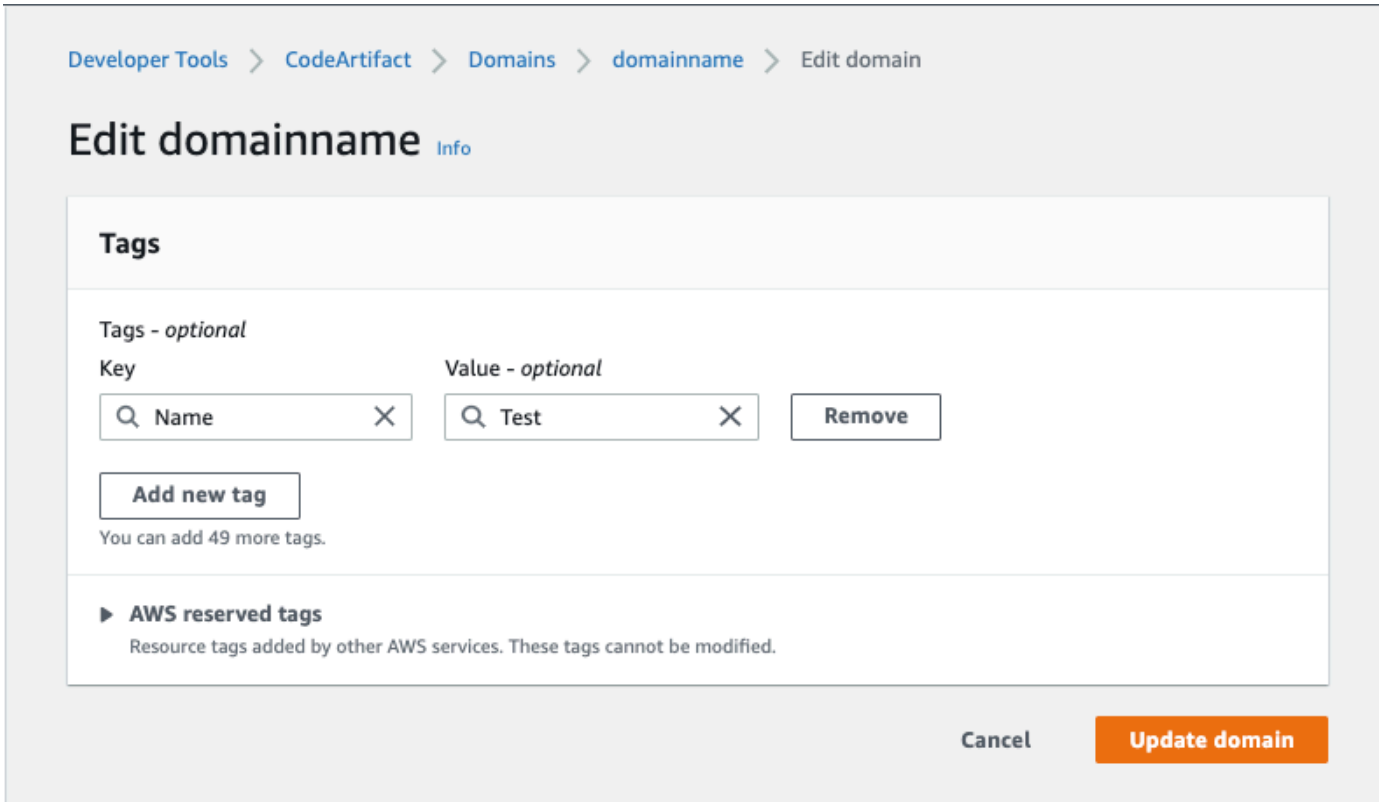
주제

- 도메인에 태그 추가(콘솔)
- 도메인의 태그 보기(콘솔)
- 도메인의 태그 편집(콘솔)
- 도메인에서 태그 제거(콘솔)

도메인에 태그 추가(콘솔)

콘솔을 사용하여 기존 도메인에 태그를 추가할 수 있습니다.

1. https://console.aws.amazon.com/codesuite/codeartifact/home AWS CodeArtifact 콘솔을 엽니다.
2. 도메인 페이지에서 태그를 추가할 도메인을 선택합니다.
3. 세부 정보 섹션을 확장합니다.
4. 도메인 태그에서 도메인에 태그가 없다면 도메인 태그 추가를 선택하고, 태그가 있다면 도메인 태그 보기 및 편집을 선택합니다.
5. 새 태그 추가를 선택합니다.
6. 키 및 값 필드에, 추가할 각 태그의 텍스트를 입력합니다. (값 필드는 선택 사항입니다.) 예를 들어 키에 **Name**을 입력합니다. 값에는 **Test**를 입력합니다.



7. (선택 사항) 행을 추가하고 태그를 더 입력하려면 태그 추가를 선택합니다.
8. 도메인 업데이트를 선택합니다.

도메인의 태그 보기(콘솔)

콘솔을 사용하여 기존 도메인에 대한 태그를 나열할 수 있습니다.

1. https://console.aws.amazon.com/codesuite/codeartifact/home AWS CodeArtifact 콘솔을 엽니다.
2. 도메인 페이지에서 태그를 확인할 도메인을 선택합니다.
3. 세부 정보 섹션을 확장합니다.
4. 도메인 태그에서 도메인 태그 보기 및 편집을 선택합니다.

> **Note**
>
> 이 도메인에 추가된 태그가 없는 경우 콘솔에 도메인 태그 추가가 표시됩니다.

도메인의 태그 편집(콘솔)

도메인에 추가된 태그를 콘솔을 사용하여 편집할 수 있습니다.

1. https://console.aws.amazon.com/codesuite/codeartifact/home AWS CodeArtifact 콘솔을 엽니다.
2. 도메인 페이지에서 태그를 업데이트할 도메인을 선택합니다.
3. 세부 정보 섹션을 확장합니다.
4. 도메인 태그에서 도메인 태그 보기 및 편집을 선택합니다.

> **Note**
>
> 이 도메인에 추가된 태그가 없는 경우 콘솔에 도메인 태그 추가가 표시됩니다.

5. 키 및 값 필드에서 필요에 따라 각 필드의 값을 업데이트합니다. 예를 들어 **`Name`** 키의 값에서 **`Test`**를 **`Prod`**로 변경합니다.
6. 도메인 업데이트를 선택합니다.

도메인에서 태그 제거(콘솔)

콘솔을 사용하여 도메인에서 태그를 삭제할 수 있습니다.

1. https://console.aws.amazon.com/codesuite/codeartifact/home AWS CodeArtifact 콘솔을 엽니다.
2. 도메인 페이지에서 태그를 제거할 도메인을 선택합니다.
3. 세부 정보 섹션을 확장합니다.
4. 도메인 태그에서 도메인 태그 보기 및 편집을 선택합니다.

> **Note**
>
> 이 도메인에 추가된 태그가 없는 경우 콘솔에 도메인 태그 추가가 표시됩니다.

5. 삭제할 각 태그의 키와 값 옆에 있는 제거를 선택합니다.
6. 도메인 업데이트를 선택합니다.

CodeArtifact를 Cargo와 함께 사용

이 항목에서는 Rust 패키지 관리자인 Cargo를 CodeArtifact와 함께 사용하는 방법을 설명합니다.

> **Note**
>
> CodeArtifact는 Cargo 1.74.0 이상만 지원합니다. Cargo 1.74.0은 CodeArtifact 리포지토리에서 인증을 지원하는 가장 빠른 버전입니다.

주제

CodeArtifact로 Cargo 구성 및 사용

Cargo를 사용하여 CodeArtifact 리포지토리에서 크레이트를 게시 및 다운로드하거나 Rust 커뮤니티의 크레이트 레지스트리인 crates.io에서 크레이트를 가져올 수 있습니다. 이 항목에서는 CodeArtifact 리포지토리를 인증하고 사용하도록 Cargo를 구성하는 방법을 설명합니다.

CodeArtifact를 사용하여 Cargo 설정

Cargo를 사용하여 AWS CodeArtifact에서 크레이트를 설치하고 게시하려면 먼저 CodeArtifact 리포지토리 정보를 사용하여 구성해야 합니다. 다음 절차 중 하나의 단계에 따라 CodeArtifact 리포지토리 엔드포인트 정보 및 자격 증명으로 Cargo를 구성합니다.

콘솔 지침을 사용하여 Cargo 구성

콘솔의 구성 지침을 사용하여 Cargo를 CodeArtifact 리포지토리에 연결할 수 있습니다. 콘솔 지침은 CodeArtifact 리포지토리에 맞게 사용자 지정된 Cargo 구성을 제공합니다. CodeArtifact 정보를 찾아 입력할 필요 없이 이 사용자 지정 구성을 사용하여 Cargo를 설정할 수 있습니다.

1. https://console.aws.amazon.com/codesuite/codeartifact/home에서 AWS CodeArtifact 콘솔을 엽니다.
2. 탐색 창에서 리포지토리를 선택한 다음, Cargo에 연결할 리포지토리를 선택합니다.
3. 연결 지침 보기를 선택합니다.

4. 운영 체제를 선택합니다.
5. Cargo를 선택합니다.
6. 생성된 지침에 따라 Cargo를 CodeArtifact 리포지토리에 연결합니다.

Cargo 수동 구성

콘솔의 구성 지침을 사용할 수 없거나 사용하지 않으려는 경우 다음 지침을 사용하여 Cargo를 CodeArtifact 리포지토리에 수동으로 연결할 수 있습니다.

macOS and Linux

CodeArtifact로 Cargo를 구성하려면 CodeArtifact 리포지토리를 Cargo 구성의 레지스트리로 정의하고 자격 증명을 제공해야 합니다.

- *my_registry*를 레지스트리 이름으로 변경합니다.
- *my_domain*을 CodeArtifact 도메인 이름으로 변경합니다.
- *111122223333*을 도메인 소유자의 AWS 계정 ID로 변경합니다. 소유한 도메인의 리포지토리에 액세스하는 경우 `--domain-owner`를 포함하지 않아도 됩니다. 자세한 내용은 크로스 계정 도메인 섹션을 참조하세요.
- *my_repo*를 CodeArtifact 리포지토리 이름으로 변경합니다.

구성을 복사하여 Cargo 패키지를 리포지토리에 게시 및 다운로드하고 시스템 수준 구성을 위해 `~/.cargo/config.toml` 파일에 저장하거나 프로젝트 수준 구성을 위해 `.cargo/config.toml`에 저장합니다.

```
[registries.my_registry]
index = "sparse+https://my_domain-111122223333.d.codeartifact.us-
west-2.amazonaws.com/cargo/my_repo/"
credential-provider = "cargo:token-from-stdout aws codeartifact get-authorization-
token --domain my_domain --domain-owner 111122223333 --region us-west-2 --query
 authorizationToken --output text"

[registry]
default = "my_registry"

[source.crates-io]
replace-with = "my_registry"
```

Windows: Download packages only

CodeArtifact로 Cargo를 구성하려면 CodeArtifact 리포지토리를 Cargo 구성의 레지스트리로 정의하고 자격 증명을 제공해야 합니다.

- *my_registry*를 레지스트리 이름으로 변경합니다.
- *my_domain*을 CodeArtifact 도메인 이름으로 변경합니다.
- *111122223333*을 도메인 소유자의 AWS 계정 ID로 변경합니다. 소유한 도메인의 리포지토리에 액세스하는 경우 `--domain-owner`를 포함하지 않아도 됩니다. 자세한 내용은 크로스 계정 도메인 섹션을 참조하세요.
- *my_repo*를 CodeArtifact 리포지토리 이름으로 변경합니다.

구성을 복사하여 리포지토리에서 Cargo 패키지만 다운로드하고 시스템 수준 구성을 위해 `%USERPROFILE%\.cargo\config.toml` 파일에 저장하거나 프로젝트 수준 구성을 위해 `.cargo\config.toml`에 저장합니다.

```
[registries.my_registry]
index = "sparse+https://my_domain-111122223333.d.codeartifact.us-
west-2.amazonaws.com/cargo/my_repo/"
credential-provider = "cargo:token-from-stdout aws codeartifact get-authorization-
token --domain my_domain --domain-owner 111122223333 --region us-west-2 --query
 authorizationToken --output text"

[registry]
default = "my_registry"

[source.crates-io]
replace-with = "my_registry"
```

Windows: Publish and download packages

1. CodeArtifact로 Cargo를 구성하려면 CodeArtifact 리포지토리를 Cargo 구성의 레지스트리로 정의하고 자격 증명을 제공해야 합니다.

   - *my_registry*를 레지스트리 이름으로 변경합니다.
   - *my_domain*을 CodeArtifact 도메인 이름으로 변경합니다.
   - *111122223333*을 도메인 소유자의 AWS 계정 ID로 변경합니다. 소유한 도메인의 리포지토리에 액세스하는 경우 `--domain-owner`를 포함하지 않아도 됩니다. 자세한 내용은 크로스 계정 도메인 섹션을 참조하세요.

- *my_repo*를 CodeArtifact 리포지토리 이름으로 변경합니다.

구성을 복사하여 Cargo 패키지를 리포지토리에 게시 및 다운로드하고 시스템 수준 구성을 위해 %USERPROFILE%\.cargo\config.toml 파일에 저장하거나 프로젝트 수준 구성을 위해 .cargo\config.toml에 저장합니다.

~/.cargo/credentials.toml 파일에 저장된 자격 증명을 사용하는 자격 증명 공급자 cargo:token을 사용하는 것이 좋습니다. cargo publish 중에 Cargo 클라이언트가 권한 부여 토큰을 제대로 잘라내지 않기 때문에 cargo:token-from-stdout를 사용하는 경우 cargo publish 동안 오류가 발생할 수 있습니다.

```
[registries.my_registry]
index = "sparse+https://my_domain-111122223333.d.codeartifact.us-
west-2.amazonaws.com/cargo/my_repo/"
credential-provider = "cargo:token"

[registry]
default = "my_registry"

[source.crates-io]
replace-with = "my_registry"
```

2. Windows를 사용하여 리포지토리에 Cargo 패키지를 게시하려면 CodeArtifact get-authorization-token 명령 및 Cargo login 명령을 사용하여 권한 부여 토큰과 자격 증명을 가져와야 합니다.

   - *my_registry*를 [registries.*my_registry*]에 정의된 레지스트리 이름으로 변경합니다.
   - *my_domain*을 CodeArtifact 도메인 이름으로 변경합니다.
   - *111122223333*을 도메인 소유자의 AWS 계정 ID로 변경합니다. 소유한 도메인의 리포지토리에 액세스하는 경우 --domain-owner를 포함하지 않아도 됩니다. 자세한 내용은 크로스 계정 도메인 섹션을 참조하세요.

```
aws codeartifact get-authorization-token --domain my_domain --domain-
owner 111122223333 --region us-west-2 --query authorizationToken --output text |
 cargo login --registry my_registry
```

> **Note**
>
> 생성된 권한 부여 토큰은 12시간 동안 유효합니다. 토큰이 생성된 후 12시간이 경과한 경우 새 토큰을 생성해야 합니다.

이전 예제의 `[registries.my_registry]` 섹션은 *my_registry*을 사용하여 레지스트리를 정의하고 `index` 및 `credential-provider` 정보를 제공합니다.

- `index`는 레지스트리에 대한 인덱스의 URL을 지정하며, 이 URL은 `/`로 끝나는 CodeArtifact 리포지토리 엔드포인트입니다. `sparse+` 접두사는 Git 리포지토리가 아닌 레지스트리에 필요합니다.

  > **Note**
  >
  > 듀얼 스택 엔드포인트를 사용하려면 `codeartifact.region.on.aws` 엔드포인트를 사용합니다.

- `credential-provider`는 지정된 레지스트리에 대한 자격 증명 공급자를 지정합니다. `credential-provider`가 설정되지 않은 경우 `registry.global-credential-providers`의 공급자가 사용됩니다. `credential-provider`를 `cargo:token-from-stdout`로 설정하면 CodeArtifact 리포지토리에서 게시하거나 다운로드할 때 Cargo 클라이언트가 새 권한 부여 토큰을 자동으로 가져오므로 12시간마다 권한 부여 토큰을 수동으로 새로 고칠 필요가 없습니다.

`[registry]` 섹션에서는 사용되는 기본 레지스트리를 정의합니다.

- `default`는 CodeArtifact 리포지토리에서 게시하거나 다운로드할 때 기본적으로 사용할 `[registries.my_registry]`에 정의된 레지스트리의 이름을 지정합니다.

`[source.crates-io]` 섹션에서는 지정되지 않은 경우 사용되는 기본 레지스트리를 정의합니다.

- `replace-with = "my_registry"`는 퍼블릭 레지스트리 crates.io를 `[registries.my_registry]`에 정의된 CodeArtifact 리포지토리로 바꿉니다. 이 구성은 crates.io와 같은 외부 연결에서 패키지를 요청해야 하는 경우에 권장됩니다.

  종속성 혼동 공격을 방지하는 패키지 원본 제어와 같은 CodeArtifact의 모든 이점을 얻으려면 소스 교체를 사용하는 것이 좋습니다. 소스 교체를 통해 CodeArtifact는 외부 연결에 대한 모든 요청을 프록시하고 외부 연결에서 리포지토리로 패키지를 복사합니다. 소스 교체가 없으면 Cargo 클라이언

트는 프로젝트의 `Cargo.toml` 파일에 있는 구성을 기반으로 패키지를 직접 검색합니다. 종속성이 `registry=my_registry`로 표시되지 않은 경우 Cargo 클라이언트는 CodeArtifact 리포지토리와 통신하지 않고 crates.io를 직접 검색합니다.

> **Note**
>
> 소스 교체 사용을 시작한 다음, 소스 교체를 사용하지 않도록 구성 파일을 업데이트하면 오류가 발생할 수 있습니다. 반대 시나리오에서도 오류가 발생할 수 있습니다. 따라서 프로젝트의 구성을 변경하지 않는 것이 좋습니다.

Cargo 크레이트 설치

다음 절차를 사용하여 CodeArtifact 리포지토리 또는 crates.io에서 Cargo 크레이트를 설치합니다.

CodeArtifact에서 Cargo 크레이트 설치

Cargo(`cargo`) CLI를 사용하여 CodeArtifact 리포지토리에서 특정 버전의 Cargo 크레이트를 빠르게 설치할 수 있습니다.

`cargo`를 사용하여 CodeArtifact 리포지토리에서 Cargo 크레이트를 설치하려면

1. 아직 구성하지 않았다면 CodeArtifact로 Cargo 구성 및 사용의 단계에 따라 적절한 자격 증명과 함께 CodeArtifact 리포지토리를 사용하도록 `cargo` CLI를 구성합니다.
2. CodeArtifact에서 Cargo 크레이트를 설치하려면 다음 명령을 사용합니다.

   ```
   cargo add my_cargo_package@1.0.0
   ```

   자세한 내용은 Cargo Book에서 cargo add를 참조하세요.

CodeArtifact에 Cargo 크레이트 게시

다음 절차에 따라 `cargo` CLI를 사용하여 Cargo 크레이트를 CodeArtifact 리포지토리에 게시합니다.

1. 아직 구성하지 않았다면 CodeArtifact로 Cargo 구성 및 사용의 단계에 따라 적절한 자격 증명과 함께 CodeArtifact 리포지토리를 사용하도록 `cargo` CLI를 구성합니다.
2. 다음 명령을 사용하여 Cargo 크레이트를 CodeArtifact 리포지토리에 게시합니다.

```
cargo publish
```

자세한 내용은 Cargo Book에서 cargo publish를 참조하세요.

Cargo 명령 지원

다음 섹션에는 지원되지 않는 특정 명령 외에도 CodeArtifact 리포지토리에서 지원하는 Cargo 명령이 요약되어 있습니다.

목차

레지스트리에 액세스해야 하는 지원되는 명령

이 섹션에는 Cargo 클라이언트가 구성된 레지스트리에 액세스해야 하는 Cargo 명령이 나열되어 있습니다. CodeArtifact 리포지토리에 대해 이러한 명령을 간접적으로 호출했을 때 제대로 작동하는 것으로 확인되었습니다.

명령	설명
build	로컬 패키지와 해당 종속성을 빌드합니다.
check	로컬 패키지와 해당 종속성에 오류가 있는지 확인합니다.
fetch	패키지의 종속성을 가져옵니다.
publish	패키지를 레지스트리에 게시합니다.

지원되지 않는 명령

다음 Cargo 명령은 CodeArtifact 리포지토리에서 지원되지 않습니다.

명령	설명	
owner	레지스트리의 크레이트 소유자를 관리합니다.	
search	레지스트리에서 패키지를 검색합니다.	

CodeArtifact를 Maven과 함께 사용

Maven 리포지토리 형식은 Java, Kotlin, Scala, Clojure를 비롯한 다양한 언어에서 사용됩니다. Maven, Gradle, Scala SBT, Apache Ivy, Leiningen을 비롯한 다양한 빌드 도구에서 지원됩니다.

CodeArtifact와 다음 버전과의 호환성을 테스트하고 확인했습니다.

- 최신 Maven 버전: 3.6.3.
- 최신 Gradle 버전: 6.4.1. 5.5.1도 테스트되었습니다.
- 최신 Clojure 버전: 1.11.1도 테스트되었습니다.

주제

Gradle과 함께 CodeArtifact 사용

환경 변수를 사용하여 인증 토큰 전달에 설명된 대로 CodeArtifact 인증 토큰을 환경 변수에 저장한 후, 다음 지침에 따라 CodeArtifact 리포지토리에서 Maven 패키지를 사용하고 새 패키지를 CodeArtifact 리포지토리에 게시합니다.

주제

종속성 가져오기

Gradle 빌드의 CodeArtifact에서 종속 항목을 가져오려면 다음 절차를 사용하세요.

Gradle 빌드의 CodeArtifact에서 종속 항목을 가져오려면

1. CodeArtifact 인증 토큰을 아직 만들지 않았다면 환경 변수를 사용하여 인증 토큰 전달의 절차에 따라 이 인증 토큰을 만들어 환경 변수에 저장하세요.
2. 프로젝트 `build.gradle` 파일의 `repositories` 섹션에 `maven` 섹션을 추가합니다.

```
maven {
         url 'https://my_domain-111122223333.d.codeartifact.region.amazonaws.com/
maven/my_repo/'
         credentials {
             username "aws"
             password System.env.CODEARTIFACT_AUTH_TOKEN
         }
}
```

위 예제에서 `url`은 CodeArtifact 리포지토리의 엔드포인트입니다. Gradle은 엔드포인트를 사용하여 리포지토리에 연결합니다. 샘플에서 `my_domain`은 도메인 이름이고 `111122223333`은 도메인 소유자 ID이며 `my_repo`는 리포지토리 이름입니다. `get-repository-endpoint` AWS CLI 명령을 사용하여 리포지토리의 엔드포인트를 검색할 수 있습니다.

예를 들어 *my_domain*이라는 도메인 내에 *my_repo*라는 리포지토리가 있는 경우, 명령은 다음과 같습니다.

```
aws codeartifact get-repository-endpoint --domain my_domain --domain-
owner 111122223333 --repository my_repo --format maven
```

이 `get-repository-endpoint` 명령은 리포지토리 엔드포인트를 반환합니다.

```
url 'https://my_domain-111122223333.d.codeartifact.region.amazonaws.com/
maven/my_repo/'
```

위 예제의 `credentials` 객체에는 1단계에서 생성한 CodeArtifact 인증 토큰이 포함되어 있습니다. 이 토큰은 Gradle이 CodeArtifact를 인증하는 데 사용합니다.

> **Note**
>
> 듀얼 스택 엔드포인트를 사용하려면 `codeartifact.region.on.aws` 엔드포인트를 사용합니다.

3. (선택 사항) - CodeArtifact 리포지토리를 프로젝트 종속성의 유일한 소스로 사용하려면 `build.gradle`에서 `repositories`의 다른 섹션을 모두 제거합니다. 리포지토리가 두 개 이상인 경우, Gradle은 나열된 순서대로 각 리포지토리에서 종속성을 검색합니다.
4. 리포지토리를 구성한 후, 표준 Gradle 구문을 사용하여 `dependencies` 섹션에 프로젝트 종속성을 추가할 수 있습니다.

```
dependencies {
    implementation 'com.google.guava:guava:27.1-jre'
    implementation 'commons-cli:commons-cli:1.4'
    testImplementation 'org.testng:testng:6.14.3'
}
```

플러그인 가져오기

기본적으로 Gradle은 퍼블릭 Gradle 플러그인 포털에서 플러그인을 확인합니다. CodeArtifact 리포지토리에서 플러그인을 가져오려면 다음 절차를 사용합니다.

CodeArtifact 리포지토리에서 플러그인을 가져오려면

1. CodeArtifact 인증 토큰을 아직 만들지 않았다면 환경 변수를 사용하여 인증 토큰 전달의 절차에 따라 이 인증 토큰을 만들어 환경 변수에 저장하세요.
2. `settings.gradle` 파일에 `pluginManagement` 블록을 추가합니다. `pluginManagement` 블록은 `settings.gradle`에서 다른 문 앞에 나타나야 합니다. 다음 코드 조각을 참조하세요.

```
pluginManagement {
    repositories {
        maven {
            name 'my_repo'
            url
 'https://my_domain-111122223333.d.codeartifact.region.amazonaws.com/
maven/my_repo/'
            credentials {
                username 'aws'
```

```
                password System.env.CODEARTIFACT_AUTH_TOKEN
            }
        }
    }
}
```

이렇게 하면 Gradle이 지정된 리포지토리의 플러그인을 확인할 수 있습니다. 이 리포지토리에는 일반적으로 필요한 Gradle 플러그인을 빌드에서 사용할 수 있도록 Gradle 플러그인 포털(예: `gradle-plugins-store`)에 대한 외부 연결이 있는 업스트림 리포지토리가 있어야 합니다. 자세한 내용은 Gradle 설명서를 참조하세요.

아티팩트 게시

이 섹션에서는 Gradle로 빌드한 Java 라이브러리를 CodeArtifact 리포지토리에 게시하는 방법을 설명합니다.

먼저 프로젝트의 `build.gradle` 파일 중 `plugins` 섹션에 `maven-publish` 플러그인을 추가합니다.

```
plugins {
    id 'java-library'
    id 'maven-publish'
}
```

다음으로 프로젝트 `build.gradle` 파일에 `publishing` 섹션을 추가합니다.

```
publishing {
    publications {
        mavenJava(MavenPublication) {
            groupId = 'group-id'
            artifactId = 'artifact-id'
            version = 'version'
            from components.java
        }
    }
    repositories {
        maven {
            url 'https://my_domain-111122223333.d.codeartifact.region.amazonaws.com/
maven/my_repo/'
            credentials {
                username "aws"
```

```
                password System.env.CODEARTIFACT_AUTH_TOKEN
            }
        }
    }
}
```

maven-publish 플러그인은 publishing 섹션에 지정된 groupId, artifactId, version을 기반으로 POM 파일을 생성합니다.

build.gradle에 대한 이러한 변경이 완료되면 다음 명령을 실행하여 프로젝트를 빌드하여 리포지토리에 업로드합니다.

```
./gradlew publish
```

list-package-versions를 사용하여 패키지가 성공적으로 게시되었는지 확인합니다.

```
aws codeartifact list-package-versions --domain my_domain --domain-owner 111122223333
 --repository my_repo --format maven\
  --namespace com.company.framework --package my-package-name
```

샘플 출력:

```
{
    "format": "maven",
    "namespace": "com.company.framework",
    "package": "example",
    "versions": [
        {
            "version": "1.0",
            "revision": "REVISION-SAMPLE-1-C7F4S5E9B772FC",
            "status": "Published"
        }
    ]
}
```

자세한 내용은 Gradle 웹 사이트에서 이 주제를 참조하세요.

- Java 라이브러리 빌드
- 프로젝트를 모듈로 게시

IntelliJ IDEA에서 Gradle 빌드 실행

CodeArtifact에서 종속성을 가져오는 IntelliJ IDEA에서 Gradle 빌드를 실행할 수 있습니다. CodeArtifact로 인증하려면 Gradle에 CodeArtifact 인증 토큰을 제공해야 합니다. 인증 토큰을 제공하는 방법은 세 가지가 있습니다.

- 방법 1: 인증 토큰을 `gradle.properties`에 저장. `gradle.properties` 파일의 내용을 덮어쓰거나 추가할 수 있는 경우, 이 방법을 사용합니다.
- 방법 2: 인증 토큰을 별도의 파일에 저장. `gradle.properties` 파일을 수정하지 않으려면 이 방법을 사용하세요.
- 방법 3: `build.gradle`에서 `aws`를 인라인 스크립트로 실행하여 실행할 때마다 새 인증 토큰을 생성. 실행할 때마다 Gradle 스크립트가 새 토큰을 가져오도록 하려면 이 방법을 사용하세요. 토큰은 파일 시스템에 저장되지 않습니다.

Token stored in gradle.properties

방법 1: 인증 토큰을 **`gradle.properties`**에 저장.

> **Note**
>
> 예제는 `GRADLE_USER_HOME`에 있는 `gradle.properties` 파일을 보여줍니다.

1. 다음 코드 조각을 사용하여 `build.gradle` 파일을 업데이트합니다.

```
repositories {
    maven {
            url
 'https://my_domain-111122223333.d.codeartifact.region.amazonaws.com/
maven/my_repo/'
            credentials {
                username "aws"
                password "$codeartifactToken"
            }
    }
}
```

2. CodeArtifact에서 플러그인을 가져오려면 `settings.gradle` 파일에 `pluginManagement` 블록을 추가하세요. `pluginManagement` 블록은 `settings.gradle`에서 다른 문 앞에 나타나야 합니다.

```
pluginManagement {
    repositories {
        maven {
            name 'my_repo'
            url
 'https://my_domain-111122223333.codeartifact.region.amazonaws.com/
maven/my_repo/'
            credentials {
                username 'aws'
                password "$codeartifactToken"
            }
        }
    }
}
```

3. CodeArtifact 인증 토큰 가져오기:

```
export CODEARTIFACT_AUTH_TOKEN=`aws codeartifact get-authorization-token --
domain my_domain --domain-owner 111122223333 --query authorizationToken --output
 text --profile profile-name`
```

4. 인증 토큰을 `gradle.properties` 파일에 작성:

```
echo "codeartifactToken=$CODEARTIFACT_AUTH_TOKEN" > ~/.gradle/gradle.properties
```

Token stored in separate file

방법 2: 인증 토큰을 별도의 파일에 저장

1. 다음 코드 조각을 사용하여 `build.gradle` 파일을 업데이트합니다.

```
def props = new Properties()
file("file").withInputStream { props.load(it) }

repositories {

    maven {
```

```
            url
 'https://my_domain-111122223333.d.codeartifact.region.amazonaws.com/
maven/my_repo/'
            credentials {
                username "aws"
                password props.getProperty("codeartifactToken")
            }
    }
}
```

2. CodeArtifact에서 플러그인을 가져오려면 `settings.gradle` 파일에 `pluginManagement` 블록을 추가하세요. `pluginManagement` 블록은 `settings.gradle`에서 다른 문 앞에 나타나야 합니다.

```
pluginManagement {
    def props = new Properties()
    file("file").withInputStream { props.load(it) }
    repositories {
        maven {
            name 'my_repo'
            url
 'https://my_domain-111122223333.codeartifact.region.amazonaws.com/
maven/my_repo/'
            credentials {
                username 'aws'
                password props.getProperty("codeartifactToken")
            }
        }
    }
}
```

3. CodeArtifact 인증 토큰 가져오기:

```
export CODEARTIFACT_AUTH_TOKEN=`aws codeartifact get-authorization-token --
domain my_domain --domain-owner 111122223333 --query authorizationToken --output
 text --profile profile-name`
```

4. `build.gradle` 파일에 지정된 파일에 인증 토큰을 작성:

```
echo "codeartifactToken=$CODEARTIFACT_AUTH_TOKEN" > file
```

Token generated for each run in build.gradle

방법 3: **build.gradle**에서 **aws**를 인라인 스크립트로 실행하여 실행할 때마다 새 인증 토큰을 생성

1. 다음 코드 조각을 사용하여 build.gradle 파일을 업데이트:

```
def codeartifactToken = "aws codeartifact get-authorization-token --
domain my_domain --domain-owner 111122223333 --query authorizationToken --output
 text --profile profile-name".execute().text
    repositories {
        maven {
            url
 'https://my_domain-111122223333.d.codeartifact.region.amazonaws.com/
maven/my_repo/'
            credentials {
                username "aws"
                password codeartifactToken
            }
        }
    }
```

2. CodeArtifact에서 플러그인을 가져오려면 settings.gradle 파일에 pluginManagement 블록을 추가하세요. pluginManagement 블록은 settings.gradle에서 다른 문 앞에 나타나야 합니다.

```
pluginManagement {
    def codeartifactToken = "aws codeartifact get-authorization-token --
domain my_domain --domain-owner 111122223333 --query authorizationToken --output
 text --profile profile-name".execute().text
    repositories {
        maven {
            name 'my_repo'
            url
 'https://my_domain-111122223333.codeartifact.region.amazonaws.com/
maven/my_repo/'
            credentials {
                username 'aws'
                password codeartifactToken
            }
        }
    }
```

```
}
```

mvn과 함께 CodeArtifact 사용

mvn 명령을 사용하여 Maven 빌드를 실행합니다. 이 섹션에서는 CodeArtifact 리포지토리를 사용하도록 mvn을 구성하는 방법에 대해 설명합니다.

주제

- 종속성 가져오기
- 아티팩트 게시
- 타사 아티팩트 게시
- CodeArtifact 리포지토리에서만 Maven 종속성을 다운로드
- Apache Maven 프로젝트 정보

종속성 가져오기

CodeArtifact 리포지토리에서 종속성을 가져오도록 mvn을 구성하려면 Maven 구성 파일, settings.xml 및 프로젝트의 POM(선택 사항)을 편집해야 합니다.

1. 아직 생성 및 저장을 하지 않았다면 CodeArtifact 리포지토리에 대한 인증을 설정하기 위해 환경 변수를 사용하여 인증 토큰 전달에 설명된 대로 환경 변수에 CodeArtifact 인증 토큰을 만들고 저장하십시오.
2. Maven이 HTTP 요청에서 토큰을 전달하도록 settings.xml(대체로 ~/.m2/settings.xml에서 볼 수 있음)에서 CODEARTIFACT_AUTH_TOKEN 환경 변수에 대한 참조가 있는 <servers> 섹션을 추가합니다.

```
<settings>
...
    <servers>
        <server>
            <id>codeartifact</id>
            <username>aws</username>
            <password>${env.CODEARTIFACT_AUTH_TOKEN}</password>
        </server>
    </servers>
...
```

```
</settings>
```

3. <repository> 요소에서 CodeArtifact 리포지토리의 URL 엔드포인트를 추가합니다. settings.xml 또는 프로젝트의 POM 파일에서 이 작업을 수행할 수 있습니다.

   get-repository-endpoint AWS CLI 명령을 사용하여 리포지토리의 엔드포인트를 검색할 수 있습니다.

   예를 들어 *my_domain*이라는 도메인 내에 *my_repo*라는 리포지토리가 있는 경우, 명령은 다음과 같습니다.

   ```
   aws codeartifact get-repository-endpoint --domain my_domain --repository my_repo --
   format maven
   ```

   이 get-repository-endpoint 명령은 리포지토리 엔드포인트를 반환합니다.

   ```
   url 'https://my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/
   maven/my_repo/'
   ```

   > **Note**
   >
   > 듀얼 스택 엔드포인트를 사용하려면 codeartifact.*region*.on.aws 엔드포인트를 사용합니다.

   리포지토리 엔드포인트를 다음과 같이 settings.xml에 추가합니다.

   ```
   <settings>
   ...
       <profiles>
           <profile>
               <id>default</id>
               <repositories>
                   <repository>
                       <id>codeartifact</id>
                       <url>https://my_domain-111122223333.d.codeartifact.us-
   west-2.amazonaws.com/maven/my_repo/</url>
                   </repository>
               </repositories>
           </profile>
   ```

```
    </profiles>
    <activeProfiles>
        <activeProfile>default</activeProfile>
    </activeProfiles>
    ...
</settings>
```

또는 <repositories> 섹션을 프로젝트 POM 파일에 추가하여 해당 프로젝트에만 CodeArtifact를 사용할 수 있습니다.

```
<project>
...
    <repositories>
        <repository>
            <id>codeartifact</id>
            <name>codeartifact</name>
            <url>https://my_domain-111122223333.d.codeartifact.us-
west-2.amazonaws.com/maven/my_repo/</url>
        </repository>
    </repositories>
...
</project>
```

Important

<id> 요소에서 어떤 값이든 사용할 수 있지만 그 값은 <server> 및 <repository> 요소에서 모두 동일해야 합니다. 이렇게 하면 지정된 보안 인증을 CodeArtifact에 대한 요청에 포함할 수 있습니다.

이러한 구성을 변경한 후 프로젝트를 빌드할 수 있습니다.

```
mvn compile
```

Maven은 콘솔에 다운로드한 모든 종속성의 전체 URL을 기록합니다.

```
[INFO] ------------------< com.example.example:myapp >------------------
[INFO] Building myapp 1.0
[INFO] --------------------------------[ jar ]---------------------------------
```

```
Downloading from codeartifact: https://<domain>.d.codeartifact.us-west-2.amazonaws.com/
maven/myrepo/commons-cli/commons-cli/1.4/commons-cli-1.4.pom
Downloaded from codeartifact: https://<domain>.d.codeartifact.us-west-2.amazonaws.com/
maven/myrepo/commons-cli/commons-cli/1.4/commons-cli-1.4.pom (11 kB at 3.9 kB/s)
Downloading from codeartifact: https://<domain>.d.codeartifact.us-west-2.amazonaws.com/
maven/myrepo/org/apache/commons/commons-parent/42/commons-parent-42.pom
Downloading from codeartifact: https://<domain>.d.codeartifact.us-west-2.amazonaws.com/
maven/myrepo/org/apache/commons/commons-parent/42/commons-parent-42.pom
Downloaded from codeartifact: https://<domain>.d.codeartifact.us-west-2.amazonaws.com/
maven/myrepo/org/apache/commons/commons-parent/42/commons-parent-42.pom (68 kB at 123
 kB/s)
Downloading from codeartifact: https://<domain>.d.codeartifact.us-west-2.amazonaws.com/
maven/myrepo/commons-cli/commons-cli/1.4/commons-cli-1.4.jar
Downloaded from codeartifact: https://<domain>.d.codeartifact.us-west-2.amazonaws.com/
maven/myrepo/commons-cli/commons-cli/1.4/commons-cli-1.4.jar (54 kB at 134 kB/s)
```

아티팩트 게시

mvn과 함께 Maven 아티팩트를 CodeArtifact 리포지토리에 게시하려면 ~/.m2/settings.xml 및 프로젝트 POM도 편집해야 합니다.

1. 아직 생성 및 저장을 하지 않았다면 CodeArtifact 리포지토리에 대한 인증을 설정하기 위해 환경 변수를 사용하여 인증 토큰 전달에 설명된 대로 환경 변수에 CodeArtifact 인증 토큰을 만들고 저장하십시오.
2. Maven이 HTTP 요청에서 토큰을 전달하도록 <servers> 섹션을 CODEARTIFACT_AUTH_TOKEN 환경 변수에 대한 참조가 있는 settings.xml에 추가합니다.

   ```
   <settings>
   ...
       <servers>
           <server>
               <id>codeartifact</id>
               <username>aws</username>
               <password>${env.CODEARTIFACT_AUTH_TOKEN}</password>
           </server>
       </servers>
   ...
   </settings>
   ```

3. 프로젝트의 pom.xml에 <distributionManagement> 섹션을 추가합니다.

   ```
   <project>
   ```

```
...
     <distributionManagement>
         <repository>
             <id>codeartifact</id>
             <name>codeartifact</name>
             <url>https://my_domain-111122223333.d.codeartifact.us-
west-2.amazonaws.com/maven/my_repo/</url>
         </repository>
     </distributionManagement>
...
</project>
```

이러한 구성을 변경한 후 프로젝트를 빌드하여 지정된 리포지토리에 게시할 수 있습니다.

```
mvn deploy
```

`list-package-versions`를 사용하여 패키지가 성공적으로 게시되었는지 확인합니다.

```
aws codeartifact list-package-versions --domain my_domain --domain-owner 111122223333
 --repository my_repo --format maven \
  --namespace com.company.framework --package my-package-name
```

샘플 출력:

```
{
    "defaultDisplayVersion": null,
    "format": "maven",
    "namespace": "com.company.framework",
    "package": "my-package-name",
    "versions": [
        {
            "version": "1.0",
            "revision": "REVISION-SAMPLE-1-C7F4S5E9B772FC",
            "status": "Published"
        }
    ]
}
```

타사 아티팩트 게시

`mvn deploy:deploy-file`을 사용하여 타사 Maven 아티팩트를 CodeArtifact 리포지토리에 게시할 수 있습니다. 이는 JAR 파일만 가지고 있고 패키지 소스 코드나 POM 파일에는 액세스할 수 없는 사용자가 아티팩트를 게시하려고 할 때 도움이 될 수 있습니다.

이 `mvn deploy:deploy-file` 명령을 실행하면 명령줄에 전달된 정보를 기반으로 POM 파일이 생성됩니다.

타사 Maven 아티팩트 게시

1. 아직 생성 및 저장을 하지 않았다면 CodeArtifact 리포지토리에 대한 인증을 설정하기 위해 환경 변수를 사용하여 인증 토큰 전달에 설명된 대로 환경 변수에 CodeArtifact 인증 토큰을 만들고 저장하십시오.
2. 다음 콘텐츠가 포함된 `~/.m2/settings.xml` 파일을 생성합니다.

```
<settings>
    <servers>
        <server>
            <id>codeartifact</id>
            <username>aws</username>
            <password>${env.CODEARTIFACT_AUTH_TOKEN}</password>
        </server>
    </servers>
</settings>
```

3. `mvn deploy:deploy-file` 명령을 실행합니다.

```
mvn deploy:deploy-file -DgroupId=commons-cli          \
-DartifactId=commons-cli       \
-Dversion=1.4                  \
-Dfile=./commons-cli-1.4.jar   \
-Dpackaging=jar                \
-DrepositoryId=codeartifact    \
-Durl=https://my_domain-111122223333.d.codeartifact.region.amazonaws.com/
maven/repo-name/
```

> **Note**
>
> 위 예시는 `commons-cli 1.4`를 게시합니다. groupId, artifactID, 버전 및 파일 인수를 수정하여 다른 JAR을 게시합니다.

이 지침은 Apache Maven 설명서의 타사 JAR을 원격 리포지토리에 배포하기 위한 가이드의 예제를 기반으로 합니다.

CodeArtifact 리포지토리에서만 Maven 종속성을 다운로드

구성된 리포지토리에서 패키지를 가져올 수 없는 경우, 기본적으로 `mvn` 명령을 실행하면 Maven Central에서 패키지를 가져옵니다. `mvn`이 CodeArtifact 리포지토리를 항상 사용할 수 있도록 `mirrors` 요소를 `settings.xml`에 추가합니다.

```
<settings>
  ...
    <mirrors>
      <mirror>
        <id>central-mirror</id>
        <name>CodeArtifact Maven Central mirror</name>
        <url>https://my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/
maven/my_repo/</url>
        <mirrorOf>central</mirrorOf>
      </mirror>
    </mirrors>
  ...
</settings>
```

`mirrors` 요소를 추가하는 경우 `settings.xml` 또는 `pom.xml`에도 `pluginRepository` 요소가 있어야 합니다. 다음 예제는 CodeArtifact 리포지토리에서 애플리케이션 종속성과 Maven 플러그인을 가져옵니다.

```
<settings>
...
  <profiles>
    <profile>
      <pluginRepositories>
        <pluginRepository>
          <id>codeartifact</id>
```

```
          <name>CodeArtifact Plugins</name>
          <url>https://my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/
maven/my_repo/</url>
          <releases>
            <enabled>true</enabled>
          </releases>
          <snapshots>
            <enabled>true</enabled>
          </snapshots>
        </pluginRepository>
      </pluginRepositories>
    </profile>
  </profiles>
...
</settings>
```

다음 예제는 CodeArtifact 리포지토리에서 애플리케이션 종속성을 가져오며 Maven Central에서 Maven 플러그인을 가져옵니다.

```
<profiles>
   <profile>
     <id>default</id>
     ...
     <pluginRepositories>
       <pluginRepository>
         <id>central-plugins</id>
         <name>Central Plugins</name>
         <url>https://repo.maven.apache.org/maven2/</url>
         <releases>
             <enabled>true</enabled>
         </releases>
         <snapshots>
             <enabled>true</enabled>
         </snapshots>
       </pluginRepository>
     </pluginRepositories>
   ....
   </profile>
 </profiles>
```

Apache Maven 프로젝트 정보

Maven에 관한 자세한 내용은 Apache Maven Project 웹 사이트에서 다음 주제를 참조하세요.

- 여러 리포지토리 설정
- 설정 참조
- 배포 관리
- 프로파일

CodeArtifact를 deps.edn과 함께 사용

deps.edn을 clj와 함께 사용하여 Clojure 프로젝트의 종속성을 관리할 수 있습니다. 이 섹션에서는 CodeArtifact 리포지토리를 사용하도록 deps.edn을 구성하는 방법에 대해 설명합니다.

주제

- 종속성 가져오기
- 아티팩트 게시

종속성 가져오기

CodeArtifact 리포지토리에서 종속성을 가져오도록 Clojure를 구성하려면 Maven 구성 파일 settings.xml을 편집해야 합니다.

1. Clojure가 HTTP 요청에서 토큰을 전달하도록 settings.xml에서 CODEARTIFACT_AUTH_TOKEN 환경 변수에 대한 참조가 있는 <servers> 섹션을 추가합니다.

> **Note**
>
> Clojure는 settings.xml 파일이 ~/.m2/settings.xml에 있을 것으로 예상합니다. 다른 곳에 있다면 이 위치에 파일을 생성하세요.

```
<settings>
...
    <servers>
        <server>
            <id>codeartifact</id>
            <username>aws</username>
            <password>${env.CODEARTIFACT_AUTH_TOKEN}</password>
        </server>
    </servers>
```

```
...
</settings>
```

2. 아직 파일이 없는 경우, `clj -Spom`을 사용하여 프로젝트에 대해 POM.xml을 생성합니다.
3. `deps.edn` 구성 파일에 Maven `settings.xml`의 서버 ID와 일치하는 리포지토리를 추가합니다.

```
:mvn/repos {
  "clojars" nil
  "central" nil
  "codeartifact" {:url "https://my_domain-111122223333.d.codeartifact.us-
west-2.amazonaws.com/maven/my_repo/"}
}
```

> **Note**
>
> - `tools.deps`를 실행하면 `central` 및 `clojars` 리포지토리에서 Maven 라이브러리를 먼저 확인하게 됩니다. 그런 다음, `deps.edn`에 나열된 다른 리포지토리도 검사됩니다.
> - Clojars와 Maven Central에서 직접 다운로드하는 것을 방지하려면 `central`과 `clojars`를 `nil`로 설정해야 합니다.

환경 변수에 CodeArtifact 인증 토큰이 있는지 확인합니다(환경 변수를 사용하여 인증 토큰 전달 참조). 이러한 변경 후 패키지를 빌드하면 CodeArtifact에서 `deps.edn`의 종속성을 가져옵니다.

> **Note**
>
> 듀얼 스택 엔드포인트를 사용하려면 `codeartifact.region.on.aws` 엔드포인트를 사용합니다.

아티팩트 게시

1. CodeArtifact를 Maven이 인식하는 서버로 포함하도록 Maven 설정과 `deps.edn`을 업데이트합니다(종속성 가져오기 참조). deps-deploy 같은 도구를 사용하여 CodeArtifact에 아티팩트를 업로드할 수 있습니다.
2. `build.clj`에서 이전에 설정한 `codeartifact` 리포지토리에 필수 아티팩트를 업로드하는 `deploy` 작업을 추가합니다.

```
(ns build
(:require [deps-deploy.deps-deploy :as dd]))

(defn deploy [_]
  (dd/deploy {:installer :remote
          :artifact "PATH_TO_JAR_FILE.jar"
          :pom-file "pom.xml" ;; pom containing artifact coordinates
          :repository "codeartifact"}))
```

3. `clj -T:build deploy` 명령을 실행하여 아티팩트를 게시합니다.

기본 리포지토리 수정에 관한 자세한 내용은 Clojure Deps 및 CLI 참조 근거에서 기본 리포지토리 수정을 참조합니다.

curl을 사용한 게시

이 섹션에서는 HTTP 클라이언트 `curl`을 사용하여 CodeArtifact 리포지토리에 Maven 아티팩트를 게시하는 방법을 보여줍니다. 환경에 Maven 클라이언트가 없거나 이 클라이언트를 설치하려는 경우, `curl`을 사용하여 아티팩트를 게시하는 것이 도움이 될 수 있습니다.

`curl`을 사용하여 Maven 아티팩트를 게시

1. 환경 변수를 사용하여 인증 토큰 전달의 단계에 따라 CodeArtifact 인증 토큰을 가져오고 이 단계로 돌아갑니다.
2. 다음 `curl` 명령을 사용하여 JAR을 CodeArtifact 리포지토리에 게시합니다.

   이 절차의 각 `curl` 명령에서 다음 자리 표시자를 변경합니다.

   - *`my_domain`*을 CodeArtifact 도메인 이름으로 변경합니다.
   - *`111122223333`*을 CodeArtifact 도메인 소유자의 ID로 변경합니다.
   - *`us-west-2`*를 CodeArtifact 도메인이 위치한 리전으로 변경합니다.
   - *`my_repo`*를 CodeArtifact 리포지토리 이름으로 변경합니다.

```
curl --request PUT https://my_domain-111122223333.d.codeartifact.us-
west-2.amazonaws.com/maven/my_repo/com/mycompany/app/my-app/1.0/my-app-1.0.jar \
     --user "aws:$CODEARTIFACT_AUTH_TOKEN" --header "Content-Type: application/
octet-stream" \
```

```
    --data-binary @my-app-1.0.jar
```

> **Important**
>
> `--data-binary` 파라미터 값 앞에 문자 `@`을 붙여야 합니다. 값을 따옴표로 묶을 때는 따옴표 안에 `@`을 포함해야 합니다.

3. 다음 `curl` 명령을 사용하여 POM을 CodeArtifact 리포지토리에 게시합니다.

```
curl --request PUT https://my_domain-111122223333.d.codeartifact.us-
west-2.amazonaws.com/maven/my_repo/com/mycompany/app/my-app/1.0/my-app-1.0.pom \
    --user "aws:$CODEARTIFACT_AUTH_TOKEN" --header "Content-Type: application/
octet-stream" \
    --data-binary @my-app-1.0.pom
```

4. 이때 Maven 아티팩트는 상태가 `Unfinished`인 CodeArtifact 리포지토리에 저장됩니다. 패키지를 사용하려면 패키지가 `Published` 상태에 있어야 합니다. 패키지에 `maven-metadata.xml` 파일을 업로드하거나 UpdatePackageVersionsStatus API를 직접적으로 호출하여 상태를 변경함으로써 패키지를 `Unfinished`에서 `Published`로 이동할 수 있습니다.

   a. 옵션 1: 다음 `curl` 명령을 사용하여 패키지에 `maven-metadata.xml` 파일을 추가합니다.

```
curl --request PUT
 https://my_domain-111122223333.d.codeartifact.region.amazonaws.com/
maven/my_repo/com/mycompany/app/my-app/maven-metadata.xml \
    --user "aws:$CODEARTIFACT_AUTH_TOKEN" --header "Content-Type: application/
octet-stream" \
    --data-binary @maven-metadata.xml
```

   다음은 `maven-metadata.xml` 파일의 내용을 보여주는 예제입니다.

```
<metadata modelVersion="1.1.0">
    <groupId>com.mycompany.app</groupId>
    <artifactId>my-app</artifactId>
    <versioning>
        <latest>1.0</latest>
        <release>1.0</release>
        <versions>
            <version>1.0</version>
        </versions>
        <lastUpdated>20200731090423</lastUpdated>
```

```
    </versioning>
</metadata>
```

b. 옵션 2: UpdatePackageVersionsStatus API를 사용하여 패키지 상태를 Published로 업데이트합니다.

```
aws codeartifact update-package-versions-status \
    --domain my_domain \
    --domain-owner 111122223333 \
    --repository my_repo \
    --format maven \
    --namespace com.mycompany.app \
    --package my-app \
    --versions 1.0 \
    --target-status Published
```

아티팩트의 JAR 파일만 있는 경우 mvn을 사용하여 CodeArtifact 리포지토리에 사용 가능한 패키지 버전을 게시할 수 있습니다. 이는 아티팩트의 소스 코드나 POM에 액세스할 수 없는 경우에 도움이 될 수 있습니다. 세부 정보는 타사 아티팩트 게시 섹션을 참조하세요.

Maven 체크섬 사용

Maven 아티팩트가 an AWS CodeArtifact 리포지토리에 게시되면 패키지의 각 자산 또는 파일과 연결된 체크섬이 업로드를 검증하는 데 사용됩니다. 자산의 예로는 jar, pom, war 파일 등이 있습니다. 각 자산의 경우, md5 또는 sha1 같은 추가 확장자가 있는 자산 이름을 사용하는 여러 개의 체크섬 파일이 Maven 아티팩트에 포함되어 있습니다. 예를 들어, my-maven-package.jar라는 이름이 지정된 파일의 체크섬 파일은 my-maven-package.jar.md5 및 my-maven-package.jar.sha1일 수 있습니다.

> **Note**
>
> Maven은 artifact라는 용어를 사용합니다. 이 가이드에서 Maven 패키지는 Maven 아티팩트와 동일합니다. 자세한 내용은 AWS CodeArtifact 패키지를 참조하세요.

체크섬 스토리지

CodeArtifact는 Maven 체크섬을 자산으로 저장하지 않습니다. 즉, ListPackageVersionAssets API의 출력에서는 체크섬이 개별 자산으로 표시되지 않습니다. 그 대신 CodeArtifact로 계산된 체크섬은 지

원되는 모든 체크섬 유형의 각 자산에 사용할 수 있습니다. 예를 들어, Maven 패키지 버전 commons-lang:commons-lang 2.1에서 ListPackageVersionAssets를 직접적으로 호출할 때의 응답 중 일부는 다음과 같습니다.

```
{
    "name": "commons-lang-2.1.jar",
    "size": 207723,
    "hashes": {
        "MD5": "51591549f1662a64543f08a1d4a0cf87",
        "SHA-1": "4763ecc9d78781c915c07eb03e90572c7ff04205",
        "SHA-256": "2ded7343dc8e57decd5e6302337139be020fdd885a2935925e8d575975e480b9",
        "SHA-512":
 "a312a5e33b17835f2e82e74ab52ab81f0dec01a7e72a2ba58bb76b6a197ffcd2bb410e341ef7b3720f3b595ce49fd
    }
},
{
    "name": "commons-lang-2.1.pom",
    "size": 9928,
    "hashes": {
        "MD5": "8e41bacdd69de9373c20326d231c8a5d",
        "SHA-1": "a34d992202615804c534953aba402de55d8ee47c",
        "SHA-256": "f1a709cd489f23498a0b6b3dfbfc0d21d4f15904791446dec7f8a58a7da5bd6a",
        "SHA-512":
 "1631ce8fe4101b6cde857f5b1db9b29b937f98ba445a60e76cc2b8f2a732ff24d19b91821a052c1b56b73325104e9
    }
},
        {
    "name": "maven-metadata.xml",
    "size": 121,
    "hashes": {
        "MD5": "11bb3d48d984f2f49cea1e150b6fa371",
        "SHA-1": "7ef872be17357751ce65cb907834b6c5769998db",
        "SHA-256": "d04d140362ea8989a824a518439246e7194e719557e8d701831b7f5a8228411c",
        "SHA-512":
 "001813a0333ce4b2a47cf44900470bc2265ae65123a8c6b5ac5f2859184608596baa4d8ee0696d0a497755dade0f6
    }
}
```

체크섬이 자산으로 저장되지는 않지만 Maven 클라이언트는 여전히 예상 위치에 체크섬을 게시하고 다운로드할 수 있습니다. 예를 들어, commons-lang:commons-lang 2.1이 maven-repo라는 이름의 리포지토리에 있는 경우, JAR 파일의 SHA-256 체크섬 URL 경로는 다음과 같습니다.

```
/maven/maven-repo/commons-lang/commons-lang/2.1/commons-lang-2.1.jar.sha256
```

curl과 같은 일반 HTTP 클라이언트를 사용하여 기존 Maven 패키지(예: 이전에 Amazon S3에 저장된 패키지)를 CodeArtifact에 업로드하는 경우, 체크섬을 업로드할 필요가 없습니다. CodeArtifact는 체크섬을 자동으로 생성합니다. 자산이 올바르게 업로드되었는지 확인하려는 경우, ListPackageVersionAssets API 작업을 사용하여 각 자산의 원래 체크섬 값에 대한 응답의 체크섬을 비교할 수 있습니다.

게시 중 체크섬 불일치

자산과 체크섬 외에도 Maven 아티팩트에는 maven-metadata.xml 파일도 포함되어 있습니다. Maven 패키지의 일반적인 게시 순서는 모든 자산과 체크섬을 먼저 업로드한 후 maven-metadata.xml을 업로드하는 것입니다. 예를 들어, 클라이언트가 SHA-256 체크섬 파일을 게시하도록 구성되었다고 할 때 앞서 commons-lang 2.1에 설명된 Maven 패키지 버전의 게시 순서는 다음과 같습니다.

```
PUT commons-lang-2.1.jar
PUT commons-lang-2.1.jar.sha256
PUT commons-lang-2.1.pom
PUT commons-lang-2.1.pom.sha256
PUT maven-metadata.xml
PUT maven-metadata.xml.sha256
```

자산에 대한 체크섬 파일(예:JAR 파일)을 업로드할 때 업로드된 체크섬 값과 CodeArtifact에서 계산한 체크섬 값이 일치하지 않으면 체크섬 업로드 요청이 실패하며 400(잘못된 요청) 응답이 함께 표시됩니다. 해당 자산이 없는 경우, 요청은 실패하며 404(찾을 수 없음) 응답이 함께 표시됩니다. 이 오류를 방지하려면 먼저 자산을 업로드한 후 체크섬을 업로드해야 합니다.

maven-metadata.xml이 업로드되면 CodeArtifact는 일반적으로 Maven 패키지 버전의 상태를 Unfinished에서 Published로 변경합니다. 자산에 대해 체크섬 불일치가 감지되면 CodeArtifact는 maven-metadata.xml 게시 요청에 대한 응답으로 400(잘못된 요청)을 반환합니다. 이 오류로 인해 클라이언트가 해당 패키지 버전에 대한 파일 업로드를 중단할 수 있습니다. 이러한 문제가 발생하여 maven-metadata.xml 파일이 업로드되지 않을 경우, 이미 업로드된 패키지 버전의 자산을 다운로드할 수 없습니다. 이는 패키지 버전의 상태가 Published로 설정되지 않고 Unfinished로 계속 유지되기 때문입니다.

CodeArtifact를 사용하면 maven-metadata.xml이 업로드되고 패키지 버전 상태가 Published로 설정된 후에도 Maven 패키지 버전에 자산을 더 추가할 수 있습니다. 이 상태에서는 불일치한 체크섬 파일을 업로드하라는 요청도 실패하며 400(잘못된 요청) 응답이 함께 표시됩니다. 하지만 패키지 버전 상

태가 이미 Published로 설정되었으므로 체크섬 파일 업로드가 실패한 자산을 포함하여 패키지에서 모든 자산을 다운로드할 수 있습니다. 체크섬 파일 업로드가 실패한 자산의 체크섬을 다운로드할 때 클라이언트가 수신하는 체크섬 값은 업로드된 자산 데이터를 기반으로 CodeArtifact에서 계산한 체크섬 값입니다.

CodeArtifact 체크섬 비교는 대소문자를 구분하며 CodeArtifact로 계산된 체크섬은 소문자 형식으로 표시됩니다. 따라서 909FA780F76DA393E992A3D2D495F468 체크섬이 업로드되면 CodeArtifact가 이를 909fa780f76da393e992a3d2d495f468과 같은 것으로 처리하지 않기 때문에 체크섬 불일치로 실패합니다.

체크섬 불일치 복구하기

체크섬 불일치로 인해 체크섬 업로드가 실패하는 경우, 다음 중 하나를 시도하여 복구합니다.

- Maven 아티팩트를 다시 게시하는 명령을 실행합니다. 네트워크 문제로 인해 체크섬 파일이 손상된 경우, 이러한 명령 실행이 효과적일 수 있습니다. 이러한 방법으로 네트워크 문제가 해결되면 체크섬이 일치하여 다운로드가 완료됩니다.
- 패키지 버전을 삭제한 후 다시 게시합니다. 자세한 내용은 AWS CodeArtifact API 참조의 DeletePackageVersions를 참조하세요.

Maven 스냅샷 사용

Maven 스냅샷은 최신 프로덕션 브랜치 코드를 참조하는 Maven 패키지의 특수 버전입니다. 이 스냅샷은 최종 릴리스 버전보다 앞서는 개발 버전입니다. 패키지 버전에 추가된 접미사 SNAPSHOT으로 Maven 패키지의 스냅샷 버전을 식별할 수 있습니다. 예를 들어, 버전 1.1의 스냅샷은 1.1-SNAPSHOT입니다. 자세한 내용은 Apache Maven 프로젝트 웹사이트에서 SNAPSHOT 버전이란 무엇입니까?를 참조하세요.

AWS CodeArtifact는 Maven 스냅샷 게시 및 사용을 지원합니다. 지원되는 스냅샷은 시간 기반 버전 번호를 사용하는 고유 스냅샷 밖에 없습니다. CodeArtifact는 Maven 2 클라이언트에서 생성된 고유하지 않은 스냅샷을 지원하지 않습니다. 지원되는 Maven 스냅샷을 모든 CodeArtifact 리포지토리에 게시할 수 있습니다.

주제

- CodeArtifact에서의 스냅샷 게시
- 스냅샷 버전 사용
- 스냅샷 버전 삭제

CodeArtifact에서의 스냅샷 게시

AWS CodeArtifact는 스냅샷을 게시할 때와 같은 클라이언트가 mvn사용하는 요청 패턴을 지원합니다. 따라서 Maven 스냅샷이 게시되는 방식을 자세히 이해하지 못했더라도 빌드 도구 또는 패키지 관리자의 설명서를 따르면 됩니다. 좀 더 복잡한 작업을 수행하는 경우, 이 섹션에서는 CodeArtifact가 스냅샷을 처리하는 방법을 자세히 설명합니다.

Maven 스냅샷이 CodeArtifact 리포지토리에 게시되면 이전 버전이 빌드라는 새 버전에 보존됩니다. Maven 스냅샷이 게시될 때마다 새 빌드 버전이 생성됩니다. 스냅샷의 모든 이전 버전은 빌드 버전에서 유지 관리됩니다. Maven 스냅샷이 게시되면 패키지 버전 상태가 Published로 설정되고 이전 버전이 포함된 빌드의 상태가 Unlisted로 설정됩니다. 이 동작은 패키지 버전에 접미사 -SNAPSHOT이 있는 Maven 패키지 버전에만 적용됩니다.

예를 들어 com.mycompany.myapp:pkg-1이라는 Maven 패키지의 스냅샷 버전이 my-maven-repo라는 CodeArtifact 리포지토리에 업로드됩니다. 스냅샷 버전은 1.0-SNAPSHOT입니다. 지금까지 게시된 com.mycompany.myapp:pkg-1의 버전은 없습니다. 먼저, 초기 빌드의 자산은 다음 경로에 게시됩니다.

```
PUT maven/my-maven-repo/com/mycompany/myapp/pkg-1/1.0-SNAPSHOT/
pkg-1-1.0-20210728.194552-1.jar
PUT maven/my-maven-repo/com/mycompany/myapp/pkg-1/1.0-SNAPSHOT/
pkg-1-1.0-20210728.194552-1.pom
```

참고로 20210728.194552-1 타임스탬프는 스냅샷 빌드를 게시하는 클라이언트에 의해 생성됩니다.

.pom 및 .jar 파일을 업로드한 후 리포지토리에 존재하는 com.mycompany.myapp:pkg-1의 유일한 버전은 1.0-20210728.194552-1입니다. 이는 이전 경로에 지정된 버전이 1.0-SNAPSHOT인 경우에도 발생합니다. 현재 패키지 버전 상태는 Unfinished입니다.

```
aws codeartifact list-package-versions --domain my-domain --repository \
  my-maven-repo --package pkg-1 --namespace com.mycompany.myapp --format maven
{
    "versions": [
        {
```

```
            "version": "1.0-20210728.194552-1",
            "revision": "GipMW+599JmwTcTLaXo9YvDsVQ2bcrrk/02rWJhoKUU=",
            "status": "Unfinished"
        }
    ],
    "defaultDisplayVersion": null,
    "format": "maven",
    "package": "pkg-1",
    "namespace": "com.mycompany.myapp"
}
```

그런 다음, 클라이언트가 패키지 버전용 maven-metadata.xml 파일을 업로드합니다.

```
PUT my-maven-repo/com/mycompany/myapp/pkg-1/1.0-SNAPSHOT/maven-metadata.xml
```

maven-metadata.xml 파일이 성공적으로 업로드되면 CodeArtifact는 1.0-SNAPSHOT 패키지 버전을 생성하고 1.0-20210728.194552-1 버전을 Unlisted로 설정합니다.

```
aws codeartifact list-package-versions --domain my-domain --repository \
  my-maven-repo --package pkg-1 --namespace com.mycompany.myapp --format maven
{
    "versions": [
        {
            "version": "1.0-20210728.194552-1",
            "revision": "GipMW+599JmwTcTLaXo9YvDsVQ2bcrrk/02rWJhoKUU=",
            "status": "Unlisted"
        },
        {
            "version": "1.0-SNAPSHOT",
            "revision": "tWu8n3IX5HR82vzVZQAxlwcvvA4U/+S80edWNAkil24=",
            "status": "Published"
        }
    ],
    "defaultDisplayVersion": "1.0-SNAPSHOT",
    "format": "maven",
    "package": "pkg-1",
    "namespace": "com.mycompany.myapp"
}
```

이제 스냅샷 버전 1.0-SNAPSHOT을 빌드에서 사용할 수 있습니다. 리포지토리 my-maven-repo에는 두 가지 버전의 com.mycompany.myapp:pkg-1이 있지만 두 버전 모두 동일한 자산을 포함하고 있습니다.

```
aws codeartifact list-package-version-assets --domain my-domain --repository \
  my-maven-repo --format maven --namespace com.mycompany.myapp \
 --package pkg-1 --package-version 1.0-SNAPSHOT--query 'assets[*].name'
[
     "pkg-1-1.0-20210728.194552-1.jar",
     "pkg-1-1.0-20210728.194552-1.pom"
]
```

`--package-version` 파라미터를 `1.0-20210728.194552-1`로 변경하여 이전에 표시된 것과 동일한 `list-package-version-assets` 명령을 실행하면 동일한 출력이 표시됩니다.

`1.0-SNAPSHOT`의 추가 빌드가 리포지토리에 추가되면 새 빌드마다 새 `Unlisted` 패키지 버전이 생성됩니다. 버전 `1.0-SNAPSHOT`의 자산은 매번 업데이트되므로 버전은 항상 해당 버전의 최신 빌드를 참조합니다. 최신 자산으로 `1.0-SNAPSHOT`을 업데이트하는 절차는 새 빌드용 `maven-metadata.xml` 파일을 업로드하는 것으로 시작됩니다.

스냅샷 버전 사용

스냅샷을 요청하면 상태 `Published`의 버전이 반환됩니다. 이 버전은 항상 Maven 스냅샷의 최신 버전입니다. URL 경로의 스냅샷 버전(예: `1.0-SNAPSHOT`) 대신 빌드 버전 번호(예: `1.0-20210728.194552-1`)를 사용하여 스냅샷의 특정 빌드를 요청할 수도 있습니다. Maven 스냅샷의 빌드 버전을 보려면 CodeArtifact API 안내서의 ListPackageVersions API를 사용하고 상태 파라미터를 `Unlisted`로 설정합니다.

스냅샷 버전 삭제

Maven 스냅샷의 모든 빌드 버전을 삭제하려면 DeletePackageVersions API를 사용하여 삭제하려는 버전을 지정합니다.

curl을 사용한 스냅샷 게시

Amazon Simple Storage Service(Amazon S3) 또는 다른 아티팩트 리포지토리 제품에 저장된 기존 스냅샷 버전이 있는 경우 이를 AWS CodeArtifact에 다시 게시할 수 있습니다. CodeArtifact가 Maven 스냅샷을 지원하는 방식 때문에(CodeArtifact에서의 스냅샷 게시 참조) 일반 HTTP 클라이언트(예: `curl`)를 사용하여 스냅샷을 게시하는 것은 curl을 사용한 게시에 설명된 대로 Maven 릴리스 버전을 게시하는 것보다 더 복잡합니다. `mvn` 또는 `gradle` 같은 Maven 클라이언트를 사용하여 스냅샷 버전을 빌드하고 배포하는 경우, 이 섹션은 관련성이 없습니다. Maven 클라이언트에 대한 설명서를 따라야 합니다.

스냅샷 버전을 게시하려면 스냅샷 버전의 빌드를 하나 이상 게시해야 합니다. CodeArtifact에서 스냅샷 버전의 빌드가 n개 이상 있는 경우, CodeArtifact 버전이 n+1개 즉, 상태가 `Unlisted`인 빌드 버전이 n개, 상태가 `Published`인 스냅샷 버전 하나(가장 최근에 게시된 빌드)가 있습니다. 스냅샷 버전(즉, 버전 문자열에 '-SNAPSHOT'이 포함된 버전)에는 최근에 게시된 빌드와 동일한 자산 세트가 포함되어 있습니다. `curl`을 사용하여 이 구조를 생성하는 가장 간단한 방법은 다음과 같습니다.

1. `curl`을 사용하여 모든 빌드의 모든 자산을 게시합니다.
2. `curl`을 사용하여 마지막 빌드의 `maven-metadata.xml` 파일(즉, 가장 최근의 날짜-시간 스탬프가 있는 빌드)을 게시합니다. 그러면 버전 문자열에 '`-SNAPSHOT`' 이 포함되고 올바른 자산 세트가 포함된 버전이 하나 생성됩니다.
3. UpdatePackageVersionsStatus API를 사용하여 최신 버전이 아닌 모든 빌드 버전의 상태를 `Unlisted`로 설정합니다.

다음 `curl` 명령을 사용하여 패키지 `com.mycompany.app:pkg-1`의 스냅샷 버전 `1.0-SNAPSHOT`에 대한 스냅샷 자산(예: .jar 및 .pom 파일)을 게시하세요.

```
curl --user "aws:$CODEARTIFACT_AUTH_TOKEN" -H "Content-Type: application/octet-stream"
 \
     -X PUT https://my_domain-111122223333.d.codeartifact.us-
west-2.amazonaws.com/maven/my_maven_repo/com/mycompany/app/pkg-1/1.0-SNAPSHOT/
pkg-1-1.0-20210729.171330-2.jar \
     --data-binary @pkg-1-1.0-20210728.194552-1.jar
```

```
curl --user "aws:$CODEARTIFACT_AUTH_TOKEN" -H "Content-Type: application/octet-stream"
 \
     -X PUT https://my_domain-111122223333.d.codeartifact.us-
west-2.amazonaws.com/maven/my_maven_repo/com/mycompany/app/pkg-1/1.0-SNAPSHOT/
pkg-1-1.0-20210729.171330-2.pom \
     --data-binary @pkg-1-1.0-20210728.194552-1.pom
```

다음 예제를 사용하는 경우:

- *my_domain*을 CodeArtifact 도메인 이름으로 변경합니다.
- *111122223333*을 CodeArtifact 도메인 소유자의 AWS 계정 ID로 바꿉니다.
- *us-west-2*를 CodeArtifact 도메인이 위치한 AWS 리전 로 바꿉니다.
- *my_maven_repo*를 CodeArtifact 리포지토리 이름으로 변경합니다.

> **Important**
>
> `--data-binary` 파라미터의 값 앞에 문자 `@`을 붙여야 합니다. 값을 따옴표로 묶을 때는 따옴표 안에 `@`을 포함해야 합니다.

각 빌드에 업로드할 자산이 두 개 이상 존재할 수 있습니다. 예를 들어, 기본 JAR 및 `pom.xml` 외에도 Javadoc 및 소스 JAR 파일이 존재할 수 있습니다. CodeArtifact는 업로드된 각 자산에 대해 체크섬을 자동으로 생성하므로 패키지 버전 자산에 대한 체크섬 파일을 게시할 필요가 없습니다. 자산이 제대로 업로드되었는지 확인하려면 `list-package-version-assets` 명령을 사용하여 생성된 체크섬을 가져와서 원본 체크섬과 비교합니다. CodeArtifact가 Maven 체크섬을 처리하는 방법에 관한 자세한 내용은 Maven 체크섬 사용 섹션을 참조하세요.

다음 curl 명령어를 사용하여 최신 빌드 버전용 `maven-metadata.xml` 파일을 게시하세요.

```
curl --user "aws:$CODEARTIFACT_AUTH_TOKEN" -H "Content-Type: application/octet-stream"
 \
     -X PUT https://my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/
maven/my_maven_repo/com/mycompany/app/pkg-1/1.0-SNAPSHOT/maven-metadata.xml \
     --data-binary @maven-metadata.xml
```

`maven-metadata.xml` 파일은 `<snapshotVersions>` 요소의 최신 빌드 버전에 있는 자산 중 한 가지 이상을 참조해야 합니다. 또한 `<timestamp>` 값이 있어야 하며 이 값은 자산 파일 이름의 타임스탬프와 일치해야 합니다. 예를 들어, 이전에 게시된 `20210729.171330-2` 빌드의 경우 `maven-metadata.xml`의 내용은 다음과 같습니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<metadata>
  <groupId>com.mycompany.app</groupId>
  <artifactId>pkg-1</artifactId>
  <version>1.0-SNAPSHOT</version>
  <versioning>
    <snapshot>
      <timestamp>20210729.171330</timestamp>
      <buildNumber>2</buildNumber>
    </snapshot>
    <lastUpdated>20210729171330</lastUpdated>
    <snapshotVersions>
      <snapshotVersion>
        <extension>jar</extension>
```

```
        <value>1.0-20210729.171330-2</value>
        <updated>20210729171330</updated>
      </snapshotVersion>
      <snapshotVersion>
        <extension>pom</extension>
        <value>1.0-20210729.171330-2</value>
        <updated>20210729171330</updated>
      </snapshotVersion>
    </snapshotVersions>
  </versioning>
</metadata>
```

maven-metadata.xml이 게시된 후 마지막 단계는 기타 모든 빌드 버전(즉, 최신 빌드를 제외한 모든 빌드 버전)의 패키지 버전 상태를 Unlisted로 설정하는 데 그 목적이 있습니다. 예를 들어, 1.0-SNAPSHOT 버전에 두 개의 빌드가 있고 첫 번째 빌드가 20210728.194552-1인 경우 해당 빌드를 Unlisted로 설정하는 명령은 다음과 같습니다.

```
aws codeartifact update-package-versions-status --domain my-domain --domain-owner
 111122223333 \
   --repository my-maven-repo --format maven --namespace com.mycompany.app --package
 pkg-1 \
   --versions 1.0-20210728.194552-1 --target-status Unlisted
```

스냅샷과 외부 연결

Maven 스냅샷은 외부 연결을 통해 Maven 퍼블릭 리포지토리에서 가져올 수 없습니다. AWS CodeArtifact는 Maven 릴리스 버전 가져오기만 지원합니다.

스냅샷 및 업스트림 리포지토리

일반적으로 Maven 스냅샷은 업스트림 리포지토리와 함께 사용할 때 Maven 릴리스 버전과 동일한 방식으로 작동하지만, 업스트림 관계가 있는 두 리포지토리에 동일한 패키지 버전의 스냅샷을 게시하려는 경우 제한이 있습니다. 예를 들어 an AWS CodeArtifact 도메인에 리포지토리가 두 개 R 있고U, 여기서 U는의 업스트림입니다R. R에 새 빌드를 게시하는 경우 Maven 클라이언트가 해당 스냅샷 버전의 최신 빌드를 요청하면 CodeArtifact는 U에서 최신 버전을 반환합니다. 이제 최신 버전이 U가 아닌 R에 있기 때문에 이로 인해 예기치 않은 상황이 발생할 수 있습니다. 이를 피하는 방법은 두 가지입니다.

1. U에 1.0-SNAPSHOT이 있는 경우 R의 1.0-SNAPSHOT과 같은 스냅샷 버전의 빌드를 게시하지 마십시오.

2. CodeArtifact 패키지 원본 제어를 사용하여 R에서 해당 패키지의 업스트림을 비활성화합니다. 후자는 R에서 `1.0-SNAPSHOT`의 빌드를 게시하도록 허용하지만 R이 아직 보존되지 않은 U에서 해당 패키지의 다른 버전을 가져오지 못하게 합니다.

업스트림 및 외부 연결에서 Maven 패키지 요청

표준 자산 이름 가져오기

Maven Central과 같은 퍼블릭 리포지토리에서 Maven 패키지 버전을 가져올 때 AWS CodeArtifact는 해당 패키지 버전의 모든 자산을 가져오려고 시도합니다. 업스트림 리포지토리가 포함된 패키지 버전 요청에 설명된 대로 다음과 같은 경우에 가져오기가 실행됩니다.

- 클라이언트는 CodeArtifact 리포지토리에서 Maven 자산을 요청합니다.
- 패키지 버전은 리포지토리 또는 해당 업스트림에 아직 없습니다.
- 퍼블릭 Maven 리포지토리에 연결할 수 있는 외부 연결이 있습니다.

클라이언트가 하나의 자산만 요청했더라도 CodeArtifact는 해당 패키지 버전에서 찾을 수 있는 모든 자산을 가져오려고 시도합니다. CodeArtifact가 Maven 패키지 버전에 사용할 수 있는 자산을 찾는 방법은 특정 퍼블릭 리포지토리에 따라 달라집니다. 일부 퍼블릭 Maven 리포지토리는 자산 목록 요청을 지원하지만 나머지 리포지토리는 지원하지 않습니다. 자산을 나열하는 방법을 제공하지 않는 리포지토리의 경우, CodeArtifact는 존재할 가능성이 있는 자산 이름의 세트를 생성합니다. 예를 들어, Maven 패키지 버전 `junit 4.13.2`의 자산이 요청되면 CodeArtifact는 다음 자산을 가져오려고 시도합니다.

- `junit-4.13.2.pom`
- `junit-4.13.2.jar`
- `junit-4.13.2-javadoc.jar`
- `junit-4.13.2-sources.jar`

비표준 자산 이름 가져오기

Maven 클라이언트가 위에서 설명한 패턴 중 하나와 일치하지 않는 자산을 요청하면 CodeArtifact는 해당 자산이 퍼블릭 리포지토리에 있는지 확인합니다. 해당 자산이 있는 경우, 그 자산을 가져와서 기존 패키지 버전 레코드(있는 경우)에 추가합니다. 예를 들어, Maven 패키지 버전 `com.android.tools.build:aapt2 7.3.1-8691043`에는 다음과 같은 자산이 포함되어 있습니다.

- `aapt2-7.3.1-8691043.pom`
- `aapt2-7.3.1-8691043-windows.jar`
- `aapt2-7.3.1-8691043-osx.jar`
- `aapt2-7.3.1-8691043-linux.jar`

클라이언트가 POM 파일을 요청할 때 CodeArtifact가 패키지 버전의 자산을 나열할 수 없는 경우, 가져오기가 완료된 자산은 POM 밖에 없습니다. 이는 다른 어떤 자산도 표준 자산 이름 패턴과 일치하지 않기 때문입니다. 하지만 클라이언트가 JAR 자산 중 하나를 요청하면 해당 자산을 가져와서 CodeArtifact에 저장된 기존 패키지 버전에 추가합니다. 업스트림 리포지토리의 패키지 보존에 설명된 대로 최대 다운스트림 리포지토리(클라이언트가 요청한 리포지토리)와 외부 연결이 연결된 리포지토리의 패키지 버전이 모두 새 자산을 포함하도록 업데이트됩니다.

일반적으로 패키지 버전이 CodeArtifact 리포지토리에 보존되면 업스트림 리포지토리의 변경에 영향을 받지 않습니다. 자세한 내용은 업스트림 리포지토리의 패키지 보존 단원을 참조하십시오. 하지만 앞에서 설명한 비표준 이름을 가진 Maven 자산의 동작은 이 규칙의 예외에 속합니다. 클라이언트가 추가 자산을 요청하지 않으면 다운스트림 패키지 버전이 변경되지 않지만 이 경우 보존된 패키지 버전은 맨 처음 보존된 후에 수정되므로 변경 불가능하지 않습니다. 그렇지 않으면 CodeArtifact를 통해 비표준 이름을 가진 Maven 자산에 액세스할 수 없기 때문에 이 동작이 필요합니다. 이 동작은 패키지 버전이 CodeArtifact 리포지토리에 보존된 후 퍼블릭 리포지토리의 Maven 패키지 버전에 추가된 경우에도 활성화됩니다.

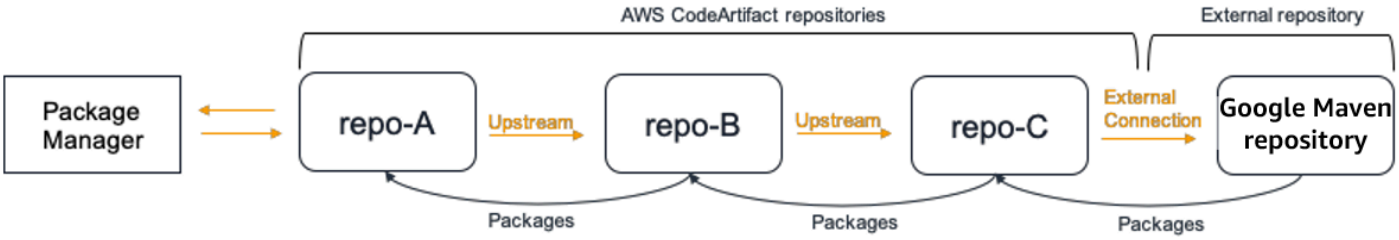
자산 출처 확인

이전에 보존된 Maven 패키지 버전에 새 자산을 추가할 때 CodeArtifact는 보존된 패키지 버전의 원본이 새 자산의 원본과 동일한지 확인합니다. 이렇게 하면 서로 다른 퍼블릭 리포지토리에서 여러 자산이 생성되는 '혼합' 패키지 버전을 만들 수 없습니다. 이 검사를 하지 않으면 Maven 패키지 버전이 둘 이상의 퍼블릭 리포지토리에 게시되고 해당 리포지토리가 CodeArtifact 리포지토리의 업스트림 그래프에 속하는 경우 자산 혼합이 발생할 수 있습니다.

업스트림 리포지토리에서 새 자산 가져오기 및 패키지 버전 상태

업스트림 리포지토리에 있는 패키지 버전의 패키지 버전 상태로 인해 CodeArtifact가 다운스트림 리포지토리에 해당 버전을 보존하지 못할 수 있습니다.

예를 들어, 하나의 도메인에 세 개의 리포지토리 즉, `repo-A`, `repo-B`, `repo-C`가 있는데 여기서 `repo-B`는 `repo-A`의 업스트림이고 `repo-C`는 `repo-B`의 업스트림이라고 가정해 보겠습니다.



Maven 패키지 `com.android.tools.build:aapt2`의 패키지 버전 `7.3.1`은 `repo-B`에 있으며 상태는 `Published`입니다. 이 버전은 `repo-A`에 있지 않습니다. 클라이언트가 `repo-A`를 출처로 한 이 패키지 버전의 자산을 요청하는 경우, 응답은 200(OK)이며 Maven 패키지 버전 `7.3.1`은 `repo-A`에 그대로 유지됩니다. 그러나 `repo-B`에서 패키지 버전 `7.3.1`의 상태가 `Archived` 또는 `Disposed`인 경우, 404(찾을 수 없음)라는 응답이 표시됩니다. 이 두 상태에서 패키지 버전의 자산은 다운로드할 수 없기 때문입니다.

`repo-A`, `repo-B` 및 `repo-C`에서 `com.android.tools.build:aapt2`에 대한 패키지 오리진 제어를 `upstream=BLOCK`로 설정하면 패키지 버전 상태와 관계없이 `repo-A`에서 해당 패키지의 모든 버전에 대한 새 자산을 가져오지 못하게 됩니다.

Maven 문제 해결

다음은 Maven을 CodeArtifact와 함께 사용할 때 일반적으로 발생하는 문제를 해결하는 데 유용한 정보입니다.

병렬 입력을 비활성화하여 오류 429 수정: 요청이 너무 많음

버전 3.9.0부터 Maven은 패키지 아티팩트를 병렬로 업로드합니다(한 번에 최대 5개 파일). 이로 인해 CodeArtifact가 오류 응답 코드 429(요청이 너무 많음)로 응답하는 경우가 있을 수 있습니다. 이 오류가 발생한 경우 병렬 입력을 비활성화하여 수정할 수 있습니다.

병렬 입력을 비활성화하려면 다음 예와 같이 `settings.xml` 파일의 프로필에서 `aether.connector.basic.parallelPut` 속성을 `false`로 설정합니다.

```
<settings>
    <profiles>
        <profile>
            <id>default</id>
            <properties>
                <aether.connector.basic.parallelPut>false</
aether.connector.basic.parallelPut>
            </properties>
```

```
        </profile>
    </profiles>
<settings>
```

자세한 내용은 Maven 설명서의 아티팩트 해석기 구성 옵션을 참조하세요.

CodeArtifact를 npm과 함께 사용

이 항목에서는 Node.js 패키지 관리자인 npm을 CodeArtifact와 함께 사용하는 방법을 설명합니다.

> **Note**
>
> CodeArtifact는 `node v4.9.1` 이상 및 `npm v5.0.0` 이상을 지원합니다.

주제

CodeArtifact로 npm 구성 및 사용

CodeArtifact에서 리포지토리를 만든 후 npm 클라이언트를 사용하여 패키지를 설치하고 게시할 수 있습니다. 리포지토리 엔드포인트와 인증 토큰을 사용하여 npm을 구성하기 위해 권장되는 방법은 `aws codeartifact login` 명령을 사용하는 것입니다. npm을 수동으로 구성할 수도 있습니다.

목차

로그인 명령으로 npm 구성하기

`aws codeartifact login` 명령을 사용하여 npm에 사용할 보안 인증 정보를 가져올 수 있습니다.

> **Note**
>
> 소유한 도메인의 리포지토리에 액세스하는 경우 `--domain-owner`를 포함하지 않아도 됩니다. 자세한 내용은 크로스 계정 도메인 단원을 참조하십시오.

> **Important**
>
> npm 10.x 이상을 사용하는 경우 AWS CLI 버전 2.9.5 이상을 사용하여 `aws codeartifact login` 명령을 성공적으로 실행해야 합니다.

```
aws codeartifact login --tool npm --domain my_domain --domain-owner 111122223333 --
repository my_repo
```

이 명령은 ~/.npmrc 파일을 다음과 같이 변경합니다.

- AWS 보안 인증을 사용하여 CodeArtifact에서 가져온 인증 토큰을 추가합니다.
- npm 레지스트리를 `--repository` 옵션으로 지정된 리포지토리로 설정합니다.
- npm 6 이하의 경우: 모든 npm 명령에 대해 인증 토큰이 전송되도록 `"always-auth=true"`를 추가합니다.

`login` 직접 호출 후의 기본 승인 기간은 12시간이며, 토큰을 주기적으로 재발급하려면 `login`를 직접적으로 호출해야 합니다. `login` 명령으로 만든 인증 토큰에 관한 자세한 내용은 `login` 명령으로 생성된 토큰을 참조하세요.

로그인 명령을 실행하지 않고 npm 구성하기

npm 구성을 수동으로 업데이트하여 `aws codeartifact login` 명령 없이 CodeArtifact 리포지토리로 npm을 구성할 수 있습니다.

로그인 명령을 실행하지 않고 npm을 구성하려면

1. 명령줄에서 CodeArtifact 인증 토큰을 가져와 환경 변수에 저장합니다. npm은 이 토큰을 사용하여 CodeArtifact 리포지토리로 인증합니다.

> **Note**
>
> 다음 명령은 macOS 또는 Linux 시스템에 사용됩니다. Windows 시스템에서 환경 변수를 구성하는 방법에 관한 자세한 내용은 환경 변수를 사용하여 인증 토큰 전달을 참조합니다.

```
CODEARTIFACT_AUTH_TOKEN=`aws codeartifact get-authorization-token --
domain my_domain --domain-owner 111122223333 --query authorizationToken --output
 text`
```

2. 다음 명령을 실행하여 CodeArtifact 리포지토리의 엔드포인트를 가져올 수 있습니다. 리포지토리 엔드포인트는 패키지를 설치하거나 게시하도록 npm을 리포지토리로 이동하는 데 사용됩니다.

   - *my_domain*을 CodeArtifact 도메인 이름으로 변경합니다.
   - *111122223333*을 도메인 소유자의 AWS 계정 ID로 바꿉니다. 소유한 도메인의 리포지토리에 액세스하는 경우 `--domain-owner`를 포함하지 않아도 됩니다. 자세한 내용은 크로스 계정 도메인 단원을 참조하십시오.
   - *my_repo*를 CodeArtifact 리포지토리 이름으로 변경합니다.

```
aws codeartifact get-repository-endpoint --domain my_domain --domain-
owner 111122223333 --repository my_repo --format npm
```

   다음 URL은 리포지토리 엔드포인트의 예입니다.

```
https://my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/npm/my_repo/
```

> **Important**
>
> 레지스트리 URL은 슬래시(/)로 끝나야 합니다. 그렇지 않으면 리포지토리에 연결할 수 없습니다.

3. `npm config set` 명령을 실행하여 레지스트리를 CodeArtifact 리포지토리로 설정합니다. URL을 이전 단계의 리포지토리 엔드포인트 URL로 바꿉니다.

```
npm config set
 registry=https://my_domain-111122223333.d.codeartifact.region.amazonaws.com/
npm/my_repo/
```

> **Note**
>
> 듀얼 스택 엔드포인트를 사용하려면 `codeartifact.region.on.aws` 엔드포인트를 사용합니다.

4. `npm config set` 명령을 사용하여 npm 구성에 인증 토큰을 추가합니다.

```
npm config set //my_domain-111122223333.d.codeartifact.region.amazonaws.com/
npm/my_repo/:_authToken=$CODEARTIFACT_AUTH_TOKEN
```

npm 6 이하의 경우: `GET` 요청이 있을 경우에도 npm이 항상 CodeArtifact에 인증 토큰을 전달하도록 하려면 `always-auth` 구성 변수를 `npm config set`로 설정합니다.

```
npm config set //my_domain-111122223333.d.codeartifact.region.amazonaws.com/
npm/my_repo/:always-auth=true
```

npm 구성 파일 예(**`.npmrc`**)

다음은 위의 지침에 따라 CodeArtifact 레지스트리 엔드포인트를 설정하고 인증 토큰을 추가하며 `always-auth`를 구성한 후의 `.npmrc` 예제 파일입니다.

```
registry=https://my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/npm/my-
cli-repo/
//my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/npm/
my_repo/:_authToken=eyJ2ZX...
//my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/npm/my_repo/:always-
auth=true
```

npm 명령 실행

npm 클라이언트를 구성한 후 npm 명령을 실행할 수 있습니다. 패키지가 리포지토리 또는 업스트림 리포지토리 중 하나에 있다고 할 경우, `npm install`을 사용하여 패키지를 설치할 수 있습니다. 예를 들어, 다음 명령을 사용해 `lodash` 패키지를 설치합니다.

```
npm install lodash
```

다음 명령을 사용하여 새 npm 패키지를 CodeArtifact 리포지토리에 게시하세요.

```
npm publish
```

npm 패키지를 만드는 방법에 관한 자세한 내용은 npm 설명서 웹 사이트에서 Node.js 모듈 생성 섹션을 참조하세요. CodeArtifact에서 지원하는 npm 명령 목록은 npm 명령 지원을 참조하세요.

npm 인증 및 권한 부여 확인

`npm ping` 명령을 간접적으로 호출하면 다음을 확인할 수 있습니다.

- CodeArtifact 리포지토리에 인증할 수 있도록 보안 인증을 올바르게 구성했습니다.
- 권한 부여 구성 시 `ReadFromRepository` 권한이 부여됩니다.

`npm ping`을 성공적으로 간접 호출한 결과는 다음과 같습니다.

```
$ npm -d ping
npm info it worked if it ends with ok
npm info using npm@6.4.1
npm info using node@v9.5.0
npm info attempt registry request try #1 at 4:30:59 PM
npm http request GET https://<domain>.d.codeartifact.us-west-2.amazonaws.com/npm/
shared/-/ping?write=true
npm http 200 https:///npm/shared/-/ping?write=true
Ping success: {}
npm timing npm Completed in 716ms
npm info ok
```

이 `-d` 옵션을 사용하면 npm이 리포지토리 URL을 포함한 추가 디버그 정보를 인쇄합니다. 이 정보를 통해 npm이 사용자가 예상한 리포지토리를 사용하도록 구성되어 있는지 쉽게 확인할 수 있습니다.

기본 npm 레지스트리로 다시 변경

CodeArtifact로 npm을 구성하면 npm 레지스트리는 지정된 CodeArtifact 리포지토리로 설정됩니다. CodeArtifact에 대한 연결이 완료되면 다음 명령을 실행하여 npm 레지스트리를 기본 레지스트리로 다시 설정할 수 있습니다.

```
npm config set registry https://registry.npmjs.com/
```

npm 8.x 이상을 사용한 느린 설치 속도 문제 해결

패키지 리포지토리에 요청을 보내고 리포지토리가 자산을 직접 스트리밍하는 대신 클라이언트를 Amazon S3로 리디렉션하는 경우 npm 클라이언트가 종속성별로 몇 분 동안 중단될 수 있는 문제가 npm 버전 8.x 이상에 있다고 알려져 있습니다.

CodeArtifact 리포지토리는 요청을 항상 Amazon S3로 리디렉션하도록 설계되었으므로 이 문제가 발생하여 npm 설치 시간이 길어져 구축 시간이 길어지는 경우가 있습니다. 이 동작의 인스턴스는 몇 분 동안 진행률 표시줄로 표시됩니다.

이 문제를 방지하려면 다음 예시와 같이 npm cli 명령과 함께 `--no-progress` 또는 `progress=false` 플래그를 사용하십시오.

```
npm install lodash --no-progress
```

CodeArtifact로 Yarn 구성 및 사용

리포지토리를 만든 후 Yarn 클라이언트를 사용하여 npm 패키지를 관리할 수 있습니다.

> **Note**
>
> `Yarn 1.X`는 npm 구성 파일(.npmrc)에서 정보를 읽고 사용하지만 `Yarn 2.X`는 그렇지 않습니다. `Yarn 2.X`에 대한 구성은 .yarnrc.yml 파일에 정의해야 합니다.

목차

`aws codeartifact login` 명령을 사용하여 Yarn 1.X를 구성합니다.

`Yarn 1.X`의 경우, `aws codeartifact login` 명령을 실행하여 CodeArtifact로 Yarn을 구성할 수 있습니다. 이 `login` 명령을 실행하면 CodeArtifact 리포지토리 엔드포인트 정보 및 보안 인증으로

~/.npmrc 파일이 구성됩니다. `Yarn 1.X`를 사용하면 `yarn` 명령은 ~/.npmrc 파일의 구성 정보를 사용합니다.

login 명령으로 **`Yarn 1.X`**를 구성하려면

1. AWS 보안 인증을 아직 구성하지 않았다면 CodeArtifact 시작하기에 설명된 대로 AWS CLI에서 사용할 이 보안 인증을 구성합니다.
2. `aws codeartifact login` 명령을 성공적으로 실행하려면 npm을 설치해야 합니다. 설치 지침에 관한 내용은 npm 설명서의 Node.js 및 npm 다운로드 및 설치를 참조합니다.
3. `aws codeartifact login` 명령을 실행하여 CodeArtifact 보안 인증을 가져오고 ~/.npmrc 파일을 구성합니다.

   - *`my_domain`*을 CodeArtifact 도메인 이름으로 변경합니다.
   - *`111122223333`*을 도메인 소유자의 AWS 계정 ID로 변경합니다. 소유한 도메인의 리포지토리에 액세스하는 경우 `--domain-owner`를 포함하지 않아도 됩니다. 자세한 내용은 크로스 계정 도메인 섹션을 참조하세요.
   - *`my_repo`*를 CodeArtifact 리포지토리 이름으로 변경합니다.

   ```
   aws codeartifact login --tool npm --domain my_domain --domain-owner 111122223333 --
   repository my_repo
   ```

   이 `login` 명령은 ~/.npmrc 파일을 다음과 같이 변경합니다.

   - AWS 보안 인증을 사용하여 CodeArtifact에서 가져온 인증 토큰을 추가합니다.
   - npm 레지스트리를 `--repository` 옵션으로 지정된 리포지토리로 설정합니다.
   - npm 6 이하의 경우: 모든 npm 명령에 대해 인증 토큰이 전송되도록 `"always-auth=true"`를 추가합니다.

   `login` 직접 호출 후 기본 인증 기간은 12시간이며 토큰을 주기적으로 새로 고치려면 `login`을 직접적으로 호출해야 합니다. `login` 명령으로 만든 인증 토큰에 관한 자세한 내용은 `login` 명령으로 생성된 토큰을 참조하세요.
4. npm 7.X 및 8.X의 경우 Yarn을 사용하려면 ~/.npmrc 파일에 `always-auth=true`를 추가해야 합니다.

   - 텍스트 편집기에서 ~/.npmrc 파일을 열고 `always-auth=true`를 새 줄에 추가합니다.

`yarn config list` 명령을 실행하면 Yarn이 올바른 구성을 사용하고 있는지 확인할 수 있습니다. 명령을 실행한 후 `info npm config` 섹션의 값을 확인합니다. 내용은 다음 코드 조각과 비슷해야 합니다.

```
info npm config
{
  registry: 'https://my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/npm/
my_repo/',
  '//my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/npm/
my_repo/:_authToken': 'eyJ2ZXI...',
  'always-auth': true
}
```

`yarn config set` 명령을 사용하여 Yarn 2.X를 구성합니다.

다음 절차는 `yarn config set` 명령을 사용해 명령줄에서 `.yarnrc.yml` 구성을 업데이트하여 `Yarn 2.X`를 구성하는 방법을 자세히 설명합니다.

명령줄에서 **`yarnrc.yml`** 구성을 업데이트하려면

1. AWS 보안 인증을 아직 구성하지 않았다면 CodeArtifact 시작하기에 설명된 대로 AWS CLI에서 사용할 이 보안 인증을 구성합니다.
2. `aws codeartifact get-repository-endpoint` 명령을 실행하여 CodeArtifact 리포지토리의 엔드포인트를 가져올 수 있습니다.

   - *`my_domain`*을 CodeArtifact 도메인 이름으로 변경합니다.
   - *`111122223333`*을 도메인 소유자의 AWS 계정 ID로 변경합니다. 소유한 도메인의 리포지토리에 액세스하는 경우 `--domain-owner`를 포함하지 않아도 됩니다. 자세한 내용은 크로스 계정 도메인 섹션을 참조하세요.
   - *`my_repo`*를 CodeArtifact 리포지토리 이름으로 변경합니다.

   ```
   aws codeartifact get-repository-endpoint --domain my_domain --domain-
   owner 111122223333 --repository my_repo --format npm
   ```

3. .yarnrc.yml 파일의 npmRegistryServer 값을 리포지토리 엔드포인트로 업데이트합니다.

```
yarn config set npmRegistryServer
 "https://my_domain-111122223333.d.codeartifact.region.amazonaws.com/npm/my_repo/"
```

4. CodeArtifact 인증 토큰을 가져와 환경 변수에 저장합니다.

> **Note**
>
> 다음 명령은 macOS 또는 Linux 시스템에 사용됩니다. Windows 시스템에서 환경 변수를 구성하는 방법에 관한 자세한 내용은 환경 변수를 사용하여 인증 토큰 전달을 참조합니다.

- *my_domain*을 CodeArtifact 도메인 이름으로 변경합니다.
- *111122223333*을 도메인 소유자의 AWS 계정 ID로 변경합니다. 소유한 도메인의 리포지토리에 액세스하는 경우 `--domain-owner`를 포함하지 않아도 됩니다. 자세한 내용은 크로스 계정 도메인 섹션을 참조하세요.
- *my_repo*를 CodeArtifact 리포지토리 이름으로 변경합니다.

```
export CODEARTIFACT_AUTH_TOKEN=`aws codeartifact get-authorization-token --
domain my_domain --domain-owner 111122223333 --query authorizationToken --output
 text`
```

5. `yarn config set` 명령을 사용하여 CodeArtifact 인증 토큰을 .yarnrc.yml 파일에 추가합니다. 다음 명령의 URL을 2단계의 리포지토리 엔드포인트 URL로 변경합니다.

```
yarn config set
 'npmRegistries["https://my_domain-
111122223333.d.codeartifact.region.amazonaws.com/npm/my_repo/"].npmAuthToken'
 "${CODEARTIFACT_AUTH_TOKEN}"
```

6. `yarn config set` 명령을 실행하여 `npmAlwaysAuth`의 값을 `true`로 설정합니다. 다음 명령의 URL을 2단계의 리포지토리 엔드포인트 URL로 변경합니다.

```
yarn config set
 'npmRegistries["https://my_domain-
111122223333.d.codeartifact.region.amazonaws.com/npm/my_repo/"].npmAlwaysAuth'
 "true"
```

구성한 후에는 .yarnrc.yml 구성 파일에 다음 코드 조각과 비슷한 내용이 포함되어야 합니다.

```
npmRegistries:
  "https://my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/npm/my_repo/":
    npmAlwaysAuth: true
    npmAuthToken: eyJ2ZXI...

npmRegistryServer: "https://my_domain-111122223333.d.codeartifact.us-
west-2.amazonaws.com/npm/my_repo/"
```

`yarn config` 명령을 실행하여 `npmRegistries` 및 `npmRegistryServer`의 값을 확인할 수도 있습니다.

npm 명령 지원

다음 섹션에는 지원되지 않는 특정 명령 외에도 CodeArtifact 리포지토리에서 지원하는 npm 명령이 요약되어 있습니다.

목차

리포지토리와 상호 작용하는 지원되는 명령

이 섹션에는 npm 클라이언트가 구성될 때 사용된 레지스트리(예: `npm config set registry` 포함)에 하나 이상의 요청을 보내는 npm 명령이 나열되어 있습니다. CodeArtifact 리포지토리에 대해 이러한 명령을 간접적으로 호출했을 때 제대로 작동하는 것으로 확인되었습니다.

명령	설명
bugs	패키지의 버그 추적기 URL 위치를 추측한 후 URL 열기를 시도합니다.
ci	프로젝트를 새로 다시 설치합니다.
deprecate	패키지 버전을 더 이상 사용하지 않습니다.

명령	설명
dist-tag	패키지 배포 태그를 수정합니다.
docs	패키지 설명서 URL의 위치를 추측한 다음 `--browser` config 파라미터를 사용하여 URL 열기를 시도합니다.
doctor	일련의 검사를 실행하여 JavaScript 패키지를 관리하는 데 필요한 것이 npm 설치에 있는지 확인합니다.
install	패키지를 설치합니다.
install-ci-test	프로젝트를 새로 다시 설치하고 테스트를 실행합니다. 별칭: `npm cit`. 이 명령은 `npm ci`를 실행한 후 즉시 `npm test`를 실행합니다.
install-test	패키지를 설치하고 테스트를 실행합니다. `npm install`을 실행한 후 즉시 `npm test`를 실행합니다.
outdated	구성된 레지스트리를 검사하여 설치된 패키지가 현재 만료되었는지 확인합니다.
ping	구성되거나 지정된 npm 레지스트리를 ping하고 인증을 확인합니다.
publish	패키지 버전을 레지스트리에 게시합니다.
update	패키지의 리포지토리 URL 위치를 추측한 다음, `--browser` config 파라미터를 사용하여 URL 열기를 시도합니다.
view	패키지 메타데이터를 표시합니다. 메타데이터 속성을 인쇄하는 데 사용할 수 있습니다.

지원되는 클라이언트 측 명령

이러한 명령은 리포지토리와 직접 상호 작용할 필요가 없으므로 CodeArtifact는 명령을 지원하기 위해 아무 것도 할 필요가 없습니다.

명령	설명
build	패키지를 빌드합니다.
cache	패키지 캐시를 조작합니다.
completion	모든 npm 명령에서 탭 완성을 활성화합니다.
config	사용자 및 글로벌 `npmrc` 파일의 내용을 업데이트합니다.
dedupe	로컬 패키지 트리를 검색하고 종속성을 트리 위로 이동하여 구조를 단순화하려고 합니다. 여기서 종속성을 여러 종속 패키지에서 더 효과적으로 공유할 수 있습니다.
edit	설치된 패키지를 편집합니다. 현재 작업 디렉터리에서 종속성을 선택하고 기본 편집기에서 패키지 폴더를 엽니다.
explore	설치된 패키지를 찾아봅니다. 설치된 특정 패키지의 디렉터리에 서브셸을 생성합니다. 명령이 지정되면 해당 명령은 서브셸에서 실행된 후 즉시 종료됩니다.
help	npm에 관한 도움말을 가져옵니다.
help-search	npm 도움말 설명서를 검색합니다.
init	`package.json` 파일을 생성합니다.
link	패키지 폴더를 symlink합니다.
ls	설치된 패키지를 나열합니다.

명령	설명
pack	패키지에서 tarball을 생성합니다.
prefix	접두사를 표시합니다. -g도 지정되지 않는 한, 이 디렉터리는 package.json 파일을 포함하는 가장 가까운 상위 디렉터리입니다.
prune	상위 패키지의 종속성 목록에 나열되지 않은 패키지를 제거합니다.
rebuild	일치하는 폴더에서 npm build 명령을 실행합니다.
restart	패키지의 중지, 재시작, 시작 스크립트와 관련 사전/사후 스크립트를 실행합니다.
root	유효 node_modules 폴더를 표준 출력으로 출력합니다.
run-script	임의의 패키지 스크립트를 실행합니다.
shrinkwrap	게시할 종속 버전을 잠급니다.
uninstall	패키지를 제거합니다.

지원되지 않는 명령

이러한 npm 명령은 CodeArtifact 리포지토리에서 지원하지 않습니다.

명령	설명	참고
access	게시된 패키지에서 액세스 수준을 설정합니다.	CodeArtifact는 퍼블릭 npmjs 리포지토리와는 다른 권한 모델을 사용합니다.

명령	설명	참고
adduser	레지스트리 사용자 계정을 추가합니다.	CodeArtifact는 퍼블릭 npmjs 리포지토리와는 다른 사용자 모델을 사용합니다.
audit	보안 감사를 실행합니다.	CodeArtifact는 현재 보안 취약성 데이터를 제공하지 않습니다.
hook	추가, 제거, 나열 및 업데이트를 포함하여 npm 후크를 관리합니다.	CodeArtifact는 현재 어떠한 종류의 변경 알림 메커니즘도 지원하지 않습니다.
login	사용자를 인증합니다. `npm adduser`에 대한 별칭입니다.	CodeArtifact는 퍼블릭 npmjs 리포지토리와는 다른 인증 모델을 사용합니다. 자세한 내용은 npm으로 인증을 참조하세요.
logout	레지스트리에서 로그아웃합니다.	CodeArtifact는 퍼블릭 npmjs 리포지토리와는 다른 인증 모델을 사용합니다. CodeArtifact 리포지토리에서 로그아웃할 수 있는 방법은 없지만 인증 토큰은 구성 가능한 만료 시간이 지나면 만료됩니다. 기본 토큰 지속 시간은 12시간입니다.
owner	패키지 소유자를 관리합니다.	CodeArtifact는 퍼블릭 npmjs 리포지토리와는 다른 권한 모델을 사용합니다.
profile	레지스트리 프로필의 설정을 변경합니다.	CodeArtifact는 퍼블릭 npmjs 리포지토리와는 다른 사용자 모델을 사용합니다.

명령	설명	참고
search	레지스트리에서 검색어와 일치하는 패키지를 검색합니다.	CodeArtifact는 list-packages 명령으로 제한된 검색 기능을 지원합니다.
star	좋아하는 패키지를 표시합니다.	CodeArtifact는 현재 어떠한 종류의 즐겨찾기 메커니즘도 지원하지 않습니다.
stars	즐겨찾기로 표시된 패키지를 조회합니다.	CodeArtifact는 현재 어떠한 종류의 즐겨찾기 메커니즘도 지원하지 않습니다.
team	조직 팀 및 팀 멤버십을 관리합니다.	CodeArtifact는 퍼블릭 npmjs 리포지토리와는 다른 사용자 및 그룹 구성원 모델을 사용합니다. 자세한 내용은 IAM 사용 설명서의 자격 증명(사용자, 그룹 및 역할) 섹션을 참조하세요.
token	인증 토큰을 관리합니다.	CodeArtifact는 인증 토큰을 가져오기 위해 다른 모델을 사용합니다. 자세한 내용은 npm으로 인증을 참조하세요.
unpublish	레지스트리에서 패키지를 제거합니다.	CodeArtifact는 npm 클라이언트를 사용하여 리포지토리에서 패키지 버전을 제거하는 것을 지원하지 않습니다. delete-package-version 명령을 사용할 수 있습니다.
whoami	npm 사용자 이름을 표시합니다.	CodeArtifact는 퍼블릭 npmjs 리포지토리와는 다른 사용자 모델을 사용합니다.

npm 태그 처리

npm 레지스트리는 패키지 버전의 문자열 별칭인 태그를 지원합니다. 태그를 사용하여 버전 번호 대신 별칭을 제공할 수 있습니다. 예를 들어, 여러 개발 스트림이 있는 프로젝트에서 각 스트림마다 다른 태그(예: `stable`, `beta`, `dev`, `canary`)를 사용할 수 있습니다. 자세한 내용은 npm 웹 사이트에서 dist-tag를 참조하세요.

기본적으로 npm은 `latest` 태그를 사용하여 패키지의 현재 버전을 식별합니다. `npm install` *`pkg`*(`@`*`version`* 또는 `@`*`tag`* 지정자가 없는)는 최신 태그를 설치합니다. 일반적으로 프로젝트는 안정적인 릴리스 버전에만 최신 태그를 사용합니다. 그 밖의 태그는 불안정한 버전이나 프리릴리스 버전에 사용됩니다.

npm 클라이언트로 태그를 편집

CodeArtifact 리포지토리에서는 세 개의 `npm dist-tag` 명령(`add`, `rm` 및 `ls`)이 기본 npm 레지스트리에서와 동일하게 작동합니다.

npm 태그와 CopyPackageVersions API

`CopyPackageVersions` API를 사용하여 npm 패키지 버전을 복사하면 해당 버전의 별칭을 지정하는 모든 태그가 대상 리포지토리에 복사됩니다. 대상에 있는 태그가 복사 중인 버전에도 있을 때 복사 작업은 대상 리포지토리의 태그 값을 소스 리포지토리의 값과 일치하도록 설정합니다.

예를 들어, 다음 표에 나와 있는 것처럼 리포지토리 S와 리포지토리 D에 모두 최신 태그가 설정된 단일 버전의 `web-helper` 패키지가 포함되어 있다고 가정해 보겠습니다.

리포지토리	패키지 이름	패키지 태그
S	`web-helper`	latest(버전 1.0.1의 별칭)
D	`web-helper`	latest(버전 1.0.0의 별칭)

`CopyPackageVersions`는 `web-helper` 1.0.1을 S에서 D로 복사하기 위해 간접적으로 호출됩니다. 작업이 완료된 후 리포지토리 D의 `web-helper`에서 `latest` 태그는 1.0.0이 아닌 1.0.1로 별칭을 지정합니다.

복사 후 태그를 변경해야 하는 경우, `npm dist-tag` 명령을 사용하여 대상 리포지토리에서 직접 태그를 수정합니다. `CopyPackageVersions` API에 관한 자세한 내용은 리포지토리 간 패키지 복사를 참조합니다.

npm 태그와 업스트림 리포지토리

npm이 패키지에 대한 태그를 요청하고 해당 패키지의 버전이 업스트림 리포지토리에도 있을 때 CodeArtifact는 태그를 병합한 후 클라이언트에 반환합니다. 예를 들면, R이라는 리포지토리에 U라는 업스트림 리포지토리가 있습니다. 다음 표에는 두 리포지토리에 모두 존재하는 `web-helper`라는 패키지의 태그가 나와 있습니다.

리포지토리	패키지 이름	패키지 태그
R	`web-helper`	latest(버전 1.0.0의 별칭)
U	`web-helper`	alpha(버전 1.0.1의 별칭)

이 경우, npm 클라이언트가 리포지토리 R에서 `web-helper` 패키지의 태그를 가져오면 latest 태그와 alpha 태그를 모두 받습니다. 태그가 가리키는 버전은 변경되지 않습니다.

업스트림 리포지토리와 다운스트림 리포지토리의 동일한 패키지에 동일한 태그가 있는 경우, CodeArtifact는 업스트림 리포지토리에 있는 태그를 사용합니다. 예를 들어, webhelper의 태그가 다음과 같이 수정되었다고 가정해 보겠습니다.

리포지토리	패키지 이름	패키지 태그
R	`web-helper`	latest(버전 1.0.0의 별칭)
U	`web-helper`	latest(버전 1.0.1의 별칭)

이 경우, npm 클라이언트가 리포지토리 R에서 패키지 web-helper의 태그를 가져오면 latest 태그는 버전 1.0.1에 별칭을 지정합니다. 이는 업스트림 리포지토리에 있기 때문입니다. 이렇게 하면 `npm update`를 실행하여 다운스트림 리포지토리에 아직 없는 업스트림 리포지토리의 새 패키지 버전을 쉽게 사용할 수 있습니다.

다운스트림 리포지토리에 패키지의 새 버전을 게시할 때 업스트림 리포지토리의 태그를 사용하면 문제가 될 수 있습니다. 예를 들어, 패키지 web-helper의 latest 태그가 R과 U에서 모두 동일하다고 가정해 보겠습니다.

리포지토리	패키지 이름	패키지 태그
R	web-helper	latest(버전 1.0.1의 별칭)
U	web-helper	latest(버전 1.0.1의 별칭)

버전 1.0.2가 R에 게시되면 npm은 latest 태그를 1.0.2로 업데이트합니다.

리포지토리	패키지 이름	패키지 태그
R	web-helper	latest(버전 1.0.2의 별칭)
U	web-helper	latest(버전 1.0.1의 별칭)

하지만 U의 latest 값이 1.0.1이기 때문에 npm 클라이언트는 이 태그 값을 볼 수 없습니다. 1.0.2를 게시한 직후 리포지토리 R을 대상으로 `npm install`을 실행하면 방금 게시된 버전 대신 1.0.1이 설치됩니다. 가장 최근에 게시된 버전을 설치하려면 다음과 같이 정확한 패키지 버전을 지정해야 합니다.

```
npm install web-helper@1.0.2
```

npm 호환 패키지 관리자 지원

다음과 같은 기타 패키지 관리자는 CodeArtifact와 호환되며 npm 패키지 형식 및 npm 와이어 프로토콜과 함께 작동합니다.

- pnpm package manager. CodeArtifact와 연동하는 것으로 확인된 최신 버전은 2019년 5월 18일에 릴리스된 3.3.4 버전입니다.
- Yarn package manager. CodeArtifact와 연동하는 것으로 확인된 최신 버전은 2019년 12월 11일에 릴리스된 1.21.1 버전입니다.

> **Note**
>
> CodeArtifact와 함께 Yarn 2.x를 사용하는 것이 좋습니다. Yarn 1.x에는 HTTP 재시도가 없으므로 500수준 상태 코드나 오류가 발생하는 간헐적인 서비스 오류에 더 취약합니다. Yarn 1.x에 대해 다른 재시도 전략을 구성할 수 있는 방법은 없지만 Yarn 2.x에서는 이것이 추가되었습니

다. Yarn 1.x를 사용할 수 있지만 빌드 스크립트에 더 높은 수준의 재시도를 추가해야 할 수도 있습니다. 예를 들어, 루프에서 yarn 명령을 실행하면 패키지 다운로드가 실패할 경우 다시 시도하게 됩니다.

CodeArtifact를 NuGet과 함께 사용

이 항목에서는 CodeArtifact를 사용하여 NuGet 패키지를 사용하고 게시하는 방법을 설명합니다.

> **Note**
>
> AWS CodeArtifact는 NuGet.exe 버전 4.8 이상만 지원합니다.

주제

Visual Studio와 함께 CodeArtifact 사용하기

CodeArtifact 보안 인증 공급자를 사용하여 Visual Studio에서 직접 CodeArtifact의 패키지를 사용할 수 있습니다. 보안 인증 공급자는 Visual Studio에서 CodeArtifact 리포지토리의 설정 및 인증을 단순화하며 AWS Toolkit for Visual Studio에서 사용할 수 있습니다.

> **Note**
>
> AWS Toolkit for Visual Studio은 Mac용 Visual Studio에서는 사용할 수 없습니다.

NuGet을 CLI 도구와 함께 구성하고 사용하려면 CodeArtifact를 nuget 또는 dotnet CLI와 함께 사용하기을 참조하십시오.

주제

CodeArtifact 보안 인증 공급자를 사용하여 Visual Studio를 구성하기

CodeArtifact 보안 인증 공급자는 CodeArtifact와 Visual Studio 간의 설정 및 지속적인 인증을 간소화합니다. CodeArtifact 인증 토큰은 최대 12시간 동안 유효합니다. Visual Studio에서 작업하는 동안 토큰을 수동으로 새로 고칠 필요가 없도록 보안 인증 공급자는 현재 토큰이 만료되기 전에 계속 새 토큰으로 갱신합니다.

> **Important**
>
> 보안 인증 공급자를 사용하려면 수동으로 `nuget.config`에 추가했거나 이전에 NuGet을 구성하고 `aws codeartifact login`를 구동하여 추가했을 수 있는 기존 AWS CodeArtifact 보안 인증 정보를 파일에서 지워야 합니다.

Visual Studio의 CodeArtifact를 AWS Toolkit for Visual Studio과 함께 사용하기

1. 다음 단계를 수행하여 AWS Toolkit for Visual Studio를 설치하세요. 이 툴킷은 다음 단계를 사용하면 Visual Studio 2017 및 2019와 호환됩니다. AWS CodeArtifact는 Visual Studio 2015 및 이전 버전을 지원하지 않습니다.

   1. Visual Studio 2017 및 Visual Studio 2019용 Visual Studio용 툴킷은 Visual Studio 마켓플레이스에서 배포합니다. 도구 ≫ 확장 및 업데이트 (Visual Studio 2017) 또는 확장 ≫ 확장 관리 (Visual Studio 2019) 를 사용하여 Visual Studio 내에 도구 모음을 설치하고 업데이트할 수도 있습니다.
   2. 툴킷을 설치한 후 보기 메뉴에서 AWS 탐색기를 선택하여 툴킷을 엽니다.
2. AWS Toolkit for Visual Studio 사용 설명서의AWS 보안 인증 제공에 나와 있는 단계에 따라 AWS 보안 인증을 사용하여 Visual Studio용 도구 모음을 구성하십시오.
3. (선택 사항) CodeArtifact와 함께 사용할 AWS 프로필을 설정합니다. 설정하지 않으면 CodeArtifact는 기본 프로필을 사용합니다. 프로필을 설정하려면 도구 > NuGet 패키지 관리자 > CodeArtifact AWS 프로필 선택으로 이동합니다.
4. CodeArtifact 리포지토리를 Visual Studio에서 패키지 소스로 추가합니다.

   1. AWS탐색기 창에서 리포지토리 창으로 이동한 다음 마우스 오른쪽 버튼을 클릭하여 `Copy NuGet Source Endpoint`를 선택합니다.
   2. 도구 > 옵션 명령을 사용하고 NuGet 패키지 관리자로 스크롤합니다.
   3. 패키지 소스 노드를 선택합니다.

4. +를 선택하고 이름을 편집한 다음 3a단계에서 복사한 리포지토리 URL 엔드포인트를 소스 상자에 붙여넣고 업데이트를 선택합니다.
5. 새로 추가한 패키지 소스의 확인란을 선택하여 활성화합니다.

> **Note**
>
> 이후 CodeArtifact 리포지토리에 Nuget.org에 대한 외부 연결을 추가하고 Visual Studio에서 nuget.org 패키지 소스를 비활성화하는 것을 권고드립니다. 외부 연결을 사용하는 경우 NuGet.org에서 가져온 모든 패키지는 CodeArtifact 리포지토리에 저장됩니다. NuGet.org를 사용할 수 없게 되더라도 애플리케이션 종속 항목을 CI 빌드 및 로컬 개발에 계속 사용할 수 있습니다. 연결에 대한 자세한 정보는 CodeArtifact 저장소를 공용 저장소에 연결하기을 참조하세요.

5. 변경 사항을 적용하려면 Visual Studio를 다시 시작합니다.

구성 후 Visual Studio는 CodeArtifact 리포지토리, 모든 업스트림 리포지토리 또는 외부 연결을 추가한 경우 NuGet.org의 패키지를 사용할 수 있습니다. Visual Studio에서 NuGet 패키지를 검색하고 설치하는 방법에 대한 자세한 내용은 NuGet 설명서의 NuGet 패키지 관리자를 사용한 Visual Studio에서 패키지 설치 및 관리를 참조하십시오.

Visual Studio의 Package Manager Console에서 NuGet 사용하기

Visual Studio 패키지 관리자 콘솔은 CodeArtifact 보안 인증 공급자의 Visual Studio 버전을 사용하지 않습니다. 이를 사용하려면 명령줄 보안 인증 공급자를 구성해야 합니다. 자세한 내용은 CodeArtifact를 nuget 또는 dotnet CLI와 함께 사용하기를 참조하세요.

CodeArtifact를 nuget 또는 dotnet CLI와 함께 사용하기

`nuget` 및 `dotnet`와 같은 CLI 도구를 사용하여 CodeArtifact에서 패키지를 게시하고 사용할 수 있습니다. 이 문서에서는 CLI 도구를 구성하고 이를 사용하여 패키지를 게시하거나 사용하는 방법에 관한 정보를 제공합니다.

주제

- nuget 또는 dotnet CLI 구성하기
- CodeArtifact의 NuGet 패키지 사용하기
- NuGet 패키지를 CodeArtifact에 게시하기

- CodeArtifact NuGet 보안 인증 공급자 참조
- CodeArtifact NuGet 보안 인증 공급자 버전

nuget 또는 dotnet CLI 구성하기

CodeArtifact NuGet 자격 증명 공급자를 사용하거나를 사용하거나 수동으로 nuget AWS CLI또는 dotnet CLI를 구성할 수 있습니다. 간단한 설정과 지속적인 인증을 위해 보안 인증 공급자를 통해 NuGet을 구성하는 것을 권장합니다.

방법 1: CodeArtifact NuGet 보안 인증 공급자를 통해 구성하기

CodeArtifact NuGet 보안 인증 공급자는 NuGet CLI 도구를 사용하여 CodeArtifact의 인증 및 구성을 단순화합니다. CodeArtifact 인증 토큰은 최대 12시간 동안 유효합니다. nuget 또는 dotnet CLI를 사용하는 동안 토큰을 수동으로 갱신할 필요가 없도록 보안 인증 공급자는 현재 토큰이 만료되기 전에 정기적으로 새 토큰을 발행합니다.

> **Important**
>
> 자격 증명 공급자를 사용하려면 수동으로 추가되었거나 이전에 NuGet을 구성aws `codeartifact login`하기 위해를 실행하여 파일에서 기존 AWS CodeArtifact 자격 증명을 지워야 `nuget.config` 합니다.

CodeArtifact NuGet 보안 인증 공급자 설치 및 구성

dotnet

1. 다음 `dotnet` 명령을 사용하여 Nuget.org에서 AWS.CodeArtifact.Nuget.CredentialProvider 도구의 최신 버전을 다운로드하십시오.

   ```
   dotnet tool install -g AWS.CodeArtifact.NuGet.CredentialProvider
   ```

2. `codeartifact-creds install` 명령을 사용하여 보안 인증 공급자를 NuGet 플러그인 폴더에 복사합니다.

   ```
   dotnet codeartifact-creds install
   ```

3. (선택 사항): 자격 증명 공급자와 함께 사용할 AWS 프로필을 설정합니다. 설정하지 않으면 보안 인증 공급자가 기본 프로파일을 사용합니다. AWS CLI 프로파일에 대한 자세한 내용은 명명된 프로파일을 참조하세요.

```
dotnet codeartifact-creds configure set profile profile_name
```

nuget

NuGet CLI를 사용하여 Amazon S3 버킷에서 CodeArtifact NuGet 보안 인증 공급자를 설치하고 이를 구성하여 다음 단계를 진행하세요. 자격 증명 공급자는 기본 AWS CLI 프로필을 사용합니다. 프로필에 대한 자세한 내용은 명명된 프로필을 참조하세요.

1. Amazon S3 버킷에서 CodeArtifact NuGet 보안 인증 공급자 (codeartifact-nuget-credentialprovider.zip) 의 최신 버전을 다운로드합니다.

   이전 버전을 보고 다운로드하려면 CodeArtifact NuGet 보안 인증 공급자 버전를 참조하십시오.
2. 파일 압축을 풉니다.
3. netfx 폴더의 AWS.CodeArtifact.nugetCredentialProvider 폴더를 Windows는 `%user_profile%/.nuget/plugins/netfx/`, Linux 또는 macOS의 경우`~/.nuget/plugins/netfx`에 복사합니다.
4. netcore 폴더의 AWS.CodeArtifact.nugetCredentialProvider 폴더를 Windows는 `%user_profile%/.nuget/plugins/netcore/`, Linux 또는 macOS의 경우`~/.nuget/plugins/netcore`에 복사합니다.

리포지토리를 생성하고 보안 인증 공급자를 구성한 후 `nuget` 또는 `dotnet` CLI 도구를 사용하여 패키지를 설치하고 게시할 수 있습니다. 자세한 내용은 CodeArtifact의 NuGet 패키지 사용하기 및 NuGet 패키지를 CodeArtifact에 게시하기 섹션을 참조하세요.

방법 2: 로그인 명령으로 nuget 또는 dotnet을 구성하기

의 `codeartifact login` 명령은 NuGet 구성 파일에 리포지토리 엔드포인트와 권한 부여 토큰을 AWS CLI 추가하여 nuget 또는 dotnet이 CodeArtifact 리포지토리에 연결할 수 있도록 합니다. 이렇게 하면 Windows `%appdata%\NuGet\NuGet.Config` 및/또는 Mac/Linux 내 `~/.config/NuGet/NuGet.Config`, `~/.nuget/NuGet/NuGet.Config`에 있는 사용자 수준의 NuGet 구성이 수정됩니다. NuGet 구성에 대한 자세한 내용은 일반 NuGet 구성을 참조하십시오.

`login` 명령을 사용하여 nuget 또는 dotnet을 구성하기

1. 에 설명된 AWS CLI대로와 함께 사용할 자격 AWS 증명을 구성합니다CodeArtifact 시작하기.
2. NuGet CLI 도구 (`nuget`또는`dotnet`) 가 제대로 설치 및 구성되었는지 확인하십시오. 지침은 nuget 또는 dotnet 설명서를 참조하십시오.
3. CodeArtifact `login` 명령을 사용하여 NuGet에 사용할 보안 인증 정보를 가져올 수 있습니다.

> **Note**
>
> 자신이 소유한 도메인의 저장소에 접근하는 경우 `--domain-owner`를 포함하지 않아도 됩니다. 자세한 내용은 크로스 계정 도메인 단원을 참조하십시오.

dotnet

> **Important**
>
> Linux 및 macOS 사용자: Windows 이외의 플랫폼에서는 암호화를 지원하지 않으므로 가져온 보안 인증은 구성 파일에 일반 텍스트로 저장됩니다.

```
aws codeartifact login --tool dotnet --domain my_domain --domain-
owner 111122223333 --repository my_repo
```

nuget

```
aws codeartifact login --tool nuget --domain my_domain --domain-
owner 111122223333 --repository my_repo
```

로그인 명령은 다음과 같습니다.

- AWS 자격 증명을 사용하여 CodeArtifact에서 권한 부여 토큰을 가져옵니다.
- NuGet 패키지 소스의 새 항목으로 사용자 수준 NuGet 구성을 업데이트하십시오. CodeArtifact 리포지토리 엔드포인트를 가리키는 소스가 *domain_name/repo_name*을 호출합니다.

`login` 직접 호출 후의 기본 승인 기간은 12시간이며, 토큰을 주기적으로 새로 고치려면 `login`을 직접적으로 호출해야 합니다. `login` 명령으로 만든 인증 토큰에 대한 자세한 내용은 `login` 명령으로 생성된 토큰을 참조하십시오.

리포지토리를 생성하고 인증을 구성한 후 `nuget`, `dotnet` 또는 `msbuild` CLI 클라이언트를 사용하여 패키지를 설치하고 게시할 수 있습니다. 자세한 정보는 CodeArtifact의 NuGet 패키지 사용하기 및 NuGet 패키지를 CodeArtifact에 게시하기를 참조하세요.

방법 3: 로그인 명령 없이 nuget 또는 dotnet 구성하기

수동 구성의 경우 NuGet 구성 파일에 리포지토리 엔드포인트와 인증 토큰을 추가하여 nuget 또는 dotnet이 CodeArtifact 리포지토리에 연결할 수 있도록 해야 합니다.

CodeArtifact 리포지토리에 연결하도록 nuget 또는 dotnet을 수동으로 구성하세요.

1. `get-repository-endpoint` AWS CLI 명령을 사용하여 CodeArtifact 리포지토리 엔드포인트를 확인합니다.

```
aws codeartifact get-repository-endpoint --domain my_domain --domain-
owner 111122223333 --repository my_repo --format nuget
```

출력 예시:

```
{
   "repositoryEndpoint": "https://my_domain-111122223333.d.codeartifact.us-
west-2.amazonaws.com/nuget/my_repo/"
}
```

2. `get-authorization-token` AWS CLI 명령을 사용하여 패키지 관리자에서 리포지토리에 연결할 권한 부여 토큰을 가져옵니다.

```
aws codeartifact get-authorization-token --domain my_domain
```

출력 예시:

```
{
   "authorizationToken": "eyJ2I...viOw",
   "expiration": 1601616533.0
}
```

3. 3단계에서 `get-repository-endpoint`가 반환한 URL에 `/v3/index.json`을 추가하여 전체 리포지토리 엔드포인트 URL을 생성합니다.
4. 1단계의 리포지토리 엔드포인트와 2단계의 인증 토큰을 사용하도록 nuget 또는 dotnet을 구성합니다.

> **Note**
>
> nuget 또는 dotnet이 CodeArtifact 리포지토리에 성공적으로 연결하려면 소스 URL이 `/v3/index.json`로 끝나야 합니다.

dotnet

Linux 및 macOS 사용자: Windows 이외의 플랫폼에서는 암호화를 지원하지 않으므로 다음 명령에 `--store-password-in-clear-text` 플래그를 추가해야 합니다. 단, 이렇게 하면 비밀번호가 구성 파일에 일반 텍스트로 저장됩니다.

```
dotnet nuget add source https://my_domain-111122223333.d.codeartifact.us-
west-2.amazonaws.com/nuget/my_repo/v3/index.json --name packageSourceName --
password eyJ2I...vi0w --username aws
```

> **Note**
>
> 기존 소스를 업데이트하려면 `dotnet nuget update source` 명령을 사용하세요.

nuget

```
nuget sources add -name domain_name/repo_name -Source
 https://my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/
nuget/my_repo/v3/index.json -password eyJ2I...vi0w -username aws
```

출력 예시:

```
Package source with Name: domain_name/repo_name added successfully.
```

> **Note**
>
> 듀얼 스택 엔드포인트를 사용하려면 `codeartifact.region.on.aws` 엔드포인트를 사용합니다.

CodeArtifact의 NuGet 패키지 사용하기

CodeArtifact로 NuGet을 구성한 후에는 CodeArtifact 리포지토리 또는 업스트림 리포지토리 중 하나에 저장된 NuGet 패키지를 사용할 수 있습니다.

CodeArtifact 리포지토리 또는 해당 업스트림 리포지토리 `nuget` `dotnet` 중 하나에서 패키지 버전을 사용하거나 사용하려면 NuGet 구성 파일에서 *`PackageName`*을 사용하려는 패키지 이름으로, *`PackageSourceName`*을 CodeArtifact 리포지토리의 소스 이름으로 대체하는 다음 명령을 실행합니다. `login` 명령을 사용하여 NuGet 구성을 구성한 경우 소스 이름은 *`###_##/###_##`*입니다.

> **Note**
>
> 패키지를 요청하면 NuGet 클라이언트는 해당 패키지의 어떤 버전이 존재하는지 캐시를 확인합니다. 이 동작으로 인해 원하는 버전이 제공되기 전에 이전에 요청된 패키지의 설치가 실패할 수 있습니다. 이러한 실패를 방지하고 기존 패키지를 성공적으로 설치하려면 `nuget locals all --clear` 또는 `dotnet nuget locals all --clear` 를 사용하여 설치하기 전에 NuGet 캐시를 지우거나 `-NoCache`나 `--no-cache`과 같은 옵션을 `nuget` 또는 `dotnet`에게 제공하여 `install` 및 `restore` 명령 중에 캐시를 사용하지 않도록 할 수 있습니다.

dotnet

```
dotnet add package packageName --source packageSourceName
```

nuget

```
nuget install packageName -Source packageSourceName
```

특정 버전 패키지 설치 방법

dotnet

```
dotnet add package packageName --version 1.0.0 --source packageSourceName
```

nuget

```
nuget install packageName -Version 1.0.0 -Source packageSourceName
```

자세한 내용은 Microsoft 설명서의 nuget.exe CLI를 사용하여 패키지 관리 또는 dotnet CLI를 사용하여 패키지 설치 및 관리를 참조하십시오.

Nuget.org의 NuGet 패키지를 사용하기

NuGet.org에 대한 외부 연결을 통해 리포지토리를 구성하여 CodeArtifact 리포지토리를 통해 NuGet.org의 NuGet 패키지를 사용할 수 있습니다. Nuget.org에서 사용하는 패키지는 CodeArtifact 리포지토리로 수집하여 저장됩니다. 외부 연결 추가에 대한 자세한 내용은 CodeArtifact 저장소를 공용 저장소에 연결하기을 참조하세요.

NuGet 패키지를 CodeArtifact에 게시하기

CodeArtifact로 NuGet을 구성한 후에는 `nuget` 또는 `dotnet`을 사용하여 CodeArtifact 리포지토리에 패키지 버전을 게시할 수 있습니다.

패키지 버전을 CodeArtifact 리포지토리로 불러오려면 NuGet 구성 파일에 `.nupkg` 파일의 전체 경로와 CodeArtifact 리포지토리의 소스 이름을 포함하여 다음 명령을 실행합니다. `login` 명령을 사용하여 NuGet 구성을 구성한 경우 소스 이름은 `domain_name/repo_name`입니다.

> **Note**
>
> 게시할 패키지가 없는 경우 NuGet 패키지를 생성할 수 있습니다. 자세한 내용은 Microsoft 설명서의 패키지 생성 워크플로를 참조하십시오.

dotnet

```
dotnet nuget push path/to/nupkg/SamplePackage.1.0.0.nupkg --source packageSourceName
```

nuget

```
nuget push path/to/nupkg/SamplePackage.1.0.0.nupkg -Source packageSourceName
```

CodeArtifact NuGet 보안 인증 공급자 참조

CodeArtifact NuGet 보안 인증 공급자를 사용하면 CodeArtifact 리포지토리를 사용하여 NuGet을 쉽게 구성하고 인증할 수 있습니다.

CodeArtifact NuGet 보안 인증 공급자 명령

이 섹션에는 CodeArtifact NuGet 보안 인증 정보 공급자에 대한 명령 목록이 포함되어 있습니다. 다음 예제와 같이 이러한 명령 앞에 접두사 `dotnet codeartifact-creds` 를 붙여야 합니다.

```
dotnet codeartifact-creds command
```

- `configure set profile profile`: 제공된 AWS 프로필을 사용하도록 자격 증명 공급자를 구성합니다.
- `configure unset profile`: 구성된 프로필이 설정된 경우 해당 프로필을 제거합니다.
- `install`: 보안 인증 공급자를 `plugins` 폴더에 복사합니다.
- `install --profile profile`: 자격 증명 공급자를 `plugins` 폴더에 복사하고 제공된 AWS 프로필을 사용하도록 구성합니다.
- `uninstall`: 보안 인증 공급자를 제거합니다. 이렇게 해도 구성 파일에 대한 변경 내용은 제거되지 않습니다.
- `uninstall --delete-configuration`: 보안 인증 공급자를 제거하고 구성 파일의 모든 변경 사항을 제거합니다.

CodeArtifact NuGet 보안 인증 공급자 로그

CodeArtifact NuGet 보안 인증 공급자 로깅을 활성화하려면 사용자 환경에서 로그 파일을 설정해야 합니다. 보안 인증 공급자 로그에는 다음과 같은 유용한 디버깅 정보가 포함되어 있습니다.

- 연결을 만드는 데 사용되는 AWS 프로필
- 모든 인증 오류
- 제공된 엔드포인트가 CodeArtifact URL이 아닌 경우

CodeArtifact NuGet 보안 인증 공급자 로그 파일 설정하기

```
export AWS_CODEARTIFACT_NUGET_LOGFILE=/path/to/file
```

로그 파일이 설정되면 모든 `codeartifact-creds` 명령은 해당 로그 출력을 해당 파일의 내용에 추가합니다.

CodeArtifact NuGet 보안 인증 공급자 버전

다음 표에는 CodeArtifact NuGet 보안 인증 공급자에 관한 버전 기록 정보와 다운로드 링크가 나와 있습니다.

버전	변경 사항	게시 날짜	다운로드 링크(S3)
1.0.2(최신 버전)	업그레이드된 종속성	06/26/2024	v1.0.2 다운로드
1.0.1	net5, net6 및 SSO 프로파일 지원이 추가되었습니다.	03/05/2022	v1.0.1 다운로드
1.0.0	초기 CodeArtifact NuGet 보안 인증 공급자 출시	11/20/2020	v1.0.0 다운로드

NuGet 패키지 이름, 버전 및 자산 이름 표준화

CodeArtifact는 패키지 및 자산 이름과 패키지 버전을 저장하기 전에 표준화합니다. 즉, CodeArtifact의 이름 또는 버전은 패키지 또는 자산이 게시될 때 제공된 것과 다를 수 있습니다.

패키지 이름 표준화: CodeArtifact는 모든 문자를 소문자로 변환하여 NuGet 패키지 이름을 표준화합니다.

패키지 버전 표준화: CodeArtifact는 NuGet과 동일한 패턴을 사용하여 NuGet 패키지 버전을 표준화합니다. 다음 정보는 NuGet 설명서의 표준화된 버전 번호에서 가져왔습니다.

- 버전 번호에서 앞에 오는 0은 제거되었습니다.
  - `1.00`은 `1.0`으로 간주합니다.

  - `1.01.1`은 `1.1.1`으로 간주합니다.
  - `1.00.0.1`은 `1.0.0.1`으로 간주합니다.
- 버전 번호의 네 번째 부분에 있는 0은 생략합니다.
  - `1.0.0.0`은 `1.0.0`으로 간주합니다.
  - `1.0.01.0`은 `1.0.1`으로 간주합니다.
- SemVer 2.0.0 빌드 메타데이터가 제거되었습니다.
  - `1.0.7+r3456`은 `1.0.7`으로 간주합니다.

패키지 에셋 이름 표준화: CodeArtifact는 표준화된 패키지 이름과 패키지 버전을 기반으로 NuGet 패키지 에셋 이름을 생성합니다.

표준화되지 않은 패키지 이름과 버전 이름은 CodeArtifact가 해당 요청의 패키지 이름과 버전 입력을 표준화하므로, API 및 CLI 요청에도 사용할 수 있습니다. 예를 들어, `--package Newtonsoft.JSON`과 `--version 12.0.03.0`를 입력하면 값을 표준화해 표준화된 패키지 이름인 `newtonsoft.json`과 버전 `12.0.3`을 가진 패키지로 반환합니다.

CodeArtifact는 `--asset` 입력은 표준화를 수행하지 않으므로 API 및 CLI 요청에서 표준화한 패키지 자산 이름을 사용해야 합니다.

ARN에서는 표준화한 이름과 버전을 사용해야 합니다.

패키지의 표준화된 이름을 찾으려면 `aws codeartifact list-packages` 명령을 사용합니다. 자세한 내용은 패키지 이름 나열을 참조하세요.

표준화되지 않은 패키지 이름을 찾으려면 `aws codeartifact describe-package-version` 명령을 사용합니다. 표준화되지 않은 패키지 이름이 `displayName` 필드에 반환됩니다. 자세한 내용은 패키지 버전 세부 정보 및 종속성 보기 및 업데이트 섹션을 참조하세요.

NuGet 호환성

이 안내서에는 CodeArtifact의 다양한 NuGet 도구 및 버전과의 호환성에 관한 정보가 포함되어 있습니다.

주제

- 일반 NuGet 호환성
- NuGet 명령줄 지원

일반 NuGet 호환성

AWS CodeArtifact는 NuGet 4.8 이상을 지원합니다.

AWS CodeArtifact는 NuGet HTTP 프로토콜의 V3만 지원합니다. 즉, 프로토콜의 V2를 사용하는 일부 CLI 명령은 지원하지 않습니다. 자세한 내용은 nuget.exe 명령 지원 섹션을 참조하세요.

AWS CodeArtifact는 PowerShellGet 2.x를 지원하지 않습니다.

NuGet 명령줄 지원

AWS CodeArtifact는 NuGet(`nuget.exe`) 및 .NET Core(`dotnet`) CLI 도구를 지원합니다.

nuget.exe 명령 지원

CodeArtifact는 NuGet HTTP 프로토콜 V3만 지원하므로 CodeArtifact 리소스에 대해 다음 명령을 사용해도 작동하지 않습니다.

- `list`: 이 `nuget list` 명령은 지정된 소스의 패키지 목록을 표시합니다. CodeArtifact 리포지토리의 패키지 목록을 가져오려면 AWS CLI의 패키지 이름 나열 명령을 사용하세요.

CodeArtifact를 Python과 함께 사용

이 항목에서는 CodeArtifact와 함께 Python 패키지 관리자인 `pip` , `twine` 및 Python 패키지 게시 유틸리티를 사용하는 방법에 대해 설명합니다.

주제

CodeArtifact로 pip 구성 및 사용

pip는 Python 패키지용 패키지 설치관리자 프로그램입니다. pip를 사용하여 CodeArtifact 리포지토리에서 Python 패키지를 설치하려면 먼저 CodeArtifact 리포지토리 정보와 보안 인증 정보로 pip 클라이언트를 구성해야 합니다.

pip는 Python 패키지를 설치하는 용도로만 사용할 수 있습니다. Python 패키지를 게시하기 위해 twine을 사용할 수 있습니다. 자세한 내용은 CodeArtifact로 twine 구성 및 사용 단원을 참조하십시오.

`login` 명령으로 pip 구성

먼저에 설명된 AWS CLI대로와 함께 사용할 자격 AWS 증명을 구성합니다CodeArtifact 시작하기. 그런 다음 CodeArtifact `login` 명령을 사용하여 보안 인증을 가져오고 해당 인증을 사용하여 `pip`를 구성합니다.

> **Note**
>
> 소유한 도메인의 리포지토리에 액세스하는 경우 `--domain-owner`를 포함하지 않아도 됩니다. 자세한 내용은 크로스 계정 도메인 단원을 참조하십시오.

pip를 구성하려면 다음 명령을 실행합니다.

```
aws codeartifact login --tool pip --domain my_domain --domain-owner 111122223333 --
repository my_repo
```

login는 AWS 자격 증명을 사용하여 CodeArtifact에서 권한 부여 토큰을 가져옵니다. login 명령은 --repository 옵션을 통해 지정된 리포지토리로 index-url를 설정하도록 ~/.config/pip/pip.conf를 편집하여 CodeArtifact와 함께 사용할 수 있도록 pip를 구성합니다.

login 직접 호출 후의 기본 승인 기간은 12시간이며, 토큰을 주기적으로 새로 고치려면 login을 직접적으로 호출해야 합니다. login 명령으로 만든 인증 토큰에 관한 자세한 내용은 login 명령으로 생성된 토큰을 참조하세요.

로그인 명령 없이 pip를 구성하려면

login 명령을 사용하여 pip를 구성할 수 없는 경우 pip config를 사용할 수 있습니다.

1. AWS CLI 를 사용하여 새 권한 부여 토큰을 가져옵니다.

   > **Note**
   >
   > 소유한 도메인의 리포지토리에 액세스하는 경우 --domain-owner를 포함하지 않아도 됩니다. 자세한 내용은 크로스 계정 도메인 단원을 참조하십시오.

   ```
   CODEARTIFACT_AUTH_TOKEN=`aws codeartifact get-authorization-token --
   domain my_domain --domain-owner 111122223333 --query authorizationToken --output
    text`
   ```

2. pip config를 사용하여 CodeArtifact 레지스트리 URL 및 보안 인증을 설정합니다. 다음 명령은 현재 환경 구성 파일만 업데이트합니다. 시스템 전체 구성 파일을 업데이트하려면 site를 global로 바꿔야 합니다.

   ```
   pip config set site.index-url https://aws:
   $CODEARTIFACT_AUTH_TOKEN@my_domain-
   111122223333.d.codeartifact.region.amazonaws.com/pypi/my_repo/simple/
   ```

> **Note**
>
> 듀얼 스택 엔드포인트를 사용하려면 `codeartifact.region.on.aws` 엔드포인트를 사용합니다.

> **Important**
>
> 레지스트리 URL은 슬래시(/)로 끝나야 합니다. 그렇지 않으면 리포지토리에 연결할 수 없습니다.

pip 구성 파일 예

다음은 CodeArtifact 레지스트리 URL 및 보안 인증을 설정한 후의 `pip.conf` 파일 예입니다.

```
[global]
index-url = https://aws:eyJ2ZX...@my_domain-111122223333.d.codeartifact.us-
west-2.amazonaws.com/pypi/my_repo/simple/
```

pip 실행

`pip` 명령을 실행하려면 CodeArtifact를 사용하여 `pip`를 구성해야 합니다. 자세한 내용은 다음 문서를 참조하세요.

1. with AWS CodeArtifact 설정 섹션의 단계에 따라 AWS 계정, 도구 및 권한을 구성합니다.
2. CodeArtifact로 twine 구성 및 사용의 단계에 따라 `twine`을 구성합니다.

패키지가 리포지토리 또는 업스트림 리포지토리 중 하나에 있다고 할 경우, `pip install`을 사용하여 패키지를 설치할 수 있습니다. 예를 들어 다음 명령을 사용하여 `requests` 패키지를 설치할 수 있습니다.

```
pip install requests
```

`-i` 옵션을 사용하여 CodeArtifact 리포지토리 대신 https://pypi.org에서 패키지를 설치하는 것으로 일시적으로 되돌릴 수 있습니다.

```
pip install -i https://pypi.org/simple requests
```

CodeArtifact로 twine 구성 및 사용

twine은 Python 패키지용 패키지 게시 유틸리티입니다. twine을 사용하여 CodeArtifact 리포지토리에 Python 패키지를 게시하려면 먼저 CodeArtifact 리포지토리 정보와 보안 인증 정보로 twine을 구성해야 합니다.

twine은 Python 패키지를 게시하는 용도로만 사용할 수 있습니다. Python 패키지를 설치하기 위해 pip를 사용할 수 있습니다. 자세한 내용은 CodeArtifact로 pip 구성 및 사용 단원을 참조하십시오.

`login` 명령으로 twine 구성

먼저에 설명된 AWS CLI대로와 함께 사용할 자격 AWS 증명을 구성합니다CodeArtifact 시작하기. 그런 다음 CodeArtifact `login` 명령을 사용하여 보안 인증을 가져오고 해당 인증을 사용하여 twine을 구성합니다.

> **Note**
>
> 소유한 도메인의 리포지토리에 액세스하는 경우 `--domain-owner`를 포함하지 않아도 됩니다. 자세한 내용은 크로스 계정 도메인 단원을 참조하십시오.

twine을 구성하려면 다음 명령을 실행합니다.

```
aws codeartifact login --tool twine --domain my_domain --domain-owner 111122223333 --
repository my_repo
```

`login`는 AWS 자격 증명을 사용하여 CodeArtifact에서 권한 부여 토큰을 가져옵니다. `login` 명령은 `--repository` 옵션에서 지정한 리포지토리를 보안 인증과 함께 추가하도록 `~/.pypirc`를 편집하여 CodeArtifact와 함께 사용할 twine을 구성합니다.

`login` 직접 호출 후의 기본 승인 기간은 12시간이며, 토큰을 주기적으로 새로 고치려면 `login`을 직접적으로 호출해야 합니다. `login` 명령으로 만든 인증 토큰에 관한 자세한 내용은 `login` 명령으로 생성된 토큰을 참조하세요.

`login` 명령 없이 twine 구성

`login` 명령을 사용하여 twine을 구성할 수 없는 경우 `~/.pypirc` 파일 또는 환경 변수를 사용할 수 있습니다. `~/.pypirc` 파일을 사용하려면 다음 항목을 파일에 추가합니다. 암호는 `get-authorization-token` API에서 획득한 인증 토큰이어야 합니다.

```
[distutils]
index-servers =
 codeartifact
[codeartifact]
repository = https://my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/
pypi/my_repo/
password = auth-token
username = aws
```

> **Note**
>
> 듀얼 스택 엔드포인트를 사용하려면 `codeartifact.region.on.aws` 엔드포인트를 사용합니다.

환경 변수를 사용하려면 다음을 수행합니다.

> **Note**
>
> 자신이 소유한 도메인의 리포지토리에 액세스하는 경우 `--domain-owner`를 포함하지 않아도 됩니다. 자세한 내용은 크로스 계정 도메인 단원을 참조하십시오.

```
export TWINE_USERNAME=aws
export TWINE_PASSWORD=`aws codeartifact get-authorization-token --domain my_domain --
domain-owner 111122223333 --query authorizationToken --output text`
export TWINE_REPOSITORY_URL=`aws codeartifact get-repository-endpoint --
domain my_domain --domain-owner 111122223333 --repository my_repo --format pypi --query
 repositoryEndpoint --output text`
```

twine 실행

twine을 사용하여 Python 패키지 자산을 게시하려면 먼저 CodeArtifact 권한 및 리소스를 구성해야 합니다.

1. with AWS CodeArtifact 설정 섹션의 단계에 따라 AWS 계정, 도구 및 권한을 구성합니다.
2. `login` 명령으로 twine 구성 또는 `login` 명령 없이 twine 구성 의 단계에 따라 twine을 구성합니다.

twine을 구성한 후 `twine` 명령을 실행할 수 있습니다. 다음 명령을 사용하여 Python 패키지 자산을 게시합니다.

```
twine upload --repository codeartifact mypackage-1.0.tgz
```

Python 애플리케이션을 빌드하고 패키징하는 방법에 관한 자세한 내용은 Python Packaging Authority 웹 사이트에서 Generating Distribution Archives를 참조하세요.

Python 패키지 이름 정규화

CodeArtifact는 패키지 이름을 저장하기 전에 정규화합니다. 즉, CodeArtifact의 패키지 이름은 패키지가 게시될 때 제공된 이름과 다를 수 있습니다.

Python 패키지의 경우, 정규화를 수행할 때 패키지 이름은 소문자와 `.`, `-` 및 `_` 문자의 모든 인스턴스가 단일 `-` 문자로 대체됩니다. 따라서 `pigeon_cli` 및 `pigeon.cli` 패키지 이름은 `pigeon-cli`과 같이 정규화되어 저장됩니다. 정규화되지 않은 이름은 pip 및 twine에서 사용할 수 있지만 CodeArtifact CLI 또는 API 요청(예: `list-package-versions`) 및 ARN에서는 정규화된 이름을 사용해야 합니다. Python 패키지 이름 정규화에 관한 자세한 내용은 Python 설명서에서 PEP 503을 참조하세요.

Python 호환성

CodeArtifact는 PyPI의 `XML-RPC` 또는 `JSON` API를 지원하지 않습니다.

CodeArtifact는 `simple` API를 제외한 PyPI의 `Legacy` API를 지원합니다. CodeArtifact는 `/simple/` API 엔드포인트를 지원하지 않지만 `/simple/<project>/` 엔드포인트는 지원합니다.

자세한 내용은 Python Packaging Authority의 GitHub 리포지토리에서 다음 섹션을 참조하세요.

- XML-RPC API
- JSON API

- Legacy API

pip 명령 지원

다음 섹션에는 지원되지 않는 특정 명령 외에도 CodeArtifact 리포지토리에서 지원하는 pip 명령이 요약되어 있습니다.

주제

- 리포지토리와 상호 작용하는 지원되는 명령
- 지원되는 클라이언트 측 명령

리포지토리와 상호 작용하는 지원되는 명령

이 섹션에는 `pip` 클라이언트가 구성된 레지스트리에 하나 이상의 요청을 보내는 `pip` 명령이 나열되어 있습니다. CodeArtifact 리포지토리에 대해 이러한 명령을 호출했을 때 제대로 작동하는 것으로 확인되었습니다.

명령	설명
install	패키지를 설치합니다.
download	패키지를 다운로드합니다.

CodeArtifact는 `pip search`를 실행하지 않습니다. `pip`를 CodeArtifact 리포지토리로 구성한 경우 `pip search`를 실행하면 PyPI에서 패키지를 검색하고 표시합니다.

지원되는 클라이언트 측 명령

이러한 명령은 리포지토리와 직접 상호 작용할 필요가 없으므로 CodeArtifact는 명령을 지원하기 위해 아무 것도 할 필요가 없습니다.

명령	설명
uninstall	패키지를 제거합니다.
freeze	설치된 패키지를 요구 사항 형식으로 출력합니다.

명령	설명
list	설치된 패키지를 나열합니다.
show	설치된 패키지에 대한 정보를 표시합니다.
check	설치된 패키지에 호환되는 종속성이 있는지 확인합니다.
config	로컬 및 글로벌 구성을 관리합니다.
wheel	요구 사항에 맞게 휠을 빌드합니다.
hash	패키지 아카이브의 해시를 계산합니다.
completion	명령 완성을 돕습니다.
debug	디버깅에 유용한 정보를 표시합니다.
help	명령에 대한 도움말을 표시합니다.

업스트림 및 외부 연결에서 Python 패키지 요청하기

pypi.org에서 파이썬 패키지 버전을 불러올 때, CodeArtifact는 해당 패키지 버전의 모든 에셋을 불러옵니다. 대부분의 Python 패키지에는 적은 수의 에셋이 포함되어 있지만 일부 패키지는 일반적으로 여러 하드웨어 구조와 Python 인터프리터를 지원하기 위해 100개가 넘는 에셋을 포함합니다.

기존 패키지 버전의 경우 새 에셋이 pypi.org에 게시되는 것이 일반적입니다. 예를 들어, 일부 프로젝트는 새 버전의 Python이 출시되면 새 에셋을 게시합니다. `pip install`로 CodeArtifact에서 Python 패키지를 설치하면 CodeArtifact 저장소에 유지 중인 패키지 버전이 pypi.org의 최신 자산 세트를 반영하도록 업데이트됩니다.

마찬가지로, 현재 CodeArtifact 리포지토리에 없는 업스트림 CodeArtifact 리포지토리의 패키지 버전에 새 자산을 사용할 수 있는 경우에는 새 자산이 `pip install` 실행 시 현재 리포지토리에 보존됩니다.

삭제된 패키지 버전

pypi.org의 일부 패키지 버전은 삭제된 것으로 표시되는데, 이는 패키지 설치 프로그램 (예: pip) 에게 버전 지정자와 일치하는 유일한 버전이 아니면 설치하지 말아야 한다고 알려주는 역할을 합니다(==또는 === 사용). 자세한 내용은 PEP_592를 참조하십시오.

CodeArtifact의 패키지 버전이 원래 pypi.org에 대한 외부 연결에서 가져온 경우 CodeArtifact 리포지토리에서 패키지 버전을 설치하면 CodeArtifact는 패키지 버전의 업데이트된 삭제 메타데이터를 pypi.org에서 가져오는지 확인합니다.

패키지 버전이 삭제되었는지 확인하는 방법

CodeArtifact에서 패키지 버전이 삭제되었는지 확인하려면 `pip install packageName===packageVersion`를 사용하여 설치를 시도하세요. 패키지 버전이 삭제되었다면 다음과 유사한 경고 메시지가 표시됩니다.

```
WARNING: The candidate selected for download or install is a yanked version
```

pypi.org에서 패키지 버전이 삭제되었는지 확인하려면 `https://pypi.org/project/packageName/packageVersion/`에서 패키지 버전의 pypi.org 목록을 확인하세요.

비공개 패키지에 제거 생태 설정하기

CodeArtifact는 CodeArtifact 리포지토리에 직접 게시된 패키지의 제거된 메타데이터 설정을 지원하지 않습니다.

CodeArtifact가 패키지 버전의 제거된 최신 메타데이터 또는 자산을 가져오지 않는 이유는 무엇입니까?

일반적으로 CodeArtifact는 CodeArtifact 리포지토리에서 Python 패키지 버전을 가져올 때 삭제한 메타데이터를 pypi.org의 최신 상태로 유지되도록 합니다. 또한 패키지 버전의 자산 목록도 pypi.org 및 업스트림 CodeArtifact 리포지토리의 최신 세트로 업데이트됩니다. 이는 패키지 버전을 처음 설치하고 CodeArtifact가 pypi.org에서 CodeArtifact 리포지토리로 가져오는 경우나 이전에 패키지를 설치한 적이 있는 경우 모두 해당됩니다. 하지만 pip와 같은 패키지 관리자 클라이언트가 pypi.org 또는 업스트림 리포지토리에서 제거된 최신 메타데이터를 가져오지 않는 경우도 있습니다. 대신, CodeArtifact는 리포지토리에 이미 저장된 메타데이터를 반환합니다. 이 섹션에서는 이 문제가 발생할 수 있는 세 가지 방법에 대해 설명하겠습니다.

업스트림 구성: disassociate-external-connection을 사용하여 pypi.org에 대한 외부 연결을 리포지토리 또는 업스트림에서 제거하면 제거된 메타데이터가 더 이상 pypi.org에서 새로 고침되지 않습니다. 마찬가지로 업스트림 리포지토리를 제거하면 제거된 리포지토리의 자산과 제거된 리포지토리의 업스트림은 현재 리포지토리에서 더 이상 사용할 수 없습니다. CodeArtifact 패키지 원본 컨트롤을 사용하여 특정 패키지의 새 버전을 가져오는 것을 방지하는 경우에도 마찬가지입니다. 이 `upstream=BLOCK` 설정은 삭제된 메타데이터를 갱신하는 것을 차단합니다.

패키지 버전 상태: 패키지 버전의 상태를 Published 또는 Unlisted를 제외한 상태로 설정하면 해당 패키지 버전의 제거된 메타데이터와 자산은 새로 고침되지 않습니다. 마찬가지로 특정 패키지 버전(예:`torch 2.0.1`)을 가져오는데 업스트림 리포지토리에 동일한 패키지 버전이 있고 그 상태가 Published 또는 Unlisted가 아닌 경우에는 제거된 메타데이터 및 자산의 업스트림 리포지토리에서 현재 리포지토리로의 전파가 차단됩니다. 이것은 다른 패키지 버전 상태가 해당 버전을 더 이상 어떤 리포지토리에서도 사용할 수 없음을 가리키기 때문입니다.

직접 게시: 특정 패키지 버전을 CodeArtifact 리포지토리에 직접 게시하면 해당 패키지 버전의 제거된 메타데이터와 자산이 업스트림 리포지토리 및 pypi.org에서 새로 고침되지 않습니다. 예를 들어 웹 브라우저를 사용하여 패키지 `torch-2.0.1-cp311-none-macosx_11_0_arm64.whl` 버전에서 에셋 `torch 2.0.1`을 다운로드한 다음 twine을 사용하여 CodeArtifact 리포지토리에 `torch 2.0.1`을 게시한다고 가정해 보겠습니다. CodeArtifact는 패키지 버전이 pypi.org 외부 연결 또는 업스트림 리포지토리가 아닌 리포지토리에 직접 게시하여 도메인에 진입했음을 인지하고 추적합니다. 이 경우 CodeArtifact는 제거된 메타데이터를 업스트림 리포지토리 또는 pypi.org와 동기화된 상태로 유지하지 않습니다. 업스트림 리포지토리에 `torch 2.0.1`을 게시하는 경우에도 마찬가지입니다. 해당 패키지 버전이 있으면 제거된 메타데이터와 자산이 업스트림 그래프 아래의 리포지토리로 전파되는 것이 차단됩니다.

CodeArtifact를 Ruby와 함께 사용

다음 항목에서는 RubyGems 및 Bundler 도구를 CodeArtifact와 함께 사용하여 Ruby gem을 설치하고 게시하는 방법을 설명합니다.

> **Note**
>
> CodeArtifact는 Ruby 3.3 이상을 권장하며 Ruby 2.6 이하에서는 작동하지 않습니다.

주제

CodeArtifact로 RubyGems와 Bundler 구성 및 사용

CodeArtifact에서 리포지토리를 생성한 후 RubyGems(`gem`) 및 Bundler(`bundle`)를 사용하여 gem을 설치하고 게시할 수 있습니다. 이 항목에서는 CodeArtifact 리포지토리를 인증하고 사용하도록 패키지 관리자를 구성하는 방법을 설명합니다.

CodeArtifact로 RubyGems(**`gem`**) 및 Bundler(**`bundle`**) 구성

RubyGems(`gem`) 또는 Bundler(`bundle`)를 사용하여 AWS CodeArtifact에 gem을 게시하거나 CodeArtifact에서 gem을 사용하려면 먼저 액세스할 수 있는 자격 증명을 포함한 CodeArtifact 리포지토리 정보로 구성해야 합니다. 다음 절차 중 하나의 단계에 따라 CodeArtifact 리포지토리 엔드포인트 정보 및 자격 증명으로 `gem` 및 `bundle` CLI 도구를 구성합니다.

콘솔 지침을 사용하여 RubyGems 및 Bundler 구성

콘솔의 구성 지침을 사용하여 Ruby 패키지 관리자를 CodeArtifact 리포지토리에 연결할 수 있습니다. 콘솔 지침은 CodeArtifact 정보를 찾아 입력할 필요 없이 패키지 관리자를 설정하기 위해 실행할 수 있는 사용자 지정 명령을 제공합니다.

1. https://console.aws.amazon.com/codesuite/codeartifact/home에서 AWS CodeArtifact 콘솔을 엽니다.

2. 탐색 창에서 리포지토리를 선택한 다음, Ruby gem을 설치하거나 게시하는 데 사용할 리포지토리를 선택합니다.
3. 연결 지침 보기를 선택합니다.
4. 운영 체제를 선택합니다.
5. CodeArtifact 리포지토리로 구성할 Ruby 패키지 관리자 클라이언트를 선택합니다.
6. 생성된 지침에 따라 Ruby gem을 리포지토리에서 설치하거나 Ruby gem을 리포지토리에 게시하도록 패키지 관리자 클라이언트를 구성합니다.

RubyGems 및 Bundler 수동 구성

콘솔의 구성 지침을 사용할 수 없거나 사용하지 않으려는 경우 다음 지침을 사용하여 Ruby 패키지 관리자를 CodeArtifact 리포지토리에 수동으로 연결할 수 있습니다.

1. 명령줄에서 다음 명령을 사용하여 CodeArtifact 인증 토큰을 가져와 환경 변수에 저장합니다.

   - *my_domain*을 CodeArtifact 도메인 이름으로 변경합니다.
   - *111122223333*을 도메인 소유자의 AWS 계정 ID로 변경합니다. 소유한 도메인의 리포지토리에 액세스하는 경우 `--domain-owner`를 포함하지 않아도 됩니다. 자세한 내용은 크로스 계정 도메인 섹션을 참조하세요.

   macOS and Linux

   ```
   export CODEARTIFACT_AUTH_TOKEN=`aws codeartifact get-authorization-token --
   domain my_domain --domain-owner 111122223333 --query authorizationToken --output
    text`
   ```

   Windows

   - Windows(기본 명령 쉘 사용):

     ```
     for /f %i in ('aws codeartifact get-authorization-token --domain my_domain --
     domain-owner 111122223333 --query authorizationToken --output text') do set
      CODEARTIFACT_AUTH_TOKEN=%i
     ```

   - Windows PowerShell:

```
$env:CODEARTIFACT_AUTH_TOKEN = aws codeartifact get-authorization-token --
domain my_domain --domain-owner 111122223333 --query authorizationToken --
output text
```

2. Ruby gem을 리포지토리에 게시하려면 다음 명령을 사용하여 CodeArtifact 리포지토리의 엔드포인트를 가져와 RUBYGEMS_HOST 환경 변수에 저장합니다. gem CLI는 이 환경 변수를 사용하여 gem이 게시되는 위치를 결정합니다.

> **Note**
>
> 또는 RUBYGEMS_HOST 환경 변수를 사용하는 대신 gem push 명령을 사용할 때 리포지토리 엔드포인트에 --host 옵션을 제공할 수 있습니다.

- *my_domain*을 CodeArtifact 도메인 이름으로 변경합니다.
- *111122223333*을 도메인 소유자의 AWS 계정 ID로 변경합니다. 소유한 도메인의 리포지토리에 액세스하는 경우 --domain-owner를 포함하지 않아도 됩니다. 자세한 내용은 크로스 계정 도메인 섹션을 참조하세요.
- *my_repo*를 CodeArtifact 리포지토리 이름으로 변경합니다.

macOS and Linux

```
export RUBYGEMS_HOST=`aws codeartifact get-repository-endpoint --
domain my_domain --domain-owner 111122223333 --repository my_repo --format ruby
 --query repositoryEndpoint --output text | sed 's:/*$::'`
```

Windows

다음 명령은 리포지토리 엔드포인트를 검색하고 후행 /를 잘라낸 다음, 환경 변수에 저장합니다.

- Windows(기본 명령 쉘 사용):

```
for /f %i in ('aws codeartifact get-repository-endpoint --domain my_domain
 --domain-owner 111122223333 --repository my_repo --format ruby --query
 repositoryEndpoint --output text') do set RUBYGEMS_HOST=%i
```

```
set RUBYGEMS_HOST=%RUBYGEMS_HOST:~0,-1%
```

- Windows PowerShell:

```
$env:RUBYGEMS_HOST = (aws codeartifact get-repository-endpoint --
domain my_domain --domain-owner 111122223333 --repository my_repo --format
 ruby --query repositoryEndpoint --output text).TrimEnd("/")
```

다음 URL은 리포지토리 엔드포인트의 예입니다.

```
https://my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/ruby/my_repo/
```

> **Note**
>
> 듀얼 스택 엔드포인트를 사용하려면 `codeartifact.region.on.aws` 엔드포인트를 사용합니다.

3. Ruby gem을 리포지토리에 게시하려면 인증 토큰을 포함하도록 `~/.gem/credentials` 파일을 편집하여 RubyGems로 CodeArtifact에 인증해야 합니다. `~/.gem/` 디렉터리 또는 `~/.gem/credentials` 파일이 없는 경우 해당 디렉터리와 파일을 생성합니다.

   macOS and Linux

   ```
   echo ":codeartifact_api_key: Bearer $CODEARTIFACT_AUTH_TOKEN" >> ~/.gem/
   credentials
   ```

   Windows

   - Windows(기본 명령 쉘 사용):

     ```
     echo :codeartifact_api_key: Bearer %CODEARTIFACT_AUTH_TOKEN% >> %USERPROFILE
     %/.gem/credentials
     ```

   - Windows PowerShell:

     ```
     echo ":codeartifact_api_key: Bearer $env:CODEARTIFACT_AUTH_TOKEN" | Add-
     Content ~/.gem/credentials
     ```

4. gem을 사용하여 리포지토리에서 Ruby gem을 설치하려면 리포지토리 엔드포인트 정보와 인증 토큰을 .gemrc 파일에 추가해야 합니다. 전역 파일(~/.gemrc) 또는 프로젝트 .gemrc 파일에 추가할 수 있습니다. .gemrc에 추가해야 하는 CodeArtifact 정보는 리포지토리 엔드포인트와 인증 토큰의 조합입니다. 형식은 다음과 같습니다.

```
https://aws:${CODEARTIFACT_AUTH_TOKEN}@my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/ruby/my_repo/
```

- 인증 토큰의 경우 이전 단계에서 설정한 CODEARTIFACT_AUTH_TOKEN 환경 변수를 사용할 수 있습니다.
- 리포지토리 엔드포인트를 가져오려면 이전에 설정한 RUBYGEMS_HOST 환경 변수의 값을 읽거나 다음 get-repository-endpoint 명령을 사용하여 필요에 따라 값을 대체할 수 있습니다.

```
aws codeartifact get-repository-endpoint --domain my_domain --domain-owner 111122223333 --repository my_repo --format ruby --query repositoryEndpoint --output text
```

엔드포인트가 있으면 텍스트 편집기를 사용하여 적절한 위치에 aws:${CODEARTIFACT_AUTH_TOKEN}@를 추가합니다. 리포지토리 엔드포인트와 인증 토큰 문자열이 생성되면 다음과 같이 echo 명령을 사용하여 .gemrc 파일의 :sources: 섹션에 추가합니다.

> **Warning**
>
> CodeArtifact는 gem sources -add 명령을 사용하여 리포지토리를 소스로 추가하는 것을 지원하지 않습니다. 소스는 파일에 직접 추가해야 합니다.

macOS and Linux

```
echo ":sources:
    - https://aws:${CODEARTIFACT_AUTH_TOKEN}@my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/ruby/my_repo/" > ~/.gemrc
```

Windows

- Windows(기본 명령 쉘 사용):

```
echo ":sources:
    - https://aws:%CODEARTIFACT_AUTH_TOKEN
%@my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/ruby/my_repo/"
 > "%USERPROFILE%\.gemrc"
```

- Windows PowerShell:

```
echo ":sources:
    - https://aws:
$env:CODEARTIFACT_AUTH_TOKEN@my_domain-111122223333.d.codeartifact.us-
west-2.amazonaws.com/ruby/my_repo/" | Add-Content ~/.gemrc
```

5. Bundler를 사용하려면 다음 `bundle config` 명령을 실행하여 리포지토리 엔드포인트 URL 및 인증 토큰으로 Bundler를 구성해야 합니다.

   macOS and Linux

   ```
   bundle config $RUBYGEMS_HOST aws:$CODEARTIFACT_AUTH_TOKEN
   ```

   Windows

   - Windows(기본 명령 쉘 사용):

     ```
     bundle config %RUBYGEMS_HOST% aws:%CODEARTIFACT_AUTH_TOKEN%
     ```

   - Windows PowerShell:

     ```
     bundle config $Env:RUBYGEMS_HOST aws:$Env:CODEARTIFACT_AUTH_TOKEN
     ```

이제 CodeArtifact 리포지토리로 RubyGems(`gem`) 및 Bundler(`bundle`)를 구성했으므로 이를 사용하여 해당 리포지토리에서 Ruby gem을 게시하고 사용할 수 있습니다.

CodeArtifact에서 Ruby gem 설치

다음 절차에 따라 `gem` 또는 `bundle` CLI 도구를 사용하여 CodeArtifact 리포지토리에서 Ruby gem을 설치합니다.

`gem`을 사용하여 Ruby gem 설치

RubyGems(gem) CLI를 사용하여 CodeArtifact 리포지토리에서 특정 버전의 Ruby gem을 빠르게 설치할 수 있습니다.

`gem`을 사용하여 CodeArtifact 리포지토리에서 Ruby gem을 설치하려면

1. 아직 구성하지 않았다면 CodeArtifact로 RubyGems(gem) 및 Bundler(bundle) 구성의 단계에 따라 적절한 자격 증명과 함께 CodeArtifact 리포지토리를 사용하도록 `gem` CLI를 구성합니다.

   > **Note**
   >
   > 생성된 권한 부여 토큰은 12시간 동안 유효합니다. 토큰이 생성된 후 12시간이 경과한 경우 새 토큰을 생성해야 합니다.

2. CodeArtifact에서 Ruby gem을 설치하려면 다음 명령을 사용합니다.

   ```
   gem install my_ruby_gem --version 1.0.0
   ```

`bundle`을 사용하여 Ruby gem 설치

Bundler(`bundle`) CLI를 사용하여 `Gemfile`에 구성된 Ruby gem을 설치할 수 있습니다.

`bundle`을 사용하여 CodeArtifact 리포지토리에서 Ruby gem을 설치하려면

1. 아직 구성하지 않았다면 CodeArtifact로 RubyGems(gem) 및 Bundler(bundle) 구성의 단계에 따라 적절한 자격 증명과 함께 CodeArtifact 리포지토리를 사용하도록 `bundle` CLI를 구성합니다.

   > **Note**
   >
   > 생성된 권한 부여 토큰은 12시간 동안 유효합니다. 토큰이 생성된 후 12시간이 경과한 경우 새 토큰을 생성해야 합니다.

2. CodeArtifact 리포지토리 엔드포인트 URL을 `Gemfile`에 `source`로 추가하여 CodeArtifact 리포지토리 및 업스트림에서 구성된 Ruby gem을 설치합니다.

   ```
   source "https://my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/
   ruby/my_repo/"
   ```

```
gem 'my_ruby_gem'
```

3. 다음 명령을 사용하여 Gemfile에 지정된 대로 Ruby gem을 설치합니다.

```
bundle install
```

CodeArtifact에 Ruby gem 게시

다음 절차에 따라 gem CLI를 사용하여 Ruby gem을 CodeArtifact 리포지토리에 게시합니다.

1. 아직 구성하지 않았다면 CodeArtifact로 RubyGems(gem) 및 Bundler(bundle) 구성의 단계에 따라 적절한 자격 증명과 함께 CodeArtifact 리포지토리를 사용하도록 gem CLI를 구성합니다.

> **Note**
>
> 생성된 권한 부여 토큰은 12시간 동안 유효합니다. 토큰이 생성된 후 12시간이 경과한 경우 새 토큰을 생성해야 합니다.

2. 다음 명령을 사용하여 Ruby gem을 CodeArtifact 리포지토리에 게시합니다. RUBYGEMS_HOST 환경 변수를 설정하지 않은 경우 --host 옵션에 CodeArtifact 리포지토리 엔드포인트를 제공해야 합니다.

```
gem push --key codeartifact_api_key my_ruby_gem-0.0.1.gem
```

RubyGems 명령 지원

CodeArtifact는 gem install 및 gem push 명령을 지원합니다. CodeArtifact는 다음 gem 명령을 지원하지 않습니다.

- gem fetch
- gem info --remote
- gem list --remote
- gem mirror
- gem outdated
- gem owner
- gem query

- `gem search`
- `gem signin`
- `gem signout`
- `gem sources --add`
- `gem sources --update`
- `gem specification --remote`
- `gem update`
- `gem yank`

Bundler 호환성

이 안내서에는 CodeArtifact와 Bundler의 호환성에 관한 정보가 포함되어 있습니다.

Bundler 호환성

AWS CodeArtifact는 Bundler 2.4.11 이상을 권장합니다. 설치에 문제가 발생하면 Bundler CLI를 최신 버전으로 업데이트합니다.

Bundler 버전 지원

2.4.11보다 낮은 Bundler 버전에서는 Bundler가 전체 인덱스 `specs.4.8.gz`를 쿼리하기로 결정하기 전에 Gemfile에 정의할 수 있는 종속성이 500개로 제한됩니다. CodeArtifact는 전체 인덱스를 지원하지 않으므로 2.4.11보다 낮은 Bundler 버전을 사용할 경우 500개 이상의 종속성을 지정하면 CodeArtifact에서 작동하지 않습니다.

CodeArtifact를 사용하여 Gemfile에서 500개 이상의 종속성을 정의하려면 Bundler를 버전 2.4.11 이상으로 업데이트합니다.

Bundler 작업 지원

RubyGems에 대한 CodeArtifact의 지원에는 Bundler Compact Index API가 포함되지 않습니다(`/versions` API는 지원되지 않음). CodeArtifact는 종속성 API만 지원합니다.

또한 CodeArtifact는 `specs.4.8.gz`와 같은 다양한 사양의 API를 지원하지 않습니다.

CodeArtifact를 Swift와 함께 사용

이 항목에서는 CodeArtifact와 함께 Swift 패키지 관리자를 사용하여 Swift 패키지를 설치하고 게시하는 방법을 설명합니다.

> **Note**
>
> CodeArtifact는 Swift 5.8 이상 및 Xcode 14.3 이상을 지원합니다.
> CodeArtifact는 Swift 5.9 이상 및 Xcode 15 이상을 권장합니다.

주제

CodeArtifact를 사용하여 Swift Package Manager 설정

Swift Package Manager를 사용하여 AWS CodeArtifact에 패키지를 게시하거나 사용하려면 먼저 CodeArtifact 리포지토리에 액세스하기 위한 자격 증명을 설정해야 합니다. CodeArtifact 보안 인증 및 리포지토리 엔드포인트를 사용하여 Swift Package Manager CLI를 구성하는 권장 방법은 `aws codeartifact login` 명령을 사용하는 것입니다. Swift Package Manager를 수동으로 구성할 수도 있습니다.

로그인 명령으로 Swift 구성

`aws codeartifact login` 명령을 사용하여 CodeArtifact로 Swift Package Manager를 구성합니다.

> **Note**
>
> 로그인 명령을 사용하려면 Swift 5.8 이상이 필요하며 Swift 5.9 이상을 사용하는 것이 좋습니다.

`aws codeartifact login` 명령은 다음을 수행합니다.

1. CodeArtifact에서 인증 토큰을 가져와 사용자 환경에 저장합니다. 보안 인증이 저장되는 방식은 환경의 운영 체제에 따라 달라집니다.

   a. macOS: macOS 키체인 애플리케이션에 항목이 생성됩니다.

   b. Linux 및 Windows: `~/.netrc` 파일에 항목이 생성됩니다.

   모든 운영 체제에서 보안 인증 항목이 있는 경우 이 명령은 해당 항목을 새 토큰으로 대체합니다.
2. CodeArtifact 리포지토리 엔드포인트 URL을 가져와서 Swift 구성 파일에 추가합니다. 이 명령은 `/path/to/project/.swiftpm/configuration/registries.json`에 있는 프로젝트 수준 구성 파일에 리포지토리 엔드포인트 URL을 추가합니다.

> **Note**
>
> 이 `aws codeartifact login` 명령은 `Package.swift` 파일이 포함된 디렉터리에서 실행해야 하는 `swift package-registry` 명령을 호출합니다. 따라서 `aws codeartifact login` 명령은 Swift 프로젝트 내에서 실행해야 합니다.

로그인 명령으로 Swift를 구성하려면

1. `Package.swift` 프로젝트 파일이 포함된 Swift 프로젝트 디렉터리로 이동합니다.
2. 다음 `aws codeartifact login` 명령을 실행합니다.

   자신이 소유한 도메인의 리포지토리에 액세스하는 경우 `--domain-owner`를 포함하지 않아도 됩니다. 자세한 내용은 크로스 계정 도메인 단원을 참조하십시오.

   ```
   aws codeartifact login --tool swift --domain my_domain \
   --domain-owner 111122223333 --repository my_repo \
   [--namespace my_namespace]
   ```

`--namespace` 옵션은 CodeArtifact 리포지토리의 패키지가 지정된 네임스페이스에 있는 경우에만 해당 패키지를 사용하도록 애플리케이션을 구성합니다. CodeArtifact 네임스페이스는 범위와 동의어로, 코드를 논리적 그룹으로 구성하고 코드베이스에 여러 라이브러리가 포함된 경우 발생할 수 있는 이름 충돌을 방지하는 데 사용됩니다.

`login` 호출 후의 기본 승인 기간은 12시간이며, 토큰을 주기적으로 재발급하려면 `login`를 호출해야 합니다. `login` 명령으로 만든 인증 토큰에 관한 자세한 내용은 `login` 명령으로 생성된 토큰을 참조하세요.

로그인 명령 없이 Swift 구성

aws codeartifact login 명령을 사용하여 Swift를 구성하는 것이 좋지만 Swift Package Manager 구성을 수동으로 업데이트하여 로그인 명령 없이 Swift Package Manager를 구성할 수도 있습니다.

다음 절차에서는 AWS CLI 를 사용하여 다음을 수행합니다.

1. CodeArtifact에서 인증 토큰을 가져와 사용자 환경에 저장합니다. 보안 인증이 저장되는 방식은 환경의 운영 체제에 따라 달라집니다.
   a. macOS: macOS 키체인 애플리케이션에 항목이 생성됩니다.
   b. Linux 및 Windows: `~/.netrc` 파일에 항목이 생성됩니다.
2. CodeArtifact 리포지토리 엔드포인트 URL을 가져옵니다.
3. `~/.swiftpm/configuration/registries.json` 구성 파일에 리포지토리 엔드포인트 URL 및 인증 유형이 포함된 항목을 추가합니다.

로그인 명령 없이 Swift를 구성하려면

1. 명령줄에서 다음 명령을 사용하여 CodeArtifact 인증 토큰을 가져와 환경 변수에 저장합니다.
   - *my_domain*을 CodeArtifact 도메인 이름으로 변경합니다.
   - *111122223333*을 도메인 소유자의 AWS 계정 ID로 바꿉니다. 소유한 도메인의 리포지토리에 액세스하는 경우 `--domain-owner`를 포함하지 않아도 됩니다. 자세한 내용은 크로스 계정 도메인 단원을 참조하십시오.

   macOS and Linux

   ```
   export CODEARTIFACT_AUTH_TOKEN=`aws codeartifact get-authorization-token --
   domain my_domain --domain-owner 111122223333 --query authorizationToken --output
    text`
   ```

   Windows

   - Windows(기본 명령 쉘 사용):

     ```
     for /f %i in ('aws codeartifact get-authorization-token --domain my_domain --
     domain-owner 111122223333 --query authorizationToken --output text') do set
      CODEARTIFACT_AUTH_TOKEN=%i
     ```

- Windows PowerShell:

```
$env:CODEARTIFACT_AUTH_TOKEN = aws codeartifact get-authorization-token --
domain my_domain --domain-owner 111122223333 --query authorizationToken --
output text
```

2. 다음 명령을 실행하여 CodeArtifact 리포지토리의 엔드포인트를 가져올 수 있습니다. 리포지토리 엔드포인트는 패키지를 사용하거나 게시하도록 Swift Package Manager를 리포지토리로 이동하는 데 사용됩니다.

   - *my_domain*을 CodeArtifact 도메인 이름으로 변경합니다.
   - *111122223333*을 도메인 소유자의 AWS 계정 ID로 바꿉니다. 소유한 도메인의 리포지토리에 액세스하는 경우 `--domain-owner`를 포함하지 않아도 됩니다. 자세한 내용은 크로스 계정 도메인 단원을 참조하십시오.
   - *my_repo*를 CodeArtifact 리포지토리 이름으로 변경합니다.

   macOS and Linux

```
export CODEARTIFACT_REPO=`aws codeartifact get-repository-endpoint --
domain my_domain --domain-owner 111122223333 --repository my_repo --format swift
 --query repositoryEndpoint --output text`
```

   Windows

   - 윈도우(기본 커맨드 쉘 사용):

```
for /f %i in ('aws codeartifact get-repository-endpoint --domain my_domain
 --domain-owner 111122223333 --repository my_repo --format swift --query
 repositoryEndpoint --output text') do set CODEARTIFACT_REPO=%i
```

   - Windows PowerShell:

```
$env:CODEARTIFACT_REPO = aws codeartifact get-repository-endpoint --
domain my_domain --domain-owner 111122223333 --repository my_repo --format
 swift --query repositoryEndpoint --output text
```

   다음 URL은 리포지토리 엔드포인트의 예입니다.

```
https://my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com/
swift/my_repo/
```

> **Note**
>
> 듀얼 스택 엔드포인트를 사용하려면 `codeartifact.region.on.aws` 엔드포인트를 사용합니다.

> **Important**
>
> Swift Package Manager를 구성하는 데 사용할 때는 리포지토리 URL 엔드포인트 끝에 `login`을 추가해야 합니다. 이 절차는 이 절차의 명령으로 자동으로 수행됩니다.

3. 이 두 값을 환경 변수에 저장한 후 `swift package-registry login` 명령을 사용하여 Swift에 전달합니다.

   macOS and Linux

   ```
   swift package-registry login ${CODEARTIFACT_REPO}login --token
    ${CODEARTIFACT_AUTH_TOKEN}
   ```

   Windows

   - 윈도우(기본 커맨드 쉘 사용):

     ```
     swift package-registry login %CODEARTIFACT_REPO%login --token
      %CODEARTIFACT_AUTH_TOKEN%
     ```

   - Windows PowerShell:

     ```
     swift package-registry login $Env:CODEARTIFACT_REPO+"login" --token
      $Env:CODEARTIFACT_AUTH_TOKEN
     ```

4. 그런 다음 애플리케이션에서 사용하는 패키지 레지스트리를 업데이트하여 CodeArtifact 리포지토리에서 모든 종속성을 가져올 수 있도록 합니다. 이 명령은 패키지 종속성을 해결하려는 프로젝트 디렉터리에서 실행해야 합니다.

macOS and Linux

```
$ swift package-registry set ${CODEARTIFACT_REPO} [--scope my_scope]
```

Windows

- Windows(기본 명령 쉘 사용):

```
$ swift package-registry set %CODEARTIFACT_REPO% [--scope my_scope]
```

- Windows PowerShell:

```
$ swift package-registry set $Env:CODEARTIFACT_REPO [--scope my_scope]
```

이 `--scope` 옵션은 CodeArtifact 리포지토리의 패키지가 지정된 범위 내에 있는 경우에만 해당 패키지를 사용하도록 애플리케이션을 구성합니다. 범위는 CodeArtifact 네임스페이스와 동의어이며 코드를 논리적 그룹으로 구성하고 코드 베이스에 여러 라이브러리가 포함된 경우 발생할 수 있는 이름 충돌을 방지하는 데 사용됩니다.

5. 프로젝트 디렉터리에서 다음 명령을 실행하여 프로젝트 수준 `.swiftpm/configuration/registries.json` 파일의 내용을 보면 구성이 올바르게 설정되었는지 확인할 수 있습니다.

```
$ cat .swiftpm/configuration/registries.json
{
  "authentication" : {

  },
  "registries" : {
    "[default]" : {
      "url" : "https://my-domain-111122223333.d.codeartifact.us-
west-2.amazonaws.com/swift/my-repo/"
    }
  },
  "version" : 1
}
```

CodeArtifact 리포지토리로 Swift Package Manager를 구성했으니, 이제 이 리포지토리를 사용하여 Swift 패키지를 게시하고 이 리포지토리에서 Swift 패키지를 사용할 수 있습니다. 자세한 내용은 Swift 패키지 사용 및 게시 단원을 참조하십시오.

Swift 패키지 사용 및 게시

CodeArtifact에서의 Swift 패키지 사용

다음 절차에 따라 a AWS CodeArtifact 리포지토리의 Swift 패키지를 사용합니다.

CodeArtifact 리포지토리에서 Swift 패키지를 사용하려면

1. 아직 구성하지 않았다면, CodeArtifact를 사용하여 Swift Package Manager 설정의 단계에 따라 적절한 보안 인증과 함께 CodeArtifact 리포지토리를 사용하도록 Swift Package Manager를 구성합니다.

   > **Note**
   >
   > 생성된 권한 부여 토큰은 12시간 동안 유효합니다. 토큰이 생성된 후 12시간이 경과한 경우 새 토큰을 생성해야 합니다.

2. 애플리케이션 프로젝트 폴더의 `Package.swift` 파일을 편집하여 프로젝트에서 사용할 패키지 종속성을 업데이트합니다.

   a. `Package.swift` 파일에 `dependencies` 섹션이 없는 경우 해당 섹션을 추가합니다.

   b. `Package.swift` 파일의 `dependencies` 섹션에서 패키지 식별자를 추가하여 사용하려는 패키지를 추가합니다. 패키지 식별자는 범위와 패키지 이름을 마침표로 구분하여 구성됩니다. 예제는 이후 단계 다음에 나오는 코드 스니펫을 참조하세요.

      > **Tip**
      >
      > CodeArtifact 콘솔을 사용하여 패키지 식별자를 찾을 수 있습니다. 사용하려는 특정 패키지 버전을 찾고 패키지 버전 페이지의 설치 단축키 지침을 참조하세요.

   c. `Package.swift` 파일에 `targets` 섹션이 없는 경우 해당 섹션을 추가합니다.

   d. `targets` 섹션에서 종속성을 사용해야 하는 대상을 추가합니다.

      다음 스니펫은 `Package.swift` 파일의 구성된 `dependencies` 섹션과 `targets` 섹션을 보여 주는 예제 스니펫입니다.

```
...
    ],
    dependencies: [
        .package(id: "my_scope.package_name", from: "1.0.0")
    ],
    targets: [
      .target(
         name: "MyApp",
         dependencies: ["package_name"]
      ),...
    ],
...
```

3. 이제 모든 구성이 완료되었으므로 다음 명령을 사용하여 CodeArtifact에서 패키지 종속성을 다운로드합니다.

```
swift package resolve
```

Xcode에서 CodeArtifact의 Swift 패키지 사용

다음 절차를 사용하여 Xcode의 CodeArtifact 리포지토리에서 Swift 패키지를 사용할 수 있습니다.

Xcode의 CodeArtifact 리포지토리에서 Swift 패키지를 사용하려면

1. 아직 구성하지 않았다면, CodeArtifact를 사용하여 Swift Package Manager 설정의 단계에 따라 적절한 보안 인증과 함께 CodeArtifact 리포지토리를 사용하도록 Swift Package Manager를 구성합니다.

> **Note**
>
> 생성된 권한 부여 토큰은 12시간 동안 유효합니다. 토큰이 생성된 후 12시간이 경과한 경우 새 토큰을 생성해야 합니다.

2. 패키지를 Xcode의 프로젝트에 종속 항목으로 추가합니다.

   a. 파일 > 패키지 추가를 선택합니다.

   b. 검색 창을 사용하여 패키지를 검색합니다. 검색은 `package_scope.package_name` 형식으로 합니다.

c. 패키지를 찾으면 패키지를 선택한 다음 패키지 저장을 선택합니다.

d. 패키지가 확인되면 종속 항목으로 추가할 패키지 제품을 선택한 다음 패키지 추가를 선택합니다.

CodeArtifact 리포지토리를 Xcode와 함께 사용할 때 문제가 발생하는 경우 일반적인 문제 및 가능한 해결 방법은 Swift 문제 해결를 참조하세요.

CodeArtifact에 Swift 패키지 게시

CodeArtifact에서는 Swift 5.9 이상을 사용하고 `swift package-registry publish` 명령을 사용하여 Swift 패키지를 게시하는 것이 좋습니다. 이전 버전을 사용하는 경우 Curl 명령을 사용하여 Swift 패키지를 CodeArtifact에 게시해야 합니다.

`swift package-registry publish` 명령을 사용하여 CodeArtifact 패키지 게시

다음 절차에 따라 Swift 5.9 이상 버전에서Swift Package Manager를 사용하여 CodeArtifact 리포지토리에 Swift 패키지를 게시할 수 있습니다.

1. 아직 구성하지 않았다면, CodeArtifact를 사용하여 Swift Package Manager 설정의 단계에 따라 적절한 보안 인증과 함께 CodeArtifact 리포지토리를 사용하도록 Swift Package Manager를 구성합니다.

   > **Note**
   >
   > 생성된 권한 부여 토큰은 12시간 동안 유효합니다. 토큰이 생성된 후 12시간이 경과한 경우 새 토큰을 생성해야 합니다.

2. 패키지의 `Package.swift` 파일이 포함된 Swift 프로젝트 디렉터리로 이동합니다.

3. 다음 `swift package-registry publish` 명령을 실행하여 패키지를 게시합니다. 이 명령은 패키지 소스 아카이브를 생성한 다음 CodeArtifact 리포지토리에 게시합니다.

   ```
   swift package-registry publish packageScope.packageName packageVersion
   ```

   예제:

   ```
   swift package-registry publish myScope.myPackage 1.0.0
   ```

4. 패키지가 게시되었고 리포지토리에 존재하는지 여부는 콘솔에서 확인하거나 다음과 같은 `aws codeartifact list-packages` 명령을 사용하여 확인할 수 있습니다.

```
aws codeartifact list-packages --domain my_domain --repository my_repo
```

다음과 같이 `aws codeartifact list-package-versions` 명령을 사용하여 패키지의 단일 버전을 나열할 수 있습니다.

```
aws codeartifact list-package-versions --domain my_domain --repository my_repo \
--format swift --namespace my_scope --package package_name
```

Curl을 사용하여 CodeArtifact 패키지 게시

Swift 5.9 이상과 함께 제공되는 `swift package-registry publish` 명령을 사용하는 것이 좋지만 Curl을 사용하여 CodeArtifact에 Swift 패키지를 게시할 수도 있습니다.

다음 절차에 따라 Curl을 사용하여 Swift 패키지를 a AWS CodeArtifact 리포지토리에 게시합니다.

1. 아직 업데이트하지 않았다면 CodeArtifact를 사용하여 Swift Package Manager 설정의 단계에 따라 `CODEARTIFACT_AUTH_TOKEN` 및 `CODEARTIFACT_REPO` 환경 변수를 만들고 업데이트합니다.

> **Note**
>
> 권한 부여 토큰은 12시간 동안 유효합니다. `CODEARTIFACT_AUTH_TOKEN` 환경 변수를 생성한 후 12시간이 경과한 경우 새 보안 인증으로 환경 변수를 새로 고쳐야 합니다.

2. 먼저 Swift 패키지를 생성하지 않은 경우 다음 명령을 실행하여 생성할 수 있습니다.

```
mkdir testDir && cd testDir
swift package init
git init .
swift package archive-source
```

3. 다음 Curl 명령을 사용하여 CodeArtifact에 Swift 패키지를 게시합니다.

macOS and Linux

```
curl -X PUT  --user "aws:$CODEARTIFACT_AUTH_TOKEN" \
```

```
-H "Accept: application/vnd.swift.registry.v1+json" \
-F source-archive="@test_dir_package_name.zip" \
"${CODEARTIFACT_REPO}my_scope/package_name/packageVersion"
```

Windows

```
curl -X PUT  --user "aws:%CODEARTIFACT_AUTH_TOKEN%" \
-H "Accept: application/vnd.swift.registry.v1+json" \
-F source-archive="@test_dir_package_name.zip" \
"%CODEARTIFACT_REPO%my_scope/package_name/packageVersion"
```

4. 패키지가 게시되었고 리포지토리에 존재하는지 여부는 콘솔에서 확인하거나 다음과 같은 `aws codeartifact list-packages` 명령을 사용하여 확인할 수 있습니다.

```
aws codeartifact list-packages --domain my_domain --repository my_repo
```

다음과 같이 `aws codeartifact list-package-versions` 명령을 사용하여 패키지의 단일 버전을 나열할 수 있습니다.

```
aws codeartifact list-package-versions --domain my_domain --repository my_repo \
--format swift --namespace my_scope --package package_name
```

Swift 패키지를 GitHub에서 가져온 후 CodeArtifact에 다시 게시

다음 절차를 사용하여 GitHub에서 Swift 패키지를 가져와서 CodeArtifact 리포지토리에 다시 게시할 수 있습니다.

Swift 패키지를 GitHub에서 가져와서 CodeArtifact에 다시 게시하려면

1. 아직 구성하지 않았다면, CodeArtifact를 사용하여 Swift Package Manager 설정의 단계에 따라 적절한 보안 인증과 함께 CodeArtifact 리포지토리를 사용하도록 Swift Package Manager를 구성합니다.

> **Note**
>
> 생성된 권한 부여 토큰은 12시간 동안 유효합니다. 토큰이 생성된 후 12시간이 경과한 경우 새 토큰을 생성해야 합니다.

2. 다음 `git clone` 명령을 사용하여 가져오고 다시 게시하려는 Swift 패키지의 git 저장소를 복제합니다. GitHub 리포지토리 복제에 관한 자세한 내용은 GitHub 문서에서 리포지토리 복제를 참조하세요.

```
git clone repoURL
```

3. 방금 복제한 리포지토리로 이동합니다.

```
cd repoName
```

4. 패키지를 생성하여 CodeArtifact에 게시합니다.

   a. 권장: Swift 5.9 이상을 사용하는 경우 다음 `swift package-registry publish` 명령을 사용하여 패키지를 생성하고 구성된 CodeArtifact 저장소에 게시할 수 있습니다.

   ```
   swift package-registry publish packageScope.packageName versionNumber
   ```

   예제:

   ```
   swift package-registry publish myScope.myPackage 1.0.0
   ```

   b. 5.9 이전의 Swift 버전을 사용하는 경우 `swift archive-source` 명령을 사용하여 패키지를 생성한 다음 Curl 명령을 사용하여 패키지를 게시해야 합니다.

      i. `CODEARTIFACT_AUTH_TOKEN` 및 `CODEARTIFACT_REPO` 환경 변수를 구성하지 않았거나 구성한 지 12시간이 넘었다면 로그인 명령 없이 Swift 구성의 다음 단계를 따르세요.

      ii. `swift package archive-source` 명령을 사용하여 Swift 패키지 생성:

      ```
      swift package archive-source
      ```

      성공하면 명령줄에 `Created package_name.zip`이 표시됩니다.

      iii. 다음 Curl 명령을 사용하여 CodeArtifact에 Swift 패키지를 게시합니다.

      macOS and Linux

      ```
      curl -X PUT  --user "aws:$CODEARTIFACT_AUTH_TOKEN" \
      -H "Accept: application/vnd.swift.registry.v1+json" \
      -F source-archive="@package_name.zip" \
      "${CODEARTIFACT_REPO}my_scope/package_name/packageVersion"
      ```

Windows

```
curl -X PUT  --user "aws:%CODEARTIFACT_AUTH_TOKEN%" \
-H "Accept: application/vnd.swift.registry.v1+json" \
-F source-archive="@package_name.zip" \
"%CODEARTIFACT_REPO%my_scope/package_name/packageVersion"
```

5. 패키지가 게시되었고 리포지토리에 존재하는지 여부는 콘솔에서 확인하거나 다음과 같은 `aws codeartifact list-packages` 명령을 사용하여 확인할 수 있습니다.

```
aws codeartifact list-packages --domain my_domain --repository my_repo
```

다음과 같이 `aws codeartifact list-package-versions` 명령을 사용하여 패키지의 단일 버전을 나열할 수 있습니다.

```
aws codeartifact list-package-versions --domain my_domain --repository my_repo \
--format swift --namespace my_scope --package package_name
```

Swift 패키지 이름 및 네임스페이스 정규화

CodeArtifact는 패키지 이름과 네임스페이스를 저장하기 전에 정규화합니다. 즉, CodeArtifact의 이름은 패키지가 게시될 때 제공된 이름과 다를 수 있습니다.

패키지 이름 및 네임스페이스 정규화: CodeArtifact는 모든 문자를 소문자로 변환하여 Swift 패키지 이름과 네임스페이스를 정규화합니다.

패키지 버전 정규화: CodeArtifact는 Swift 패키지 버전을 정규화하지 않습니다. CodeArtifact는 Semantic Versioning 2.0 버전 패턴만 지원합니다. 시맨틱 버전 관리에 관한 자세한 내용은 Semantic Versioning 2.0.0을 참조하세요.

정규화되지 않은 패키지 이름 및 네임스페이스는 CodeArtifact가 해당 요청의 입력을 정규화하므로, API 및 CLI 요청에도 사용할 수 있습니다. 예를 들어, `--package myPackage`과 `--namespace myScope`를 입력하면 값을 정규화하여 정규화된 패키지 이름인 `mypackage`과 `myscope` 네임스페이스를 가진 패키지로 반환합니다.

IAM 정책과 같은 ARN에서는 정규화된 이름을 사용해야 합니다.

패키지의 정규화된 이름을 찾으려면 `aws codeartifact list-packages` 명령을 사용합니다. 자세한 내용은 패키지 이름 나열 단원을 참조하십시오.

Swift 문제 해결

다음은 CodeArtifact를 Swift와 함께 사용할 때 일반적으로 발생하는 문제를 해결하는 데 유용한 정보입니다.

Swift Package Manager를 구성한 후에도 Xcode에서 401 오류가 발생합니다.

문제: CodeArtifact 리포지토리의 패키지를 Xcode의 Swift 프로젝트에 대한 종속 항목으로 추가하려고 하면 Swift를 CodeArtifact에 연결하는 지침을 따랐는데도 401 무단 오류가 발생합니다.

가능한 해결 방법: 이 문제는 CodeArtifact 보안 인증이 저장된 macOS 키체인 애플리케이션의 문제로 인해 발생할 수 있습니다. 이 문제를 해결하려면 키체인 애플리케이션을 열고 CodeArtifact 항목을 모두 삭제하고 CodeArtifact를 사용하여 Swift Package Manager 설정의 지침에 따라 CodeArtifact 리포지토리를 사용하여 Swift Package Manager를 다시 구성하는 것이 좋습니다.

암호에 대한 키체인 프롬프트로 인해 Xcode가 CI 시스템에서 중단됨

문제: GitHub Actions와 같은 지속적 통합(CI) 서버의 Xcode 빌드의 일부로 CodeArtifact에서 Swift 패키지를 가져오려고 하면 CodeArtifact를 사용한 인증이 중단되고 결국 다음과 유사한 오류 메시지와 함께 실패할 수 있습니다.

```
Failed to save credentials for
\'https://my_domain-111122223333.d.codeartifact.us-west-2.amazonaws.com\'
to keychain: status -60008
```

가능한 해결 방법: 이 문제는 자격 증명이 CI 시스템의 키체인에 저장되지 않고 Xcode가 키체인에 저장된 자격 증명만 지원하므로 발생합니다. 이 문제를 해결하려면 다음 단계를 사용하여 키체인 항목을 수동으로 생성하는 것이 좋습니다.

1. 키체인을 준비합니다.

```
KEYCHAIN_PASSWORD=$(openssl rand -base64 20)
KEYCHAIN_NAME=login.keychain
SYSTEM_KEYCHAIN=/Library/Keychains/System.keychain

if [ -f $HOME/Library/Keychains/"${KEYCHAIN_NAME}"-db ]; then
    echo "Deleting old ${KEYCHAIN_NAME} keychain"
    security delete-keychain "${KEYCHAIN_NAME}"
fi
```

```
echo "Create Keychain"
security create-keychain -p "${KEYCHAIN_PASSWORD}" "${KEYCHAIN_NAME}"

EXISTING_KEYCHAINS=( $( security list-keychains | sed -e 's/ *//' | tr '\n' ' ' |
 tr -d '"') )
sudo security list-keychains -s "${KEYCHAIN_NAME}" "${EXISTING_KEYCHAINS[@]}"

echo "New keychain search list :"
security list-keychain

echo "Configure keychain : remove lock timeout"
security unlock-keychain -p "${KEYCHAIN_PASSWORD}" "${KEYCHAIN_NAME}"
security set-keychain-settings "${KEYCHAIN_NAME}"
```

2. CodeArtifact 인증 토큰과 리포지토리 엔드포인트를 가져옵니다.

```
export CODEARTIFACT_AUTH_TOKEN=`aws codeartifact get-authorization-token \
                                --region us-west-2 \
                                --domain my_domain   \
                                --domain-owner 111122223333   \
                                --query authorizationToken  \
                                --output text`

export CODEARTIFACT_REPO=`aws codeartifact get-repository-endpoint  \
                            --region us-west-2   \
                            --domain my_domain   \
                            --domain-owner 111122223333   \
                            --format swift     \
                            --repository my_repo    \
                            --query repositoryEndpoint   \
                            --output text`
```

3. 키체인 항목을 수동으로 생성합니다.

```
SERVER=$(echo $CODEARTIFACT_REPO | sed  's/https:\/\///g' | sed 's/.com.*$/.com/g')
AUTHORIZATION=(-T /usr/bin/security -T /usr/bin/codesign -T /usr/bin/xcodebuild -
T /usr/bin/swift \
               -T /Applications/Xcode-15.2.app/Contents/Developer/usr/bin/
xcodebuild)

security delete-internet-password -a token -s $SERVER -r htps "${KEYCHAIN_NAME}"
```

```
security add-internet-password -a token \
                               -s $SERVER \
                               -w $CODEARTIFACT_AUTH_TOKEN \
                               -r htps \
                               -U \
                               "${AUTHORIZATION[@]}" \
                               "${KEYCHAIN_NAME}"

security set-internet-password-partition-list \
             -a token \
             -s $SERVER \
             -S "com.apple.swift-
package,com.apple.security,com.apple.dt.Xcode,apple-tool:,apple:,codesign" \
             -k "${KEYCHAIN_PASSWORD}" "${KEYCHAIN_NAME}"

security find-internet-password   "${KEYCHAIN_NAME}"
```

이 오류 및 솔루션에 대한 자세한 내용은 https://github.com/apple/swift-package-manager/issues/7236을 참조하세요.

CodeArtifact를 일반 패키지와 함께 사용

이 항목에서는 AWS CodeArtifact를 사용하여 일반 패키지를 사용하고 게시하는 방법을 보여줍니다.

주제

일반 패키지 개요

`generic` 패키지 형식을 사용하면 모든 유형의 파일을 업로드하여 CodeArtifact 리포지토리에 패키지를 만들 수 있습니다. 일반 패키지는 특정 프로그래밍 언어, 파일 유형 또는 패키지 관리 에코시스템과 관련이 없습니다. 이는 애플리케이션 설치 프로그램, 기계 학습 모델, 구성 파일 등과 같은 임의의 빌드 아티팩트를 저장하고 버전을 관리하는 데 유용할 수 있습니다.

일반 패키지는 패키지 이름, 네임스페이스, 버전 및 하나 이상의 자산(또는 파일)으로 구성됩니다. 일반 패키지는 단일 CodeArtifact 리포지토리에서 다른 형식의 패키지와 함께 존재할 수 있습니다.

AWS CLI 또는 SDK를 사용하여 일반 패키지로 작업할 수 있습니다. 일반 패키지에서 작동하는 AWS CLI 명령의 전체 목록은 섹션을 참조하세요일반 패키지에 지원되는 명령.

일반 패키지 제약 조건

- 업스트림 리포지토리에서는 절대 가져오지 않습니다. 게시된 리포지토리에서만 가져올 수 있습니다.
- ListPackageVersionDependencies에서 반환되거나 AWS Management Console 에 표시되는 종속성을 선언할 수는 없습니다.
- README 및 LICENSE 파일을 저장할 수 있지만 CodeArtifact에서는 이를 해석하지 않습니다. 이러한 파일의 정보는 GetPackageVersionReadme 또는 DescribePackageVersion에서 반환되지 않으며 AWS Management Console에도 표시되지 않습니다.
- CodeArtifact의 모든 패키지와 마찬가지로 자산 크기 및 패키지당 자산 수에는 제한이 있습니다. CodeArtifact의 제한과 할당량에 대한 자세한 내용은 in AWS CodeArtifact 할당량를 참조하세요.
- 자산 이름은 다음 규칙에 따라 생성해야 합니다.

- 자산 이름에는 유니코드 문자와 숫자를 사용할 수 있습니다. 특히 소문자(`Ll`), 수정자(`Lm`), 기타 문자(`Lo`), 제목 대문자(`Lt`), 대문자(`Lu`), 문자 번호(`Nl`), 십진수(`Nd`) 등의 유니코드 문자 범주가 허용됩니다.
- 다음 특수 문자가 허용됩니다. `~!@^&()-_+[]{};,.`
- 자산에는 `.` 또는 `..` 이름을 지정할 수 없습니다.
- 스페이스는 허용되는 유일한 공백 문자입니다. 자산 이름은 공백 문자로 시작하거나 끝날 수 없으며 연속된 공백 문자가 포함될 수 없습니다.

일반 패키지에 지원되는 명령

AWS CLI 또는 SDK를 사용하여 일반 패키지로 작업할 수 있습니다. 다음 CodeArtifact 명령은 일반 패키지에서 사용할 수 있습니다.

- copy-package-versions (리포지토리 간 패키지 복사 참조)
- delete-package (패키지 삭제(AWS CLI) 참조)
- delete-package-versions (패키지 버전 삭제(AWS CLI) 참조)
- describe-package
- describe-package-version (패키지 버전 세부 정보 및 종속성 보기 및 업데이트 참조)
- dispose-package-versions (패키지 버전 폐기 참조)
- get-package-version-asset (패키지 버전 자산 다운로드 참조)
- list-package-version-assets (패키지 버전 자산 나열 참조)
- list-package-versions (패키지 버전 나열 참조)
- list-packages (패키지 이름 나열 참조)
- publish-package-version (일반 패키지 게시 참조)
- put-package-origin-configuration (패키지 원본 제어 편집 참조)

> **Note**
>
> `publish` 오리진 제어 설정을 사용하여 리포지토리에 일반 패키지 이름을 게시하는 것을 허용하거나 차단할 수 있습니다. 그러나 일반 패키지는 업스트림 저장소에서 가져올 수 없으므로 일반 패키지에는 이 `upstream` 설정이 적용되지 않습니다.

- update-package-versions-status (패키지 버전 상태 업데이트 참조)

일반 패키지 게시 및 사용

일반 패키지 버전과 관련 자산을 게시하려면 `publish-package-version` 명령을 사용합니다. `list-package-version-asset` 명령을 사용하여 일반 패키지의 자산을 나열하고 `get-package-version-asset`를 사용하여 이를 다운로드할 수 있습니다. 다음 항목에는 이러한 명령을 사용하여 일반 패키지를 게시하거나 일반 패키지 자산을 다운로드하는 방법에 대한 단계별 지침이 포함되어 있습니다.

일반 패키지 게시

일반 패키지는 패키지 이름, 네임스페이스, 버전 및 하나 이상의 자산(또는 파일)으로 구성됩니다. 이 항목에서는 `my-ns` 네임스페이스와 버전 `1.0.0`을 사용하고 `asset.tar.gz` 이름이 지정된 자산 하나를 포함하는 `my-package` 이름이 지정된 패키지를 게시하는 방법을 보여줍니다.

사전 조건:

- CodeArtifact를 AWS Command Line Interface 사용하여 설정 및 구성(참조with AWS CodeArtifact 설정)
- CodeArtifact 도메인 및 리포지토리 보유(AWS CLI를 사용하여 시작하기 참조)

일반 패키지를 게시하려면

1. 다음 명령을 사용하여 패키지 버전에 업로드할 각 파일의 SHA256 해시를 생성하고 환경 변수에 값을 입력합니다. 이 값은 파일 내용이 처음 전송된 후 변경되지 않았는지 확인하기 위한 무결성 검사로 사용됩니다.

   Linux

   ```
   export ASSET_SHA256=$(sha256sum asset.tar.gz | awk '{print $1;}')
   ```

   macOS

   ```
   export ASSET_SHA256=$(shasum -a 256 asset.tar.gz | awk '{print $1;}')
   ```

   Windows

   ```
   for /f "tokens=*" %G IN ('certUtil -hashfile asset.tar.gz SHA256 ^| findstr /v
    "hash"') DO SET "ASSET_SHA256=%G"
   ```

2. publish-package-version를 호출하여 자산을 업로드하고 새 패키지 버전을 생성합니다.

> **Note**
>
> 패키지에 자산이 두 개 이상 포함된 경우 업로드할 자산마다 한 번씩 publish-package-version를 호출할 수 있습니다. 최종 자산을 업로드하는 경우를 제외하고 각 publish-package-version 호출에 대한 --unfinished 인수를 포함합니다. --unfinished를 생략하면 패키지 버전의 상태가 Published로 설정되고 추가 자산이 업로드되지 않습니다.
> 또는 publish-package-version를 호출할 때마다 --unfinished를 포함하여 실행한 다음 update-package-versions-status 명령을 사용하여 패키지 버전의 상태를 Published로 설정하도록 할 수도 있습니다.

Linux/macOS

```
aws codeartifact publish-package-version --domain my_domain --repository my_repo
 \
      --format generic --namespace my-ns --package my-package --package-
version 1.0.0 \
      --asset-content asset.tar.gz --asset-name asset.tar.gz \
      --asset-sha256 $ASSET_SHA256
```

Windows

```
aws codeartifact publish-package-version --domain my_domain --repository my_repo
 ^
      --format generic --namespace my-ns --package my-package --package-
version 1.0.0 ^
      --asset-content asset.tar.gz --asset-name asset.tar.gz ^
      --asset-sha256 %ASSET_SHA256%
```

다음은 출력값을 보여줍니다.

```
{
    "format": "generic",
    "namespace": "my-ns",
    "package": "my-package",
    "version": "1.0.0",
```

```
    "versionRevision": "REVISION-SAMPLE-1-C7F4S5E9B772FC",
    "status": "Published",
    "asset": {
        "name": "asset.tar.gz",
        "size": 11,
        "hashes": {
            "MD5": "41bba98d5b9219c43089eEXAMPLE-MD5",
            "SHA-1": "69b215c25dd4cda1d997a786ec6EXAMPLE-SHA-1",
            "SHA-256": "43f24850b7b7b7d79c5fa652418518fbdf427e602b1edabe6EXAMPLE-
SHA-256",
            "SHA-512":
 "3947382ac2c180ee3f2aba4f8788241527c8db9dfe9f4b039abe9fc560aaf5a1fced7bd1e80a0dca9ce320d95
SHA-512"
        }
    }
}
```


일반 패키지 자산 목록

일반 패키지에 포함된 자산을 나열하려면 `list-package-version-assets` 명령을 사용합니다. 자세한 내용은 패키지 버전 자산 나열 단원을 참조하십시오.

다음 예제는 `my-package` 패키지 버전 `1.0.0`의 자산을 나열합니다.

패키지 버전 자산을 나열하려면

- 일반 패키지에 포함된 자산을 나열하려면 `list-package-version-assets`을 호출합니다.

  Linux/macOS

  ```
  aws codeartifact list-package-version-assets --domain my_domain \
    --repository my_repo --format generic --namespace my-ns \
    --package my-package --package-version 1.0.0
  ```

  Windows

  ```
  aws codeartifact list-package-version-assets --domain my_domain ^
    --repository my_repo --format generic --namespace my-ns ^
    --package my-package --package-version 1.0.0
  ```

다음은 출력값을 보여줍니다.

```
{
    "assets": [
        {
            "name": "asset.tar.gz",
            "size": 11,
            "hashes": {
                "MD5": "41bba98d5b9219c43089eEXAMPLE-MD5",
                "SHA-1": "69b215c25dd4cda1d997a786ec6EXAMPLE-SHA-1",
                "SHA-256":
 "43f24850b7b7b7d79c5fa652418518fbdf427e602b1edabe6EXAMPLE-SHA-256",
                "SHA-512":
 "3947382ac2c180ee3f2aba4f8788241527c8db9dfe9f4b039abe9fc560aaf5a1fced7bd1e80a0dca9ce320d95
SHA-512"
            }
        }
    ],
    "package": "my-package",
    "format": "generic",
    "namespace": "my-ns",
    "version": "1.0.0",
    "versionRevision": "REVISION-SAMPLE-1-C7F4S5E9B772FC"
}
```

일반 패키지 자산 다운로드

일반 패키지에서 자산을 다운로드하려면 `get-package-version-asset` 명령을 사용합니다. 자세한 내용은 패키지 버전 자산 다운로드 단원을 참조하십시오.

다음 예제에서는 `my-package` 패키지 버전 `1.0.0`의 `asset.tar.gz` 자산을 현재 작업 디렉터리의 `asset.tar.gz` 이름이 지정된 파일로 다운로드합니다.

패키지 버전 자산을 다운로드하려면

- `get-package-version-asset`를 호출하여 일반 패키지에서 자산을 다운로드합니다.

  Linux/macOS

  ```
  aws codeartifact get-package-version-asset --domain my_domain \
  ```

```
  --repository my_repo --format generic --namespace my-ns --package my-package \
  --package-version 1.0.0 --asset asset.tar.gz \
  asset.tar.gz
```

Windows

```
aws codeartifact get-package-version-asset --domain my_domain ^
  --repository my_repo --format generic --namespace my-ns --package my-package ^
  --package-version 1.0.0 --asset asset.tar.gz ^
  asset.tar.gz
```

다음은 출력값을 보여줍니다.

```
{
    "assetName": "asset.tar.gz",
    "packageVersion": "1.0.0",
    "packageVersionRevision": "REVISION-SAMPLE-1-C7F4S5E9B772FC"
}
```

CodeArtifact를 CodeBuild와 함께 사용

이 주제에서는 AWS CodeBuild 빌드 프로젝트의 CodeArtifact 리포지토리에서 패키지를 사용하는 방법을 설명합니다.

주제

CodeBuild의 npm 패키지 사용

다음 단계는 CodeBuild에서 제공하는 도커 이미지에 나열된 운영 체제를 사용하여 테스트되었습니다.

IAM 역할을 사용하여 권한 설정

이러한 단계는 CodeBuild에서 CodeArtifact의 npm 패키지를 사용할 때 필요합니다.

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/iam/ IAM 콘솔을 엽니다.
2. 탐색 창에서 역할을 선택합니다. 역할 페이지에서 CodeBuild 빌드 프로젝트에 사용하는 역할을 편집합니다. 이 역할에는 다음 권한이 있어야 합니다.

   JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
      {
          "Effect": "Allow",
          "Action": [ "codeartifact:GetAuthorizationToken",
                      "codeartifact:GetRepositoryEndpoint",
                      "codeartifact:ReadFromRepository"
                      ],
          "Resource": "*"
```

```
        },
        {
            "Effect": "Allow",
            "Action": "sts:GetServiceBearerToken",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "sts:AWSServiceName": "codeartifact.amazonaws.com"
                }
            }
        }
    ]
}
```

> **Important**
>
> CodeBuild를 사용하여 패키지를 게시하려면 **codeartifact:PublishPackageVersion** 권한을 추가해야 합니다.

자세한 내용은 IAM 사용 설명서의 역할 수정 섹션을 참조하세요.

로그인 및 npm 사용

CodeBuild에서 npm 패키지를 사용하려면, 프로젝트 `buildspec.yaml`의 `pre-build` 섹션에서 `login` 명령을 실행하여 CodeArtifact에서 패키지를 가져오도록 npm을 구성합니다. 자세한 내용은 npm으로 인증을 참조하세요.

`login`이 성공적으로 실행되면 `build` 섹션에서 npm 명령을 실행하여 npm 패키지를 설치할 수 있습니다.

Linux

> **Note**
>
> 이전 CodeBuild 이미지를 사용하는 `pip3 install awscli --upgrade --user` 경우에만를 AWS CLI 로 업그레이드하면 됩니다. 최신 이미지 버전을 사용한다면 이 줄을 제거해도 됩니다.

```
pre_build:
  commands:
    - pip3 install awscli --upgrade --user
    - aws codeartifact login --tool npm --domain my_domain --domain-owner 111122223333
 --repository my_repo
build:
  commands:
    - npm install
```

Windows

```
version: 0.2
phases:
  install:
    commands:
      - '[Net.ServicePointManager]::SecurityProtocol = "Tls12"; Invoke-WebRequest
 https://awscli.amazonaws.com/AWSCLIV2.msi -OutFile $env:TEMP/AWSCLIV2.msi'
      - Start-Process -Wait msiexec "/i $env:TEMP\AWSCLIV2.msi /quiet /norestart"
  pre_build:
    commands:
      - '&"C:\Program Files\Amazon\AWSCLIV2\aws" codeartifact login --tool npm --
domain my_domain --domain-owner 111122223333 --repository my_repo'
  build:
    commands:
      - npm install
```

CodeBuild에서 Python 패키지 사용

다음 단계는 CodeBuild에서 제공하는 도커 이미지에 나열된 운영 체제를 사용하여 테스트되었습니다.

IAM 역할을 사용하여 권한 설정

이러한 단계는 CodeBuild에서 CodeArtifact의 Python 패키지를 사용할 때 필요합니다.

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/iam/ IAM 콘솔을 엽니다.
2. 탐색 창에서 역할을 선택합니다. 역할 페이지에서 CodeBuild 빌드 프로젝트에 사용하는 역할을 편집합니다. 이 역할에는 다음 권한이 있어야 합니다.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
      {
          "Effect": "Allow",
          "Action": [ "codeartifact:GetAuthorizationToken",
                      "codeartifact:GetRepositoryEndpoint",
                      "codeartifact:ReadFromRepository"
                      ],
          "Resource": "*"
      },
      {
          "Effect": "Allow",
          "Action": "sts:GetServiceBearerToken",
          "Resource": "*",
          "Condition": {
              "StringEquals": {
                  "sts:AWSServiceName": "codeartifact.amazonaws.com"
              }
          }
      }
  ]
}
```

> **Important**
>
> CodeBuild를 사용하여 패키지를 게시하려면 **codeartifact:PublishPackageVersion** 권한을 추가해야 합니다.

자세한 내용은 IAM 사용 설명서의 역할 수정 섹션을 참조하세요.

로그인하여 pip 또는 twine 사용

CodeBuild에서 Python 패키지를 사용하려면, 프로젝트 `buildspec.yaml` 파일의 `pre-build` 섹션에서 `login` 명령을 실행하여 CodeArtifact에서 패키지를 가져오도록 `pip`을 구성합니다. 자세한 내용은 CodeArtifact를 Python과 함께 사용 단원을 참조하십시오.

`login`이 성공적으로 실행되면 `build` 섹션에서 `pip` 명령을 실행하여 Python 패키지를 설치하거나 게시할 수 있습니다.

Linux

> **Note**
>
> 이전 CodeBuild 이미지를 사용하는 `pip3 install awscli --upgrade --user` 경우에만를 AWS CLI 로 업그레이드하면 됩니다. 최신 이미지 버전을 사용한다면 이 줄을 제거해도 됩니다.

`pip`를 사용하여 Python 패키지를 설치하는 방법은 다음과 같습니다.

```
pre_build:
  commands:
    - pip3 install awscli --upgrade --user
    - aws codeartifact login --tool pip --domain my_domain --domain-owner 111122223333
 --repository my_repo
build:
  commands:
    - pip install requests
```

`twine`을 사용하여 Python 패키지를 게시하는 방법은 다음과 같습니다.

```
pre_build:
  commands:
    - pip3 install awscli --upgrade --user
    - aws codeartifact login --tool twine --domain my_domain --domain-
owner 111122223333 --repository my_repo
build:
  commands:
    - twine upload --repository codeartifact mypackage
```

Windows

pip를 사용하여 Python 패키지를 설치하는 방법은 다음과 같습니다.

```
version: 0.2
phases:
  install:
    commands:
      - '[Net.ServicePointManager]::SecurityProtocol = "Tls12"; Invoke-WebRequest
 https://awscli.amazonaws.com/AWSCLIV2.msi -OutFile $env:TEMP/AWSCLIV2.msi'
      - Start-Process -Wait msiexec "/i $env:TEMP\AWSCLIV2.msi /quiet /norestart"
  pre_build:
    commands:
      - '&"C:\Program Files\Amazon\AWSCLIV2\aws" codeartifact login --tool pip --
domain my_domain --domain-owner 111122223333 --repository my_repo'
  build:
    commands:
      - pip install requests
```

twine을 사용하여 Python 패키지를 게시하는 방법은 다음과 같습니다.

```
version: 0.2
phases:
  install:
    commands:
      - '[Net.ServicePointManager]::SecurityProtocol = "Tls12"; Invoke-WebRequest
 https://awscli.amazonaws.com/AWSCLIV2.msi -OutFile $env:TEMP/AWSCLIV2.msi'
      - Start-Process -Wait msiexec "/i $env:TEMP\AWSCLIV2.msi /quiet /norestart"
  pre_build:
    commands:
      - '&"C:\Program Files\Amazon\AWSCLIV2\aws" codeartifact login --tool twine --
domain my_domain --domain-owner 111122223333 --repository my_repo'
  build:
    commands:
      - twine upload --repository codeartifact mypackage
```

CodeBuild에서 Maven 패키지 사용

IAM 역할을 사용하여 권한 설정

이러한 단계는 CodeBuild에서 CodeArtifact의 Maven 패키지를 사용할 때 필요합니다.

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/iam/ IAM 콘솔을 엽니다.
2. 탐색 창에서 역할을 선택합니다. 역할 페이지에서 CodeBuild 빌드 프로젝트에 사용하는 역할을 편집합니다. 이 역할에는 다음 권한이 있어야 합니다.

   JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
      {
          "Effect": "Allow",
          "Action": [ "codeartifact:GetAuthorizationToken",
                      "codeartifact:GetRepositoryEndpoint",
                      "codeartifact:ReadFromRepository"
                      ],
          "Resource": "*"
      },
      {
          "Effect": "Allow",
          "Action": "sts:GetServiceBearerToken",
          "Resource": "*",
          "Condition": {
              "StringEquals": {
                  "sts:AWSServiceName": "codeartifact.amazonaws.com"
              }
          }
      }
  ]
}
```

> **Important**
>
> CodeBuild를 사용하여 패키지를 게시하려면 **codeartifact:PublishPackageVersion** 및 **codeartifact:PutPackageMetadata** 권한을 추가해야 합니다.

자세한 내용은 IAM 사용 설명서의 역할 수정 섹션을 참조하세요.

gradle 또는 mvn 사용

`gradle` 또는 `mvn`과 함께 Maven 패키지를 사용하려면, 환경 변수에 인증 토큰 전달에서 설명하는 것처럼 CodeArtifact 인증 토큰을 환경 변수에 저장하세요. 다음은 예입니다.

> **Note**
>
> 이전 CodeBuild 이미지를 사용하는 `pip3 install awscli --upgrade --user` 경우에만를 AWS CLI 로 업그레이드하면 됩니다. 최신 이미지 버전을 사용한다면 이 줄을 제거해도 됩니다.

```
pre_build:
  commands:
    - pip3 install awscli --upgrade --user
    - export CODEARTIFACT_AUTH_TOKEN=`aws codeartifact get-authorization-token --
domain my_domain --domain-owner 111122223333 --query authorizationToken --output text`
```

Gradle을 사용하는 방법은 다음과 같습니다.

CodeArtifact를 Gradle과 함께 사용에서 설명하는 것처럼 Gradle `build.gradle` 파일에서 `CODEARTIFACT_AUTH_TOKEN` 변수를 참조한다면, `buildspec.yaml` `build` 섹션에서 Gradle 빌드를 호출할 수 있습니다.

```
build:
  commands:
    - gradle build
```

mvn을 사용하는 방법은 다음과 같습니다.

CodeArtifact를 mvn과 함께 사용에 나오는 지침에 따라 Maven 구성 파일(`settings.xml` 및 `pom.xml`)을 구성해야 합니다.

CodeBuild에서 NuGet 패키지 사용

다음 단계는 CodeBuild에서 제공하는 도커 이미지에 나열된 운영 체제를 사용하여 테스트되었습니다.

주제

IAM 역할을 사용하여 권한 설정

이러한 단계는 CodeBuild에서 CodeArtifact의 NuGet 패키지를 사용할 때 필요합니다.

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/iam/ IAM 콘솔을 엽니다.
2. 탐색 창에서 역할을 선택합니다. 역할 페이지에서 CodeBuild 빌드 프로젝트에 사용하는 역할을 편집합니다. 이 역할에는 다음 권한이 있어야 합니다.

   JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
      {
          "Effect": "Allow",
          "Action": [ "codeartifact:GetAuthorizationToken",
                      "codeartifact:GetRepositoryEndpoint",
                      "codeartifact:ReadFromRepository"
                      ],
          "Resource": "*"
      },
      {
          "Effect": "Allow",
          "Action": "sts:GetServiceBearerToken",
          "Resource": "*",
          "Condition": {
              "StringEquals": {
                  "sts:AWSServiceName": "codeartifact.amazonaws.com"
              }
          }
      }
  ]
}
```

> **Important**
>
> CodeBuild를 사용하여 패키지를 게시하려면 **`codeartifact:PublishPackageVersion`** 권한을 추가해야 합니다.

자세한 내용은 IAM 사용 설명서의 역할 수정 섹션을 참조하세요.

NuGet 패키지 사용

CodeBuild에서 NuGet 패키지를 사용하려면 프로젝트의 `buildspec.yaml` 파일에 다음을 포함해야 합니다.

1. `install` 섹션에서 CodeArtifact 자격 증명 공급자를 설치하여, 패키지를 빌드하고 CodeArtifact에 게시하도록 `msbuild` 및 `dotnet` 같은 명령줄 도구를 구성합니다.
2. `pre-build` 섹션에서 CodeArtifact 리포지토리를 NuGet 구성에 추가합니다.

다음 `buildspec.yaml` 예시를 참조하세요. 자세한 내용은 CodeArtifact를 NuGet과 함께 사용 단원을 참조하십시오.

보안 인증 공급자를 설치하고 리포지토리 소스를 추가한 후에는, `build` 섹션에서 NuGet CLI 도구 명령을 실행하여 NuGet 패키지를 사용할 수 있습니다.

Linux

`dotnet`을 사용하여 NuGet 패키지를 사용하는 방법은 다음과 같습니다.

```
version: 0.2

phases:
  install:
    runtime-versions:
      dotnet: latest
    commands:
      - export PATH="$PATH:/root/.dotnet/tools"
      - dotnet tool install -g AWS.CodeArtifact.NuGet.CredentialProvider
      - dotnet codeartifact-creds install
```

```
  pre_build:
    commands:
      -  dotnet nuget add source -n codeartifact $(aws codeartifact get-repository-
endpoint --domain my_domain --domain-owner 111122223333 --repository my_repo --format
 nuget --query repositoryEndpoint --output text)"v3/index.json"
  build:
    commands:
      - dotnet add package <packageName> --source codeartifact
```

Windows

dotnet을 사용하여 NuGet 패키지를 사용하는 방법은 다음과 같습니다.

```
version: 0.2

phases:
  install:
    commands:
      - dotnet tool install -g AWS.CodeArtifact.NuGet.CredentialProvider
      - dotnet codeartifact-creds install
  pre_build:
    commands:
      - dotnet nuget add source -n codeartifact "$(aws codeartifact get-repository-
endpoint --domain my_domain --domain-owner 111122223333 --repository my_repo --format
 nuget --query repositoryEndpoint --output text)v3/index.json"
  build:
    commands:
      - dotnet add package <packageName> --source codeartifact
```

NuGet 패키지를 이용한 빌드

CodeBuild의 NuGet 패키지를 이용해 빌드하려면 프로젝트의 buildspec.yaml 파일에 다음을 포함해야 합니다.

1. install 섹션에서 CodeArtifact 자격 증명 공급자를 설치하여, 패키지를 빌드하고 CodeArtifact에 게시하도록 msbuild 및 dotnet 같은 명령줄 도구를 구성합니다.
2. pre-build 섹션에서 CodeArtifact 리포지토리를 NuGet 구성에 추가합니다.

다음 buildspec.yaml 예시를 참조하세요. 자세한 내용은 CodeArtifact를 NuGet과 함께 사용 단원을 참조하십시오.

보안 인증 공급자를 설치하고 리포지토리 소스를 추가한 후에는, `build` 섹션에서 `dotnet build` 같은 NuGet CLI 도구 명령을 실행할 수 있습니다.

Linux

`dotnet`을 사용하여 NuGet 패키지를 빌드하는 방법은 다음과 같습니다.

```
version: 0.2

phases:
  install:
    runtime-versions:
      dotnet: latest
    commands:
      - export PATH="$PATH:/root/.dotnet/tools"
      - dotnet tool install -g AWS.CodeArtifact.NuGet.CredentialProvider
      - dotnet codeartifact-creds install
  pre_build:
    commands:
      -  dotnet nuget add source -n codeartifact $(aws codeartifact get-repository-
endpoint --domain my_domain --domain-owner 111122223333 --repository my_repo --format
 nuget --query repositoryEndpoint --output text)"v3/index.json"
  build:
    commands:
      - dotnet build
```

`msbuild`를 사용하여 NuGet 패키지를 빌드하는 방법은 다음과 같습니다.

```
version: 0.2

phases:
  install:
    runtime-versions:
      dotnet: latest
    commands:
      - export PATH="$PATH:/root/.dotnet/tools"
      - dotnet tool install -g AWS.CodeArtifact.NuGet.CredentialProvider
      - dotnet codeartifact-creds install
  pre_build:
    commands:
      - dotnet nuget add source -n codeartifact $(aws codeartifact get-repository-
endpoint --domain my_domain --domain-owner 111122223333 --repository my_repo --format
 nuget --query repositoryEndpoint --output text)"v3/index.json"
```

```
  build:
    commands:
      - msbuild -t:Rebuild -p:Configuration=Release
```

Windows

dotnet을 사용하여 NuGet 패키지를 빌드하는 방법은 다음과 같습니다.

```
version: 0.2

phases:
  install:
    commands:
      - dotnet tool install -g AWS.CodeArtifact.NuGet.CredentialProvider
      - dotnet codeartifact-creds install
  pre_build:
    commands:
      - dotnet nuget add source -n codeartifact "$(aws codeartifact get-repository-
endpoint --domain my_domain --domain-owner 111122223333 --repository my_repo --format
 nuget --query repositoryEndpoint --output text)v3/index.json"
  build:
    commands:
      - dotnet build
```

msbuild를 사용하여 NuGet 패키지를 빌드하는 방법은 다음과 같습니다.

```
version: 0.2

phases:
  install:
    commands:
      - dotnet tool install -g AWS.CodeArtifact.NuGet.CredentialProvider
      - dotnet codeartifact-creds install
  pre_build:
    commands:
      - dotnet nuget add source -n codeartifact "$(aws codeartifact get-repository-
endpoint --domain my_domain --domain-owner 111122223333 --repository my_repo --format
 nuget --query repositoryEndpoint --output text)v3/index.json"
  build:
    commands:
      - msbuild -t:Rebuild -p:Configuration=Release
```

NuGet 패키지 게시

CodeBuild의 NuGet 패키지를 게시하려면 프로젝트의 `buildspec.yaml` 파일에 다음을 포함해야 합니다.

1. `install` 섹션에서 CodeArtifact 보안 인증 공급자를 설치하여, 패키지를 빌드하고 CodeArtifact에 게시하도록 `msbuild` 및 `dotnet` 같은 명령줄 도구를 구성합니다.
2. `pre-build` 섹션에서 CodeArtifact 리포지토리를 NuGet 구성에 추가합니다.

다음 `buildspec.yaml` 예시를 참조하세요. 자세한 내용은 CodeArtifact를 NuGet과 함께 사용 단원을 참조하십시오.

보안 인증 공급자를 설치하고 리포지토리 소스를 추가한 후에는, `build` 섹션에서 NuGet CLI 도구 명령을 실행하고 NuGet 패키지를 게시할 수 있습니다.

Linux

`dotnet`을 사용하여 NuGet 패키지를 게시하는 방법은 다음과 같습니다.

```
version: 0.2

phases:
  install:
    runtime-versions:
      dotnet: latest
    commands:
      - export PATH="$PATH:/root/.dotnet/tools"
      - dotnet tool install -g AWS.CodeArtifact.NuGet.CredentialProvider
      - dotnet codeartifact-creds install
  pre_build:
    commands:
      - dotnet nuget add source -n codeartifact $(aws codeartifact get-repository-
endpoint --domain my_domain --domain-owner 111122223333 --repository my_repo --format
 nuget --query repositoryEndpoint --output text)"v3/index.json"
  build:
    commands:
      - dotnet pack -o .
      - dotnet nuget push *.nupkg -s codeartifact
```

Windows

dotnet을 사용하여 NuGet 패키지를 게시하는 방법은 다음과 같습니다.

```
version: 0.2

phases:
  install:
    commands:
      - dotnet tool install -g AWS.CodeArtifact.NuGet.CredentialProvider
      - dotnet codeartifact-creds install
  pre_build:
    commands:
      - dotnet nuget add source -n codeartifact "$(aws codeartifact get-repository-
endpoint --domain my_domain --domain-owner 111122223333 --repository my_repo --format
 nuget --query repositoryEndpoint --output text)v3/index.json"
  build:
    commands:
      - dotnet pack -o .
      - dotnet nuget push *.nupkg -s codeartifact
```

종속성 캐싱

CodeBuild에서 로컬 캐싱을 활성화하면 각 빌드에 대해 CodeArtifact에서 가져와야 하는 종속성 수를 줄일 수 있습니다. 자세한 내용은 AWS CodeBuild 사용 설명서의 AWS CodeBuild에서의 빌드 캐싱을 참조하세요. 사용자 지정 로컬 캐시를 활성화한 후 프로젝트의 buildspec.yaml 파일에 캐시 디렉터리를 추가합니다.

예를 들어 mvn을 사용하는 경우에는 다음을 사용하세요.

```
cache:
  paths:
    - '/root/.m2/**/*'
```

다른 도구를 사용하는 경우에는 이 표에 나오는 캐시 폴더를 사용하세요.

도구	캐시 디렉터리
mvn	/root/.m2/**/*

도구	캐시 디렉터리
gradle	/root/.gradle/caches/**/*
pip	/root/.cache/pip/**/*
npm	/root/.npm/**/*
nuget	/root/.nuget/**/*
yarn (classic)	/root/.cache/yarn/**/*

CodeArtifact 모니터링

모니터링은 CodeArtifact 및 기타 AWS 솔루션의 신뢰성, 가용성 및 성능을 유지하는 데 중요한 부분입니다.는 CodeArtifact를 모니터링하고, 이상이 있을 때 보고하고, 필요한 경우 자동 조치를 취할 수 있도록 다음 모니터링 도구를 AWS 제공합니다.

- Amazon EventBridge를 사용하여 AWS 서비스를 자동화하고 애플리케이션 가용성 문제 또는 리소스 변경과 같은 시스템 이벤트에 자동으로 대응할 수 있습니다. AWS 서비스의 이벤트는 거의 실시간으로 EventBridge로 전송됩니다. 원하는 이벤트만 표시하도록 간단한 규칙을 작성한 후 규칙과 일치하는 이벤트 발생 시 실행할 자동화 작업을 지정할 수 있습니다. 자세한 내용은 Amazon EventBridge 사용 설명서 및 CodeArtifact 이벤트 형식 및 예제 섹션을 참조하세요.
- Amazon CloudWatch 지표를 사용하여 작업별 CodeArtifact 사용량을 볼 수 있습니다. CloudWatch 지표에는 CodeArtifact에 대한 모든 요청이 포함되며, 요청은 계정별로 표시됩니다. Usage/By AWS 리소스 AWS 네임스페이스로 이동하여 CloudWatch 지표에서 이러한 지표를 볼 수 있습니다. 자세한 내용은 Amazon CloudWatch 사용 설명서의 Amazon CloudWatch 지표 사용 섹션을 참조하십시오.

주제

CodeArtifact 이벤트 모니터링

CodeArtifact는 CodeArtifact 리포지토리 변경을 비롯한 이벤트를 자동화하고 반응하는 서비스인 Amazon EventBridge와 통합되어 있습니다. 이벤트 규칙을 생성하고, 이벤트가 규칙과 일치할 때 일어나는 일을 구성할 수 있습니다. 이전에는 EventBridge를 CloudWatch Events라고 했습니다.

이벤트가 트리거할 수 있는 작업은 다음과 같습니다.

- AWS Lambda 함수 호출.
- AWS Step Functions 상태 시스템 활성화.
- SNS 주제 또는 Amazon SQS 대기열 알림.
- 에서 파이프라인을 시작합니다 AWS CodePipeline.

CodeArtifact는 패키지 버전이 생성, 수정 또는 삭제될 때 이벤트를 생성합니다. 다음은 CodeArtifact 이벤트의 예제입니다.

- (`npm publish` 실행 등의 방법을 통한) 새 패키지 버전 게시.
- (새 JAR 파일을 기존 Maven 패키지로 푸시하는 방법 등을 통한) 기존 패키지 버전에 새 자산 추가.
- `copy-package-versions`를 사용하여 한 리포지토리의 패키지 버전을 다른 리포지토리로 복사. 자세한 내용은 리포지토리 간 패키지 복사 단원을 참조하십시오.
- `delete-package-versions`를 사용하여 패키지 버전 삭제. 자세한 내용은 패키지 또는 패키지 버전 삭제 단원을 참조하십시오.
- `delete-package`를 사용하여 패키지 버전 삭제. 삭제된 패키지의 각 버전에 대해 이벤트 하나가 게시됩니다. 자세한 내용은 패키지 또는 패키지 버전 삭제 단원을 참조하십시오.
- 업스트림 리포지토리에서 가져온 패키지 버전을 다운스트림 리포지토리에 유지. 자세한 내용은 CodeArtifact에서의 업스트림 리포지토리 작업 단원을 참조하십시오.
- 외부 리포지토리의 패키지 버전을 CodeArtifact 리포지토리에 수집. 자세한 내용은 CodeArtifact 저장소를 공용 저장소에 연결하기 단원을 참조하십시오.

이벤트는 도메인을 소유한 계정과 리포지토리를 관리하는 계정 모두에 전달됩니다. 예를 들어 계정 `111111111111`이 `my_domain` 도메인을 소유하고 있다고 가정해 보겠습니다. 계정 `222222222222`는 `repo2`라는 이름의 `my_domain`에서 리포지토리를 생성합니다. 새 패키지 버전이 `repo2`에 게시되면 두 계정 모두 EventBridge 이벤트를 수신합니다. 도메인 소유 계정(`111111111111`)은 도메인 내 모든 리포지토리에 대한 이벤트를 수신합니다. 단일 계정이 도메인과 도메인 내 리포지토리를 모두 소유하는 경우 단일 이벤트만 전달됩니다.

다음 주제에서는 CodeArtifact 이벤트 형식을 설명합니다. CodeArtifact 이벤트를 구성하는 방법과 다른 AWS 서비스와 함께 이벤트를 사용하는 방법을 보여줍니다. 자세한 정보는 Amazon EventBridge 사용 설명서의 Amazon EventBridge 시작하기를 참조하세요.

CodeArtifact 이벤트 형식 및 예제

다음은 이벤트 필드 및 설명과 CodeArtifact 이벤트 예제입니다.

CodeArtifact 이벤트 형식

모든 CodeArtifact 이벤트에는 다음 필드가 포함됩니다.

이벤트 필드	설명
version	이벤트 형식의 버전입니다. 현재는 버전 `0`만 존재합니다.
id	이벤트의 고유 식별자입니다.
detail-type	이벤트의 유형입니다. `detail` 객체의 필드를 결정합니다. 현재 지원되는 유일한 `detail-type` 은 `CodeArtifact Package Version State Change`입니다.
source	이벤트의 원본입니다. CodeArtifact의 경우에는 `aws.codeartifact` 입니다.
계정	이벤트를 수신하는 AWS 계정의 계정 ID입니다.
시간	이벤트가 트리거된 정확한 시간입니다.
region	이벤트가 트리거된 지역입니다.
resources	변경된 패키지의 ARN이 포함된 목록입니다. 목록에는 항목 하나가 포함되어 있습니다. 패키지 ARN 형식에 대한 자세한 내용은 패키지에 쓰기 액세스 권한 부여 섹션을 참조하세요.
domainName	패키지가 포함된 리포지토리가 있는 도메인입니다.
domainOwner	도메인 소유자의 AWS 계정 ID입니다.
repositoryName	패키지를 포함하는 리포지토리입니다.
repositoryAdministrator	리포지토리 관리자의 AWS 계정 ID입니다.
packageFormat	이벤트를 트리거한 패키지의 형식입니다.
packageNamespace	이벤트를 트리거한 패키지의 네임스페이스입니다.

이벤트 필드	설명
packageName	이벤트를 트리거한 패키지의 이름입니다.
packageVersion	이벤트를 트리거한 패키지의 버전입니다.
packageVersionState	이벤트를 트리거한 패키지 버전의 상태입니다. 가능한 값은 `Unfinished` , `Published` , `Unlisted`, `Archived`, `Disposed`입니다.
packageVersionRevision	이벤트가 트리거되었을 때 패키지 버전의 자산 및 메타데이터 상태를 고유하게 식별하는 값입니다. 패키지 버전이 수정되면(예: 다른 JAR 파일을 Maven 패키지에 추가) `packageVersionRevision` 이 변경됩니다.
changes.assetsAdded	이벤트를 트리거한 패키지에 추가된 자산의 수입니다. 대표적인 자산은 Maven JAR 파일 또는 Python 휠입니다.
changes.assetsRemoved	이벤트를 트리거한 패키지에서 제거된 자산의 수입니다.
changes.assetsUpdated	이벤트를 트리거한 패키지에서 수정된 자산의 수입니다.
changes.metadataUpdated	이벤트에 수정된 패키지 수준 메타데이터가 포함된 경우 `true`에 설정되는 부울 값입니다. 예를 들어 이벤트는 Maven `pom.xml` 파일을 수정할 수 있습니다.
changes.statusChanged	이벤트의 `packageVersionStatus` 가 수정된 경우(예: `packageVersionStatus` 가 `Unfinished` 에서 `Published` 로 변경되는 경우) `true`에 설정되는 부울 값입니다.
operationType	패키지 버전 변경의 상위 수준 유형을 설명합니다. 가능한 값은 `Created`, `Updated`, `Deleted`입니다.

이벤트 필드	설명
sequenceNumber	패키지의 이벤트 번호를 지정하는 정수입니다. 패키지의 각 이벤트는 `sequenceNumber` 를 증가시키므로, 이벤트를 순차적으로 정렬할 수 있습니다. 이벤트는 `sequenceNumber` 를 아무 정수만큼 증가시킬 수 있습니다. **Note** EventBridge 이벤트가 잘못된 순서로 수신될 수 있습니다. `sequenceNumber` 를 사용하면 실제 순서를 결정할 수 있습니다.
eventDeduplicationId	중복된 EventBridge 이벤트를 구별하는 데 사용하는 ID입니다. 드물지만 EventBridge는 단일 이벤트 또는 예정된 시간에 대해 동일한 규칙을 두 번 이상 트리거하기도 합니다. 또는 트리거된 규칙에 대해 동일한 대상을 두 번 이상 호출할 수도 있습니다.

CodeArtifact 이벤트 예제

다음은 npm 패키지가 게시될 때 트리거될 수 있는 CodeArtifact 이벤트 예제입니다.

```
{
      "version":"0",
      "id":"73f03fec-a137-971e-6ac6-07c8ffffffff",
      "detail-type":"CodeArtifact Package Version State Change",
      "source":"aws.codeartifact",
      "account":"123456789012",
      "time":"2019-11-21T23:19:54Z",
      "region":"us-west-2",
      "resources":["arn:aws:codeartifact:us-west-2:111122223333:package/my_domain/
myrepo/npm//mypackage"],
      "detail":{
        "domainName":"my_domain",
```

```
        "domainOwner":"111122223333",
        "repositoryName":"myrepo",
        "repositoryAdministrator":"123456789012",
        "packageFormat":"npm",
        "packageNamespace":null,
        "packageName":"mypackage",
        "packageVersion":"1.0.0",
        "packageVersionState":"Published",
        "packageVersionRevision":"0E5DE26A4CD79FDF3EBC4924FFFFFFFF",
        "changes":{
          "assetsAdded":1,
          "assetsRemoved":0,
          "metadataUpdated":true,
          "assetsUpdated":0,
          "statusChanged":true
        },
        "operationType":"Created",
        "sequenceNumber":1,
        "eventDeduplicationId":"2mEO0A2Ke07rWUTBXk3CAiQhdTXF4N94LNaT/ffffff="
      }
    }
```

이벤트를 사용하여 CodePipeline 실행 시작

이 예제에서는 CodeArtifact 리포지토리의 패키지 버전이 게시, 수정 또는 삭제될 때 AWS CodePipeline 실행이 시작되도록 Amazon EventBridge 규칙을 구성하는 방법을 보여줍니다.

주제

- EventBridge 권한 구성
- EventBridge 규칙 생성
- EventBridge 규칙 대상 생성

EventBridge 권한 구성

생성한 규칙을 호출하기 위해 CodePipeline을 사용하려면 EventBridge에 대한 권한을 추가해야 합니다. AWS Command Line Interface (AWS CLI)를 사용하여 이러한 권한을 추가하려면 AWS CodePipeline 사용 설명서의 CodeCommit 소스(CLI)에 대한 CloudWatch Events 규칙 생성의 1단계를 따릅니다.

EventBridge 규칙 생성

규칙을 생성하려면 `put-rule` 명령을 `--name` 및 `--event-pattern` 파라미터와 함께 사용하세요. 이벤트 패턴은 각 이벤트의 내용과 일치하는 값을 지정합니다. 패턴이 이벤트와 일치하면 대상이 트리거됩니다. 예를 들어 다음 패턴은 `my_domain` 도메인의 `myrepo` 리포지토리에서 발생한 CodeArtifact 이벤트와 일치합니다.

```
aws events put-rule --name MyCodeArtifactRepoRule --event-pattern \
    '{"source":["aws.codeartifact"],"detail-type":["CodeArtifact Package Version State
 Change"],
    "detail":{"domainName":["my_domain"],"domainOwner":
["111122223333"],"repositoryName":["myrepo"]}}'
```

EventBridge 규칙 대상 생성

다음 명령은 대상을 규칙에 추가하여, 규칙과 일치하는 이벤트가 있을 때 CodePipeline 실행이 트리거되게 합니다. `RoleArn` 파라미터의 경우 이 주제 앞부분에서 생성한 역할의 Amazon 리소스 이름(ARN)을 지정하세요.

```
aws events put-targets --rule MyCodeArtifactRepoRule --targets \
  'Id=1,Arn=arn:aws:codepipeline:us-west-2:111122223333:pipeline-name,
  RoleArn=arn:aws:iam::123456789012:role/MyRole'
```

이벤트를 사용하여 Lambda 함수 실행

이 예제에서는 CodeArtifact 리포지토리의 패키지 버전이 게시, 수정 또는 삭제될 때 AWS Lambda 함수가 시작되도록 EventBridge 규칙을 구성하는 방법을 보여줍니다.

자세한 내용은 Amazon EventBridge 사용 설명서의 자습서: EventBridge를 사용하여 AWS Lambda 함수 예약을 참조하세요.

주제

- EventBridge 규칙 생성
- EventBridge 규칙 대상 생성
- EventBridge 권한 구성

EventBridge 규칙 생성

Lambda 함수를 시작하는 규칙을 생성하려면 `put-rule` 명령을 `--name` 및 `--event-pattern` 옵션과 함께 사용하세요. 다음 패턴은 `my_domain` 도메인 내 모든 리포지토리의 `@types` 범위에 npm 패키지를 지정합니다.

```
aws events put-rule --name "MyCodeArtifactRepoRule" --event-pattern \
  '{"source":["aws.codeartifact"],"detail-type":["CodeArtifact Package Version State
 Change"],
  "detail":{"domainName":["my_domain"],"domainOwner":
["111122223333"],"packageNamespace":["types"],"packageFormat":["npm"]}}'
```

EventBridge 규칙 대상 생성

다음 명령은 이벤트가 규칙과 일치할 때 Lambda 함수를 실행하는 규칙에 대상을 추가합니다. `arn` 파라미터의 경우 Lambda 함수의 Amazon 리소스 이름(ARN)을 지정합니다.

```
aws events put-targets --rule MyCodeArtifactRepoRule --targets \
  Id=1,Arn=arn:aws:lambda:us-west-2:111122223333:function:MyLambdaFunction
```

EventBridge 권한 구성

`add-permission` 명령을 사용하여, Lambda 함수를 호출할 수 있는 권한을 규칙에 부여합니다. `--source-arn` 파라미터의 경우 이 예제의 앞부분에서 생성한 규칙의 ARN을 지정합니다.

```
aws lambda add-permission --function-name MyLambdaFunction \\
  --statement-id my-statement-id --action 'lambda:InvokeFunction' \\
  --principal events.amazonaws.com \\
  --source-arn arn:aws:events:us-west-2:111122223333:rule/MyCodeArtifactRepoRule
```

CodeArtifact 보안

의 클라우드 보안 AWS 이 최우선 순위입니다. AWS 고객은 보안에 가장 민감한 조직의 요구 사항을 충족하도록 구축된 데이터 센터 및 네트워크 아키텍처의 이점을 누릴 수 있습니다.

보안은 AWS 와 사용자 간의 공동 책임입니다. 공동 책임 모델은 이 사항을 클라우드의 보안 및 클라우드 내 보안으로 설명합니다.

- 클라우드 보안 - AWS 는 AWS 클라우드에서 AWS 서비스를 실행하는 인프라를 보호할 책임이 있습니다. AWS 또한는 안전하게 사용할 수 있는 서비스를 제공합니다. 타사 감사자는 AWS 규정 준수 프로그램 일환으로 보안의 효과를 정기적으로 테스트하고 확인합니다. CodeArtifact에 적용되는 규정 준수 프로그램에 대한 자세한 내용은 규정 준수 프로그램 제공 범위 내 AWS 서비스를 참조하세요.
- 클라우드의 보안 - 사용자의 책임은 사용하는 AWS 서비스에 따라 결정됩니다. 또한 귀하는 귀사의 데이터 민감도, 귀사의 요구 사항, 관련 법률 및 규정을 비롯한 기타 요소에 대해서도 책임이 있습니다.

이 설명서는 CodeArtifact 사용 시 공동 책임 모델을 적용하는 방법을 이해하는 데 도움이 됩니다. 다음 주제에서는 보안 및 규정 준수 목표를 충족하도록 CodeArtifact를 구성하는 방법을 보여줍니다. 또한 CodeArtifact 리소스를 모니터링하고 보호하는 데 도움이 되는 다른 서비스를 사용하는 방법을 알아봅니다.

주제

- in AWS CodeArtifact의 데이터 보호
- CodeArtifact 모니터링
- AWS CodeArtifact의 규정 준수 검증
- AWS CodeArtifact 인증 및 토큰
- Resilience in AWS CodeArtifact
- 인프라 보안 in AWS CodeArtifact
- 종속성 대체 공격
- Identity and Access Management for AWS CodeArtifact

in AWS CodeArtifact의 데이터 보호

AWS 공동 책임 모델 AWS CodeArtifact의 데이터 보호에 적용됩니다. 이 모델에 설명된 대로 AWS 는 모든를 실행하는 글로벌 인프라를 보호할 책임이 있습니다 AWS 클라우드. 사용자는 이 인프라에 호스팅되는 콘텐츠에 대한 통제 권한을 유지할 책임이 있습니다. 사용하는 AWS 서비스 의 보안 구성과 관리 태스크에 대한 책임도 사용자에게 있습니다. 데이터 프라이버시에 관한 자세한 내용은 데이터 프라이버시 FAQ를 참조하세요. 유럽의 데이터 보호에 대한 자세한 내용은 AWS 보안 블로그의 AWS 공동 책임 모델 및 GDPR 블로그 게시물을 참조하세요.

데이터 보호를 위해 자격 증명을 보호하고 AWS 계정 AWS IAM Identity Center 또는 AWS Identity and Access Management (IAM)를 사용하여 개별 사용자를 설정하는 것이 좋습니다. 이렇게 하면 개별 사용자에게 자신의 직무를 충실히 이행하는 데 필요한 권한만 부여됩니다. 또한 다음과 같은 방법으로 데이터를 보호하는 것이 좋습니다.

- 각 계정에 다중 인증(MFA)을 사용합니다.
- SSL/TLS를 사용하여 AWS 리소스와 통신합니다. TLS 1.2는 필수이며 TLS 1.3을 권장합니다.
- 를 사용하여 API 및 사용자 활동 로깅을 설정합니다 AWS CloudTrail. CloudTrail 추적을 사용하여 AWS 활동을 캡처하는 방법에 대한 자세한 내용은 AWS CloudTrail 사용 설명서의 CloudTrail 추적 작업을 참조하세요.
- 내부의 모든 기본 보안 제어와 함께 AWS 암호화 솔루션을 사용합니다 AWS 서비스.
- Amazon S3에 저장된 민감한 데이터를 검색하고 보호하는 데 도움이 되는 Amazon Macie와 같은 고급 관리형 보안 서비스를 사용합니다.
- 명령줄 인터페이스 또는 API를 AWS 통해에 액세스할 때 FIPS 140-3 검증 암호화 모듈이 필요한 경우 FIPS 엔드포인트를 사용합니다. 사용 가능한 FIPS 엔드포인트에 대한 자세한 내용은 연방 정보 처리 표준(FIPS) 140-3을 참조하세요.

고객의 이메일 주소와 같은 기밀 정보나 중요한 정보는 태그나 이름 필드와 같은 자유 형식 텍스트 필드에 입력하지 않는 것이 좋습니다. 여기에는 CodeArtifact 또는 기타 AWS 서비스 에서 콘솔 AWS CLI, API 또는 AWS SDKs를 사용하여 작업하는 경우가 포함됩니다. 이름에 사용되는 태그 또는 자유 형식 텍스트 필드에 입력하는 모든 데이터는 청구 또는 진단 로그에 사용될 수 있습니다. 외부 서버에 URL을 제공할 때 해당 서버에 대한 요청을 검증하기 위해 자격 증명을 URL에 포함해서는 안 됩니다.

데이터 암호화

암호화는 CodeArtifact 보안의 중요한 부분입니다. 전송 중인 데이터 암호화와 같은 일부 암호화 기능은 기본으로 제공되며 암호화 프로그램을 추가할 필요가 없습니다. 유휴 상태의 데이터 암호화와 같은 기타 암호화는 프로젝트나 빌드 생성 시 구성할 수 있습니다.

- 저장 데이터 암호화 - CodeArtifact에 저장된 모든 자산은 AWS KMS keys (KMS 키)를 사용하여 암호화됩니다. 여기에는 모든 리포지토리의 모든 패키지에 있는 모든 자산이 포함됩니다. 도메인별로 KMS 키 하나를 이용해 모든 자산을 암호화합니다. 기본적으로 AWS 관리형 KMS 키가 사용되므로 KMS 키를 생성할 필요가 없습니다. 원한다면 직접 만들고 구성한 고객 관리형 KMS 키를 사용해도 됩니다. 자세한 내용은 AWS Key Management Service 사용 설명서의 키 생성 및 AWS 키 관리 서비스 개념을 참조하세요. 도메인을 생성할 때 고객 관리형 키를 지정할 수 있습니다. 자세한 내용은 CodeArtifact에서의 도메인 작업 단원을 참조하십시오.
- 전송 중 데이터 암호화 - 고객과 CodeArtifact 간, CodeArtifact와 다운스트림 종속성 간의 모든 통신은 TLS 암호화를 사용하여 보호됩니다.

트래픽 개인 정보 보호

인터페이스 Virtual Private Cloud(VPC) 엔드포인트를 사용하도록 CodeArtifact를 구성하면 CodeArtifact 도메인과 도메인에 포함된 자산의 보안을 개선할 수 있습니다. 이 작업은 인터넷 게이트웨이, NAT 디바이스 또는 가상 프라이빗 게이트웨이가 없어도 할 수 있습니다. 자세한 내용은 Amazon VPC 엔드포인트 작업 단원을 참조하십시오. AWS PrivateLink 및 VPC 엔드포인트에 대한 자세한 내용은 AWS PrivateLink 및 PrivateLink를 통한 AWS 서비스 액세스를 참조하세요.

CodeArtifact 모니터링

모니터링은 AWS CodeArtifact 및 AWS 솔루션의 안정성, 가용성 및 성능을 유지하는 데 중요한 부분입니다. 다중 지점 장애가 발생할 경우 보다 쉽게 디버깅할 수 있도록 AWS 솔루션의 모든 부분으로부터 모니터링 데이터를 수집해야 합니다.는 CodeArtifact 리소스를 모니터링하고 잠재적 인시던트에 대응하기 위해 다음을 AWS 제공합니다.

주제

- 를 사용하여 CodeArtifact API 호출 로깅 AWS CloudTrail

를 사용하여 CodeArtifact API 호출 로깅 AWS CloudTrail

CodeArtifact는 CodeArtifact에서 사용자AWS CloudTrail, 역할 또는 서비스가 수행한 작업에 대한 레코드를 제공하는 AWS 서비스인와 통합됩니다. CloudTrail은 패키지 관리자 클라이언트의 호출을 포함하여 CodeArtifact에 대한 모든 API 호출을 이벤트로 캡처합니다.

추적을 생성하면 CodeArtifact에 대한 이벤트를 포함한 CloudTrail 이벤트를 Amazon Simple Storage Service(S3) 버킷에 지속적으로 전송할 수 있습니다. 추적을 구성하지 않은 경우에도 이벤트 기록에서 CloudTrail 콘솔의 최신 이벤트를 볼 수 있습니다. CloudTrail에서 수집한 정보를 사용하여 CodeArtifact

에 수행된 요청, 요청이 수행된 IP 주소, 요청을 수행한 사람, 요청이 수행된 시간 및 추가 세부 정보를 확인할 수 있습니다.

CloudTrail에 대한 자세한 내용은 AWS CloudTrail 사용 설명서를 참조하세요.

CloudTrail의 CodeArtifact 정보

AWS 계정을 생성할 때 계정에서 CloudTrail이 활성화됩니다. CodeArtifact에서 활동이 발생하면 해당 활동이 이벤트 기록의 다른 AWS 서비스 이벤트와 함께 CloudTrail 이벤트에 기록됩니다. AWS 계정에서 최근 이벤트를 보고 검색하고 다운로드할 수 있습니다. 자세한 설명은 CloudTrail 이벤트 기록으로 이벤트 보기를 참조하세요.

CodeArtifact에 대한 이벤트를 포함하여 AWS 계정에 이벤트를 지속적으로 기록하려면 추적을 생성합니다. CloudTrail은 추적을 사용하여 Amazon S3 버킷으로 로그 파일을 전송할 수 있습니다. 콘솔에서 추적을 생성하면 기본적으로 모든 리전에 추적이 적용됩니다. 추적은 AWS 파티션의 모든 리전에서 이벤트를 로깅하고 지정한 Amazon S3 버킷으로 로그 파일을 전송합니다. CloudTrail 로그에서 수집된 이벤트 데이터를 추가 분석 및 처리하도록 다른 AWS 서비스를 구성할 수도 있습니다. 자세한 내용은 다음 항목을 참조하세요.

- AWS 계정에 대한 추적 생성
- CloudTrail 지원 서비스 및 통합
- CloudTrail에서 Amazon SNS 알림 구성

AWS 계정에서 CloudTrail 로깅이 활성화되면 CodeArtifact 작업에 대한 API 호출이 CloudTrail 로그 파일에서 추적되며, 여기에서 다른 AWS 서비스 레코드와 함께 기록됩니다. CloudTrail은 기간 및 파일 크기를 기준으로 새 파일을 만들고 기록하는 시점을 결정합니다.

모든 CodeArtifact 작업은 CloudTrail에서 로깅됩니다. 예를 들어 `ListRepositories` (AWS CLI, `aws codeartifact list-repositories`), `CreateRepository` (`aws codeartifact create-repository`) 및 `ListPackages` (`aws codeartifact list-packages`) 작업에 대한 호출은 패키지 관리자 클라이언트 명령 외에도 CloudTrail 로그 파일에 항목을 생성합니다. 패키지 관리자 클라이언트 명령은 일반적으로 서버에 두 개 이상의 HTTP 요청을 보냅니다. 각 요청은 별도의 CloudTrail 로그 이벤트를 생성합니다.

CloudTrail 로그의 계정 간 전송

단일 API 호출 당 최대 세 개의 개별 계정이 CloudTrail 로그를 수신합니다.

- 요청한 계정(예: `GetAuthorizationToken`를 호출한 계정)

- 리포지토리 관리자 계정(예: ListPackages가 호출된 리포지토리를 관리하는 계정)
- 도메인 소유자 계정(예: API가 호출된 리포지토리가 포함된 도메인을 소유하는 계정)

특정 리포지토리가 아닌 도메인에 대한 작업인 ListRepositoriesInDomain과 같은 API의 경우, 호출 계정과 도메인 소유자 계정만 CloudTrail 로그를 수신합니다. 리소스에 대해 인증되지 않은 ListRepositories과 같은 API의 경우 호출자의 계정만 CloudTrail 로그를 수신합니다.

CodeArtifact 로그 파일 항목 이해

CloudTrail 로그 파일에는 하나 이상의 로그 항목이 포함될 수 있습니다. 각 항목은 여러 개의 JSON 형식 이벤트를 표시합니다. 로그 이벤트는 모든 소스의 단일 요청을 나타내며 요청된 작업, 작업 날짜와 시간, 요청 파라미터 등에 대한 정보를 포함합니다. 로그 항목은 퍼블릭 API 직접 호출의 주문 스택 트레이스가 아니기 때문에 특정 순서로 표시되지 않습니다.

주제

예: GetAuthorizationToken API 호출을 위한 로그 항목

GetAuthorizationToken에서 만든 로그 항목은 requestParameters 필드에 도메인 이름을 포함합니다.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
      "type": "AssumedRole",
      "principalId": "AIDACKCEVSQ6C2EXAMPLE",
      "arn": "arn:aws:sts::123456789012:assumed-role/Console/example",
      "accountId": "123456789012",
      "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
      "sessionContext": {
          "attributes": {
              "mfaAuthenticated": "false",
              "creationDate": "2018-12-11T13:31:37Z"
          },
          "sessionIssuer": {
              "type": "Role",
              "principalId": "AIDACKCEVSQ6C2EXAMPLE",
              "arn": "arn:aws:iam::123456789012:role/Console",
```

```
            "accountId": "123456789012",
            "userName": "Console"
        }
    }
  },
  "eventTime": "2018-12-11T13:31:37Z",
  "eventSource": "codeartifact.amazonaws.com",
  "eventName": "GetAuthorizationToken",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "205.251.233.50",
  "userAgent": "aws-cli/1.16.37 Python/2.7.10 Darwin/16.7.0 botocore/1.12.27",
  "requestParameters": {
      "domainName": "example-domain"
      "domainOwner": "123456789012"
  },
  "responseElements": {
      "sessionToken": "HIDDEN_DUE_TO_SECURITY_REASONS"
  },
  "requestID": "6b342fc0-5bc8-402b-a7f1-ffffffffffff",
  "eventID": "100fde01-32b8-4c2b-8379-ffffffffffff",
  "readOnly": false,
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}
```

예: npm 패키지 버전을 가져오기 위한 로그 항목

npm 클라이언트를 포함한 모든 패키지 관리자 클라이언트의 요청에는 `requestParameters` 필드에 도메인 이름, 리포지토리 이름, 패키지 이름을 비롯한 추가 데이터가 기록됩니다. URL 경로와 HTTP 메서드가 `additionalEventData` 필드에 기록됩니다.

```
{
   "eventVersion": "1.05",
   "userIdentity": {
       "type": "AssumedRole",
       "principalId": "AIDACKCEVSQ6C2EXAMPLE",
       "arn": "arn:aws:sts::123456789012:assumed-role/Console/example",
       "accountId": "123456789012",
       "accessKeyId": "ASIAIJIOBJIBSREXAMPLE",
       "sessionContext": {
           "attributes": {
               "mfaAuthenticated": "false",
               "creationDate": "2018-12-17T02:05:16Z"
```

```
            },
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AIDACKCEVSQ6C2EXAMPLE",
                "arn": "arn:aws:iam::123456789012:role/Console",
                "accountId": "123456789012",
                "userName": "Console"
            }
        }
    },
    "eventTime": "2018-12-17T02:05:46Z",
    "eventSource": "codeartifact.amazonaws.com",
    "eventName": "ReadFromRepository",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "205.251.233.50",
    "userAgent": "npm/6.14.15 node/v12.22.9 linux x64 ci/custom",
    "requestParameters": {
        "domainName": "example-domain",
        "domainOwner": "123456789012",
        "repositoryName": "example-repo",
        "packageName": "lodash",
        "packageFormat": "npm",
        "packageVersion": "4.17.20"
    },
    "responseElements": null,
    "additionalEventData": {
        "httpMethod": "GET",
        "requestUri": "/npm/lodash/-/lodash-4.17.20.tgz"
    },
    "requestID": "9f74b4f5-3607-4bb4-9229-ffffffffffff",
    "eventID": "c74e40dd-8847-4058-a14d-ffffffffffff",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```


AWS CodeArtifact의 규정 준수 검증

AWS 서비스가 특정 규정 준수 프로그램의 범위에 포함되는지 알아보려면 규정 준수 프로그램 제공 범위 내 AWS 서비스를 참조하고 관심 있는 규정 준수 프로그램을 선택하십시오. 일반적인 정보는 AWS 규정 준수 프로그램을 참조하세요.

AWS Artifact(을)를 사용하여 타사 감사 보고서를 다운로드할 수 있습니다. 자세한 내용은 AWS Artifact에서 보고서 다운로드를 참조하세요.

AWS 서비스 사용 시 규정 준수 책임은 데이터의 민감도, 회사의 규정 준수 목표 및 관련 법률 및 규정에 따라 결정됩니다. AWS 서비스 사용 시 규정 준수 책임에 대한 자세한 내용은 AWS 보안 설명서를 참조하세요.

AWS CodeArtifact 인증 및 토큰

CodeArtifact에서 패키지 버전을 게시하거나 사용하려면 서비스에 인증해야 합니다. AWS 보안 인증을 사용하고 인증 토큰을 생성하여 CodeArtifact 서비스에 인증해야 합니다. 인증 토큰을 생성하려면 올바른 권한이 있어야 합니다. 인증 토큰을 만드는 데 필요한 권한은 AWS CodeArtifact 권한 참조의 `GetAuthorizationToken` 항목을 참조하세요. CodeArtifact 권한에 대한 일반적인 내용은 AWS CodeArtifact가 IAM에서 작동하는 방식 섹션을 참조하세요.

CodeArtifact에서 인증 토큰을 가져오려면 GetAuthorizationToken API를 호출해야 합니다. AWS CLI를 사용하면 `login` 또는 `get-authorization-token` 명령으로 `GetAuthorizationToken`를 호출할 수 있습니다.

> **Note**
>
> 루트 사용자는 `GetAuthorizationToken`을 호출할 수 없습니다.

- `aws codeartifact login`: 이 명령을 사용하면 CodeArtifact를 단일 단계에서 사용하도록 일반 패키지 관리자를 쉽게 구성할 수 있습니다. `login`을 호출하면 `GetAuthorizationToken` 포함 토큰을 가져오고 해당 토큰과 올바른 CodeArtifact 리포지토리 엔드포인트를 사용하여 패키지 관리자를 구성합니다. 지원 패키지 관리자는 다음과 같습니다.
  - dotnet
  - npm
  - nuget
  - pip
  - swift
  - twine
- `aws codeartifact get-authorization-token`: `login`에서 지원하지 않는 패키지 관리자의 경우 직접 `get-authorization-token`을 호출한 다음, 필요에 따라 토큰을 사용하여 패키지 관리자를 구성할 수 있습니다(예: 구성 파일에 추가하거나 환경 변수에 저장).

CodeArtifact 인증 토큰은 기본 기간인 12시간 동안 유효합니다. 토큰의 수명은 15분에서 12시간 사이로 구성할 수 있습니다. 수명이 만료되면 다른 토큰을 가져와야 합니다. 토큰 수명은 `login` 또는 `get-authorization-token` 호출 이후에 시작됩니다.

역할을 수임하는 동안 `login` 또는 `get-authorization-token` 를 호출하는 경우 값을 `--duration-seconds~0`로 설정하여 토큰의 수명을 역할의 세션 지속 기간 중 남은 시간과 같게 구성할 수 있습니다. 그렇지 않으면 토큰 수명은 역할의 최대 세션 지속 기간과 무관합니다. 예를 들어, `sts assume-role`을 호출하여 세션 지속 기간을 15분으로 지정한 다음 CodeArtifact 인증 토큰을 가져오기 위해 `login`을 호출한다고 가정해 보겠습니다. 이 경우 토큰은 세션 지속 기간인 15분보다 길더라도 전체 12시간 동안 유효합니다. 세션 지속 기간 제어에 대한 자세한 내용은 IAM 사용 설명서의 IAM 역할 사용을 참조하세요.

`login` 명령으로 생성된 토큰

`aws codeartifact login` 명령은 `GetAuthorizationToken` 포함 토큰을 가져오고 해당 토큰과 올바른 CodeArtifact 리포지토리 엔드포인트를 사용하여 패키지 관리자를 구성합니다.

다음 표는 `login` 명령의 파라미터를 설명합니다.

파라미터	필수	설명
`--tool`	예	인증할 패키지 관리자입니다. 가능한 값은 `dotnet`, `npm`, `nuget`, `pip`, `swift`, `twine`입니다.
`--domain`	예	리포지토리가 속한 도메인 이름입니다.
`--domain-owner`	아니요	도메인 소유자의 ID입니다. 이 파라미터는 인증되지 않은 AWS 계정이 소유한 도메인에 액세스하는 경우 필요합니다. 자세한 내용은 크로스 계정 도메인 섹션을 참조하세요.
`--repository`	예	인증할 리포지토리의 이름입니다.
`--duration-seconds`	아니요	로그인 정보가 유효한 시간(초)입니다. 최솟값은 900*이고 최댓값은 43200입니다.

파라미터	필수	설명
`--namespace`	아니요	네임스페이스를 리포지토리 도구와 연결합니다.
`--dry-run`	아니요	구성을 변경하지 않고 도구를 리포지토리에 연결하기 위해 실행되는 명령만 출력합니다.

*0 값은 역할을 수임한 상태에서 `login`을 호출할 때도 유효합니다. `--duration-seconds 0` 를 사용하여 `login`을 호출하면 수임된 역할의 남은 세션 지속 기간과 동일한 수명을 가진 토큰이 생성됩니다.

다음 예제에서는 `login` 명령으로 인증 토큰을 가져오는 방법을 보여줍니다.

```
aws codeartifact login \
    --tool dotnet | npm | nuget | pip | swift | twine \
    --domain my_domain \
    --domain-owner 111122223333 \
    --repository my_repo
```

npm과 함께 `login` 명령을 사용하는 방법에 대한 구체적인 지침은 CodeArtifact로 npm 구성 및 사용 섹션을 참조하세요. Python의 경우 CodeArtifact를 Python과 함께 사용 섹션을 참조하세요.

`GetAuthorizationToken` API를 호출하려면 권한이 필요합니다.

CodeArtifact `GetAuthorizationToken` API를 호출하려면 `sts:GetServiceBearerToken` 및 `codeartifact:GetAuthorizationToken` 권한이 모두 필요합니다.

CodeArtifact 리포지토리와 함께 패키지 관리자를 사용하려면 IAM 사용자 또는 역할이 `sts:GetServiceBearerToken`을 허용해야 합니다. `sts:GetServiceBearerToken`을 CodeArtifact 도메인 리소스 정책에 추가할 수 있지만 권한이 해당 정책에 영향을 주지는 않습니다.

`GetAuthorizationToken` API로 생성된 토큰

CodeArtifact에서 인증 토큰을 가져오기 위해 `get-authorization-token`를 호출할 수 있습니다.

```
aws codeartifact get-authorization-token \
    --domain my_domain \
```

```
    --domain-owner 111122223333 \
    --query authorizationToken \
    --output text
```

`--duration-seconds` 인수를 사용하여 토큰의 유효 기간을 변경할 수 있습니다. 최솟값은 900이고 최댓값은 43200입니다. 다음 예제에서는 1시간(3,600초) 동안 지속되는 토큰을 생성합니다.

```
aws codeartifact get-authorization-token \
    --domain my_domain \
    --domain-owner 111122223333 \
    --query authorizationToken \
    --output text \
    --duration-seconds 3600
```

역할을 수임하면서 `get-authorization-token`을 호출하는 경우 토큰 수명은 역할의 최대 세션 기간과 무관합니다. `--duration-seconds` 값을 0으로 설정하여 위임된 역할의 세션 기간이 만료될 때 토큰이 만료되도록 구성할 수 있습니다.

```
aws codeartifact get-authorization-token \
    --domain my_domain \
    --domain-owner 111122223333 \
    --query authorizationToken \
    --output text \
    --duration-seconds 0
```

자세한 내용은 다음 설명서를 참조하세요.

- 토큰 및 환경 변수에 대한 지침은 환경 변수를 사용하여 인증 토큰 전달 섹션을 참조하세요.
- Python 사용자의 경우 로그인 명령 없이 pip를 구성하려면 또는 CodeArtifact로 twine 구성 및 사용 섹션을 참조하세요.
- Maven 사용자의 경우 Gradle과 함께 CodeArtifact 사용 또는 mvn과 함께 CodeArtifact 사용 섹션을 참조하세요.
- npm 사용자의 경우 로그인 명령을 실행하지 않고 npm 구성하기 섹션을 참조하세요.

환경 변수를 사용하여 인증 토큰 전달

AWS CodeArtifact는 `GetAuthorizationToken` API에서 제공하는 인증 토큰을 사용하여 Maven 및 Gradle과 같은 빌드 도구의 요청을 인증하고 권한을 부여합니다. 이러한 인증 토큰에 대한 자세한 내용은 `GetAuthorizationToken` API로 생성된 토큰 섹션을 참조하세요.

CodeArtifact 리포지토리에서 패키지를 가져오거나 패키지를 게시하는 데 필요한 토큰을 얻기 위해 빌드 도구가 읽을 수 있는 환경 변수에 이러한 인증 토큰을 저장할 수 있습니다.

보안상의 이유로 이 접근 방식은 다른 사용자나 프로세스가 토큰을 읽거나 실수로 소스 제어에 체크인할 수 있는 파일에 토큰을 저장하는 것보다 좋습니다.

1. 설치 또는 업그레이드 후 구성 AWS CLI에 설명된 대로 AWS 보안 인증을 구성합니다.
2. `CODEARTIFACT_AUTH_TOKEN` 환경 변수를 설정합니다.

> **Note**
>
> 일부 시나리오에서는 `--domain-owner` 인수를 포함하지 않아도 됩니다. 자세한 내용은 크로스 계정 도메인 섹션을 참조하세요.

- macOS 또는 Linux:

```
export CODEARTIFACT_AUTH_TOKEN=`aws codeartifact get-authorization-token --
domain my_domain --domain-owner 111122223333 --query authorizationToken --output
 text`
```

- Windows(기본 명령 쉘 사용):

```
for /f %i in ('aws codeartifact get-authorization-token --domain my_domain --
domain-owner 111122223333 --query authorizationToken --output text') do set
 CODEARTIFACT_AUTH_TOKEN=%i
```

- Windows PowerShell:

```
$env:CODEARTIFACT_AUTH_TOKEN = aws codeartifact get-authorization-token --
domain my_domain --domain-owner 111122223333 --query authorizationToken --output
 text
```

CodeArtifact 인증 토큰 취소

인증된 사용자가 CodeArtifact 리소스에 액세스하기 위한 토큰을 생성하면 해당 토큰은 사용자 지정 가능한 액세스 기간이 종료될 때까지 지속됩니다. 기본 액세스 기간은 12시간입니다. 액세스 기간이 만료되기 전에 토큰에 대한 액세스를 취소해야 하는 경우도 있습니다. 다음 지침에 따라 CodeArtifact 리소스에 대한 액세스 권한을 취소할 수 있습니다.

수임한 역할 또는 페더레이션 사용자 액세스와 같은 임시 보안 자격 증명을 사용하여 액세스 토큰을 생성한 경우 액세스를 거부하도록 IAM 정책을 업데이트하여 액세스를 취소할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 임시 보안 자격 증명에 대한 권한 비활성화를 참조하세요.

장기 IAM 사용자 보안 인증을 사용하여 액세스 토큰을 생성한 경우 액세스를 거부하도록 사용자 정책을 수정하거나 IAM 사용자를 삭제해야 합니다. 자세한 정보는 IAM 사용자의 권한 변경 또는 IAM 사용자 삭제를 참조하세요.

Resilience in AWS CodeArtifact

AWS 글로벌 인프라는 AWS 리전 및 가용 영역을 중심으로 구축됩니다. AWS 리전은 물리적으로 분리되고 격리된 여러 가용 영역을 제공하며,이 가용 영역은 지연 시간이 짧고 처리량이 높으며 중복성이 높은 네트워킹과 연결됩니다. AWS CodeArtifact는 여러 가용 영역에서 작동하며 아티팩트 데이터와 메타데이터를 Amazon S3와 Amazon DynamoDB에 저장합니다. 암호화된 데이터가 여러 시설과 각 시설의 여러 디바이스에 중복 저장되어 가용성과 내구성이 뛰어납니다.

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

인프라 보안 in AWS CodeArtifact

관리형 서비스인 AWS CodeArtifact는 AWS 글로벌 네트워크 보안으로 보호됩니다. AWS 보안 서비스 및가 인프라를 AWS 보호하는 방법에 대한 자세한 내용은 AWS 클라우드 보안을 참조하세요. 인프라 보안 모범 사례를 사용하여 환경을 설계하려면 보안 원칙 AWS Well-Architected Framework의 인프라 보호를 참조하세요 AWS .

AWS 에서 게시한 API 호출을 사용하여 네트워크를 통해 CodeArtifact에 액세스합니다. 클라이언트는 다음을 지원해야 합니다.

- Transport Layer Security(TLS). TLS 1.2는 필수이며 TLS 1.3을 권장합니다.
- DHE(Ephemeral Diffie-Hellman) 또는 ECDHE(Elliptic Curve Ephemeral Diffie-Hellman)와 같은 완전 전송 보안(PFS)이 포함된 암호 제품군. Java 7 이상의 최신 시스템은 대부분 이러한 모드를 지원합니다.

종속성 대체 공격

패키지 관리자는 재사용 가능한 코드를 패키징하고 공유하는 프로세스를 단순화합니다. 이러한 패키지는 애플리케이션에서 사용하기 위해 조직에서 개발한 프라이빗 패키지일 수도 있고, 조직 외부에서 개발되어 퍼블릭 패키지 리포지토리에서 배포하는 퍼블릭(대부분의 경우 오픈 소스) 패키지일 수도 있

습니다. 패키지를 요청할 때 개발자는 패키지 관리자를 이용해 종속 항목의 새 버전을 가져옵니다. 종속성 혼동 공격이라고도 하는 종속성 대체 공격은 대부분의 패키지 관리자가 패키지의 합법적인 버전과 악성 버전을 구분할 방법이 없다는 사실을 악용합니다.

종속성 대체 공격은 소프트웨어 공급망 공격이라 알려진 해킹의 하위 집합에 속합니다. 소프트웨어 공급망 공격은 소프트웨어 공급망에 존재하는 취약성을 악용하는 공격입니다.

종속성 대체 공격은 내부적으로 개발된 패키지와 퍼블릭 리포지토리에서 가져온 패키지를 모두 사용하는 사용자라면 누구나 노릴 수 있습니다. 공격자는 내부 패키지 이름을 식별한 다음 같은 이름의 악성 코드를 전략적으로 퍼블릭 패키지 리포지토리에 배치합니다. 일반적으로 악성 코드는 버전 번호가 높은 패키지에 게시됩니다. 패키지 관리자는 악성 패키지가 패키지 최신 버전이라고 생각하기 때문에 이러한 퍼블릭 피드에서 악성 코드를 가져옵니다. 결과적으로 원하는 패키지와 악성 패키지 간에 '혼동' 또는 '대체'가 발생하여 코드가 손상될 수 있습니다.

종속성 대체 공격을 방지하기 위해 AWS CodeArtifact는 패키지 원본 제어 기능을 제공합니다. 패키지 원본 제어는 패키지를 리포지토리에 추가하는 방법을 제어하는 설정입니다. 제어를 사용하면 패키지 버전을 리포지토리에 직접 게시하고 퍼블릭 소스에서 수집할 수 없으므로 종속성 대체 공격으로부터 보호할 수 있습니다. 패키지 그룹에 원본 제어를 설정하여 개별 패키지 및 여러 패키지에 원본 제어를 설정할 수 있습니다. 패키지 원본 제어 및 이를 변경하는 방법에 대한 자세한 내용은 패키지 원본 제어 편집 및 패키지 그룹 원본 제어 섹션을 참조하세요.

Identity and Access Management for AWS CodeArtifact

AWS Identity and Access Management (IAM)는 관리자가 AWS 리소스에 대한 액세스를 안전하게 제어하는 데 도움이 AWS 서비스 되는 입니다. IAM 관리자는 어떤 사용자가 CodeArtifact 리소스를 사용할 수 있는 인증(로그인) 및 권한(권한 있음)을 받을 수 있는지 제어합니다. IAM은 추가 비용 없이 사용할 수 AWS 서비스 있는 입니다.

주제

- 대상
- ID를 통한 인증
- 정책을 사용하여 액세스 관리
- AWS CodeArtifact가 IAM에서 작동하는 방식
- AWS CodeArtifact에 대한 자격 증명 기반 정책 예제
- 태그를 사용하여 CodeArtifact 리소스에 대한 액세스 제어
- AWS CodeArtifact 권한 참조
- 문제 해결 AWS CodeArtifact 자격 증명 및 액세스

대상

AWS Identity and Access Management (IAM)를 사용하는 방법은 역할에 따라 다릅니다.

- 서비스 사용자 - 기능에 액세스할 수 없는 경우 관리자에게 권한 요청(참조문제 해결 AWS CodeArtifact 자격 증명 및 액세스)
- 서비스 관리자 - 사용자 액세스 결정 및 권한 요청 제출(AWS CodeArtifact가 IAM에서 작동하는 방식 참조)
- IAM 관리자 - 액세스를 관리하기 위한 정책 작성(AWS CodeArtifact에 대한 자격 증명 기반 정책 예제 참조)

ID를 통한 인증

인증은 AWS 자격 증명으로에 로그인하는 방법입니다. AWS 계정 루트 사용자, IAM 사용자 또는 IAM 역할을 수임하여 인증되어야 합니다.

AWS IAM Identity Center (IAM Identity Center), Single Sign-On 인증 또는 Google/Facebook 자격 증명과 같은 자격 증명 소스의 자격 증명을 사용하여 페더레이션 자격 증명으로 로그인할 수 있습니다. 로그인하는 방법에 대한 자세한 내용은 AWS 로그인 사용 설명서의 AWS 계정에 로그인하는 방법 섹션을 참조하세요.

프로그래밍 방식 액세스를 위해는 요청에 암호화 방식으로 서명할 수 있는 SDK 및 CLI를 AWS 제공합니다. 자세한 내용은 IAM 사용 설명서의 API 요청용AWS Signature Version 4 섹션을 참조하세요.

AWS 계정 루트 사용자

를 생성할 때 모든 AWS 서비스 및 리소스에 대한 완전한 액세스 권한이 있는 AWS 계정 theroot 사용자라는 하나의 로그인 자격 증명으로 AWS 계정시작합니다. 일상적인 태스크에 루트 사용자를 사용하지 않을 것을 강력히 권장합니다. 루트 사용자 자격 증명이 필요한 작업은 IAM 사용 설명서의 루트 사용자 자격 증명이 필요한 작업 섹션을 참조하세요.

페더레이션 ID

가장 좋은 방법은 인간 사용자에게 자격 증명 공급자와의 페더레이션을 사용하여 임시 자격 증명을 AWS 서비스 사용하여에 액세스하도록 요구하는 것입니다.

페더레이션 자격 증명은 엔터프라이즈 디렉터리, 웹 자격 증명 공급자 또는 자격 증명 소스의 자격 증명을 AWS 서비스 사용하여 Directory Service 에 액세스하는 사용자입니다. 페더레이션 ID는 임시 자격 증명을 제공하는 역할을 수임합니다.

중앙 집중식 액세스 관리를 위해 AWS IAM Identity Center를 추천합니다. 자세한 정보는 AWS IAM Identity Center 사용 설명서의 What is IAM Identity Center?를 참조하세요.

IAM 사용자 및 그룹

IAM 사용자는 단일 개인 또는 애플리케이션에 대한 특정 권한을 가진 ID입니다. 장기 자격 증명이 있는 IAM 사용자 대신 임시 자격 증명을 사용하는 것이 좋습니다. 자세한 내용은 IAM 사용 설명서의 자격 증명 공급자와의 페더레이션을 사용하여 임시 자격 증명을 AWS 사용하여에 액세스하도록 인간 사용자에게 요구하기를 참조하세요.

IAM 그룹은 IAM 사용자 모음을 지정하고 대규모 사용자 집합에 대한 관리 권한을 더 쉽게 만듭니다. 자세한 내용은 IAM 사용 설명서의 IAM 사용자 사용 사례 섹션을 참조하세요.

IAM 역할

IAM 역할은 임시 자격 증명을 제공하는 특정 권한이 있는 자격 증명입니다. 사용자에서 IAM 역할(콘솔)로 전환하거나 또는 API 작업을 호출하여 역할을 수임할 수 있습니다. AWS CLI AWS 자세한 내용은 IAM 사용 설명서의 역할 수임 방법을 참조하세요.

IAM 역할은 페더레이션 사용자 액세스, 임시 IAM 사용자 권한, 교차 계정 액세스, 교차 서비스 액세스 및 Amazon EC2에서 실행되는 애플리케이션에 유용합니다. 자세한 내용은 IAM 사용 설명서의 교차 계정 리소스 액세스를 참조하세요.

정책을 사용하여 액세스 관리

정책을 AWS 생성하고 자격 증명 또는 리소스에 연결하여 AWS 에서 액세스를 제어합니다. 정책은 자격 증명 또는 리소스와 연결될 때 권한을 정의합니다.는 보안 주체가 요청할 때 이러한 정책을 AWS 평가합니다. 대부분의 정책은에 JSON 문서 AWS 로 저장됩니다. JSON 정책 문서에 대한 자세한 내용은 IAM 사용 설명서의 JSON 정책 개요 섹션을 참조하세요.

정책을 사용하여 관리자는 어떤 보안 주체가 어떤 리소스에 대해 어떤 조건에서 작업을 수행할 수 있는지 정의하여 누가 무엇을 액세스할 수 있는지 지정합니다.

기본적으로 사용자 및 역할에는 어떠한 권한도 없습니다. IAM 관리자는 IAM 정책을 생성하고 사용자가 수임할 수 있는 역할에 추가합니다. IAM 정책은 작업을 수행하기 위해 사용하는 방법과 관계없이 작업에 대한 권한을 정의합니다.

ID 기반 정책

ID 기반 정책은 ID(사용자, 사용자 그룹 또는 역할)에 연결하는 JSON 권한 정책 문서입니다. 이러한 정책은 자격 증명이 수행할 수 있는 작업, 대상 리소스 및 이에 관한 조건을 제어합니다. ID 기반 정책

을 생성하는 방법을 알아보려면 IAM 사용 설명서에서 고객 관리형 정책으로 사용자 지정 IAM 권한 정의를 참조하세요.

ID 기반 정책은 인라인 정책(단일 ID에 직접 포함) 또는 관리형 정책(여러 ID에 연결된 독립 실행형 정책)일 수 있습니다. 관리형 정책 또는 인라인 정책을 선택하는 방법을 알아보려면 IAM 사용 설명서의 관리형 정책 및 인라인 정책 중에서 선택 섹션을 참조하세요.

리소스 기반 정책

리소스 기반 정책은 리소스에 연결하는 JSON 정책 설명서입니다. 예를 들어 IAM 역할 신뢰 정책 및 Amazon S3 버킷 정책이 있습니다. 리소스 기반 정책을 지원하는 서비스에서 서비스 관리자는 이러한 정책을 사용하여 특정 리소스에 대한 액세스를 통제할 수 있습니다. 리소스 기반 정책에서 보안 주체를 지정해야 합니다.

리소스 기반 정책은 해당 서비스에 있는 인라인 정책입니다. 리소스 기반 정책에서는 IAM의 AWS 관리형 정책을 사용할 수 없습니다.

기타 정책 유형

AWS 는 보다 일반적인 정책 유형에서 부여한 최대 권한을 설정할 수 있는 추가 정책 유형을 지원합니다.

- 권한 경계 - ID 기반 정책에서 IAM 엔터티에 부여할 수 있는 최대 권한을 설정합니다. 자세한 정보는 IAM 사용 설명서의 IAM 엔터티의 권한 범위를 참조하세요.
- 서비스 제어 정책(SCP) - AWS Organizations내 조직 또는 조직 단위에 대한 최대 권한을 지정합니다. 자세한 내용은AWS Organizations 사용 설명서의 서비스 제어 정책을 참조하세요.
- 리소스 제어 정책(RCP) – 계정의 리소스에 사용할 수 있는 최대 권한을 설정합니다. 자세한 내용은 AWS Organizations 사용 설명서의 리소스 제어 정책(RCP)을 참조하세요.
- 세션 정책 – 역할 또는 페더레이션 사용자에 대해 임시 세션을 프로그래밍 방식으로 생성할 때 파라미터로 전달하는 고급 정책입니다. 자세한 내용은 IAM 사용 설명서의 세션 정책을 참조하세요.

여러 정책 유형

여러 정책 유형이 요청에 적용되는 경우, 결과 권한은 이해하기가 더 복잡합니다. 에서 여러 정책 유형이 관련될 때 요청을 허용할지 여부를 AWS 결정하는 방법을 알아보려면 IAM 사용 설명서의 정책 평가 로직을 참조하세요.

AWS CodeArtifact가 IAM에서 작동하는 방식

IAM을 사용하여 CodeArtifact에 대한 액세스를 관리하기 전에 CodeArtifact와 함께 사용할 수 있는 IAM 기능을 알아보세요.

AWS CodeArtifact와 함께 사용할 수 있는 IAM 기능

IAM 특성	CodeArtifac 지원
자격 증명 기반 정책	예
리소스 기반 정책	예
정책 작업	예
정책 리소스	예
정책 조건 키(서비스별)	아니요
ACL	아니요
ABAC(정책 내 태그)	부분적
임시 자격 증명	예
엔터티 권한	예
서비스 역할	아니요
서비스 연결 역할	아니요

CodeArtifact 및 기타 AWS 서비스가 대부분의 IAM 기능과 작동하는 방식을 전체적으로 알아보려면 IAM 사용 설명서의 AWS IAM으로 작업하는 서비스를 참조하세요.

CodeArtifact에 대한 자격 증명 기반 정책

ID 기반 정책 지원: 예

ID 기반 정책은 IAM 사용자, 사용자 그룹 또는 역할과 같은 ID에 연결할 수 있는 JSON 권한 정책 문서입니다. 이러한 정책은 사용자 및 역할이 어떤 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지

를 제어합니다. 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서에서 고객 관리형 정책으로 사용자 지정 IAM 권한 정의를 참조하세요.

IAM ID 기반 정책을 사용하면 허용되거나 거부되는 작업과 리소스뿐 아니라 작업이 허용되거나 거부되는 조건을 지정할 수 있습니다. JSON 정책에서 사용할 수 있는 모든 요소에 대해 알아보려면 IAM 사용 설명서의 IAM JSON 정책 요소 참조를 참조하세요.

CodeArtifact의 자격 증명 기반 정책 예제

CodeArtifact 자격 증명 기반 정책 예제를 보려면 AWS CodeArtifact에 대한 자격 증명 기반 정책 예제 섹션을 참조하세요.

CodeArtifact 내 리소스 기반 정책

리소스 기반 정책 지원: 예

리소스 기반 정책은 리소스에 연결하는 JSON 정책 설명서입니다. 리소스 기반 정책의 예제는 IAM 역할 신뢰 정책과 Amazon S3 버킷 정책입니다. 리소스 기반 정책을 지원하는 서비스에서 서비스 관리자는 이러한 정책을 사용하여 특정 리소스에 대한 액세스를 통제할 수 있습니다. 정책이 연결된 리소스의 경우 정책은 지정된 보안 주체가 해당 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 정의합니다. 리소스 기반 정책에서 보안 주체를 지정해야 합니다. 보안 주체에는 계정, 사용자, 역할, 페더레이션 사용자 또는이 포함될 수 있습니다 AWS 서비스.

교차 계정 액세스를 활성화하려는 경우, 전체 계정이나 다른 계정의 IAM 개체를 리소스 기반 정책의 보안 주체로 지정할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 IAM에서 교차 계정 리소스 액세스를 참조하세요.

CodeArtifact 정책 작업

정책 작업 지원: 예

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

JSON 정책의 `Action`요소는 정책에서 액세스를 허용하거나 거부하는 데 사용할 수 있는 작업을 설명합니다. 연결된 작업을 수행할 수 있는 권한을 부여하기 위한 정책에 작업을 포함하세요.

CodeArtifact 작업 목록을 보려면 서비스 승인 참조의 AWS CodeArtifact에서 정의한 작업을 참조하세요.

CodeArtifact 의 정책 작업은 작업 앞에 다음 접두사를 사용합니다.

```
codeartifact
```

단일 문에서 여러 작업을 지정하려면 쉼표로 구분합니다.

```
"Action": [
      "codeartifact:action1",
      "codeartifact:action2"
         ]
```

와일드카드(*)를 사용하여 여러 작업을 지정할 수 있습니다. 예를 들어, `Describe`라는 단어로 시작하는 모든 작업을 지정하려면 다음 작업을 포함합니다.

```
"Action": "codeartifact:Describe*"
```

CodeArtifact 자격 증명 기반 정책 예제를 보려면 AWS CodeArtifact에 대한 자격 증명 기반 정책 예제 섹션을 참조하세요.

CodeArtifact 정책 리소스

정책 리소스 지원: 예

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

`Resource` JSON 정책 요소는 작업이 적용되는 하나 이상의 객체를 지정합니다. 모범 사례에 따라 Amazon 리소스 이름(ARN)을 사용하여 리소스를 지정합니다. 리소스 수준 권한을 지원하지 않는 작업의 경우, 와일드카드(*)를 사용하여 해당 문이 모든 리소스에 적용됨을 나타냅니다.

```
"Resource": "*"
```

CodeArtifact 리소스 유형 및 해당 ARNs 목록을 보려면 서비스 승인 참조의 AWS CodeArtifact에서 정의한 리소스를 참조하세요. 각 리소스의 ARN을 지정할 수 있는 작업을 알아보려면 AWS CodeArtifact에서 정의한 작업을 참조하세요. 정책에 CodeArtifact 리소스 ARN을 지정하는 방법의 예를 보려면 AWS CodeArtifact 리소스 및 작업을 참조하세요.

CodeArtifact에서 사용되는 정책 조건 키

서비스별 정책 조건 키 지원: 아니요

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

`Condition` 요소는 정의된 기준에 따라 문이 실행되는 시기를 지정합니다. 같음(equals) 또는 미만(less than)과 같은 조건 연산자를 사용하여 정책의 조건을 요청의 값과 일치시키는 조건식을 생성할 수 있습니다. 모든 AWS 전역 조건 키를 보려면 IAM 사용 설명서의 AWS 전역 조건 컨텍스트 키를 참조하세요.

> **Note**
>
> AWS CodeArtifact는 다음과 같은 AWS 전역 조건 컨텍스트 키를 지원하지 않습니다.
>
> - Referer
> - UserAgent

CodeArtifact 조건 키 목록을 보려면 서비스 승인 참조의 조건 키 for AWS CodeArtifact를 참조하세요. 조건 키를 사용할 수 있는 작업과 리소스를 알아보려면AWS CodeArtifact에서 정의한 작업을 참조하세요.

CodeArtifac 자격 증명 기반 정책 예제를 보려면 AWS CodeArtifact에 대한 자격 증명 기반 정책 예제 섹션을 참조하세요.

CodeArtifact의 ACL

ACL 지원: 아니요

액세스 제어 목록(ACL)은 어떤 보안 주체(계정 멤버, 사용자 또는 역할)가 리소스에 액세스할 수 있는 권한을 가지고 있는지를 제어합니다. ACL은 JSON 정책 문서 형식을 사용하지 않지만 리소스 기반 정책과 유사합니다.

CodeArtifact에서의 ABAC

ABAC 지원(정책의 태그): 부분적

속성 기반 액세스 제어(ABAC)는 태그라고 불리는 속성을 기반으로 권한을 정의하는 권한 부여 전략입니다. IAM 엔터티 및 AWS 리소스에 태그를 연결한 다음 보안 주체의 태그가 리소스의 태그와 일치할 때 작업을 허용하는 ABAC 정책을 설계할 수 있습니다.

태그에 근거하여 액세스를 제어하려면 `aws:ResourceTag/`*`key-name`*, `aws:RequestTag/`*`key-name`*또는 `aws:TagKeys`조건 키를 사용하여 정책의 조건 요소에 태그 정보를 제공합니다.

서비스가 모든 리소스 유형에 대해 세 가지 조건 키를 모두 지원하는 경우, 값은 서비스에 대해 예입니다. 서비스가 일부 리소스 유형에 대해서만 세 가지 조건 키를 모두 지원하는 경우, 값은 부분적입니다.

ABAC에 대한 자세한 내용은 IAM 사용 설명서의 ABAC 권한 부여를 통한 권한 정의를 참조하세요. ABAC 설정 단계가 포함된 자습서를 보려면 IAM 사용 설명서의 속성 기반 액세스 제어(ABAC) 사용을 참조하세요.

CodeArtifact 리소스의 태그를 기반으로 리소스에 대한 액세스를 제한하는 자격 증명 기반 정책을 포함하여 리소스에 태그를 지정하는 자세한 방법은 태그를 사용하여 CodeArtifact 리소스에 대한 액세스 제어 섹션을 참조하세요.

CodeArtifact에서 임시 자격 증명 사용

임시 자격 증명 지원: 예

임시 자격 증명은 AWS 리소스에 대한 단기 액세스를 제공하며 페더레이션을 사용하거나 역할을 전환할 때 자동으로 생성됩니다. 장기 액세스 키를 사용하는 대신 임시 자격 증명을 동적으로 생성하는 것이 AWS 좋습니다. 자세한 내용은 IAM 사용 설명서의 IAM의 임시 보안 자격 증명 및 IAM으로 작업하는AWS 서비스 섹션을 참조하세요.

CodeArtifact의 서비스 간 보안 주체 권한

전달 액세스 세션(FAS) 지원: 예

전달 액세스 세션(FAS)은를 호출하는 보안 주체의 권한을 다운스트림 서비스에 AWS 서비스 대한 요청과 AWS 서비스함께 사용합니다. FAS 요청 시 정책 세부 정보는 전달 액세스 세션을 참조하세요.

호출 주체가 다른 서비스에 대한 권한을 가져야 하는 두 가지 CodeArtifact API 작업이 있습니다.

1. `GetAuthorizationToken`은 `codeartifact:GetAuthorizationToken`과 함께 `sts:GetServiceBearerToken`가 필요합니다.
2. `CreateDomain`은 기본이 아닌 암호화 키를 제공하는 경우 KMS 키에 `kms:DescribeKey`와 `kms:CreateGrant`, 두 키 모두 필요하며 `codeartifact:CreateDomain`도 함께 사용해야 합니다.

CodeArtifact의 작업에 필요한 권한 및 리소스에 대한 자세한 내용은 AWS CodeArtifact 권한 참조 섹션을 참조하세요.

CodeArtifact의 서비스 역할

서비스 역할 지원: 아니요

서비스 역할은 서비스가 사용자를 대신하여 작업을 수행하는 것으로 가정하는 IAM 역할입니다. IAM 관리자는 IAM 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 AWS 서비스 AWS에 권한을 위임할 역할 생성을 참조하세요.

> **Warning**
>
> 서비스 역할에 대한 권한을 변경하면 CodeArtifact 기능이 중단될 수 있습니다. CodeArtifact가 관련 지침을 제공하는 경우에만 서비스 역할을 편집합니다.

CodeArtifact의 서비스 연결 역할

서비스 연결 역할 지원: 아니요

서비스 연결 역할은에 연결된 서비스 역할의 한 유형입니다 AWS 서비스. 서비스는 사용자를 대신하여 작업을 수행하기 위해 역할을 수임할 수 있습니다. 서비스 연결 역할은에 나타나 AWS 계정 며 서비스가 소유합니다. IAM 관리자는 서비스 연결 역할의 권한을 볼 수 있지만 편집은 할 수 없습니다.

서비스 연결 역할 생성 또는 관리에 대한 자세한 내용은 IAM으로 작업하는AWS 서비스를 참조하세요. 서비스 연결 역할 열에서 `Yes`가 포함된 서비스를 테이블에서 찾습니다. 해당 서비스에 대한 서비스 연결 역할 설명서를 보려면 예(Yes) 링크를 선택합니다.

AWS CodeArtifact에 대한 자격 증명 기반 정책 예제

기본적으로 사용자 및 역할은 CodeArtifact 리소스를 생성하거나 수정할 수 있는 권한이 없습니다. 사용자에게 사용자가 필요한 리소스에서 작업을 수행할 권한을 부여하려면 IAM 관리자가 IAM 정책을 생성하면 됩니다.

이러한 예제 JSON 정책 문서를 사용하여 IAM ID 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서의 IAM 정책 생성(콘솔)을 참조하세요.

각 리소스 유형에 대한 ARNs 형식을 포함하여 CodeArtifact에서 정의한 작업 및 리소스 유형에 대한 자세한 내용은 서비스 권한 부여 참조의 AWS CodeArtifact에 사용되는 작업, 리소스 및 조건 키를 참조하세요.

주제

- 정책 모범 사례
- CodeArtifact 콘솔 사용

- AWS CodeArtifact에 대한 AWS 관리형(미리 정의된) 정책
- 사용자가 자신의 권한을 볼 수 있도록 허용
- 사용자가 리포지토리 및 도메인 대한 정보를 가져오도록 허용
- 사용자가 특정 도메인에 대한 정보를 가져올 수 있도록 허용
- 사용자가 특정 리포지토리에 대한 정보를 가져오도록 허용
- 권한 부여 토큰 기간 제한

정책 모범 사례

자격 증명 기반 정책에 따라 계정에서 사용자가 CodeArtifac 리소스를 생성, 액세스 또는 삭제할 수 있는지 여부가 결정됩니다. 이 작업으로 인해 AWS 계정에 비용이 발생할 수 있습니다. ID 기반 정책을 생성하거나 편집할 때는 다음 지침과 권장 사항을 따르세요.

- AWS 관리형 정책을 시작하고 최소 권한으로 전환 - 사용자 및 워크로드에 권한 부여를 시작하려면 많은 일반적인 사용 사례에 대한 권한을 부여하는 AWS 관리형 정책을 사용합니다. 에서 사용할 수 있습니다 AWS 계정. 사용 사례에 맞는 AWS 고객 관리형 정책을 정의하여 권한을 추가로 줄이는 것이 좋습니다. 자세한 내용은 IAM 사용 설명서의 AWS 관리형 정책 또는 AWS 직무에 대한 관리형 정책을 참조하세요.
- 최소 권한 적용 – IAM 정책을 사용하여 권한을 설정하는 경우, 작업을 수행하는 데 필요한 권한만 부여합니다. 이렇게 하려면 최소 권한으로 알려진 특정 조건에서 특정 리소스에 대해 수행할 수 있는 작업을 정의합니다. IAM을 사용하여 권한을 적용하는 방법에 대한 자세한 정보는 IAM 사용 설명서에 있는 IAM의 정책 및 권한을 참조하세요.
- IAM 정책의 조건을 사용하여 액세스 추가 제한 – 정책에 조건을 추가하여 작업 및 리소스에 대한 액세스를 제한할 수 있습니다. 예를 들어, SSL을 사용하여 모든 요청을 전송해야 한다고 지정하는 정책 조건을 작성할 수 있습니다. AWS 서비스와 같은 특정를 통해 사용되는 경우 조건을 사용하여 서비스 작업에 대한 액세스 권한을 부여할 수도 있습니다 CloudFormation. 자세한 내용은 IAM 사용 설명서의 IAM JSON 정책 요소: 조건을 참조하세요.
- IAM Access Analyzer를 통해 IAM 정책을 확인하여 안전하고 기능적인 권한 보장 - IAM Access Analyzer에서는 IAM 정책 언어(JSON)와 모범 사례가 정책에서 준수되도록 새로운 및 기존 정책을 확인합니다. IAM Access Analyzer는 100개 이상의 정책 확인 항목과 실행 가능한 추천을 제공하여 안전하고 기능적인 정책을 작성하도록 돕습니다. 자세한 내용은 IAM 사용 설명서의 IAM Access Analyzer에서 정책 검증을 참조하세요.
- 다중 인증(MFA) 필요 -에서 IAM 사용자 또는 루트 사용자가 필요한 시나리오가 있는 경우 추가 보안을 위해 MFA를 AWS 계정켭니다. API 작업을 직접적으로 호출할 때 MFA가 필요하면 정책에 MFA 조건을 추가합니다. 자세한 내용은 IAM 사용 설명서의 MFA를 통한 보안 API 액세스를 참조하세요.

IAM의 모범 사례에 대한 자세한 내용은 IAM 사용 설명서의 IAM의 보안 모범 사례를 참조하세요.

CodeArtifact 콘솔 사용

AWS CodeArtifact 콘솔에 액세스하려면 최소 권한 집합이 있어야 합니다. 이러한 권한은에서 CodeArtifact 리소스에 대한 세부 정보를 나열하고 볼 수 있도록 허용해야 합니다 AWS 계정. 최소 필수 권한보다 더 제한적인 ID 기반 정책을 생성하는 경우, 콘솔이 해당 정책에 연결된 엔티티(사용자 또는 역할)에 대해 의도대로 작동하지 않습니다.

AWS CLI 또는 AWS API만 호출하는 사용자에게는 최소 콘솔 권한을 허용할 필요가 없습니다. 대신, 수행하려는 API 작업과 일치하는 작업에만 액세스할 수 있도록 합니다.

사용자와 역할이 CodeArtifact 콘솔을 계속 사용할 수 있도록 하려면 `AWSCodeArtifactAdminAccess` 또는 `AWSCodeArtifactReadOnlyAccess` AWS 관리형 정책도 엔터티에 연결합니다. 자세한 내용은 IAM 사용 설명서의 사용자에게 권한 추가를 참조하세요.

AWS CodeArtifact에 대한 AWS 관리형(미리 정의된) 정책

AWS 는에서 생성하고 관리하는 독립 실행형 IAM 정책을 제공하여 많은 일반적인 사용 사례를 처리합니다 AWS. 이러한 AWS 관리형 정책은 일반적인 사용 사례에 필요한 권한을 부여하므로 필요한 권한을 조사할 필요가 없습니다. 자세한 내용은 IAM 사용자 설명서의 AWS 관리형 정책을 참조하십시오.

계정의 사용자에게 연결할 수 있는 다음 AWS 관리형 정책은 AWS CodeArtifact에 고유합니다.

- `AWSCodeArtifactAdminAccess` - CodeArtifact 도메인을 관리할 수 있는 권한을 포함하여 CodeArtifact에 대한 전체 액세스 권한을 제공합니다.

  JSON

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

```
            "Resource": "*",
            "Condition": {
               "StringEquals": {
                  "sts:AWSServiceName": "codeartifact.amazonaws.com"
               }
            }
         }
      ]
}
```

- `AWSCodeArtifactReadOnlyAccess` - CodeArtifact에 대한 읽기 전용 액세스 권한을 제공합니다.

  JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Action": [
            "codeartifact:Describe*",
            "codeartifact:Get*",
            "codeartifact:List*",
            "codeartifact:ReadFromRepository"
         ],
         "Effect": "Allow",
         "Resource": "*"
      },
      {
         "Effect": "Allow",
         "Action": "sts:GetServiceBearerToken",
         "Resource": "*",
         "Condition": {
            "StringEquals": {
               "sts:AWSServiceName": "codeartifact.amazonaws.com"
            }
         }
      }
   ]
}
```

CodeArtifact 서비스 역할을 생성하고 관리하려면 라는 AWS 관리형 정책도 연결해야 합니다`IAMFullAccess`.

CodeArtifact 작업 및 리소스에 대한 권한을 허용하는 고유의 사용자 지정 IAM 정책을 생성할 수도 있습니다. 해당 권한이 필요한 IAM 사용자 또는 그룹에 이러한 사용자 지정 정책을 연결할 수 있습니다.

사용자가 자신의 권한을 볼 수 있도록 허용

이 예제는 IAM 사용자가 자신의 사용자 ID에 연결된 인라인 및 관리형 정책을 볼 수 있도록 허용하는 정책을 생성하는 방법을 보여 줍니다. 이 정책에는 콘솔에서 또는 AWS CLI 또는 AWS API를 사용하여 프로그래밍 방식으로이 작업을 완료할 수 있는 권한이 포함됩니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

사용자가 리포지토리 및 도메인 대한 정보를 가져오도록 허용

다음 정책은 IAM 사용자 또는 역할이 도메인, 리포지토리, 패키지, 자산을 비롯한 모든 유형의 CodeArtifact 리소스를 나열하고 설명할 수 있도록 허용합니다. 정책에는 보안 주체가 CodeArtifact 리포지토리에서 패키지를 가져올 수 있는 `codeArtifact:ReadFromRepository` 권한도 포함되어 있습니다. 새 도메인이나 리포지토리를 생성할 수 없으며 새 패키지를 게시하는 것도 허용되지 않습니다.

`GetAuthorizationToken` API를 호출하려면 `codeartifact:GetAuthorizationToken` 및 `sts:GetServiceBearerToken` 권한이 필요합니다.

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "codeartifact:List*",
            "codeartifact:Describe*",
            "codeartifact:Get*",
            "codeartifact:Read*"
         ],
         "Resource": "*"
      },
      {
         "Effect": "Allow",
         "Action": "sts:GetServiceBearerToken",
         "Resource": "*",
         "Condition": {
            "StringEquals": {
               "sts:AWSServiceName": "codeartifact.amazonaws.com"
            }
         }
      }
   ]
}
```

사용자가 특정 도메인에 대한 정보를 가져올 수 있도록 허용

다음은 my라는 이름으로 시작하는 모든 도메인에 대해 사용자가 123456789012 계정의 us-east-2 리전에 있는 도메인만 나열할 수 있도록 허용하는 권한 정책의 예입니다.

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": "codeartifact:ListDomains",
         "Resource": "arn:aws:codeartifact:us-east-2:111122223333:domain/my*"

      }
   ]
}
```

사용자가 특정 리포지토리에 대한 정보를 가져오도록 허용

다음은 사용자가 리포지토리에 있는 패키지에 대한 정보를 포함하여 test로 끝나는 리포지토리에 대한 정보를 가져올 수 있도록 허용하는 권한 정책의 예입니다. 사용자는 리소스를 게시, 생성 또는 삭제할 수 없습니다.

따라서 GetAuthorizationToken API를 호출하려면 codeartifact:GetAuthorizationToken 및 sts:GetServiceBearerToken 권한이 필요합니다.

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "codeartifact:List*",
            "codeartifact:Describe*",
            "codeartifact:Get*",
            "codeartifact:Read*"
```

```
            ],
            "Resource": "arn:aws:codeartifact:*:*:repository/*/*test"
        },
        {
            "Effect": "Allow",
            "Action": [
                "codeartifact:List*",
                "codeartifact:Describe*"
            ],
            "Resource": "arn:aws:codeartifact:*:*:package/*/*test/*/*/*"
        },
        {
            "Effect": "Allow",
            "Action": "sts:GetServiceBearerToken",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "sts:AWSServiceName": "codeartifact.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "codeartifact:GetAuthorizationToken",
            "Resource": "*"
        }
    ]
}
```

권한 부여 토큰 기간 제한

패키지 버전을 게시하거나 사용하려면 사용자는 인증 토큰을 사용하여 CodeArtifact에 인증해야 합니다. 인증 토큰은 구성된 수명 기간 동안만 유효합니다. 토큰의 기본 수명은 12시간입니다. 인증 토큰에 대한 자세한 내용은 AWS CodeArtifact 인증 및 토큰 섹션을 참조하세요.

토큰을 가져올 때 사용자는 토큰의 수명을 구성할 수 있습니다. 인증 토큰의 유효 수명 값은 `0` 및 `900`(15분)에서 `43200`(12시간) 사이의 숫자입니다. 값을 `0`으로 설정하면 사용자 역할의 임시 보안 인증과 동일한 기간을 가진 토큰이 생성됩니다.

관리자는 사용자 또는 그룹에 연결된 권한 정책의 `sts:DurationSeconds` 조건 키를 사용하여 인증 토큰의 수명에 대한 유효한 값을 제한할 수 있습니다. 사용자가 유효 값을 벗어난 수명을 가진 인증 토큰을 생성하려고 하면 토큰 생성에 실패합니다.

다음 예제 정책은 CodeArtifact 사용자가 생성하는 인증 토큰의 가능한 기간을 제한합니다.

예제 정책: 토큰 수명을 정확히 12시간(43,200초)으로 제한

이 정책을 사용하면 사용자는 수명이 12시간인 인증 토큰만 생성할 수 있습니다.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "codeartifact:*",
            "Resource": "*"
        },
        {
            "Sid": "sts",
            "Effect": "Allow",
            "Action": "sts:GetServiceBearerToken",
            "Resource": "*",
            "Condition": {
                "NumericEquals": {
                    "sts:DurationSeconds": 43200
                },
                "StringEquals": {
                    "sts:AWSServiceName": "codeartifact.amazonaws.com"
                }
            }
        }
    ]
}
```

정책 예: 토큰 수명을 15분에서 1시간 사이로 또는 사용자의 임시 보안 인증 기간과 같게 제한

이 정책을 통해 사용자는 15분에서 1시간 기간 동안만 유효한 토큰을 생성할 수 있습니다. 또한 사용자는 `0` 및 `--durationSeconds`를 지정하여 역할의 임시 자격 증명 기간 동안 지속되는 토큰을 생성할 수 있습니다.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "codeartifact:*",
            "Resource": "*"
        },
        {
            "Sid": "sts",
            "Effect": "Allow",
            "Action": "sts:GetServiceBearerToken",
            "Resource": "*",
            "Condition": {
                "NumericLessThanEquals": {
                    "sts:DurationSeconds": 3600
                },
                "StringEquals": {
                    "sts:AWSServiceName": "codeartifact.amazonaws.com"
                }
            }
        }
    ]
}
```

태그를 사용하여 CodeArtifact 리소스에 대한 액세스 제어

IAM 사용자 정책 설명의 조건은 CodeArtifact 작업에 필요한 리소스에 대한 권한을 지정하는 데 사용하는 구문의 일부입니다. 조건에 태그를 사용하는 것은 리소스 및 요청에 대한 액세스를 제어하는 하나의 방법입니다. CodeArtifact 리소스 태깅에 대한 자세한 내용은 리소스에 태그 지정 섹션을 참조하세요. 이 주제에서는 태그 기반 액세스 제어를 다룹니다.

IAM 정책을 설계할 때 특정 리소스에 대한 액세스 권한을 부여하여 세부적인 권한을 설정할 수 있습니다. 관리하는 리소스의 개수가 늘어날수록 이 작업은 더 어려워집니다. 리소스에 태그를 지정하고 정책 문 조건에서 태그를 사용하면 이러한 작업이 더 간단해질 수 있습니다. 특정 태그를 사용하여 리소스에

대량으로 액세스 권한을 부여합니다. 그런 다음, 생성 중 또는 나중에 이 태그를 관련 리소스에 반복해서 적용합니다.

리소스에 태그가 연결되거나 태그 지정을 지원하는 서비스에 대한 요청에서 전달될 수 있습니다. CodeArtifact에서 리소스에 태그가 있을 수 있고 일부 작업에 태그가 포함될 수 있습니다. IAM 정책을 생성하면 태그 조건 키를 사용하여 다음을 제어할 수 있습니다.

- 도메인 또는 리포지토리 리소스에 이미 있는 태그를 기반으로 해당 리소스에 대해 작업을 수행할 수 있는 사용자
- 작업의 요청에서 전달될 수 있는 태그
- 요청에서 특정 키를 사용할 수 있는지 여부를 통제합니다.

태그 조건 키의 전체 구문 및 의미는 IAM 사용 설명서의 태그를 사용한 액세스 제어를 참조하십시오.

> **Important**
>
> 리소스에 태그를 사용하여 작업을 제한하는 경우 태그는 작업이 실행되는 리소스에 있어야 합니다. 예를 들어 태그가 포함된 `DescribeRepository` 권한을 거부하려면 태그가 도메인이 아닌 각 리포지토리에 있어야 합니다. CodeArtifact의 작업 목록과 해당 작업이 작동하는 리소스는 AWS CodeArtifact 권한 참조을 참조하세요.

태그 기반 액세스 제어 정책 예제

다음 예에서는 CodeArtifact 사용자에 정책의 태그 조건을 지정하는 방법을 설명합니다.

Example 1: 요청의 태그 기반 작업 제한

`AWSCodeArtifactAdminAccess` 관리형 사용자 정책은 사용자에게 모든 리소스에서 모든 CodeArtifact 작업을 수행할 수 있는 무제한적인 권한을 제공합니다.

다음 정책은 이러한 기능을 제한하고 요청이 특정 태그를 포함하지 않는 한 권한이 없는 사용자의 리포지토리 생성 권한을 거부합니다. 이와 관련하여 정책은 요청이 1 또는 2 값 중 하나와 함께 `costcenter`라는 태그를 지정하지 않는 경우 `CreateRepository` 작업을 거부합니다. 고객의 관리자는 권한이 없는 IAM 사용자에게 관리형 사용자 정책 이외에 이 IAM 정책도 연결해야 합니다.

JSON

```
{
```

```
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": "codeartifact:CreateRepository",
      "Resource": "*",
      "Condition": {
        "Null": {
          "aws:RequestTag/costcenter": "true"
        }
      }
    },
    {
      "Effect": "Deny",
      "Action": "codeartifact:CreateRepository",
      "Resource": "*",
      "Condition": {
          "ForAnyValue:StringNotEquals": {
              "aws:RequestTag/costcenter": [
                  "1",
                  "2"
              ]
          }
      }
  }
  ]
}
```

Example 2: 리소스 태그 기반 작업 제한

`AWSCodeArtifactAdminAccess` 관리형 사용자 정책은 사용자에게 모든 리소스에서 모든 CodeArtifact 작업을 수행할 수 있는 무제한적인 권한을 제공합니다.

다음 정책은 이러한 기능을 제한하고 권한이 없는 사용자의 지정된 도메인의 리포지토리에 대한 작업 수행 권한을 거부합니다. 이와 관련하여 정책은 리소스에 `Value1` 또는 `Value2` 값 중 하나가 포함된 `Key1` 태그가 있으면 일부 작업을 거부합니다. `aws:ResourceTag` 조건 키는 해당 리소스의 태그를 기반으로 리소스에 대한 액세스를 제어하는 데 사용됩니다. 고객의 관리자는 권한이 없는 IAM 사용자에게 관리형 사용자 정책 이외에 이 IAM 정책도 연결해야 합니다.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "codeartifact:TagResource",
        "codeartifact:UntagResource",
        "codeartifact:DescribeDomain",
        "codeartifact:DescribeRepository",
        "codeartifact:PutDomainPermissionsPolicy",
        "codeartifact:PutRepositoryPermissionsPolicy",
        "codeartifact:ListRepositoriesInDomain",
        "codeartifact:UpdateRepository",
        "codeartifact:ReadFromRepository",
        "codeartifact:ListPackages",
        "codeartifact:ListTagsForResource"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Key1": ["Value1", "Value2"]
        }
      }
    }
  ]
}
```

Example 3: 리소스 태그 기반 작업 허용

다음 정책은 사용자에게 CodeArtifact에서 리포지토리 및 패키지에 대해 작업을 수행하고 관련 정보를 가져올 수 있는 권한을 부여합니다.

이와 관련하여 정책은 리포지토리에 `Value1` 값이 포함된 `Key1` 태그가 있으면 특정 작업을 허용합니다. `aws:RequestTag` 조건 키는 IAM 요청에서 전달할 수 있는 태그를 제어하는 데 사용됩니다. `aws:TagKeys` 조건은 태그 키의 대/소문자를 구분합니다. 이 정책은 `AWSCodeArtifactAdminAccess` 관리형 사용자 정책이 연결되어 있지 않은 IAM 사용자에게 유용합니다. 관리형 정책은 사용자에게 모든 리소스에서 모든 CodeArtifact 작업을 수행할 수 있는 무제한적인 권한을 제공합니다.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codeartifact:UpdateRepository",
        "codeartifact:DeleteRepository",
        "codeartifact:ListPackages"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Key1": "Value1"
        }
      }
    }
  ]
}
```

Example 4: 요청의 태그 기반 작업 허용

다음 정책은 사용자에게 CodeArtifact의 지정된 도메인에 리포지토리를 생성할 수 있는 권한을 부여합니다.

이와 관련하여 정책은 요청의 리소스 생성 API가 `Value1` 값이 포함된 `Key1` 태그를 지정하는 경우 `CreateRepository` 및 `TagResource` 작업을 허용합니다. `aws:RequestTag` 조건 키는 IAM 요청에서 전달할 수 있는 태그를 제어하는 데 사용됩니다. `aws:TagKeys` 조건은 태그 키의 대/소문자를 구분합니다. 이 정책은 `AWSCodeArtifactAdminAccess` 관리형 사용자 정책이 연결되어 있지 않은 IAM 사용자에게 유용합니다. 관리형 정책은 사용자에게 모든 리소스에서 모든 CodeArtifact 작업을 수행할 수 있는 무제한적인 권한을 제공합니다.

JSON

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

```
      "Effect": "Allow",
      "Action": [
        "codeartifact:CreateRepository",
        "codeartifact:TagResource"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/Key1": "Value1"
        }
      }
    }
  ]
}
```

AWS CodeArtifact 권한 참조

AWS CodeArtifact 리소스 및 작업

In AWS CodeArtifact에서 기본 리소스는 도메인입니다. 정책에서 Amazon 리소스 이름(ARN)을 사용하여 정책이 적용되는 리소스를 식별합니다. 리포지토리도 리소스이며 리포지토리에는 이와 연결된 ARN이 들어 있습니다. 자세한 내용은 Amazon Web Services 일반 참조의 Amazon 리소스 이름(ARN)을 참조하세요.

리소스 유형	ARN 형식
도메인	arn:aws:codeartifact: *region-ID* :*account-ID* :domain/*my_domain*
리포지토리	arn:aws:codeartifact: *region-ID* :*account-ID* :repository/ *my_domain* /*my_repo*
패키지 그룹	arn:aws:codeartifact: *region-ID* :*account-ID* :package-group/ *my_domain* /*encoded_package_group_pattern*
네임스페이스 포함 패키지	arn:aws:codeartifact: *region-ID* :*account-ID* :package/ *my_domain* /*my_repo*/*package-format* /*namespace* /*my_package*

리소스 유형	ARN 형식
네임스페이스 미포함 패키지	arn:aws:codeartifact: *region-ID* :*account-ID* :package/ *my_domain* /*my_repo*/*package-format* //*my_package*
모든 CodeArtifact 리소스	arn:aws:codeartifact:*
지정한 AWS 리전에서 지정한 계정이 소유한 모든 CodeArtifact 리소스	arn:aws:codeartifact: *region-ID* :*account-ID* :*

액세스를 제어할 작업에 따라 지정할 리소스 ARN이 달라집니다.

명령문에서 다음과 같이 ARN을 사용하여 특정 도메인(*myDomain*)을 나타낼 수 있습니다.

```
"Resource": "arn:aws:codeartifact:us-east-2:123456789012:domain/myDomain"
```

명령문에서 다음과 같이 ARN을 사용하여 특정 리포지토리(*myRepo*)를 나타낼 수 있습니다.

```
"Resource": "arn:aws:codeartifact:us-east-2:123456789012:domain/myDomain/myRepo"
```

단일 명령문에서 여러 리소스를 지정하려면 ARN을 쉼표로 구분합니다. 다음 명령문은 특정 도메인의 모든 패키지 및 리포지토리에 적용됩니다.

```
"Resource": [
  "arn:aws:codeartifact:us-east-2:123456789012:domain/myDomain",
  "arn:aws:codeartifact:us-east-2:123456789012:repository/myDomain/*",
  "arn:aws:codeartifact:us-east-2:123456789012:package/myDomain/*"
]
```

> **Note**
>
> 많은 AWS 서비스는 콜론(:) 또는 슬래시(/)를 ARNs. 그러나 CodeArtifact는 리소스 패턴 및 규칙에서 정확한 일치를 사용합니다. 따라서 이벤트 패턴을 만들 때 리소스에서 ARN 구문이 일치하도록 정확한 문자를 사용해야 합니다.

AWS CodeArtifact API 작업 및 권한

IAM 자격 증명에 연결할 수 있는 액세스 제어 및 쓰기 권한 정책(자격 증명 기반 정책)을 설정할 때 다음 표를 참조로 사용할 수 있습니다.

your AWS CodeArtifact 정책에서 AWS전체 조건 키를 사용하여 조건을 표시할 수 있습니다. 목록은 IAM 사용 설명서의 IAM JSON 정책 요소 참조를 참조하십시오.

정책의 `Action` 필드에 작업을 지정합니다. 작업을 지정하려면 `codeartifact:` 접두사 다음에 API 작업 이름을 사용합니다(예: `codeartifact:CreateDomain` 및 `codeartifact:AssociateExternalConnection`). 문장 하나에 여러 작업을 지정하려면 쉼표로 구분합니다(예: `"Action": [ "codeartifact:CreateDomain", "codeartifact:AssociateExternalConnection" ]`).

와일드카드 문자 사용

정책의 `Resource` 필드에 리소스 값으로 와일드카드 문자(*)를 사용하거나 사용하지 않고 ARN을 지정합니다. 와일드카드를 사용하여 여러 작업 또는 리소스를 지정할 수 있습니다. 예를 들어, `codeartifact:*`는 모든 CodeArtifact 작업을 지정하고, `codeartifact:Describe*`는 `Describe`이라는 단어로 시작하는 모든 CodeArtifact 작업을 지정합니다.

패키지 그룹 ARN

> **Note**
>
> 이 섹션에서는 패키지 그룹 ARN 및 패턴 인코딩이 정보를 제공하는 방법에 대해 설명합니다. 콘솔에서 ARN을 복사하거나, 패턴을 인코딩하고 ARN을 구성하는 대신 `DescribePackageGroup` API를 사용하여 ARN을 가져오는 것이 좋습니다.

IAM 정책은 와일드카드 문자 *를 사용하여 여러 IAM 작업 또는 여러 리소스를 일치시킵니다. 패키지 그룹 패턴에서도 * 문자를 사용합니다. 단일 패키지 그룹과 일치하는 IAM 정책을 보다 쉽게 작성하기 위해 패키지 그룹 ARN 형식은 인코딩된 버전의 패키지 그룹 패턴을 사용합니다.

특히 패키지 그룹 ARN 형식은 다음과 같습니다.

```
arn:aws:codeartifact:region:account-ID:package-
group/my_domain/encoded_package_group_pattern
```

여기서 인코딩된 패키지 그룹 패턴은 패키지 그룹 패턴이며, 특정 특수 문자가 백분율로 인코딩된 값으로 대체됩니다. 다음 목록에는 문자와 해당 백분율로 인코딩된 값이 포함되어 있습니다.

- * : %2a
- $: %24
- % : %25

예를 들어 도메인의 루트 패키지 그룹(/*)에 대한 ARN은 다음과 같습니다.

```
arn:aws:codeartifact:us-east-1:111122223333:package-group/my_domain/%2a
```

목록에 포함되지 않은 문자는 인코딩할 수 없으며 ARN은 대/소문자를 구분하므로 *는 %2A가 아닌 %2a로 인코딩해야 합니다.

문제 해결 AWS CodeArtifact 자격 증명 및 액세스

다음 정보를 사용하여 CodeArtifact 및 IAM 작업 시 발생할 수 있는 공통적인 문제를 진단하고 수정할 수 있습니다.

주제

- CodeArtifact에서 작업을 수행할 권한이 없습니다.
- 내 외부의 사람이 내 CodeArtifact 리소스 AWS 계정 에 액세스하도록 허용하고 싶습니다.

CodeArtifact에서 작업을 수행할 권한이 없습니다.

작업을 수행할 권한이 없다는 오류가 표시되면 작업을 수행할 수 있도록 정책을 업데이트해야 합니다.

다음의 예제 오류는 mateojackson IAM 사용자가 콘솔을 사용하여 가상 *my-example-widget* 리소스에 대한 세부 정보를 보려고 하지만 가상 codeartifact:*GetWidget* 권한이 없을 때 발생합니다.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 codeartifact:GetWidget on resource: my-example-widget
```

이 경우, codeartifact:*GetWidget* 작업을 사용하여 *my-example-widget* 리소스에 액세스할 수 있도록 mateojackson 사용자 정책을 업데이트해야 합니다.

도움이 필요한 경우 AWS 관리자에게 문의하세요. 관리자는 로그인 자격 증명을 제공한 사람입니다.

내 외부의 사람이 내 CodeArtifact 리소스 AWS 계정 에 액세스하도록 허용하고 싶습니다.

다른 계정의 사용자 또는 조직 외부의 사람이 리소스에 액세스할 때 사용할 수 있는 역할을 생성할 수 있습니다. 역할을 수임할 신뢰할 수 있는 사람을 지정할 수 있습니다. 리소스 기반 정책 또는 액세스 제어 목록(ACL)을 지원하는 서비스의 경우, 이러한 정책을 사용하여 다른 사람에게 리소스에 대한 액세스 권한을 부여할 수 있습니다.

자세한 내용은 다음을 참조하세요.

- CodeArtifact에서 이러한 기능을 지원하는지 여부를 알아보려면 AWS CodeArtifact가 IAM에서 작동하는 방식 섹션을 참조하세요.
- 소유 AWS 계정 한의 리소스에 대한 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 소유한 다른의 IAM 사용자에게 액세스 권한 제공을 참조 AWS 계정 하세요.
- 리소스에 대한 액세스 권한을 타사에 제공하는 방법을 알아보려면 IAM 사용 설명서의 타사 AWS 계정 소유에 대한 액세스 권한 제공을 AWS 계정참조하세요.
- ID 페더레이션을 통해 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 외부에서 인증된 사용자에게 액세스 권한 제공(ID 페더레이션)을 참조하세요.
- 크로스 계정 액세스에 대한 역할과 리소스 기반 정책 사용의 차이점을 알아보려면 IAM 사용 설명서의 IAM의 크로스 계정 리소스 액세스를 참조하세요.

Amazon VPC 엔드포인트 작업

인터페이스 Virtual Private Cloud(VPC) 엔드포인트를 사용하도록 CodeArtifact를 구성하여 VPC 보안을 개선할 수 있습니다.

VPC 엔드포인트는 프라이빗 IP 주소를 통해 CodeArtifact APIs에 액세스할 수 있는 서비스 AWS PrivateLink인를 사용합니다.는 VPC와 CodeArtifact 간의 모든 네트워크 트래픽을 AWS 네트워크로 AWS PrivateLink 제한합니다. 인터페이스 VPC 엔드포인트를 사용하면 인터넷 게이트웨이, NAT 디바이스 또는 가상 프라이빗 게이트웨이가 필요하지 않습니다. 자세한 내용은 Amazon Virtual Private Cloud 사용 설명서의 VPC 엔드포인트를 참조하세요.

> **Important**
>
> - VPC 엔드포인트는AWS 리전 간 요청을 지원하지 않습니다. CodeArtifact에 대한 API 호출을 실행하려는 리전과 동일한 AWS 리전에서 엔드포인트를 생성해야 합니다.
> - VPC 엔드포인트는 Amazon Route 53을 통해 Amazon이 제공하는 DNS만 지원합니다. 자체 DNS를 사용하는 경우에는 조건부 DNS 전달을 사용할 수 있습니다. 자세한 내용은 Amazon Virtual Private Cloud 사용 설명서의 DHCP 옵션 세트를 참조하세요.
> - VPC 엔드포인트에 연결된 보안 그룹은 VPC의 프라이빗 서브넷에서 443 포트로 들어오는 연결을 허용해야 합니다.

주제

CodeArtifact에 대한 VPC 엔드포인트 생성

CodeArtifact에 대한 Virtual Private Cloud(VPC) 엔드포인트를 생성하려면 Amazon EC2 `create-vpc-endpoint` AWS CLI 명령을 사용합니다. 자세한 내용은 Amazon Virtual Private Cloud 사용 설명서의 인터페이스 VPC 엔드포인트(AWS PrivateLink)를 참조하세요.

CodeArtifact에 대한 모든 요청이 AWS 네트워크에 있도록 두 개의 VPC 엔드포인트가 필요합니다. 첫 번째 엔드포인트는 CodeArtifact API(예: `GetAuthorizationToken` 및 `CreateRepository`)를 호출하는 데 사용됩니다.

```
com.amazonaws.region.codeartifact.api
```

두 번째 엔드포인트는 패키지 관리자 및 빌드 도구(예: npm 및 Gradle)를 사용하여 CodeArtifact 리포지토리에 액세스하는 데 사용됩니다.

```
com.amazonaws.region.codeartifact.repositories
```

다음 명령은 CodeArtifact 리포지토리에 액세스할 수 있는 엔드포인트를 생성합니다.

```
aws ec2 create-vpc-endpoint --vpc-id vpcid --vpc-endpoint-type Interface \
  --service-name com.amazonaws.region.codeartifact.api --subnet-ids subnetid \
  --security-group-ids groupid --private-dns-enabled
```

다음 명령은 패키지 관리자 및 빌드 도구에 액세스할 수 있는 엔드포인트를 생성합니다.

```
aws ec2 create-vpc-endpoint --vpc-id vpcid --vpc-endpoint-type Interface \
  --service-name com.amazonaws.region.codeartifact.repositories --subnet-ids subnetid \
  --security-group-ids groupid --private-dns-enabled
```

> **Note**
>
> `codeartifact.repositories` 엔드포인트를 생성할 때는 `--private-dns-enabled` 옵션을 사용하여 프라이빗 DNS 호스트 이름을 생성해야 합니다. `codeartifact.repositories` 엔드포인트를 생성할 때 프라이빗 DNS 호스트 이름을 생성할 수 없거나 생성하지 않으려는 경우, VPC에서 CodeArtifact와 함께 패키지 관리자를 사용하려면 추가 구성 단계를 따라야 합니다. 자세한 정보는 프라이빗 DNS가 없는 `codeartifact.repositories` 엔드포인트 사용을 참조하세요.

VPC 엔드포인트를 생성한 후 CodeArtifact와 함께 엔드포인트를 사용하려면 보안 그룹 규칙으로 추가 구성을 수행해야 할 수 있습니다. Amazon VPC 보안 그룹에 대한 자세한 내용은 보안 그룹을 참조하세요.

CodeArtifact에 연결하는 데 문제가 있는 경우 VPC Reachability Analyzer 도구를 사용하여 문제를 디버깅할 수 있습니다. 자세한 내용은 Reachability Analyzer가 무엇인가요?를 참조하세요.

Amazon S3 게이트웨이 엔드포인트 생성

CodeArtifact는 Amazon Simple Storage Service(S3)를 사용하여 패키지 자산을 저장합니다. CodeArtifact에서 패키지를 가져오려면 Amazon S3에 대한 게이트웨이 엔드포인트를 생성해야 합니다. 빌드 또는 배포 프로세스가 CodeArtifact에서 패키지를 다운로드할 때 CodeArtifact에 액세스하여 패키지 메타데이터를 가져오고 Amazon S3에 액세스하여 패키지 자산(예: Maven `.jar` 파일)을 다운로드해야 합니다.

> **Note**
>
> Python 또는 Swift 패키지 형식을 사용할 때는 Amazon S3 엔드포인트가 필요하지 않습니다.

CodeArtifact용 Amazon S3 게이트웨이 엔드포인트를 생성하려면 Amazon EC2 `create-vpc-endpoint` AWS CLI 명령을 사용합니다. 엔드포인트를 생성할 때는 VPC의 라우팅 테이블을 선택해야 합니다. 자세한 내용은 Amazon Virtual Private Cloud 사용 설명서의 게이트웨이 VPC 엔드포인트를 참조하세요.

다음 명령은 Amazon S3 엔드포인트를 생성합니다.

```
aws ec2 create-vpc-endpoint --vpc-id vpcid --service-name com.amazonaws.region.s3 \
  --route-table-ids routetableid
```

AWS CodeArtifact에 대한 최소 Amazon S3 버킷 권한

Amazon S3 게이트웨이 엔드포인트는 IAM 정책 문서를 사용하여 서비스에 대한 액세스를 제한합니다. CodeArtifact에 대한 최소 Amazon S3 버킷 권한만 허용하려면 엔드포인트에 대한 IAM 정책 설명을 생성할 때 CodeArtifact가 사용하는 Amazon S3 버킷에 대한 액세스를 제한하세요.

다음 표에서는 각 리전에서 CodeArtifact에 대한 액세스를 허용하려면 정책에서 참조해야 하는 Amazon S3 버킷을 설명합니다.

리전	Amazon S3 버킷 ARN
us-east-1	arn:aws:s3:::assets-193858265520-us-east-1
us-east-2	arn:aws:s3:::assets-250872398865-us-east-2
us-west-2	arn:aws:s3:::assets-787052242323-us-west-2

리전	Amazon S3 버킷 ARN
eu-west-1	arn:aws:s3:::assets-438097961670-eu-west-1
eu-west-2	arn:aws:s3:::assets-247805302724-eu-west-2
eu-west-3	arn:aws:s3:::assets-762466490029-eu-west-3
eu-north-1	arn:aws:s3:::assets-611884512288-eu-north-1
eu-south-1	arn:aws:s3:::assets-484130244270-eu-south-1
eu-central-1	arn:aws:s3:::assets-769407342218-eu-central-1
ap-northeast-1	arn:aws:s3:::assets-660291247815-ap-northeast-1
ap-southeast-1	arn:aws:s3:::assets-421485864821-ap-southeast-1
ap-southeast-2	arn:aws:s3:::assets-860415559748-ap-southeast-2
ap-south-1	arn:aws:s3:::assets-681137435769-ap-south-1

`aws codeartifact describe-domain` 명령을 사용하여 CodeArtifact 도메인에서 사용하는 Amazon S3 버킷을 가져올 수 있습니다.

```
aws codeartifact describe-domain --domain mydomain
```

```
{
  "domain": {
    "name": "mydomain",
    "owner": "111122223333",
    "arn": "arn:aws:codeartifact:us-west-2:111122223333:domain/mydomain",
    "status": "Active",
    "createdTime": 1583075193.861,
    "encryptionKey": "arn:aws:kms:us-west-2:111122223333:key/a73que8sq-ba...",
```

```
    "repositoryCount": 13,
    "assetSizeBytes": 513830295,
    "s3BucketArn": "arn:aws:s3:::assets-787052242323-us-west-2"
  }
}
```

예제

다음 예는 us-east-1 리전의 CodeArtifact 작업에 필요한 Amazon S3 버킷에 액세스 권한을 부여하는 방법입니다. 다른 리전의 경우 위 표를 기반으로 해당 리전의 올바른 권한 ARN으로 Resource 항목을 업데이트해야 합니다.

```
{
  "Statement": [
    {
      "Sid": "Access-to-specific-bucket-only",
      "Principal": "*",
      "Action": [
        "s3:GetObject"
      ],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::assets-193858265520-us-east-1/*"]
    }
  ]
}
```

VPC에서 CodeArtifact 사용

CodeArtifact에 대한 VPC 엔드포인트 생성에서 생성한 com.amazonaws.*region*.codeartifact.repositories VPC 엔드포인트에서 프라이빗 DNS를 활성화할 수 없거나 활성화하지 않으려는 경우 VPC의 CodeArtifact를 사용하려면 리포지토리 엔드포인트에 다른 구성을 사용해야 합니다. com.amazonaws.*region*.codeartifact.repositories 엔드포인트에 프라이빗 DNS가 활성화되어 있지 않은 경우 프라이빗 DNS가 없는 codeartifact.repositories 엔드포인트 사용의 지침에 따라 CodeArtifact를 구성합니다.

프라이빗 DNS가 없는 **codeartifact.repositories** 엔드포인트 사용

CodeArtifact에 대한 VPC 엔드포인트 생성에서 생성한 com.amazonaws.*region*.codeartifact.repositories VPC 엔드포인트에서 프라이빗 DNS를

활성화할 수 없거나 활성화하지 않으려는 경우, 다음 지침에 따라 올바른 CodeArtifact URL로 패키지 관리자를 구성해야 합니다.

1. 다음 명령을 실행하여 호스트 이름을 재정의하는 데 사용할 VPC 엔드포인트를 찾습니다.

```
$ aws ec2 describe-vpc-endpoints --filters Name=service-
name,Values=com.amazonaws.region.codeartifact.repositories \
  --query 'VpcEndpoints[*].DnsEntries[*].DnsName'
```

   출력은 다음과 같습니다.

```
[
  [
    "vpce-0743fe535b883ffff-76ddffff.d.codeartifact.us-west-2.vpce.amazonaws.com"
  ]
]
```

2. 패키지 형식, CodeArtifact 도메인 이름, CodeArtifact 리포지토리 이름을 포함하도록 VPC 엔드포인트 경로를 업데이트합니다. 다음 예제를 참조하세요.

```
https://vpce-0743fe535b883ffff-76ddffff.d.codeartifact.us-
west-2.vpce.amazonaws.com/format/d/domain_name-domain_owner/repo_name
```

   예제 엔드포인트에서 다음 필드를 변경합니다.

   - *##*: 유효한 CodeArtifact 패키지 형식(예: npm 또는 pypi)으로 변경합니다.
   - *domain_name*: 패키지를 호스팅하는 CodeArtifact 리포지토리가 포함된 CodeArtifact 도메인으로 대체합니다.
   - *domain_owner*: CodeArtifact 도메인 소유자의 ID(예: 111122223333)로 대체합니다.
   - *repo_name*: 패키지를 호스팅하는 CodeArtifact 리포지토리로 대체합니다.

   다음 URL은 npm 리포지토리 엔드포인트의 예입니다.

```
https://vpce-0dc4daf7fca331ed6-et36qa1d.d.codeartifact.us-
west-2.vpce.amazonaws.com/npm/d/domainName-111122223333/repoName
```

3. 이전 단계에서 업데이트된 VPC 엔드포인트를 사용하도록 패키지 관리자를 구성합니다. CodeArtifact `login` 명령을 사용하지 않고 패키지 관리자를 구성해야 합니다. 각 패키지 형식에 대한 구성 지침은 다음 설명서를 참조하세요.

- npm: 로그인 명령을 실행하지 않고 npm 구성하기
- nuget: 로그인 명령 없이 nuget 또는 dotnet 구성
- pip: 로그인 명령 없이 pip를 구성하려면
- twine: CodeArtifact로 twine 구성 및 사용
- Gradle: Gradle과 함께 CodeArtifact 사용
- mvn: mvn과 함께 CodeArtifact 사용

CodeArtifact를 위한 VPC 엔드포인트 정책 생성

CodeArtifact를 위한 VPC 엔드포인트 정책을 생성하려면 다음을 지정합니다.

- 작업을 수행할 수 있는 위탁자.
- 수행할 수 있는 작업.
- 작업을 수행할 수 있는 리소스.

다음 예제 정책은 123456789012 계정의 보안 주체가 `GetAuthorizationToken` API를 호출하고 CodeArtifact 리포지토리에서 패키지를 가져올 수 있도록 지정합니다.

```
{
  "Statement": [
    {
      "Action": [
        "codeartifact:GetAuthorizationToken",
        "codeartifact:GetRepositoryEndpoint",
        "codeartifact:ReadFromRepository",
        "sts:GetServiceBearerToken"
      ],
      "Effect": "Allow",
      "Resource": "*",
      "Principal": {
         "AWS": "arn:aws:iam::123456789012:root"
       }
    }
  ]
}
```

AWS CloudFormation을 사용하여 CodeArtifact 리소스 생성

CodeArtifact는 리소스 및 인프라를 생성하고 관리하는 데 소요되는 시간을 줄일 수 있도록 AWS 리소스를 모델링하고 설정하는 데 도움이 되는 서비스인 AWS CloudFormation과 통합됩니다. 필요한 모든 AWS 리소스를 설명하는 템플릿을 생성하면 CloudFormation이 해당 리소스의 프로비저닝과 구성을 담당합니다.

CloudFormation을 사용할 때는 템플릿을 재사용하여 CodeArtifact 리소스를 일관되고 반복적으로 설정할 수 있습니다. 리소스를 한 번만 설명하고 여러 AWS 계정 및 리전에서 동일한 리소스를 반복적으로 프로비저닝할 수 있습니다.

CodeArtifact 및 CloudFormation 템플릿

CodeArtifact 및 관련 서비스에 대한 리소스를 프로비저닝하고 구성하려면 CloudFormation 템플릿을 이해해야 합니다. 템플릿은 JSON 또는 YAML로 서식 지정된 텍스트 파일입니다. 이 템플릿은 CloudFormation 스택에서 프로비저닝할 리소스에 대해 설명합니다. JSON 또는 YAML에 익숙하지 않은 경우, CloudFormation Designer를 사용하면 CloudFormation 템플릿을 시작하는 데 도움이 됩니다. 자세한 내용은 AWS CloudFormation 사용 설명서의 AWS CloudFormation Designer란 무엇입니까?를 참조하세요.

CodeArtifact는 CloudFormation에서 도메인, 리포지토리 및 패키지 그룹을 생성하는 기능을 지원합니다. JSON 및 YAML 템플릿의 예제를 포함한 자세한 내용은 CloudFormation 사용 설명서에서 다음 항목을 참조하세요.

- AWS::CodeArtifact::Domain
- AWS::CodeArtifact::Repository
- AWS::CodeArtifact::PackageGroup

CodeArtifact 리소스 삭제 방지

CodeArtifact 리포지토리에는 분실 시 재생성하기가 쉽지 않을 수 있는 중요한 애플리케이션 종속성이 포함되어 있습니다. CloudFormation으로 CodeArtifact 리소스를 관리할 때 CodeArtifact 리소스가 실수로 삭제되지 않도록 보호하려면 모든 도메인과 리포지토리에 값이 `Retain`인 `DeletionPolicy` 및 `UpdateRetainPolicy` 속성을 포함해야 합니다. 이렇게 하면 스택 템플릿에서 리소스가 제거되거나 전체 스택이 실수로 삭제되는 경우 삭제가 방지됩니다. 다음 YAML 스니펫은 다음과 같은 속성을 가진 기본 도메인 및 리포지토리를 보여줍니다.

```
Resources:
    MyCodeArtifactDomain:
        Type: 'AWS::CodeArtifact::Domain'
        DeletionPolicy: Retain
        UpdateReplacePolicy: Retain
        Properties:
            DomainName: "my-domain"

    MyCodeArtifactRepository:
        Type: 'AWS::CodeArtifact::Repository'
        DeletionPolicy: Retain
        UpdateReplacePolicy: Retain
        Properties:
            RepositoryName: "my-repo"
            DomainName: !GetAtt MyCodeArtifactDomain.Name
```

이러한 속성에 대한 자세한 내용은 AWS CloudFormation 사용 설명서의 DeletionPolicy 및 UpdateReplacePolicy 섹션을 참조하십시오.

CloudFormation에 대해 자세히 알아보기

CloudFormation에 대한 자세한 내용은 다음 리소스를 참조하세요.

- AWS CloudFormation
- AWS CloudFormation 사용 설명서
- AWS CloudFormation 명령줄 인터페이스 사용 설명서

AWS CodeArtifact 문제 해결

다음은 CodeArtifact에서 일반적으로 발생하는 문제를 해결하는 데 유용한 정보입니다.

형식별 문제 해결 정보는 다음 주제를 참조하세요.

- Maven 문제 해결
- Swift 문제 해결

알림을 볼 수 없습니다.

문제: 개발자 도구 콘솔을 사용할 경우 설정에서 알림을 선택하면 권한 오류가 표시됩니다.

가능한 해결 방법: 알림은 개발자 도구 콘솔의 기능이지만 CodeArtifact에서는 현재 알림을 지원하지 않습니다. CodeArtifact의 모든 관리형 정책에는 사용자가 알림을 보거나 관리할 수 있는 권한이 포함되어 있지 않습니다. 개발자 도구 콘솔에서 다른 서비스를 사용하고 해당 서비스가 알림을 지원하는 경우 해당 서비스의 관리형 정책에는 해당 서비스에 대한 알림을 보고 관리하는 데 필요한 권한이 포함됩니다.

리소스에 태그 지정

태그는 사용자 또는 AWS에서 AWS 리소스에 할당하는 사용자 지정 속성 레이블입니다. 각 AWS 태그는 두 부분으로 구성됩니다.

- 태그 키(예: `CostCenter`, `Environment`, `Project` 또는 `Secret`). 태그 키는 대소문자를 구별합니다.
- 태그 값(예: `111122223333`, `Production` 또는 팀 이름)으로 알려진 선택적 필드. 태그 값을 생략하는 것은 빈 문자열을 사용하는 것과 같습니다. 태그 키처럼 태그 값은 대/소문자를 구별합니다.

태그 키와 태그 값을 합해서 키 값 페어라고 합니다.

태그를 사용하면 AWS 리소스를 식별하고 구성하는 데 도움이 됩니다. 많은 AWS 서비스가 태그 지정을 지원합니다. 따라서 서로 다른 서비스의 리소스에 동일한 태그를 할당하여 리소스가 서로 관련이 있음을 나타낼 수 있습니다. 예를 들어 AWS CodeBuild 프로젝트에 할당한 것과 동일한 태그를 리포지토리에 할당할 수 있습니다.

태그 사용에 대한 팁과 모범 사례는 AWS 리소스 태그 지정 모범 사례 백서를 참조하십시오.

CodeArtifact에서 다음 리소스 유형에 태그를 지정할 수 있습니다.

- CodeArtifact에 있는 리포지토리에 태그 추가
- CodeArtifact에서 도메인에 태그 추가

콘솔, AWS CLI, CodeArtifact API 또는 AWS SDK를 사용하여 다음을 수행할 수 있습니다.

- 도메인 또는 리포지토리를 생성할 때 도메인 또는 리포지토리에 태그를 추가할 수 있습니다*.
- 도메인 또는 리포지토리의 태그를 추가, 관리 및 제거합니다.

* 콘솔에서 도메인이나 리포지토리를 생성할 때는 도메인이나 리포지토리에 태그를 추가할 수 없습니다.

태그로 리소스를 식별, 구성 및 추적하는 것 외에도 IAM 정책의 태그를 사용하여 리소스를 보고 상호 작용할 수 있는 사용자를 제어할 수 있습니다. 태그 기반 액세스 정책의 예는 태그를 사용하여 CodeArtifact 리소스에 대한 액세스 제어 단원을 참조하세요.

태그를 사용한 CodeArtifac 비용 할당

태그를 사용하여 CodeArtifact에서 스토리지 비용과 요청 비용을 모두 할당할 수 있습니다.

CodeArtifact에서의 데이터 스토리지 비용 할당

데이터 스토리지 비용은 도메인과 연결되어 있으므로 CodeArtifact 스토리지 비용을 할당하기 위해 도메인에 적용되는 모든 태그를 사용할 수 있습니다. 도메인에 태그를 추가하는 자세한 방법은 CodeArtifact에서 도메인에 태그 추가 섹션을 참조하세요.

CodeArtifact에서의 요청 비용 할당

대부분의 요청 사용량은 리포지토리와 연결되어 있으므로 CodeArtifact 요청 비용을 할당하려면 리포지토리에 적용되는 모든 태그를 사용할 수 있습니다. 리포지토리에 태그를 추가하는 자세한 방법은 CodeArtifact에 있는 리포지토리에 태그 추가 섹션을 참조하세요.

일부 요청 유형은 리포지토리가 아닌 도메인과 연결되므로 요청과 관련된 요청 사용량 및 비용이 도메인의 태그에 할당됩니다. 요청 유형이 도메인 또는 리포지토리와 연결되어 있는지 확인하는 가장 좋은 방법은 서비스 권한 부여 참조의 AWS CodeArtifact에서 정의한 작업 테이블을 사용하는 것입니다. 작업 열에서 요청 유형을 찾고 해당 리소스 유형 열의 값을 확인하세요. 리소스 유형이 도메인인 경우 해당 유형의 요청은 도메인에 요금이 청구됩니다. 리소스 유형이 리포지토리 또는 패키지인 경우 해당 유형의 요청은 리포지토리에 요금이 청구됩니다. 일부 작업은 두 가지 리소스 유형에 모두 해당합니다. 이러한 작업의 경우 요청에서 전달된 값에 따라 청구 리소스가 달라집니다.

in AWS CodeArtifact 할당량

다음 표에서는 CodeArtifact의 리소스 할당량을 설명합니다. CodeArtifact의 서비스 엔드포인트 목록과 함께 리소스 할당량을 보려면 Amazon Web Services 일반 참조의 AWS Service Quotas를 참조하세요.

다음 CodeArtifact 리소스 할당량에 대한 서비스 할당량 증가를 요청할 수 있습니다. 할당량 증가 요청에 대한 자세한 내용은 AWS Service Quotas를 참조하세요.

이름	기본값	조정 가능	설명
자산 파일 크기	지원되는 각 리전: 5GB	예	자산당 최대 파일 크기
패키지 버전당 자산	지원되는 각 리전: 150	아니요	패키지 버전당 최대 자산 수
초당 CopyPackageVersions 요청	지원되는 각 리전: 5개	예	초당 CopyPackageVersions에 대해 할 수 있는 최대 호출 수입니다.
리포지토리당 직접 업스트림	지원되는 각 리전: 10	아니요	리포지토리당 최대 직접 업스트림 리포지토리 수입니다.
AWS 계정당 도메인	지원되는 각 리전: 10개	예	AWS 계정당 생성할 수 있는 최대 도메인 수입니다.
초당 GetAuthorizationToken 요청	지원되는 각 리전: 40	예	초당 검색되는 인증 토큰의 최대 수입니다.
초당 GetPackageVersionAsset 요청	지원되는 각 지역: 50	예	초당 GetPackageVersionAsset에 대해 할 수 있는 최대 호출 수입니다.

이름	기본값	조정 가능	설명
초당 ListPackageVersionAssets 요청	지원되는 각 리전: 200	예	초당 ListPackageVersionAssets에 대해 할 수 있는 최대 호출 수입니다.
초당 ListPackageVersions 요청	지원되는 각 리전: 200	예	초당 ListPackageVersions에 대해 할 수 있는 최대 호출 수입니다.
초당 ListPackages 요청	지원되는 각 리전: 200	예	초당 ListPackages에 대해 할 수 있는 최대 호출 수입니다.
초당 PublishPackageVersion 요청	지원되는 각 리전: 10개	예	초당 PublishPackageVersion에 대해 할 수 있는 최대 호출 수입니다.
단일 AWS 계정에서 초당 읽기 요청	지원되는 각 리전: 800	예	초당 한 AWS 계정의 최대 읽기 요청 수입니다.
도메인당 리포지토리	지원되는 각 리전: 1,000	예	도메인별로 생성할 수 있는 최대 리포지토리 수입니다.
단일 인증 토큰을 사용한 초당 요청	지원되는 각 리전: 1,200	아니요	단일 인증 토큰을 사용하는 초당 최대 요청 수입니다.
IP 주소당 인증 토큰이 없는 요청	지원되는 각 리전: 600	아니요	단일 IP 주소에서 인증 토큰이 없는 초당 최대 요청 수입니다.
업스트림 리포지토리가 검색되었습니다.	지원되는 각 리전: 25	아니요	패키지를 확인할 때 검색된 업스트림 리포지토리의 최대 수입니다.

이름	기본값	조정 가능	설명
단일 AWS 계정에서 초당 쓰기 요청	지원되는 각 리전: 100	예	초당 한 AWS 계정의 최대 쓰기 요청 수입니다.

> **Note**
>
> 일반적으로 CodeArtifact에 대한 각 읽기 요청은 할당량에 대해 하나의 요청으로 계산됩니다. 그러나 Ruby 패키지 형식의 경우 `/api/v1/dependencies` 작업에 대한 단일 읽기 요청으로 여러 패키지에 대한 데이터를 요청할 수 있습니다.
>
> 예를 들어 요청은 `https://${CODEARTIFACT_REPO_ENDPOINT}/api/v1/dependencies?gems=gem1,gem2.gem3`와 같을 수 있습니다. 이 예제에서 요청은 할당량에 대해 3개의 요청으로 계산됩니다.
>
> 여러 요청은 서비스 할당량에만 적용되며 청구에는 적용되지 않습니다. 이 예제에서는 서비스 할당량에 대한 3개의 요청으로 계산되지만 요청 1개에 대해서만 요금이 청구됩니다.

AWS CodeArtifact 사용 설명서 문서 기록

다음 표에서는 CodeArtifact 설명서의 주요 변경 사항에 대해 설명합니다.

변경 사항	설명	날짜
CodeArtifact와 함께 Cargo를 구성 및 사용하기 위한 설명서 추가	CodeArtifact는 이제 Cargo 크레이트를 지원합니다. CodeArtifact 리포지토리를 사용하도록 Cargo를 구성하는 방법에 관한 지침이 포함된 설명서가 추가되었습니다. 자세한 내용은 CodeArtifact를 Cargo와 함께 사용 단원을 참조하십시오.	2024년 6월 20일
CodeArtifact와 함께 Ruby를 구성 및 사용하기 위한 설명서 추가	CodeArtifact는 이제 Ruby Gem을 지원합니다. CodeArtifact 리포지토리를 사용하도록 Ruby 패키지 관리자를 구성하는 방법에 관한 지침이 포함된 설명서가 추가되었습니다. 자세한 내용은 CodeArtifact를 Ruby와 함께 사용 단원을 참조하십시오.	2024년 4월 30일
고객 관리형 키를 사용하여 도메인을 생성하기 위한 예제 AWS KMS 키 정책 추가	CodeArtifact 도메인에서 자산을 암호화하기 위한 고객 관리형 KMS 키를 생성하는 데 사용할 수 있는 예제 키 정책이 추가되었습니다. 자세한 내용은 AWS KMS 키 정책 예제 단원을 참조하십시오.	2024년 4월 18일
패키지 그룹 시작을 지원하는 설명서가 추가되었습니다.	CodeArtifact에서 패키지 그룹 관리 및 사용에 대한 설명서가 추가되었습니다. 자세한 내용	2024년 3월 21일

	은 CodeArtifact에서의 패키지 그룹 작업 단원을 참조하십시오.	
aws codeartifact login 명령에 대한 설명서에 유효한 패키지 관리자를 추가했습니다.	`aws codeartifact login` 명령과 함께 사용할 유효한 패키지 관리자 목록에 `dotnet`, `nuget` 및 `swift`가 추가되었습니다. 자세한 내용은 AWS CodeArtifact 인증 및 토큰 단원을 참조하십시오.	2024년 2월 18일
CI 시스템에서 중단된 Xcode에 대한 Swift 문제 해결 문서에 항목이 추가되었습니다.	암호에 대한 키체인 프롬프트로 인해 Xcode가 CI 시스템에서 중단될 수 있는 문제에 대한 솔루션을 포함한 정보가 추가되었습니다. 자세한 내용은 암호에 대한 키체인 프롬프트로 인해 Xcode가 CI 시스템에서 중단됨 단원을 참조하십시오.	2024년 2월 6일
npm 8.x 이상에서 느린 npm 패키지 설치 시간 문제를 해결하기 위한 정보가 추가되었습니다.	구축 시간이 느려질 수 있는 CodeArtifact의 느린 npm 패키지 설치 시간을 해결하는 방법에 대한 정보가 추가되었습니다. 자세한 내용은 npm 8.x 이상을 사용한 느린 설치 속도 문제 해결 단원을 참조하십시오.	2023년 12월 29일
CodeArtifact의 Python 패키지 자산과 메타데이터 동작에 대한 정보가 업데이트되었습니다	CodeArtifact 리포지토리가 Python 패키지 버전 자산 및 메타데이터를 유지하고 새로 고치는 방법에 대한 정보가 업데이트되었습니다. 자세한 내용은 업스트림 및 외부 연결에서 Python 패키지 요청하기 단원을 참조하십시오.	2023년 12월 14일

CodeArtifact 모니터링에 대한 문서를 재구성했습니다	CodeArtifact 이벤트 모니터링에 대한 정보를 재구성하고 Amazon CloudWatch 지표를 사용하여 CodeArtifact 요청을 보는 방법에 대한 정보를 추가했습니다. 자세한 내용은 CodeArtifact 모니터링 단원을 참조하십시오.	2023년 12월 14일
를 사용하여 CodeArtifact 리소스를 관리하는 방법에 대한 추가 정보 추가 CloudFormation	CloudFormation으로 관리되는 CodeArtifact 리소스의 삭제를 방지하는 방법에 대한 섹션을 포함하여, CloudFormation으로 CodeArtifact 리소스를 관리하는 방법에 관한 문서의 참조 및 링크를 추가했습니다. 자세한 내용은 CodeArtifact 리소스 삭제 방지 단원을 참조하십시오.	2023년 12월 7일
AWS KMS 외부 키 스토어(XKS)에 대한 CodeArtifact의 지원을 자세히 설명하는 설명서 추가	CodeArtifact에서 XKS 키를 사용하는 것을 포함하여 CodeArtifact의 KMS 키 지원에 관한 정보가 포함된 섹션이 추가되었습니다. 자세한 내용은 CodeArtifact에서 지원되는 AWS KMS 키 유형 단원을 참조하십시오.	2023년 10월 31일
기존 문제 해결 설명서를 업데이트하고 새 문제 해결 설명서를 추가	Maven 문제 해결 주제를 추가했으며 Swift 및 Maven 문제 해결 설명서와 연결되는 링크를 일반 문제 해결 주제에 포함했습니다. 자세한 내용은 AWS CodeArtifact 문제 해결 단원을 참조하십시오.	2023년 9월 28일

Swift Package Manager 게시 명령을 포함하도록 설명서를 업데이트	Swift 5.9에서는 Swift 패키지를 생성하여 패키지 리포지토리에 게시하는 `swift package-registry publish` 명령을 도입했습니다. 해당 명령 사용에 대한 지침을 포함하도록 Swift 설명서를 업데이트했습니다. 자세한 내용은 CodeArtifact를 Swift와 함께 사용 단원을 참조하십시오.	2023년 9월 25일
Swift로 CodeArtifact를 구성하기 위한 설명서 추가	CodeArtifact는 이제 Swift 패키지를 지원합니다. CodeArtifact 리포지토리를 사용하도록 Swift를 구성하는 방법에 관한 지침이 포함된 설명서가 추가되었습니다. 자세한 내용은 CodeArtifact를 Swift와 함께 사용 단원을 참조하십시오.	2023년 9월 20일
제거된 Python 패키지 버전을 CodeArtifact가 처리하는 방법에 관한 지침을 추가	Python 패키지 버전이 제거되었는지 확인하는 방법, 제거된 패키지 버전을 CodeArtifact가 처리하는 방법, 공통적인 질문에 대한 답변 등에 관한 정보가 포함된 설명서가 추가되었습니다. 자세한 내용은 삭제된 패키지 버전 단원을 참조하십시오.	2023년 8월 2일
Yarn 설명서의 잘못된 명령줄 명령을 수정	CodeArtifact 인증 토큰을 가져와서 Yarn 설명서의 환경 변수에 저장하는 잘못된 명령줄 명령을 수정했습니다.	2023년 7월 20일

Python 설명서에 대한 사소한 추가 및 작은 버그 수정	각 문서에 pip 및 twine 정보를 추가하고 twine과 함께 `codeartifact login` 명령을 사용할 때 발생하는 상황을 수정했습니다. 자세한 내용은 CodeArtifact로 pip 구성 및 사용 및 CodeArtifact로 twine 구성 및 사용 섹션을 참조하세요.	2023년 7월 14일
CodeBuild 설명서의 잘못된 dotnet 명령을 수정	CodeBuild에서 NuGet 패키지 사용 설명서의 `dotnet add package` 명령을 수정했습니다.	2023년 7월 13일
Updated AWS CodeArtifact 및 AWS Identity and Access Management 설명서	CodeArtifact 설명서의 IAM을 검토하여 다른 AWS 서비스에 대한 설명서와의 명확성과 일관성을 추가했습니다. Identity and Access Management for AWS CodeArtifact을(를) 참조하세요.	2023년 5월 24일
제거된 Python 패키지 버전에 관한 정보 추가	제거된 Python 패키지 버전 메타데이터를 CodeArtifact가 유지하는 방법에 관한 정보가 추가되었습니다. 자세한 내용은 삭제된 패키지 버전을 참조합니다.	2023년 4월 11일
Clojure 지원에 관한 정보 추가	Clojure 프로젝트의 종속성 관리를 포함하여 Clojure 지원에 관한 정보가 추가되었습니다. 자세한 내용은 CodeArtifact를 deps.edn과 함께 사용 단원을 참조하십시오.	2023년 3월 21일

일반 패키지 게시에 관한 정보 추가	AWS CLI를 사용하여 패키지 콘텐츠를 게시하고 다운로드하는 방법과 일반 패키지에 관한 정보가 추가되었습니다. 자세한 내용은 CodeArtifact를 일반 패키지와 함께 사용, 일반 패키지 게시 및 사용, 일반 패키지에 지원되는 명령 섹션을 참조하세요.	2023년 3월 10일
게시를 위한 자산 크기 제한에 관한 정보 추가	게시에 대한 자산 크기 제한을 설명하는 섹션을 패키지 게시에 추가했습니다.	2022년 6월 21일
외부 연결 설명서를 리팩터링	외부 연결 설명서를 옮긴 후 사용자의 최종 목표 즉, CodeArtifact 리포지토리를 퍼블릭 패키지 리포지토리에 연결하는 것에 중점을 두도록 설명서를 재구성했습니다. 또한 목표 달성을 위한 다양한 방법에 관한 지침과 정보도 더 추가되었습니다. 자세한 내용은 CodeArtifact 저장소를 공용 저장소에 연결하기 단원을 참조하십시오.	2022년 5월 9일
Amazon CloudWatch Events에 대한 CodeArtifact 이벤트 정보 업데이트	`account` 필드에 더 많은 정보를 추가하고 `repositoryAdministrator` 필드를 추가했습니다. 자세한 내용은 CodeArtifact 이벤트 형식 및 예제 단원을 참조하십시오.	2022년 3월 7일

프라이빗 DNS가 없는 VPC에서 CodeArtifact를 사용하기 위한 구성 지침을 추가	`codeartifact.repositories` VPC 엔드포인트에서 프라이빗 DNS를 활성화할 수 없거나 활성화하지 않으려는 경우, VPC의 CodeArtifact를 사용하려면 리포지토리 엔드포인트에 다른 구성을 사용해야 합니다. 자세한 정보는 프라이빗 DNS가 없는 `codeartifact.repositories` 엔드포인트 사용을 참조하세요.	2022년 2월 8일
패키지 버전 상태 업데이트를 위한 심층 설명서 추가	업데이트 패키지 버전 상태 설명서를 별도 주제로 확장했습니다. 필요한 IAM 권한, 다양한 시나리오에 대한 예제 AWS CLI 명령, 가능한 오류를 포함하여 패키지 버전의 상태를 업데이트하기 위한 설명서가 추가되었습니다. 자세한 정보는 패키지 버전 상태 업데이트을 참조하세요.	2021년 9월 1일

패키지 버전 복사 설명서에 더 자세한 권한 정보가 포함되도록 업데이트	CodeArtifact의 동일한 도메인 내에서 한 리포지토리의 패키지 버전을 다른 리포지토리로 복사하는 `aws codeartifact copy-package-versions` 명령을 직접적으로 호출하는 데 필요한 IAM 및 리소스 기반 정책 권한에 관한 자세한 정보가 추가되었습니다. 이제 소스 및 대상 리포지토리에 필요한 리소스 기반 정책의 예를 자세한 정보와 함께 확인할 수 있습니다. 자세한 정보는 패키지 복사에 필요한 IAM 권한을 참조하세요.	2021년 8월 25일
IntelliJ IDEA에서 Gradle 빌드를 실행하기 위한 설명서를 업데이트	CodeArtifact에서 플러그인을 가져오도록 Gradle을 구성하는 단계를 포함하여 IntelliJ IDEA에서 Gradle 빌드를 실행하기 위한 설명서를 업데이트했습니다. `aws codeartifact get-authorization-token` 에 대해 인라인 직접 호출을 사용하여 새로 실행할 때마다 새 CodeArtifact 인증 토큰을 생성하는 옵션도 추가되었습니다. 자세한 정보는 IntelliJ IDEA에서 Gradle 빌드 실행을 참조하세요.	2021년 8월 23일

Yarn with AWS CodeArtifact 구성 및 사용에 대한 설명서 추가	CodeArtifact로 npm 패키지를 관리하기 위해 Yarn 1.X 및 Yarn 2.X를 구성하고 사용하는 방법에 관한 설명서가 추가되었습니다. 자세한 정보는 CodeArtifact로 Yarn 구성 및 사용을 참조하세요.	2021년 7월 30일
AWS CodeArtifact는 이제 NuGet 패키지를 지원합니다.	CodeArtifact 사용자는 이제 NuGet 패키지를 게시하고 사용할 수 있습니다. CodeArtifact 리포지토리와 함께 `nuget` 및 `dotnet` 같은 Visual Studio 및 NuGet 명령줄 도구를 구성하고 사용하기 위한 설명서가 추가되었습니다. 자세한 정보는 CodeArtifact를 NuGet과 함께 사용을 참조하세요.	2020년 11월 19일
AWS CodeArtifact에서 리소스 태그 지정	AWS CodeArtifact에서 리포지토리 및 도메인 태그 지정에 대한 설명서가 추가되었습니다. 리소스에 태그 지정을(를) 참조하세요.	2020년 10월 30일
CodeArtifact에서 이제 지원 CloudFormation	이제 CodeArtifact 사용자는 CloudFormation 템플릿을 사용하여 CodeArtifact 리포지토리 및 도메인을 생성할 수 있습니다. 자세한 내용을 확인하고 시작하려면 AWS CloudFormation을 사용하여 CodeArtifact 리소스 생성을 참조합니다.	2020년 10월 8일

Amazon VPC에서 CodeArtifact를 사용하기 위한 Amazon S3 게이트웨이 엔드포인트 생성에 관한 정보 추가	Amazon EC2 명령을 사용하여 Amazon S3 게이트웨이 엔드포인트를 생성하는 방법에 대한 정보가 추가되었습니다. Amazon EC2 AWS CLI 이 설명서에는 CodeArtifact를 Amazon VPC 환경에서 사용하는 데 필요한 특정 권한에 관한 정보도 포함되어 있습니다. Amazon S3 게이트웨이 엔드포인트 생성을(를) 참조하세요.	2020년 8월 12일
curl을 사용하여 Maven 아티팩트를 게시하고 타사 Maven 아티팩트를 게시	curl을 사용한 게시 및 타사 아티팩트 게시에 대한 지침이 추가되었습니다.	2020년 8월 10일
정식 출시(GA) 릴리스	CodeArtifact 사용 설명서의 초기 버전입니다.	2020년 6월 10일

기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.