Add a permission의

AWS Proton

AWS Proton: Add a permission의

Table of Contents

지원 종료 알림: 2026년 10월 7일에 AWS 에 대한 지원이 종료됩니다 AWS Proton. 2026년 10월 7일 이후에는 AWS Proton 콘솔 또는 AWS Proton 리소스에 더 이상 액세스할 수 없습니다. 배포된 인프라는 그대로 유지됩니다. 자세한 내용은 AWS Proton 서비스 사용 중단 및 마이그레이션 안내서를 참조하세요.

기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.

AWS Proton란 무엇인가요?

AWS Proton 는 다음과 같습니다.

- 서버리스 및 컨테이너 기반 애플리케이션의 코드 프로비저닝 및 배포와 같은 자동화된 인프라

  AWS Proton 서비스는 두 갈래로 구성된 자동화 프레임워크입니다. 관리자는 서버리스 및 컨테이너 기반 애플리케이션을 위한 표준화된 인프라 및 배포 도구를 정의하는 버전이 지정된 서비스 템플릿을 생성합니다. 애플리케이션 개발자는 사용 가능한 서비스 템플릿 중에서 선택하여 애플리케이션 또는 서비스 배포를 자동화할 수 있습니다.

  AWS Proton 는 오래된 템플릿 버전을 사용하는 모든 기존 서비스 인스턴스를 식별합니다. 관리자는 클릭 한 번으로 업그레이드 AWS Proton 하도록 요청할 수 있습니다.
- 표준화된 인프라

  플랫폼 팀은 AWS Proton 및 버전이 지정된 인프라를 코드 템플릿으로 사용할 수 있습니다. 이러한 템플릿을 사용하여 아키텍처, 인프라 리소스, CI/CD 소프트웨어 배포 파이프라인이 포함된 표준 애플리케이션 스택을 정의하고 관리할 수 있습니다.
- CI/CD와 통합된 배포

  개발자는 AWS Proton 셀프 서비스 인터페이스를 사용하여 서비스 템플릿을 선택할 때 코드 배포를 위한 표준화된 애플리케이션 스택 정의를 선택합니다.는 리소스를 AWS Proton 자동으로 프로비저닝하고, CI/CD 파이프라인을 구성하고, 정의된 인프라에 코드를 배포합니다.

AWS Proton 플랫폼 팀용

관리자, 사용자 또는 플랫폼 팀 구성원은 인프라를 코드로 포함하는 환경 템플릿 및 서비스 템플릿을 생성합니다. 환경 템플릿은 여러 애플리케이션 또는 리소스에서 사용하는 공유 인프라를 정의합니다. 서비스 템플릿은 환경에 단일 애플리케이션 또는 마이크로서비스를 배포하고 유지 관리하는 데 필요한 인프라 유형을 정의합니다. An AWS Proton service는 일반적으로 여러 서비스 인스턴스와 파이프라인을 포함하는 서비스 템플릿의 인스턴스화입니다. AWS Proton 서비스 인스턴스는 특정 환경에서 서비스 템플릿을 인스턴스화한 것입니다. 사용자 또는 팀의 다른 구성원이 특정 서비스 템플릿과 호환되는 환경 템플릿을 지정할 수 있습니다. 템플릿에 대한 자세한 내용은 AWS Proton 템플릿 단원을 참조하세요.

다음 인프라를 코드 공급자로 사용할 수 있습니다 AWS Proton.

- CloudFormation

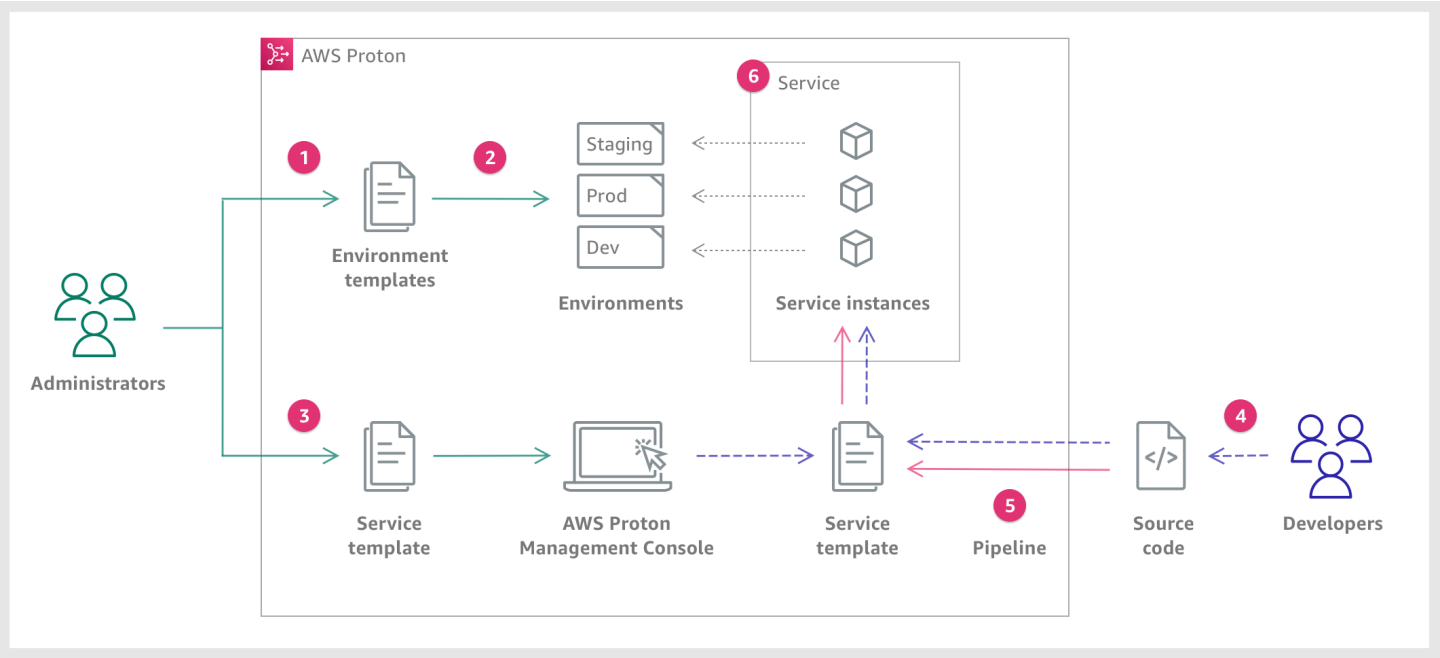
- Terraform

AWS Proton 개발자용

애플리케이션 개발자는가 서비스 인스턴스에서 애플리케이션을 배포하고 관리하는 서비스를 생성하는 데 사용하는 표준화된 서비스 템플릿을 선택합니다. AWS Proton AWS Proton 서비스는 일반적으로 여러 개의 서비스 인스턴스와 파이프라인을 포함하는 서비스 템플릿의 인스턴스화입니다.

AWS Proton 워크플로

다음 다이어그램은 이전 단락에서 설명한 주요 AWS Proton 개념을 시각화한 것입니다. 또한 간단한 AWS Proton 워크플로를 구성하는 요소에 대한 개략적인 개요도 제공합니다.



1

관

리자는 공유 리소스를 AWS Proton정의하는 환경 템플릿을 생성하고에 등록합니다.

2

AWS

Proton 는 환경 템플릿을 기반으로 하나 이상의 환경을 배포합니다.

3

관리자는 관련 인프라, 모니터링 및 CI/CD 리소스와 호환되는 환경 템플릿을 정의하는 서비스 템플릿을 생성하고에 등록합니다. AWS Proton

4

개

발자는 등록된 서비스 템플릿을 선택하고 소스 코드 리포지토리로 연결되는 링크를 제공합니다.

5

AWS Proton 는 서비스 인스턴스에 대한 CI/CD 파이프라인으로 서비스를 프로비저닝합니다.

6

AWS Proton 는 선택한 서비스 템플릿에 정의된 대로 소스 코드를 실행하는 서비스 및 서비스 인스턴스를 프로비저닝하고 관리합니다. 서비스 인스턴스는 파이프라인의 단일 단계에 대한 환경에서 선택한 서비스 템플릿을 인스턴스화하는 것입니다(예: Prod).

AWS Proton 서비스 사용 중단 및 마이그레이션 가이드

AWS 는 2026년 10월 7일에 지원이 종료 AWS Proton되어 중단하기로 결정했습니다. 신규 고객은 2025년 10월 7일 이후에는 가입할 수 없지만 기존 고객은 2026년 10월 7일까지 서비스를 계속 사용할 수 있습니다.

사용 중단까지의 서비스 상태

2026년 10월 7일까지 기존 AWS Proton 고객은 서비스를 정상적으로 계속 사용할 수 있습니다. 이 기간 동안 AWS 는 다음을 수행합니다.

1. 보안 패치 및 중요한 버그 수정 제공
2. 서비스 가용성 및 성능 유지
3. AWS Support 채널을 통해 계속 지원 제공
4. 서비스에 새 기능 추가 안 함

중요 마이그레이션 정보

AWS Proton 는 주로 인프라를 배포하기 위한 CI/CD 도구입니다. AWS Proton 가 더 이상 사용되지 않으면 배포된 CloudFormation 스택과 해당 스택에서 관리하는 리소스는 그대로 유지되고 계속 작동합니다. 사용 중단은 배포된 인프라가 아닌 전송 파이프라인과 AWS Proton 서비스 자체에만 영향을 미칩니다.

대체 솔루션

코드형 인프라 및 CI/CD 기능을 유지 관리하는 데 도움이 AWS Proton 되는에 대한 몇 가지 대안을 확인했습니다.

CloudFormation Git 동기화

가장 적합: GitOps 워크플로를 원하는 CloudFormation을 사용하는 팀

Git 동기화를 사용하면 플랫폼 팀이 개발 팀이 포크할 수 있는 Git 리포지토리에서 CloudFormation 템플릿을 모델링할 수 있습니다. 개발자는 파라미터 파일을 업데이트하고, 변경 사항을 포크된 리포지토리로 푸시하고, Git 동기화는 스택을 업데이트합니다.

주요 이점:

1. 와 유사한 개발자 경험 AWS Proton
2. 기존 CloudFormation 지식을 활용합니다.
3. 플랫폼과 개발자 팀 간의 명확한 분리

제한:

1. 환경에 대한 개념 없음
2. 고급 파이프라인 기능 없음
3. 다른 Git 공급자에서 사용할 수 없는 GitHub 기능을 사용합니다.

자세히 알아보기: Git 동기화

Harmonix 켜기 AWS

가장 적합: 포괄적인 내부 개발자 포털이 필요한 엔터프라이즈

Harmonix는 Backstage.io 기반 AWS Partner 솔루션이며 팀이 Proton과 유사한 템플릿, 환경 및 서비스를 생성할 수 있는 AWS 플러그인을 제공합니다.

주요 이점:

1. 와 유사한 기능 AWS Proton
2. 널리 사용되는 백스테이지 프레임워크를 기반으로 구축됨

3. 개발자 포털 경험 완성

제한:

1. AWS 서비스 팀에서 유지 관리하지 않음
2. 사용자 지정이 필요할 수 있는 참조 구현

자세히 알아보기: https://harmonixonaws.io/

AWS CodePipeline 및 AWS CodeBuild

가장 적합: 최대한의 유연성과 제어가 필요한 팀

AWS 기본 CI/CD 서비스를 사용하여 유연성과 제어력을 높여 AWS Proton 기능을 복제합니다.

주요 이점:

1. 최대 유연성
2. AWS 서비스와의 심층 통합
3. 활성 유지 관리 및 새로운 기능

제한:

1. 더 많은 구현 작업 필요
2. out-of-box 개발자 셀프 서비스 감소

자세히 알아보기:

정의 AWS CodePipeline

정의 AWS CodeBuild

GitHub Actions

가장 적합: 단순성을 원하는 GitHub를 사용하는 소규모 팀

>주요 이점

1. GitHub 리포지토리와의 간편한 통합

2. GitHub 사용자를 위한 간단한 설정
3. 재사용 가능한 작업의 대규모 마켓플레이스

제한:

1. GitHub 에코시스템과 연결됨
2. 플랫폼 팀 제어를 위해 더 많은 작업이 필요할 수 있습니다.

자세히 알아보기:

GitHub 작업 설명서

CI/CD 예제: GitHub 작업과 통합 - Amazon EC2에 웹 앱을 배포하기 위한 CI/CD 파이프라인

마이그레이션 지침

마이그레이션 프로세스는 구현 및 선택한 대안에 따라 달라집니다. 일반 단계:

1. Proton 리소스 인벤토리 작성:
2. 대체 솔루션을 선택합니다.
3. 템플릿 데이터 추출:
4. 선택한 대안을 구현합니다.
5. 프로덕션 워크로드 마이그레이션:

특정 마이그레이션 지원은 AWS Support 또는 계정 팀에 문의하세요.

FAQ

Q:가 AWS 중단되는 이유는 무엇입니까 AWS Proton? A: 다른 AWS 및 AWS Partner 솔루션을 통해 코드형 인프라 정책 적용에 대한 고객 요구 사항을 충족할 수 있는 더 나은 기회를 찾아냈습니다.

Q: 사용 중단 날짜 이후에도 기존 인프라가 계속 작동하나요? A: Yes. AWS Proton 는 주로 CI/CD 도구입니다. 배포된 CloudFormation 스택과 해당 스택이 관리하는 리소스는 그대로 유지되며 계속 작동합니다. 사용 중단은 배포된 인프라가 아닌 전송 파이프라인에만 영향을 미칩니다.

Q: 마이그레이션에 대한 도움을 받으려면 어떻게 해야 하나요? A: 마이그레이션을 지원할 AWS Support 수 있습니다. 에 문의AWS Support하거나 AWS 계정 관리자에게 도움을 요청할 수 있습니다.

Q: 어떤 대안을 선택해야 하나요? A: 최선의 대안은 특정 사용 사례에 따라 다릅니다.

1. 간단한 GitOps 워크플로의 경우: CloudFormation Git Sync
2. 개발자 포털이 필요한 기업의 경우: Harmonix On AWS
3. 유연성을 극대화하려면 AWS CodePipeline AWS CodeBuild
4. 이미 GitHub에 있는 팀의 경우: GitHub Actions

Q: 2026년 10월 7일까지 마이그레이션하지 않으면 어떻게 되나요? A: 더 이상에 액세스할 수 없습니다 AWS Proton. 기존 인프라는 계속 작동하지만를 사용하여 인프라를 AWS Proton 관리하거나 업데이트할 수 없습니다.

Q: 내 데이터는 얼마나 오래 보관되나요? A: 2026년 10월 7일까지. 이 날짜 이후에는 모든 데이터가 삭제됩니다.

추가 질문이 있는 경우에 문의하십시오 AWS Support.

설정

서비스 및 환경 템플릿을 만들고 등록할 수 있도록 이 섹션의 작업을 완료하세요. 를 사용하여 환경 및 서비스를 배포하려면이 값이 필요합니다 AWS Proton.

> **Note**
>
> 추가 비용 AWS Proton 없이를 제공합니다. 서비스 및 환경 템플릿을 무료로 생성, 등록 및 유지 관리할 수 있습니다. 또한 스토리지, 보안 및 배포와 같은 자체 작업을 AWS Proton 자체 관리할 수 있습니다. 사용 중에 발생하는 유일한 비용은 다음과 AWS Proton 같습니다.
>
> - 배포 및 유지 AWS Proton 관리하도록 지시한 AWS 클라우드 리소스를 배포하고 사용하는 데 드는 비용입니다.
> - 코드 리포지토리에 대한 AWS CodeStar 연결을 유지하는 데 드는 비용입니다.
> - S3 버킷을 유지 관리하는 데 드는 비용(버킷을 사용하여 AWS Proton에 입력을 제공하는 경우). the section called “템플릿 번들”에 Git 리포지토리를 사용하도록 the section called “템플릿 동기화 구성”로 전환하면 이러한 비용을 피할 수 있습니다.

주제

IAM으로 설정

에 가입하면 AWS를 AWS포함한의 모든 서비스에가 자동으로 등록 AWS 계정 됩니다 AWS Proton. 사용한 서비스에 대해서만 청구됩니다.

> **Note**
>
> 관리자와 개발자를 포함한 팀원 모두가 같은 계정을 사용해야 합니다.

에 가입 AWS

이 없는 경우 다음 단계를 AWS 계정완료하여 생성합니다.

에 가입하려면 AWS 계정

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

   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.

IAM 사용자를 생성합니다.

다음 옵션 중 하나를 선택하여 관리 사용자를 생성합니다.

관리자를 관리하는 방법 한 가지 선택	목적	By	다른 방법
IAM Identity Center에서 (권장)	단기 보안 인증 정보를 사용하여 AWS에 액세스합니다. 이는 보안 모범 사례와 일치합니다. 모범 사례에 대한 자세한 내용은 IAM 사용 설명서의 IAM의 보안 모범 사례를 참조하세요.	AWS IAM Identity Center 사용 설명서의 시작하기 지침을 따릅니다.	AWS Command Line Interface 사용 설명서에서 사용하도록 AWS CLI 를 구성 AWS IAM Identity Center하여 프로그래밍 방식 액세스를 구성합니다.
IAM에서 (권장되지 않음)	장기 보안 인증 정보를 사용하여 AWS에 액세스합니다.	IAM 사용 설명서의 비상 액세스를 위한 IAM 사용자 생성에 나와 있는 지침을 따르세요.	IAM 사용 설명서에 나온 IAM 사용자의 액세스 키 관리 단계를 수행하여 프로그래밍 방식의 액세스를 구성합니다.

AWS Proton 서비스 역할 설정

AWS Proton 솔루션의 여러 부분에 대해 생성할 수 있는 몇 가지 IAM 역할이 있습니다. IAM 콘솔을 사용하여 미리 생성하거나 AWS Proton 콘솔을 사용하여 생성할 수 있습니다.

가 사용자를 대신하여 다른 AWS 서비스, CloudFormation AWS CodeBuild, 다양한 컴퓨팅 및 스토리지 서비스에 AWS Proton API를 호출하여 리소스를 프로비저닝할 수 있도록 AWS Proton 허용하는 환경 역할을 생성합니다. 환경 또는 환경에서 실행 중인 서비스 인스턴스가 AWS관리형 프로비저닝을 사용하는 경우 AWS관리형 프로비저닝 역할이 필요합니다. 환경 또는 해당 서비스 인스턴스가 CodeBuild 프로비저닝을 사용하는 경우 CodeBuild 역할이 필요합니다. AWS Proton 환경 역할에 대한 자세한 내용은 섹션을 참조하세요the section called “IAM 역할”. 환경을 생성할 때 AWS Proton 콘솔을 사용하여 이러한 두 역할 중 하나에 대한 기존 역할을 선택하거나 관리자 권한이 있는 역할을 생성할 수 있습니다.

마찬가지로가 사용자를 대신하여 다른 서비스에 AWS Proton API를 호출하여 CI/CD 파이프라인을 프로비저닝할 수 있도록 파이프라인 AWS Proton 역할을 생성합니다. AWS Proton 파이프라인 역할에 대한 자세한 내용은 섹션을 참조하세요the section called “파이프라인 서비스 역할”. CI/CD 설정 구성에 대한 자세한 내용은 the section called “계정 CI/CD 파이프라인 설정 설정”을 참조하세요.

> **Note**
>
> AWS Proton 템플릿에서 정의할 리소스를 알 수 없으므로 콘솔을 사용하여 생성하는 역할은 광범위한 권한을 가지며 AWS Proton 파이프라인 서비스 역할과 AWS Proton 서비스 역할로 모두 사용할 수 있습니다. 프로덕션 배포의 경우 AWS Proton 파이프라인 서비스 역할과 환경 서비스 역할 모두에 대한 사용자 지정 정책을 생성하여 배포할 특정 리소스로 AWS Proton 권한 범위를 줄이는 것이 좋습니다. AWS CLI 또는 IAM을 사용하여 이러한 역할을 생성하고 사용자 지정할 수 있습니다. 자세한 내용은 에 대한 서비스 역할 AWS Proton 및 서비스 생성 섹션을 참조하세요.

를 사용하여 설정 AWS Proton

AWS CLI 를 사용하여 AWS Proton APIs 실행하려면 설치했는지 확인합니다. 항목을 설치하지 않은 경우 설정 AWS CLI을 참조하세요.

AWS Proton 특정 구성:

- 템플릿을 생성 및 관리하려면:
  - 템플릿 동기화 구성을 사용하는 경우 AWS CodeStar 연결을 설정합니다.

  - 그렇지 않으면 S3 버킷을 설정합니다.
- 인프라 프로비저닝:
  - 자체 관리형 프로비저닝의 경우 AWS CodeStar 연결을 설정해야 합니다.
- (선택 사항) 파이프라인을 프로비저하기:
  - AWS관리형 프로비저닝과 CodeBuild 기반 프로비저닝의 경우 파이프라인 역할을 설정합니다.
  - 자체 관리형 프로비저닝의 경우 파이프라인 리포지토리를 설정합니다.

프로비저닝 방법에 대한 자세한 내용은 the section called “AWS관리형 프로비저닝”을 참조하세요.

S3 버킷 설정

S3 버킷을 설정하려면 첫 번째 S3 버킷 생성의 지침에 따라 S3 버킷을 설정하세요. 가 입력을 검색할 AWS Proton 수 있는 버킷의 AWS Proton 에 입력을 배치합니다. 이러한 입력을 템플릿 번들이라고 합니다. 이 가이드의 다른 섹션의 자세한 내용은 이 가이드의 다른 섹션의 내용을 참조하세요.

AWS CodeStar 연결 설정

리포지토리 AWS Proton 에 연결하려면 타사 소스 코드 리포지토리에서 새 커밋이 생성될 때 파이프라인을 활성화하는 AWS CodeStar 연결을 생성합니다.

AWS Proton 는 연결을 사용하여 다음을 수행합니다.

- 리포지토리 소스 코드를 새로 커밋하면 서비스 파이프라인을 활성화합니다.
- 인프라형 코드 리포지토리에서 풀 리퀘스트를 생성합니다.
- 템플릿 중 하나를 변경하는 템플릿 리포지토리로 커밋이 푸시될 때마다(해당 버전이 아직 없는 경우) 새 템플릿 마이너 또는 메이저 버전을 만드세요.

CodeConnections를 사용하여 Bitbucket, GitHub, GitHub Enterprise 및 GitHub Enterprise Server 리포지토리에 연결할 수 있습니다. 자세한 내용은 AWS CodePipeline 사용 설명서의 CodeConnections를 참조하세요.

CodeStar 연결을 설정하기.

1. AWS Proton 콘솔을 엽니다.
2. 탐색 창에서 설정을 선택한 다음 리포지토리 연결을 선택하면 개발자 도구 설정의 연결 페이지로 이동합니다. 이 페이지에는 연결 목록이 표시됩니다.

3. 연결 생성을 선택하고 지침을 따릅니다.

계정 CI/CD 파이프라인 설정 설정

AWS Proton 는 애플리케이션 코드를 서비스 인스턴스에 배포하기 위한 CI/CD 파이프라인을 프로비저닝할 수 있습니다. 파이프라인 프로비저닝에 필요한 AWS Proton 설정은 파이프라인에 대해 선택한 프로비저닝 방법에 따라 다릅니다.

AWS관리형 프로비저닝과 CodeBuild 기반 프로비저닝의 경우, 파이프라인 역할을 설정합니다.

관리AWS형 프로비저닝 및 CodeBuild 프로비저닝을 사용하면 파이프라인을 AWS Proton 프로비저닝할 수 있습니다. 따라서 파이프라인 프로비저닝 권한을 제공하는 서비스 역할이 AWS Proton 필요합니다. 이 두 가지 프로비저닝 방법은 각각 고유한 서비스 역할을 사용합니다. 이러한 역할은 모든 AWS Proton 서비스 파이프라인에서 공유되며 계정 설정에서 한 번 구성합니다.

콘솔을 사용하여 파이프라인 서비스 역할 생성하기

1. AWS Proton 콘솔을 엽니다.
2. 탐색 창에서 설정을 선택한 다음 계정 설정을 선택합니다.
3. 계정 CI/CD 설정 페이지에서 구성을 선택합니다.
4. 다음 중 하나를 수행하세요.
   - 파이프라인 서비스 역할을 AWS Proton 생성하려면

     [파이프라인의 AWS관리형 프로비저닝을 활성화하려면] 계정 설정 구성 페이지의 AWS관리형 프로비저닝 파이프라인 역할 단원에서,

     a. 새 서비스 역할을 선택합니다.
     b. 역할의 이름을 입력합니다(예: **`myProtonPipelineServiceRole`**).
     c. 계정에 관리 권한이 있는 AWS Proton 역할을 생성하는 데 동의하려면 확인란을 선택합니다.

     [CodeBuild 기반 파이프라인 프로비저닝을 활성화하려면] 계정 설정 구성 페이지의 CodeBuild 파이프라인 역할 단원에서 기존 서비스 역할을 선택하고 CloudFormation 파이프라인 역할 단원에서 생성한 서비스 역할을 선택합니다. 또는 CloudFormation 파이프라인 역할을 할당하지 않은 경우 이전 3단계를 반복하여 새 서비스 역할을 생성합니다.

- 기존 파이프라인 서비스 역할을 선택하기

  [파이프라인의 AWS관리형 프로비저닝을 활성화하려면 계정 설정 구성 페이지의 AWS관리형 프로비저닝 파이프라인 역할 단원에서 기존 서비스 역할을 선택하고 AWS 계정의 서비스 역할을 선택합니다.

  [파이프라인의 CodeBuild 프로비저닝을 활성화하려면] 계정 설정 구성 페이지의 CodeBuild 파이프라인 프로비저닝 역할 섹션에서 기존 서비스 역할을 선택하고 AWS 계정에서 서비스 역할을 선택합니다.

5. 변경 사항 저장을 선택합니다.

   새 파이프라인 서비스 역할은 계정 설정 페이지에 표시됩니다.

자체 관리형 프로비저닝 - 파이프라인 리포지토리를 설정합니다.

자체 관리형 프로비저닝을 사용하면는 설정한 프로비저닝 리포지토리에 풀 요청(PR)을 AWS Proton 전송하고 자동화 코드는 파이프라인 프로비저닝을 담당합니다. 따라서 파이프라인을 프로비저닝하는 데 서비스 역할이 필요하지 AWS Proton 않습니다. 대신 등록된 프로비저닝 리포지토리가 필요합니다. 리포지토리의 자동화 코드는 프로비저닝 파이프라인에 대한 권한을 제공하는 적절한 역할을 맡아야 합니다.

콘솔을 사용하여 파이프라인 프로비저닝 리포지토리를 등록하려면

1. CI/CD 파이프라인 프로비저닝 리포지토리를 아직 만들지 않았다면 생성합니다. 자체 관리형 프로비저닝의 파이프라인에 대한 자세한 내용은 the section called “자체 관리형 프로비저닝”을 참조하세요.
2. 탐색 창에서 설정을 선택한 다음 계정 설정을 선택합니다.
3. 계정 CI/CD 설정 페이지에서 구성을 선택합니다.
4. 계정 설정 구성 페이지의 CI/CD 파이프라인 리포지토리 단원에서:

   a. 새 리포지토리를 선택한 다음 리포지토리 제공자 중 하나를 선택합니다.

   b. CodeStar 연결의 경우 연결 중 하나를 선택합니다.

> **Note**
>
> 관련 리포지토리 공급자 계정에 아직 연결되지 않은 경우 새 CodeStar 연결 추가를 선택하고 연결 생성 프로세스를 완료한 다음 CodeStar 연결 메뉴 옆에 있는 새로 고침 버튼을 선택합니다. 이제 메뉴에서 새 연결을 선택할 수 있을 것입니다.

c. 리포지토리 이름에서 파이프라인 프로비저닝 리포지토리를 선택합니다. 드롭다운 메뉴에는 제공자 계정의 리포지토리 목록이 표시됩니다.

d. 브랜치 이름에서 리포지토리 브랜치 중의 하나를 선택합니다.

5. 변경 사항 저장을 선택합니다.

파이프라인 리포지토리 역할은 계정 설정 페이지에 표시됩니다.

설정 AWS CLI

AWS CLI 를 사용하여 AWS Proton API를 호출하려면 최신 버전의를 설치했는지 확인합니다 AWS CLI. 자세한 내용은 AWS Command Line Interface 사용 설명서에서 AWS CLI시작하기를 참조하세요. 그런 다음 AWS CLI 에서 사용을 시작하려면 섹션을 AWS Proton참조하세요the section called “CLI 시작하기”.

시작하기 AWS Proton

시작하기 전에를 사용하도록 설정하고 시작하기 사전 조건을 충족했는지 AWS Proton 확인합니다.

다음 경로 중 하나 이상을 AWS Proton 선택하여를 시작합니다.

- 설명서 링크를 통해 가이드 예제 콘솔 또는 CLI 워크플로를 따르십시오.
- 안내된 예제 콘솔 워크플로를 따라 실행하세요.
- 안내형 예제 AWS CLI 워크플로를 실행합니다.

주제

사전 조건

사용을 시작하기 전에 다음 사전 조건이 충족되는지 AWS Proton확인합니다. 자세한 내용은 설정 단원을 참조하십시오.

- 관리자 권한이 있는 IAM 계정이 있습니다. 자세한 내용은 IAM으로 설정 단원을 참조하십시오.
- AWS Proton 서비스 역할이 있고 AWS Proton 파이프라인 서비스 역할이 계정에 연결되어 있습니다. 자세한 내용은 AWS Proton 서비스 역할 설정 및 에 대한 서비스 역할 AWS Proton 섹션을 참조하세요.
- AWS CodeStar 연결이 있습니다. 자세한 내용은 AWS CodeStar 연결 설정 단원을 참조하십시오.
- CloudFormation 템플릿 생성과 Jinja 파라미터화에 익숙합니다. 자세한 내용은 CloudFormation 사용 설명서 및 Jinja 웹 사이트의 What is CloudFormation?를 참조하세요.
- AWS 인프라 서비스에 대한 실무 지식이 있습니다.
- 에 로그인했습니다 AWS 계정.

워크플로 시작하기

예제 단계와 링크를 따라 템플릿 번들을 만들고, 템플릿을 만들고 등록하고, 환경 및 서비스를 만드는 방법을 알아보세요.

시작하기 전에 AWS Proton 서비스 역할을 생성했는지 확인합니다.

서비스 템플릿에 AWS Proton 서비스 파이프라인이 포함되어 있는 경우 AWS CodeStar 연결 및 AWS Proton 파이프라인 서비스 역할을 생성했는지 확인합니다.

자세한 내용은 AWS Proton 서비스 API 참조를 참조하세요.

예제: 워크플로 시작하기

1. AWS Proton 입력 및 출력에 AWS Proton 작동 방식 대한 상위 수준 보기는의 다이어그램을 참조하세요.
2. 환경 번들 및 서비스 템플릿 번들을 생성합니다.
   a. 입력 파라미터를 식별합니다.
   b. 스키마 파일을 생성합니다.
   c. 코드형 인프라(IaC) 파일을 생성합니다.
   d. 템플릿 번들을 마무리하려면 매니페스트 파일을 만들고 IaC 파일, 매니페스트 파일, 스키마 파일을 디렉터리에 마무리합니다.
   e. 템플릿 번들에 액세스할 수 있도록 합니다 AWS Proton.
3. 환경 템플릿 버전을 생성하고에 등록합니다 AWS Proton.

   콘솔을 사용하여 템플릿을 만들고 등록하면 템플릿 버전이 자동으로 생성됩니다.

   AWS CLI 를 사용하여 템플릿을 생성하고 등록하는 경우:

   a. 환경 템플릿을 생성합니다.
   b. 환경 템플릿 버전을 생성합니다.

   자세한 내용은 AWS Proton API reference의 CreateEnvironmentTemplate 및 CreateEnvironmentTemplateVersion를 참조하세요.
4. 환경 템플릿을 게시하여 사용할 수 있도록 하세요.

자세한 내용은 AWS Proton API reference에서 UpdateEnvironmentTemplateVersion을 참조하세요.

5. 환경을 만들려면 게시된 환경 템플릿 버전을 선택하고 필요한 입력 값을 제공하세요.

   자세한 내용을 알아보려면 AWS Proton API 참조의 CreateEnvironment을 참조하세요.

6. 를 사용하여 서비스 템플릿 버전을 생성하고 등록합니다 AWS Proton.

   콘솔을 사용하여 템플릿을 만들고 등록하면 템플릿 버전이 자동으로 생성됩니다.

   AWS CLI 를 사용하여 템플릿을 생성하고 등록하는 경우:

   a. 서비스 템플릿을 생성합니다.
   b. 서비스 템플릿 버전을 생성합니다.

   자세한 내용은 AWS Proton API 참조의 CreateServiceTemplate 및 CreateServiceTemplateVersion을 참조하세요.

7. 서비스 템플릿을 게시하여 사용할 수 있도록 하세요.

   에 대한 자세한 내용은 AWS Proton API Reference의 UpdateServiceTemplateVersion를 참조하세요.

8. 환경을 만들려면 게시된 서비스 템플릿 버전을 선택하고 필요한 입력 값을 제공하세요.

   자세한 내용은 AWS Proton API 참조의 CreateService 단원을 참조하세요.

시작하기 AWS Management Console

시작하기 AWS Proton

- 환경 템플릿을 생성하고 봅니다.
- 방금 만든 환경 템플릿을 사용하는 서비스 템플릿을 만들고 보고 게시하세요.
- 환경 및 서비스 만들기 (선택 사항).
- 서비스 템플릿, 환경 템플릿, 환경 및 서비스를 삭제합니다(생성된 경우).

1단계: AWS Proton 콘솔 열기

- AWS Proton 콘솔을 엽니다

2단계: 예제 템플릿 사용 준비

1. Github에 코드스타 커넥션을 생성하고 연결 이름을 my-proton-connection으로 지정합니다.
2. https://github.com/aws-samples/aws-proton-cloudformation-sample-templates로 이동합니다.
3. Github 계정에서 리포지토리의 포크를 만드세요.

3단계: 환경 템플릿을 생성합니다.

탐색 창에서 환경 템플릿을 선택합니다.

1. 환경 템플릿 페이지에서 환경 템플릿 생성을 선택합니다.
2. 환경 템플릿 생성 페이지의 템플릿 옵션 단원에서 새 환경을 프로비저닝하기 위한 템플릿 생성을 선택합니다.
3. 템플릿 번들 소스 단원에서 Git에서 템플릿 번들 동기화를 선택합니다.
4. 템플릿 정의 리포지토리 단원에서 연결된 Git 리포지토리 선택을 선택합니다.
5. 리포지토리 목록에서 my-proton-connection을 선택합니다.
6. 브랜치 목록에서 메인을 선택합니다.
7. Proton 환경 템플릿 세부 정보 단원에서.
   a. **`fargate-env`**에서 템플릿 이름을 입력합니다.
   b. 환경 템플릿 표시 이름을 **`My Fargate Environment`**로 입력합니다.
   c. (선택 사항) 환경 템플릿에 대한 설명을 입력합니다.
8. (선택 사항) 태그 단원에서 새 태그 추가를 선택하고 키와 값을 입력하여 고객 관리형 태그를 생성합니다.
9. 환경 템플릿 생성을 선택합니다.

   이제 새 환경 템플릿의 상태 및 세부 정보를 표시하는 새 페이지가 열립니다. 이러한 세부 정보에는 AWS 및 고객 관리형 태그 목록이 포함됩니다.는 리소스를 생성할 때 AWS 관리형 태그를 AWS Proton 자동으로 생성합니다 AWS Proton . 자세한 내용은 AWS Proton 리소스 및 태그 지정 단원을 참조하십시오.
10. 새 환경 템플릿 상태의 상태는 초안 상태에서 시작됩니다. 사용자 및 `proton:CreateEnvironment` 권한이 있는 다른 사용자가 보고 액세스할 수 있습니다. 다음 단계에 따라 다른 사람이 템플릿을 사용할 수 있도록 합니다.

11. 템플릿 버전 단원에서 방금 만든 템플릿의 마이너 버전 왼쪽에 있는 라디오 버튼 (1.0) 을 선택합니다. 또는 정보 알림 배너에서 게시를 선택하고 다음 단계를 건너뛸 수도 있습니다.
12. 템플릿 버전 단원에서 게시를 선택합니다.
13. 템플릿 상태가 게시됨으로 변경됩니다. 템플릿의 최신 버전이므로 권장 버전입니다.
14. 탐색 창에서 환경 템플릿을 선택합니다.

    새 페이지에는 템플릿 세부 정보와 함께 환경 템플릿 목록이 표시됩니다.

4단계: 서비스 템플릿 생성

서비스 템플릿을 생성합니다.

1. 탐색 창에서 서비스 템플릿를 선택합니다.
2. 서비스 템플릿 페이지에서 서비스 템플릿 생성을 선택합니다.
3. 서비스 템플릿 생성 페이지의 템플릿 번들 소스 단원에서 Git에서 템플릿 번들 동기화를 선택합니다.
4. 템플릿 단원에서 연결된 Git 리포지토리 선택을 선택합니다.
5. 리포지토리 목록에서 my-proton-connection을 선택합니다.
6. 브랜치 목록에서 메인을 선택합니다.
7. Proton 서비스 템플릿 세부 정보 단원에서.

    a. 서비스 템플릿 이름을 **`backend-fargate-svc`**로 입력합니다.

    b. 서비스 템플릿 표시 이름을 **`My Fargate Service`**로 입력합니다.

    c. (선택 사항) 서비스 템플릿에 대한 설명을 입력합니다.
8. 호환 가능한 환경 템플릿 단원에서.

    - 환경 템플릿 My Fargate Environment의 왼쪽에 있는 확인란을 선택하여 새 서비스 템플릿과 호환되는 환경 템플릿을 선택합니다.
9. 암호화 설정에 기본 설정을 사용합니다.
10. 파이프라인 정의 단원에서

    - 이 템플릿에는 CI/CD 파이프라인 버튼이 포함되어 있음 버튼을 선택한 상태로 유지하세요.
11. 서비스 템플릿 생성을 선택합니다.

이제 AWS 및 고객 관리형 태그 목록을 포함하여 새 서비스 템플릿의 상태 및 세부 정보를 표시하는 새 페이지가 생겼습니다.

12. 새 서비스 템플릿 상태의 상태는 초안 상태에서 시작됩니다. 관리자만 보고 액세스할 수 있습니다. 개발자가 서비스 템플릿을 사용할 수 있게 하려면 다음 단계를 따르세요.
13. 템플릿 버전 단원에서 방금 만든 템플릿의 마이너 버전 왼쪽에 있는 라디오 버튼 (1.0) 을 선택합니다. 또는 정보 알림 배너에서 게시를 선택하고 다음 단계를 건너뛸 수도 있습니다.
14. 템플릿 버전 단원에서 게시를 선택합니다.
15. 템플릿 상태가 게시됨으로 변경됩니다.

    서비스 템플릿의 첫 번째 마이너 버전이 게시되어 개발자가 사용할 수 있습니다. 템플릿의 최신 버전이므로 권장 버전입니다.
16. 탐색 창에서 서비스 템플릿를 선택합니다.

    새 페이지에 서비스 템플릿 목록과 세부 정보가 표시됩니다.

5단계: 환경 생성

탐색 창에서 환경을 선택합니다.

1. 환경 생성을 선택합니다.
2. 환경 템플릿 선택 페이지에서 방금 만든 템플릿을 선택합니다. 이름은 My Fargate 환경입니다. 그런 다음 구성을 선택합니다.
3. 환경 구성 페이지의 프로비저닝 단원에서 AWS Proton를 통한 프로비저닝을 선택합니다.
4. 배포 계정 단원에서 이 AWS 계정를 선택합니다.
5. 환경 설정 단원에서 환경 이름 **`my-fargate-environment`**로 입력합니다.
6. 환경 역할 섹션에서 새 서비스 역할을 선택하거나 서비스 역할을 이미 생성한 경우 기존 서비스 역할을 AWS Proton 선택합니다.

   a. 새 역할을 만들려면 새 서비스 역할을 선택합니다.

      i. 환경 역할 이름을 **`MyProtonServiceRole`**로 입력합니다.

      ii. 확인란을 선택하여 계정에 대한 관리 권한이 있는 AWS Proton 서비스 역할을 생성하는 데 동의합니다.

   b. 기존 역할을 사용하려면 기존 서비스 역할을 선택합니다.

- 환경 역할 이름 드롭다운 필드에서 역할을 선택합니다.

7. 다음을 선택합니다.
8. 사용자 지정 설정 구성 페이지에서 기본값을 사용합니다.
9. 다음을 선택하고 입력 내용을 검토합니다.
10. 생성(Create)을 선택합니다.

    환경 세부 정보 및 상태와 환경에 대한 AWS 관리형 태그 및 고객 관리형 태그를 확인합니다.
11. 탐색 창에서 환경을 선택합니다.

    새 페이지에는 환경 목록이 상태 및 기타 환경 세부 정보와 함께 표시됩니다.

6단계: 선택 사항 - 서비스 생성 및 애플리케이션 배포

1. AWS Proton 콘솔을 엽니다.
2. 탐색 창에서 서비스를 선택합니다.
3. 서비스 페이지에서 서비스 생성을 선택합니다.
4. 서비스 템플릿 선택 페이지에서 템플릿 카드의 오른쪽 상단에 있는 라디오 버튼을 선택하여 My Fargate 서비스 템플릿을 선택합니다.
5. 페이지 오른쪽 하단에서 구성을 선택합니다.
6. 서비스 구성 페이지의 서비스 설정 섹션에 서비스 이름을 **my-service**으로 입력합니다.
7. (선택 사항) 서비스 설명을 입력합니다.
8. 서비스 리포지토리 설정 단원에서,

    a. CodeStar 연결의 경우 목록에서 연결을 선택합니다.

    b. 리포지토리 이름의 경우 목록에서 소스 코드 리포지토리의 이름을 선택합니다.

    c. 브랜치 이름의 경우 목록에서 소스 코드 저장소 브랜치의 이름을 선택합니다.
9. (선택 사항) 태그 단원에서 새 태그 추가를 선택하고 키와 값을 입력하여 고객 관리형 태그를 생성합니다. 그리고 다음을 선택합니다.
10. 사용자 지정 설정 구성 페이지, 서비스 인스턴스 섹션, 새 인스턴스 단원에서 다음 단계에 따라 서비스 인스턴스 파라미터에 사용자 지정 값을 제공하세요.

    a. 인스턴스 이름 **my-app-service**을 입력합니다.

    b. 서비스 인스턴스의 환경 **my-fargate-environment**을 선택합니다.

c. 나머지 인스턴스 파라미터는 기본값을 유지합니다.

d. 파이프라인 입력의 기본값을 그대로 유지합니다.

e. 다음을 선택하고 입력 내용을 검토합니다.

f. 생성 를 선택하고 서비스 상태 및 세부 정보를 확인합니다.

11. 서비스 세부 정보 페이지에서 개요 및 파이프라인 탭을 선택하여 서비스 인스턴스 및 파이프라인의 상태를 확인합니다. 이 페이지에서 AWS 및 고객 관리형 태그를 볼 수도 있습니다. AWS Proton는 AWS 관리형 태그를 자동으로 생성합니다. 태그 관리를 선택하여 고객 관리 태그를 생성하고 수정합니다. 태그 지정에 대한 자세한 내용은 AWS Proton 리소스 및 태그 지정 섹션을 참조하세요.
12. 서비스가 활성화되면 개요 탭의 서비스 인스턴스 단원에서 서비스 인스턴스의 이름인 my-app-service를 선택합니다.

    이제 서비스 인스턴스 세부 정보 페이지가 나타납니다.
13. 애플리케이션을 보려면 출력 단원에서 ServiceEndpoint 링크를 브라우저에 복사하세요.

    웹 페이지에 AWS Proton 그래픽이 표시됩니다.
14. 서비스를 만든 후 탐색 창에서 서비스를 선택하여 서비스 목록을 확인합니다.

7단계: 정리

1. AWS Proton 콘솔을 엽니다.
2. 서비스 삭제 (생성한 경우)

    a. 탐색 창에서 서비스를 선택합니다.

    b. 서비스 페이지에서 서비스 이름 my-service를 선택합니다.

    이제 my-service의 서비스 세부 정보 페이지가 나타납니다.

    c. 페이지 오른쪽 상단 모서리의 작업 메뉴에서 삭제를 선택하세요.

    d. 삭제 작업을 확인하라는 모달이 표시됩니다.

    e. 지침을 따르고 예, 삭제를 선택합니다.
3. 환경을 삭제합니다.

    a. 탐색 창에서 환경을 선택합니다.

    b. 환경 페이지에서 방금 만든 환경의 왼쪽에 있는 라디오 버튼을 선택합니다.

    c. 작업을 선택한 다음 삭제를 선택합니다.

d. 삭제 작업을 확인하라는 모달이 표시됩니다.

e. 지침을 따르고 예, 삭제를 선택합니다.

4. 서비스 템플릿 삭제

a. 탐색 창에서 서비스 템플릿를 선택합니다.

b. 서비스 템플릿 페이지에서 서비스 템플릿 my-svc-template 왼쪽에 있는 라디오 버튼을 선택합니다.

c. 작업을 선택한 다음 삭제를 선택합니다.

d. 삭제 작업을 확인하라는 모달이 표시됩니다.

e. 지침을 따르고 예, 삭제를 선택합니다. 이렇게 하면 서비스 템플릿과 해당 버전이 모두 삭제됩니다.

5. 환경 템플릿을 삭제합니다.

a. 탐색 창에서 환경 템플릿을 선택합니다.

b. 환경 템플릿 페이지에서 my-env-template 왼쪽에 있는 라디오 버튼을 선택합니다.

c. 작업을 선택한 다음 삭제를 선택합니다.

d. 삭제 작업을 확인하라는 모달이 표시됩니다.

e. 지침을 따르고 예, 삭제를 선택합니다. 이렇게 하면 환경 템플릿과 해당 버전이 모두 삭제됩니다.

6. Codestar 연결을 삭제합니다.

시작하기 AWS CLI

AWS Proton 사용을 시작하려면이 자습서를 AWS CLI따르세요. 이 자습서에서는를 기반으로 하는 퍼블릭 대면 로드 밸런싱 AWS Proton 서비스를 보여줍니다 AWS Fargate. 또한 이 튜토리얼에서는 표시된 이미지가 있는 정적 웹사이트를 배포하는 CI/CD 파이프라인을 제공합니다.

시작하기 전에 제대로 설정했는지 확인합니다. 자세한 내용은 the section called “사전 조건” 단원을 참조하세요.

1단계: 환경 템플릿을 등록합니다.

이 단계에서는 관리자로서 두 개의 퍼블릭/프라이빗 서브넷이 있는 Elastic Container Service(ECS) 클러스터와 Virtual Private Cloud(VPC)가 포함된 예제 환경 템플릿을 등록합니다.

환경 템플릿을 등록하려면

1. AWS Proton 샘플 CloudFormation 템플릿 리포지토리를 GitHub 계정 또는 조직에 포크합니다. 이 리포지토리에는 이 튜토리얼에서 사용하는 환경 및 서비스 템플릿이 포함되어 있습니다.

   그런 다음 포크된 리포지토리를에 등록합니다 AWS Proton. 자세한 내용은 the section called “리포지토리 링크 생성” 단원을 참조하십시오.
2. 환경 템플릿을 생성합니다.

   환경 템플릿 리소스는 환경 템플릿 버전을 추적합니다.

   ```
   $ aws proton create-environment-template \
     --name "fargate-env" \
     --display-name "Public VPC Fargate" \
     --description "VPC with public access and ECS cluster"
   ```
3. 템플릿 동기화 구성을 생성합니다.

   AWS Proton 는 리포지토리와 환경 템플릿 간의 동기화 관계를 설정합니다. 그런 다음 DRAFT 상태의 템플릿 버전 1.0을 생성합니다.

   ```
   $ aws proton create-template-sync-config \
     --template-name "fargate-env" \
     --template-type "ENVIRONMENT" \
     --repository-name "your-forked-repo" \
     --repository-provider "GITHUB" \
     --branch "your-branch" \
     --subdirectory "environment-templates/fargate-env"
   ```
4. 환경 템플릿 버전이 성공적으로 등록될 때까지 기다리십시오.

   이 명령이 종료 상태인 0로 반환되면 버전 등록이 완료된 것입니다. 이 기능은 스크립트에서 다음 단계에서 명령을 성공적으로 실행할 수 있도록 하는 데 유용합니다.

   ```
   $ aws proton wait environment-template-version-registered \
     --template-name "fargate-env" \
     --major-version "1" \
     --minor-version "0"
   ```
5. 환경 템플릿 버전을 게시하여 환경 생성에 사용할 수 있도록 합니다.

   ```
   $ aws proton update-environment-template-version \
   ```

```
  --template-name "fargate-env" \
  --major-version "1" \
  --minor-version "0" \
  --status "PUBLISHED"
```

2단계: 서비스 템플릿 등록하기

이 단계에서는 관리자가 로드 밸런서 뒤에 있는 ECS Fargate 서비스를 프로비저닝하는 데 필요한 모든 리소스와 AWS CodePipeline를 사용하는 CI/CD 파이프라인이 포함된 예제 서비스 템플릿을 등록합니다.

서비스 템플릿을 등록하려면

1. 서비스 템플릿을 생성합니다.

   서비스 템플릿 리소스는 서비스 템플릿 버전을 추적합니다.

   ```
   $ aws proton create-service-template \
     --name "load-balanced-fargate-svc" \
     --display-name "Load balanced Fargate service" \
     --description "Fargate service with an application load balancer"
   ```

2. 템플릿 동기화 구성을 생성합니다.

   AWS Proton 는 리포지토리와 서비스 템플릿 간의 동기화 관계를 설정합니다. 그런 다음 DRAFT 상태의 템플릿 버전 1.0을 생성합니다.

   ```
   $ aws proton create-template-sync-config \
     --template-name "load-balanced-fargate-svc" \
     --template-type "SERVICE" \
     --repository-name "your-forked-repo" \
     --repository-provider "GITHUB" \
     --branch "your-branch" \
     --subdirectory "service-templates/load-balanced-fargate-svc"
   ```

3. 서비스 템플릿 버전이 성공적으로 등록될 때까지 기다립니다.

   이 명령이 종료 상태인 0로 반환되면 버전 등록이 완료된 것입니다. 이 기능은 스크립트에서 다음 단계에서 명령을 성공적으로 실행할 수 있도록 하는 데 유용합니다.

   ```
   $ aws proton wait service-template-version-registered \
   ```

```
  --template-name "load-balanced-fargate-svc" \
  --major-version "1" \
  --minor-version "0"
```

4. 서비스 템플릿 버전을 게시하여 서비스 생성에 사용할 수 있도록 합니다.

```
$ aws proton update-service-template-version \
  --template-name "load-balanced-fargate-svc" \
  --major-version "1" \
  --minor-version "0" \
  --status "PUBLISHED"
```

3단계: 환경 배포

이 단계에서는 관리자가 AWS Proton 환경 템플릿에서 환경을 인스턴스화합니다.

환경 배포하려면

1. 등록한 환경 템플릿의 예제 사양 파일을 가져옵니다.

   템플릿 예제 리포지토리에서 `environment-templates/fargate-env/spec/spec.yaml` 파일을 다운로드할 수 있습니다. 또는 전체 리포지토리를 로컬로 가져와서 `environment-templates/fargate-env` 디렉터리에서 create-environment 명령을 실행할 수도 있습니다.

2. 환경을 생성합니다.

   AWS Proton 는 환경 사양에서 입력 값을 읽고, 환경 템플릿과 결합하고, AWS Proton 서비스 역할을 사용하여 AWS 계정의 환경 리소스를 프로비저닝합니다.

```
$ aws proton create-environment \
  --name "fargate-env-prod" \
  --template-name "fargate-env" \
  --template-major-version 1 \
  --proton-service-role-arn "arn:aws:iam::123456789012:role/AWS ProtonServiceRole"
 \
  --spec "file://spec/spec.yaml"
```

3. 환경이 성공적으로 배포될 때까지 기다리세요.

```
$ aws proton wait environment-deployed --name "fargate-env-prod"
```

4단계: 서비스 배포 [애플리케이션 개발자]

이전 단계에서 관리자는 서비스 템플릿을 등록 및 게시하고 환경을 배포했습니다. 애플리케이션 개발자는 이제 AWS Proton 서비스를 생성하여 AWS Proton 환경에 배포할 수 있습니다.

서비스를 배포하려면

1. 관리자가 등록한 서비스 템플릿의 예제 사양 파일을 가져오십시오.

   템플릿 예제 리포지토리에서 `service-templates/load-balanced-fargate-svc/spec/spec.yaml` 파일을 다운로드할 수 있습니다. 또는 전체 리포지토리를 로컬로 가져와서 `service-templates/load-balanced-fargate-svc` 디렉터리에서 create-service 명령을 실행할 수도 있습니다.
2. AWS Proton 샘플 서비스 리포지토리를 GitHub 계정 또는 조직으로 포크합니다. 이 리포지토리에는 이 튜토리얼에서 사용하는 애플리케이션 소스 코드가 포함되어 있습니다.
3. 서비스를 생성합니다.

   AWS Proton 는 서비스 사양에서 입력 값을 읽고, 이를 서비스 템플릿과 결합하고, 사양에 AWS 지정된 환경의 계정에 서비스 리소스를 프로비저닝합니다. AWS CodePipeline 파이프라인은 명령에서 지정한 리포지토리에서 애플리케이션 코드를 배포합니다.

   ```
   $ aws proton create-service \
     --name "static-website" \
     --repository-connection-arn \
       "arn:aws:codestar-connections:us-east-1:123456789012:connection/your-codestar-connection-id" \
     --repository-id "your-GitHub-account/aws-proton-sample-services" \
     --branch-name "main" \
     --template-major-version 1 \
     --template-name "load-balanced-fargate-svc" \
     --spec "file://spec/spec.yaml"
   ```
4. 서비스가 성공적으로 배포될 때까지 기다립니다.

   ```
   $ aws proton wait service-created --name "static-website"
   ```
5. 출력을 검색하고 새 웹 사이트를 확인합니다.

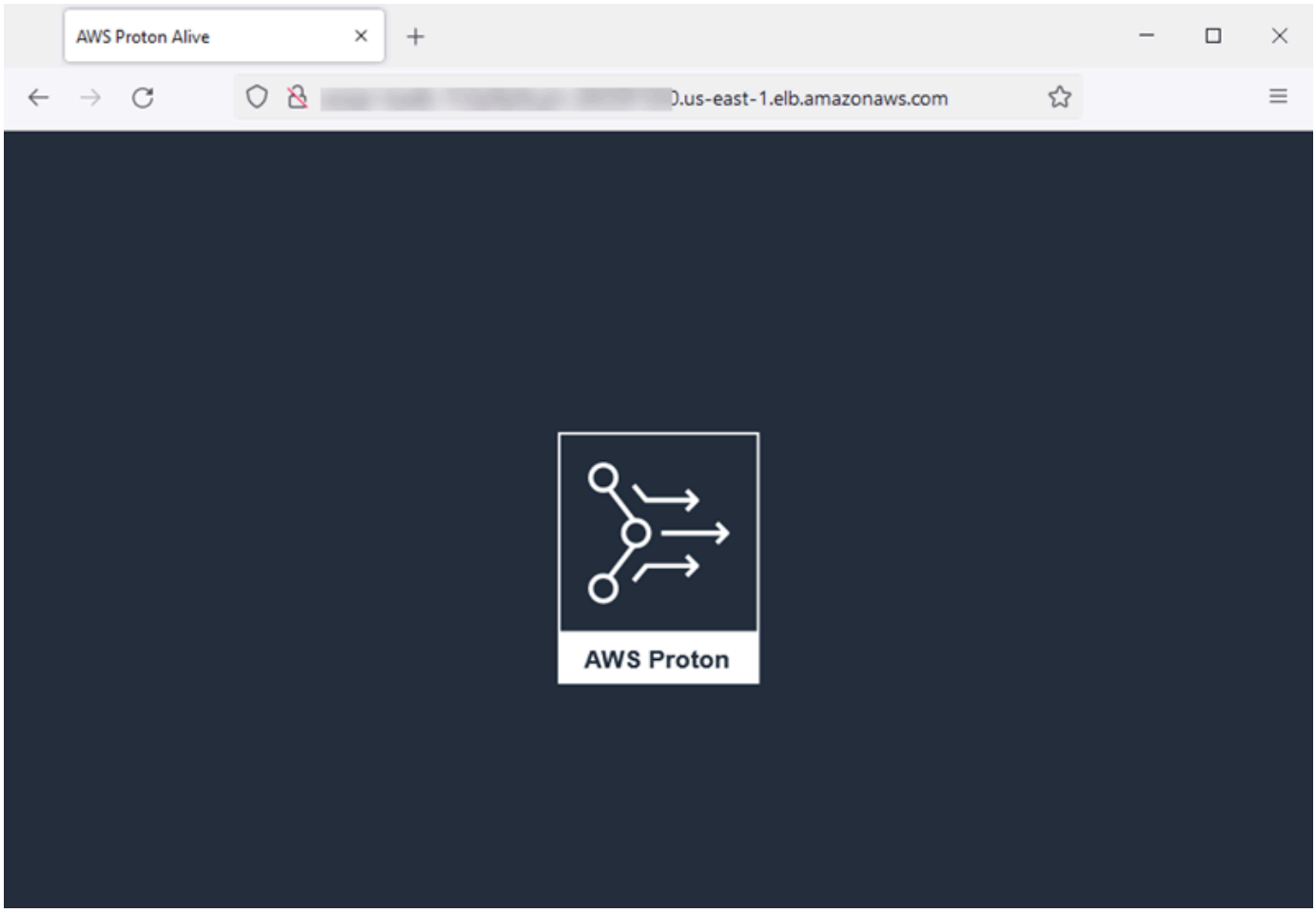
   다음 명령을 실행합니다.

```
$ aws proton list-service-instance-outputs \
  --service-name "static-website" \
  --service-instance-name load-balanced-fargate-svc-prod
```

명령 출력은 은 다음과 유사해야 합니다.

```
{
    "outputs": [
        {
            "key": "ServiceURL",
            "valueString": "http://your-service-endpoint.us-
east-1.elb.amazonaws.com"
        }
    ]
}
```

ServiceURL 인스턴스 출력 값은 새 서비스 웹 사이트의 엔드포인트입니다. 브라우저를 사용하여 탐색하세요. 정적 페이지에 다음 그래픽이 표시되어야 합니다:



5단계: 정리(선택 사항)

이 단계에서는이 자습서의 일부로 생성한 AWS 리소스를 탐색하고 이러한 리소스와 관련된 비용을 절감하기 위해 리소스를 삭제합니다.

튜토리얼 리소스 삭제

1. 서비스를 다시 시작하려면 다음 명령을 실행합니다.

```
$ aws proton delete-service --name "static-website"
```

2. 환경을 삭제하려면 다음 명령을 실행합니다.

```
$ aws proton delete-environment --name "fargate-env-prod"
```

3. 서비스 템플릿을 삭제하려면 다음 명령을 실행합니다.

```
$ aws proton delete-template-sync-config \
  --template-name "load-balanced-fargate-svc" \
  --template-type "SERVICE"
$ aws proton delete-service-template --name "load-balanced-fargate-svc"
```

4. 환경 템플릿을 삭제하려면 다음 명령을 실행합니다:

```
$ aws proton delete-template-sync-config \
  --template-name "fargate-env" \
  --template-type "ENVIRONMENT"
$ aws proton delete-environment-template --name "fargate-env"
```

AWS Proton 템플릿 라이브러리

AWS Proton 팀은 GitHub에 템플릿 예제 라이브러리를 유지합니다. 라이브러리에는 여러 가지 일반적인 환경 및 애플리케이션 인프라 시나리오에 대한 코드형 인프라(IaC) 파일 예제가 포함되어 있습니다.

템플릿 라이브러리는 두 개의 GitHub 리포지토리에 저장됩니다.

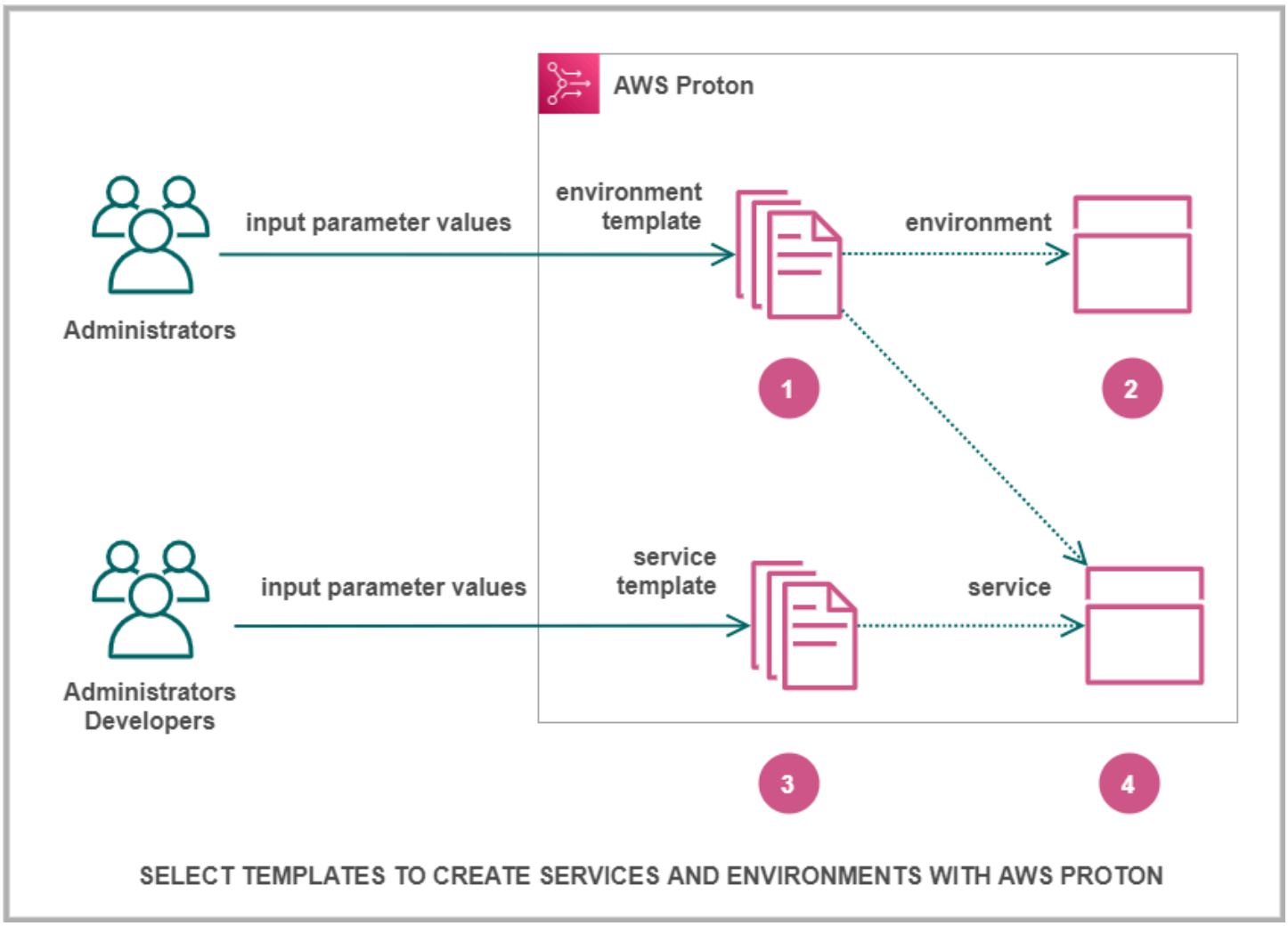
- aws-proton-cloudformation-sample-template — Jinja와 함께 AWS CloudFormation를 IaC 언어로 사용하는 템플릿 번들의 예. 이러한 예제를 AWS관리형 프로비저닝 환경에 사용할 수 있습니다.
- aws-proton-terraform-sample-templates — Jinja와 함께 Terraform를 IaC 언어로 사용하는 템플릿 번들의 예. 이러한 예제를 자체 관리형 프로비저닝 환경에 사용할 수 있습니다.

각 리포지토리에는 리포지토리의 컨텐츠 및 구조에 대한 전체 정보가 들어 있는 README 파일이 있습니다. 각 예시에는 템플릿이 다루는 사용 사례, 예시의 아키텍처 및 템플릿이 사용하는 입력 파라미터에 대한 정보가 포함되어 있습니다.

라이브러리의 리포지토리 중 하나를 GitHub 계정으로 포크하여 이 라이브러리의 템플릿을 직접 사용할 수 있습니다. 또는 이 예제를 환경 및 서비스 템플릿 개발을 위한 출발점으로 사용할 수도 있습니다.

AWS Proton 작동 방식

AWS Proton를 사용하면 환경을 프로비저닝한 다음 해당 환경에서 실행되는 서비스를 프로비저닝할 수 있습니다. 환경 및 서비스는 각각 AWS Proton 버전이 지정된 템플릿 라이브러리에서 선택한 환경 및 서비스 템플릿을 기반으로 합니다.



1

관리자는 환경 템플릿을 선택할 때 필요한 입력 파라미터 값을 AWS Proton제공합니다.

2

AWS Proton 는 환경 템플릿과 파라미터 값을 사용하여 환경을 프로비저닝합니다.

3

개발자 또는 관리자는를 사용하여 서비스 템플릿을 선택할 때 필요한 입력 파라미터 값을 AWS Proton 제공합니다. 또한 애플리케이션 또는 서비스를 배포할 환경을 선택합니다.

4

AWS Proton 는 서비스 템플릿과 서비스 및 선택한 환경 파라미터 값을 모두 사용하여 서비스를 프로비저닝합니다.

입력 파라미터 값을 제공하여 템플릿을 재사용 및 여러 사용 사례, 응용 프로그램 또는 서비스에 맞게 사용자 지정할 수 있습니다.

이 작업을 수행하려면 환경 또는 서비스 템플릿 번들을 만들어 등록된 환경 또는 서비스 템플릿에 각각 업로드해야 합니다.

템플릿 번들에는 환경 또는 서비스를 프로비저닝하는 데 AWS Proton 필요한 모든 것이 포함되어 있습니다.

환경 또는 서비스 템플릿을 만들 때는 AWS Proton 이 환경 또는 서비스를 프로비저닝하는 데 사용하는 파라미터화된 코드형 인프라(IaC) 파일이 포함된 템플릿 번들을 업로드합니다.

환경 또는 서비스를 만들거나 업데이트할 환경 또는 서비스 템플릿을 선택할 때는 템플릿 번들 IAC 파일 파라미터에 값을 제공합니다.

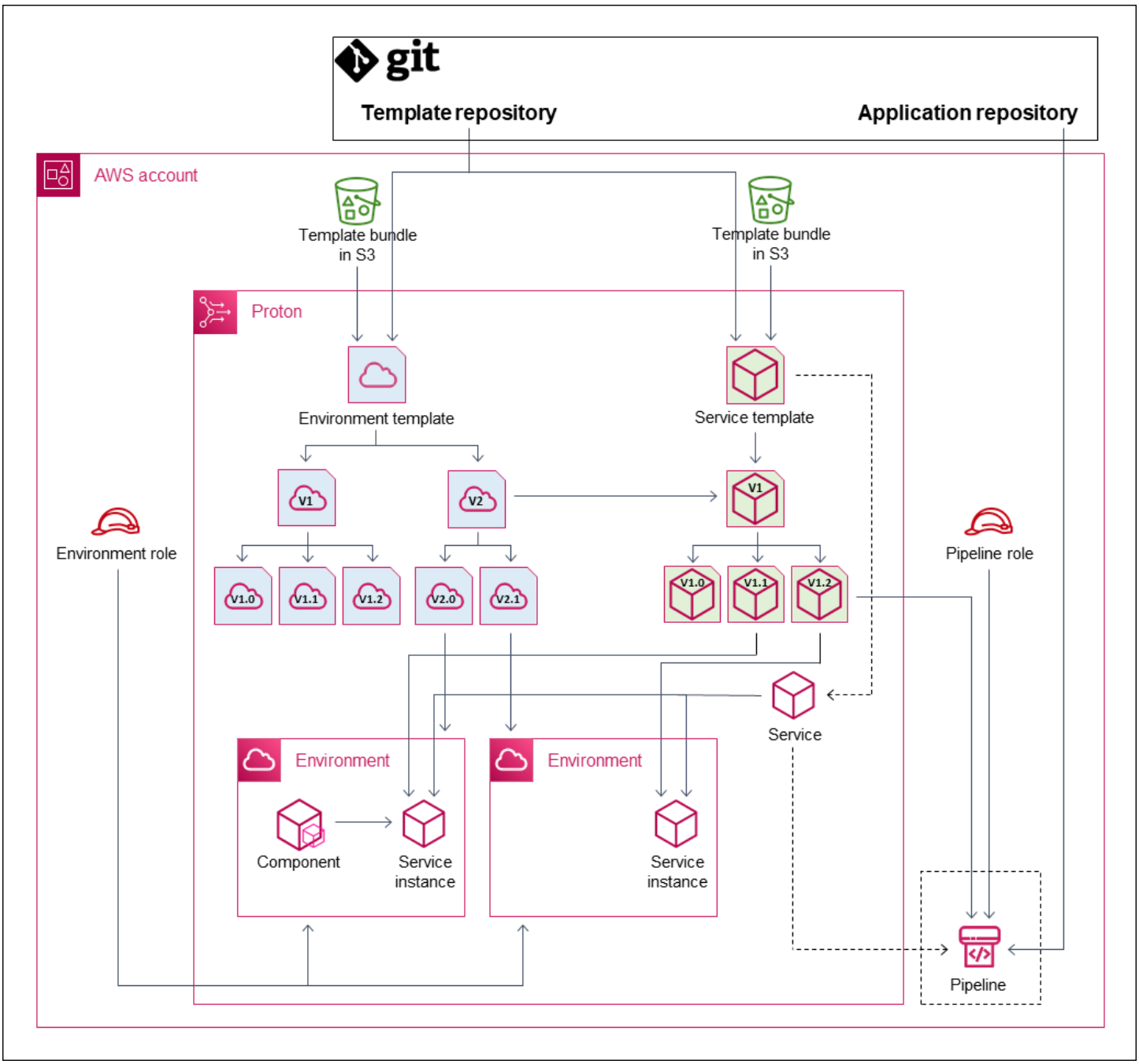
주제

- AWS Proton 객체
- 가 인프라를 AWS Proton 프로비저닝하는 방법
- AWS Proton 용어

AWS Proton 객체

다음 다이어그램은 기본 AWS Proton 객체와 다른 AWS 객체 및 타사 객체와의 관계를 보여줍니다. 화살표는 데이터 흐름의 방향(종속성의 역방향)을 나타냅니다.

이러한 AWS Proton 객체에 대한 간략한 설명과 참조 링크가 포함된 다이어그램을 따릅니다.



- 환경 템플릿 - AWS Proton 환경을 만드는 데 사용할 수 있는 환경 템플릿 버전 모음입니다.

  자세한 정보는 템플릿 작성 및 번들 및 템플릿 단원을 참조하세요.

- 환경 템플릿 버전 — 환경 템플릿의 특정 버전입니다. S3 버킷 또는 Git 리포지토리에서 템플릿 번들을 입력으로 가져옵니다. 번들은 AWS Proton 환경에 대한 코드형 인프라(IaC) 및 관련 입력 파라미터를 지정합니다.

자세한 내용은 the section called “버전”, the section called “게시”, the section called “템플릿 동기화 구성” 섹션을 참조하세요.

- 환경 - AWS Proton 서비스가 배포되는 공유 AWS 인프라 리소스 및 액세스 정책 세트입니다. AWS 리소스는 특정 파라미터 값과 함께 호출된 환경 템플릿 버전을 사용하여 프로비저닝됩니다. 액세스 정책은 서비스 역할로 제공됩니다.

  자세한 내용은 환경 단원을 참조하세요.

- 서비스 템플릿 - AWS Proton 서비스를 생성하는 데 사용할 수 있는 서비스 템플릿 버전 모음입니다.

  자세한 정보는 템플릿 작성 및 번들 및 템플릿 단원을 참조하세요.

- 서비스 템플릿 버전 — 서비스 템플릿의 특정 버전입니다. S3 버킷 또는 Git 리포지토리에서 템플릿 번들을 입력으로 가져옵니다. 번들은 AWS Proton 서비스에 대한 코드형 인프라(IaC) 및 관련 입력 파라미터를 지정합니다.

  또한 서비스 템플릿 버전은 버전을 기반으로 서비스 인스턴스에 대한 다음과 같은 제약 조건을 지정합니다.

  - 호환되는 환경 템플릿 - 인스턴스는 이러한 호환 가능한 환경 템플릿을 기반으로 하는 환경에서만 실행할 수 있습니다.
  - 지원되는 구성 요소 소스 - 개발자가 인스턴스와 연결할 수 있는 구성 요소 유형입니다.

  자세한 내용은 the section called “버전”, the section called “게시”, the section called “템플릿 동기화 구성” 단원을 참조하세요.

- 서비스 - 서비스 템플릿에 지정된 리소스를 사용하여 애플리케이션을 실행하는 서비스 인스턴스 컬렉션과 해당 인스턴스에 애플리케이션 코드를 배포하는 CI/CD 파이프라인(선택 사항).

  다이어그램에서 서비스 템플릿의 점선은 서비스가 템플릿을 서비스 인스턴스와 파이프라인으로 전달한다는 의미입니다.

  자세한 내용은 서비스 단원을 참조하십시오.

- 서비스 인스턴스 - 특정 AWS Proton 환경에서 애플리케이션을 실행하는 AWS 인프라 리소스 세트입니다. AWS 리소스는 특정 파라미터 값과 함께 호출된 서비스 템플릿 버전을 사용하여 프로비저닝됩니다.

  자세한 내용은 서비스 및 the section called “인스턴스 업데이트” 섹션을 참조하세요.

- 파이프라인 — 애플리케이션을 서비스 인스턴스에 배포하는 선택적 CI/CD 파이프라인으로, 이 파이프라인을 프로비저닝하기 위한 액세스 정책을 적용합니다. 액세스 정책은 서비스 역할로 제공됩니

다. 서비스에 항상 연결된 AWS Proton 파이프라인이 있는 것은 아닙니다. 외부에서 앱 코드 배포를 관리하도록 선택할 수 있습니다 AWS Proton.

다이어그램에서 서비스의 점선과 파이프라인 주위의 점선은 CI/CD 배포를 직접 관리하도록 선택한 경우 AWS Proton 파이프라인이 생성되지 않고 자체 파이프라인이 AWS 계정 내에 있지 않을 수 있음을 의미합니다.

자세한 내용은 서비스 및 the section called “파이프라인 업데이트” 섹션을 참조하세요.

- 구성 요소 — 개발자가 정의한 서비스 인스턴스 확장 프로그램입니다. 환경 및 서비스 인스턴스에서 제공하는 리소스 외에도 특정 애플리케이션에 필요할 수 있는 추가 AWS 인프라 리소스를 지정합니다. 플랫폼 팀은 구성 요소 역할을 환경에 연결하여 구성 요소가 제공할 수 있는 인프라를 제어합니다.

  자세한 내용은 구성 요소 단원을 참조하십시오.

가 인프라를 AWS Proton 프로비저닝하는 방법

AWS Proton 는 여러 가지 방법 중 하나로 인프라를 프로비저닝할 수 있습니다.

- AWS관리형 프로비저닝 - 사용자를 대신하여 프로비저닝 엔진을 AWS Proton 호출합니다. 이 방법은 AWS CloudFormation 템플릿 번들만 지원합니다. 자세한 내용은 the section called “CloudFormation IaC 파일” 단원을 참조하십시오.
- CodeBuild 프로비저닝 - 사용자가 제공하는 셸 명령을 실행하는 AWS CodeBuild 데 AWS Proton 사용합니다. 명령은가 AWS Proton 제공하는 입력을 읽을 수 있으며 인프라 프로비저닝 또는 프로비저닝 해제와 출력 값 생성을 담당합니다. 이 방법을 위한 템플릿 번들에는 매니페스트 파일의 명령과 이러한 명령에 필요할 수 있는 프로그램, 스크립트 또는 기타 파일이 포함되어 있습니다.

  CodeBuild 프로비저닝을 사용하는 예로를 사용하여 AWS 리소스를 AWS Cloud Development Kit (AWS CDK) 프로비저닝하는 코드와 CDK를 설치하고 CDK 코드를 실행하는 매니페스트를 포함할 수 있습니다.

  자세한 내용은 the section called “CodeBuild 번들” 단원을 참조하십시오.

> **Note**
>
> CodeBuild 프로비저닝을 환경 및 서비스와 함께 사용할 수 있습니다. 현재로서는 이 방법으로 구성 요소를 프로비저닝할 수 없습니다.

- 자체 관리형 프로비저닝 - 사용자가 제공하는 리포지토리에 풀 요청(PR)을 AWS Proton 발행합니다.이 리포지토리에서는 자체 인프라 배포 시스템이 프로비저닝 프로세스를 실행합니다. 이 방법은 Terraform 템플릿 번들만 지원합니다. 자세한 내용은 the section called “Terraform IaC 파일” 단원을 참조하십시오.

AWS Proton 는 각 환경 및 서비스에 대한 프로비저닝 방법을 별도로 결정하고 설정합니다. 환경 또는 서비스를 만들거나 업데이트할 때 AWS Proton 는 제공하는 템플릿 번들을 검사하고 템플릿 번들이 나타내는 프로비전 방법을 결정합니다. 환경 수준에서는 환경 및 잠재적 서비스가 프로비저닝 방법에 필요할 수 있는AWS Identity and Access Management (IAM) 역할, 환경 계정 연결 또는 인프라 리포지토리와 같은 파라미터를 제공합니다.

AWS Proton 를 사용하여 서비스를 프로비저닝하는 개발자는 프로비저닝 방법에 관계없이 동일한 경험을 하게 됩니다. 개발자는 프로비저닝 방법을 알 필요가 없으며 서비스 프로비전 프로세스에서 아무것도 변경할 필요가 없습니다. 서비스 템플릿은 프로비저닝 방법을 설정하며, 개발자가 서비스를 배포하는 각 환경은 서비스 인스턴스 프로비저닝에 필요한 파라미터를 제공합니다.

다음 다이어그램은 다양한 프로비전 방법의 몇 가지 주요 특성을 요약한 것입니다. 표 다음 단원에서는 각 방법에 대한 세부 정보를 제공합니다.

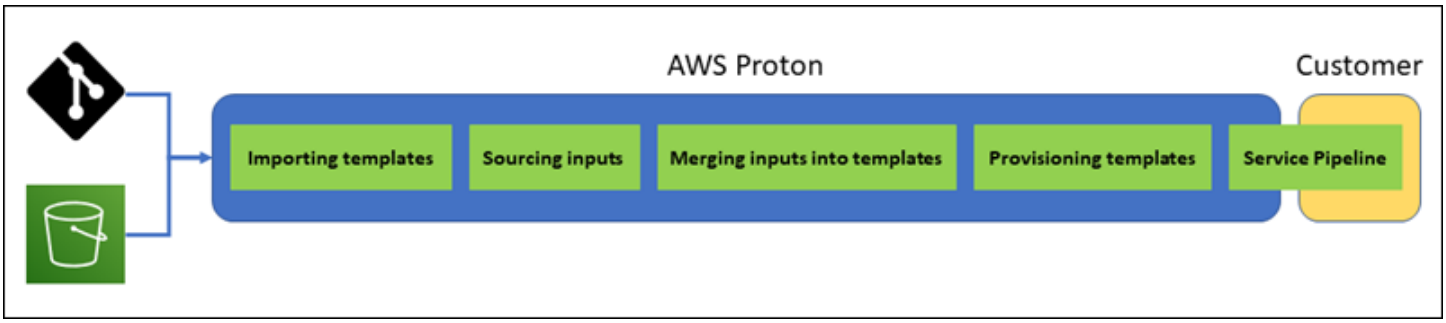
프로비저닝 방법	템플릿	프로비저닝 제공자	추적 대상 상태
AWS관리형	매니페스트, 스키마, IAC 파일(CloudFormation)	AWS Proton (CloudFormation을 통해)	AWS Proton (CloudFormation을 통해)
CodeBuild	매니페스트(명령 포함), 스키마, 명령 종속성(예: AWS CDK 코드)	AWS Proton (CodeBuild를 통해)	AWS Proton (명령은 CodeBuild를 통해 상태를 반환합니다.)
자체 관리형	매니페스트, 스키마, IAC 파일(Terraform)	사용자 코드(Git 작업을 통해)	코드(API 호출을 AWS 통해에 전달됨)

AWS관리형 프로비저닝 작동 방식

환경 또는 서비스가 AWS관리형 프로비저닝을 사용하는 경우 인프라는 다음과 같이 프로비저닝됩니다.

1. AWS Proton 고객(관리자 또는 개발자)은 AWS Proton 리소스(환경 또는 서비스)를 생성합니다. 고객은 리소스의 템플릿을 선택하고 필요한 파라미터를 제공합니다. 자세한 내용은 다음 the section called “AWS관리형 프로비저닝에 대한 고려 사항” 단원을 참조하세요.
2. AWS Proton 는 리소스를 프로비저닝하기 위한 전체 CloudFormation 템플릿을 렌더링합니다.
3. AWS Proton 는를 호출 CloudFormation 하여 렌더링된 템플릿을 사용하여 프로비저닝을 시작합니다.
4. AWS Proton 는 CloudFormation 배포를 지속적으로 모니터링합니다.
5. 프로비저닝이 완료되면 실패 시 오류를 다시 AWS Proton 보고하고 성공 시 Amazon VPC ID와 같은 프로비저닝 출력을 캡처합니다.

다음 다이어그램은 AWS Proton 가 이러한 단계 대부분을 직접 처리함을 보여줍니다.



AWS관리형 프로비저닝에 대한 고려 사항

- 인프라 프로비저닝 역할 AWS- 환경 또는 환경에서 실행되는 서비스 인스턴스가 관리형 프로비저닝을 사용할 수 있는 경우 관리자는 IAM 역할을 구성해야 합니다(직접 또는 AWS Proton 환경 계정 연결의 일부로).는이 역할을 AWS Proton 사용하여 이러한 AWS관리형 프로비저닝 리소스의 인프라를 프로비저닝합니다. 역할에는 이러한 리소스의 템플릿에 포함된 모든 리소스를 생성하는 CloudFormation 데 사용할 수 있는 권한이 있어야 합니다.

  자세한 내용은 the section called “IAM 역할” 및 the section called “서비스 역할 정책 예제” 섹션을 참조하세요.
- 서비스 프로비저닝 AWS- 개발자가 관리형 프로비저닝을 사용하는 서비스 인스턴스를 환경에 배포하면는 해당 환경에 제공된 역할을 AWS Proton 사용하여 서비스 인스턴스에 대한 인프라를 프로비저닝합니다. 개발자는 이 역할을 볼 수 없으며 변경할 수도 없습니다.
- 파이프라인이 있는 서비스 AWS- 관리형 프로비저닝을 사용하는 서비스 템플릿에는 CloudFormation YAML 스키마에 작성된 파이프라인 정의가 포함될 수 있습니다. AWS Proton 또한를 호출하여 파이프라인을 생성합니다 CloudFormation. 가 파이프라인을 생성하는 데 AWS Proton 사용하는 역할은 각 개별 환경의 역할과 별개입니다. 이 역할은 AWS 계정 수준에서 한 번만 AWS

Proton 별도로에 제공되며 모든 AWS관리형 파이프라인을 프로비저닝하고 관리하는 데 사용됩니다. 이 역할에는 파이프라인에 필요한 파이프라인 및 기타 리소스를 생성할 수 있는 권한이 있어야 합니다.

다음 절차는 AWS Proton에 파이프라인 역할을 제공하는 방법을 보여줍니다.

AWS Proton console

파이프라인 역할을 제공하려면

1. AWS Proton 콘솔의 탐색 창에서 설정 > 계정 설정을 선택한 다음 구성을 선택합니다.
2. 파이프라인 AWS관리형 역할 섹션을 사용하여 AWS관리형 프로비저닝을 위한 새 파이프라인 역할 또는 기존 파이프라인 역할을 구성합니다.

AWS Proton API

파이프라인 역할을 제공하려면

1. UpdateAccountSettings API 작업을 사용합니다.
2. `pipelineServiceRoleArn` 파라미터에서 파이프라인 서비스 역할의 리소스 이름(ARN)을 제공하세요.

AWS CLI

파이프라인 역할을 제공하려면

다음 명령을 실행합니다.

```
$ aws proton update-account-settings \
    --pipeline-service-role-arn \
        "arn:aws:iam::123456789012:role/my-pipeline-role"
```

CodeBuild 프로비저닝 작동 방식

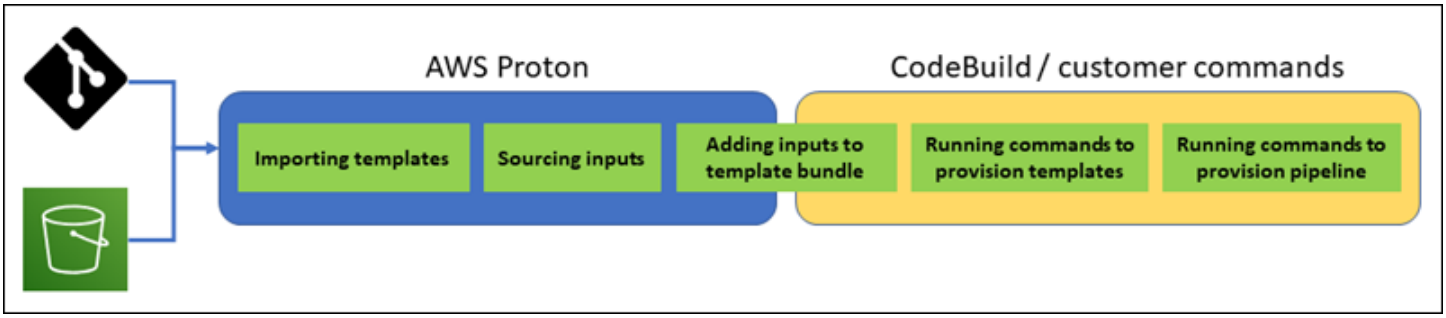
환경 또는 서비스가 관리형 프로비저닝을 사용하는 경우 인프라는 다음과 같이 프로비저닝됩니다.

1. AWS Proton 고객(관리자 또는 개발자)은 AWS Proton 리소스(환경 또는 서비스)를 생성합니다. 고객은 리소스의 템플릿을 선택하고 필요한 파라미터를 제공합니다. 자세한 내용은 다음 the section called “CodeBuild 프로비저닝 고려 사항” 단원을 참조하세요.

2. AWS Proton 는 리소스를 프로비저닝하기 위한 입력 파라미터 값으로 입력 파일을 렌더링합니다.
3. AWS Proton 는 CodeBuild를 호출하여 작업을 시작합니다. CodeBuild 작업은 템플릿에 지정된 고객 쉘 명령을 실행합니다. 이러한 명령은 원하는 인프라를 프로비저닝하는 동시에 선택적으로 입력 값을 읽습니다.
4. 프로비저닝이 완료되면 최종 고객 명령은 프로비저닝 상태를 CodeBuild에 반환하고 NotifyResourceDeploymentStatusChange AWS Proton API 작업을 호출하여 Amazon VPC ID와 같은 출력을 제공합니다.

> **Important**
>
> 명령이 CodeBuild에 프로비저닝 상태를 올바르게 반환하고 출력을 제공하는지 확인하세요. 그렇지 않으면가 프로비저닝 상태를 AWS Proton 제대로 추적할 수 없으며 서비스 인스턴스에 올바른 출력을 제공할 수 없습니다.

다음 다이어그램은가 AWS Proton 수행하는 단계와 명령이 CodeBuild 작업 내에서 수행하는 단계를 보여줍니다.



CodeBuild 프로비저닝 고려 사항

- 인프라 프로비저닝 역할 - 환경 또는 해당 환경에서 실행되는 서비스 인스턴스가 CodeBuild 기반 프로비저닝을 사용할 수 있는 경우 관리자는 IAM 역할을 구성해야 합니다(직접 또는 AWS Proton 환경 계정 연결의 일부로).는이 역할을 AWS Proton 사용하여 이러한 CodeBuild 프로비저닝 리소스의 인프라를 프로비저닝합니다. 역할에는 이러한 리소스의 템플릿에 포함된 모든 리소스를 만드는 데 사용할 수 있는 권한이 있어야 합니다.

  자세한 내용은 the section called “IAM 역할” 및 the section called “서비스 역할 정책 예제” 섹션을 참조하세요.

- 서비스 프로비저닝 - 개발자가 CodeBuild 프로비저닝을 사용하는 서비스 인스턴스를 환경에 배포하면는 해당 환경에 제공된 역할을 AWS Proton 사용하여 서비스 인스턴스에 대한 인프라를 프로비저닝합니다. 개발자는 이 역할을 볼 수 없으며 변경할 수도 없습니다.
- 파이프라인이 있는 서비스 - CodeBuild 프로비저닝을 사용하는 서비스 템플릿에는 파이프라인을 프로비저닝하는 명령이 포함될 수 있습니다. AWS Proton 또한 CodeBuild를 호출하여 파이프라인을 생성합니다. 가 파이프라인을 생성하는 데 AWS Proton 사용하는 역할은 각 개별 환경의 역할과 별개입니다. 이 역할은 AWS 계정 수준에서 한 번만 AWS Proton 별도로에 제공되며 모든 CodeBuild 기반 파이프라인을 프로비저닝하고 관리하는 데 사용됩니다. 이 역할에는 파이프라인에 필요한 파이프라인 및 기타 리소스를 생성할 수 있는 권한이 있어야 합니다.

  다음 절차는 AWS Proton에 파이프라인 역할을 제공하는 방법을 보여줍니다.

  AWS Proton console

  파이프라인 역할을 제공하려면

  1. AWS Proton 콘솔의 탐색 창에서 설정 > 계정 설정을 선택한 다음 구성을 선택합니다.
  2. Codebuild 파이프라인 프로비저닝 역할 섹션을 사용하여 CodeBuild 프로비저닝을 위한 새 파이프라인 역할 또는 기존 파이프라인 역할을 구성할 수 있습니다.

  AWS Proton API

  파이프라인 역할을 제공하려면

  1. UpdateAccountSettings API 작업을 사용합니다.
  2. `pipelineCodebuildRoleArn` 파라미터에서 파이프라인 서비스 역할의 리소스 이름(ARN)을 제공하세요.

  AWS CLI

  파이프라인 역할을 제공하려면

  다음 명령을 실행합니다.

```
$ aws proton update-account-settings \
    --pipeline-codebuild-role-arn \
        "arn:aws:iam::123456789012:role/my-pipeline-role"
```

자체 관리형 프로비저닝의 작동 방식

환경 또는 서비스가 자체 관리형 프로비저닝을 사용하는 경우 인프라는 다음과 같이 프로비저닝됩니다.

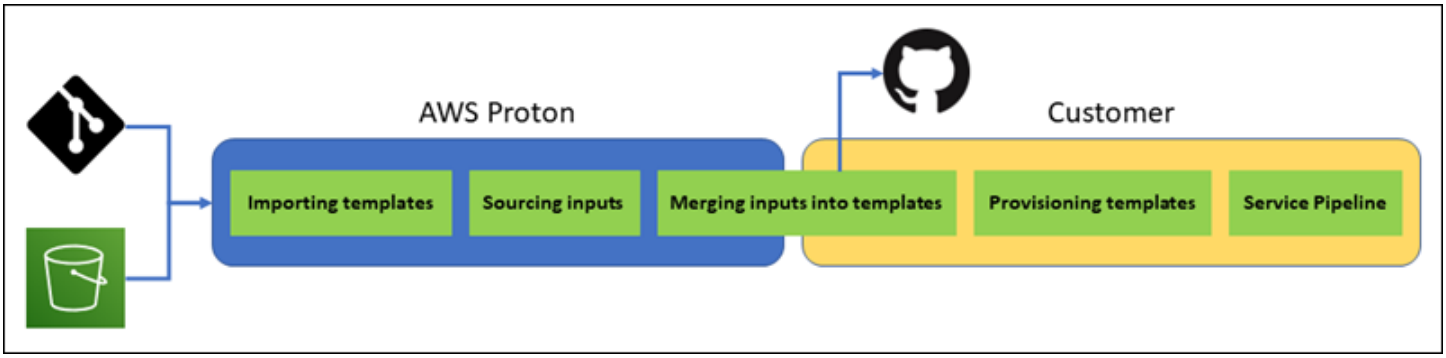
1. AWS Proton 고객(관리자 또는 개발자)은 AWS Proton 리소스(환경 또는 서비스)를 생성합니다. 고객은 리소스의 템플릿을 선택하고 필요한 파라미터를 제공합니다. 환경의 경우 고객은 연결된 인프라 리포지토리도 제공합니다. 자세한 내용은 다음 the section called “자체 관리형 프로비저닝에 대한 고려 사항” 단원을 참조하세요.
2. AWS Proton 는 전체 Terraform 템플릿을 렌더링합니다. 잠재적으로 여러 폴더에 있는 하나 이상의 Terraform 파일과 `.tfvars` 변수 파일로 구성됩니다.는 리소스 생성 호출에 제공된 파라미터 값을 이 변수 파일에 AWS Proton 씁니다.
3. AWS Proton 는 렌더링된 Terraform 템플릿을 사용하여 인프라 리포지토리에 PR을 제출합니다.
4. 고객 (관리자 또는 개발자) 이 PR을 병합하면 고객의 자동화가 프로비저닝 엔진을 트리거하여 병합된 템플릿을 사용하여 인프라 프로비저닝을 시작합니다.

> **Note**
>
> 고객(관리자 또는 개발자)이 PR을 닫으면는 PR을 닫힌 것으로 AWS Proton 인식하고 배포를 취소됨으로 표시합니다.

5. 프로비저닝이 완료되면 고객의 자동화가 NotifyResourceDeploymentStatusChange AWS Proton API 작업을 호출하여 완료를 표시하고, 상태(성공 또는 실패)를 제공하고, 존재하는 경우 Amazon VPC ID와 같은 출력을 제공합니다.

> **Important**
>
> 자동화 코드가 프로비저닝 상태 및 출력 AWS Proton 으로를 다시 호출해야 합니다. 그렇지 않으면 프로비저닝이 보류 중인 것으로 예상보다 오래 간주하고 진행 중 상태를 계속 표시할 AWS Proton 수 있습니다.

다음 다이어그램은가 AWS Proton 수행하는 단계와 자체 프로비저닝 시스템이 수행하는 단계를 보여줍니다.



자체 관리형 프로비저닝에 대한 고려 사항

- 인프라 리포지토리 - 관리자가 자체 관리형 프로비저닝을 위해 환경을 구성할 때 연결된 인프라 리포지토리를 제공해야 합니다.는 환경의 인프라와 환경에 배포된 모든 서비스 인스턴스를 프로비저닝하기 위해이 리포지토리에 PRs을 AWS Proton 제출합니다. 리포지토리의 고객 소유 자동화 작업은 환경 및 서비스 템플릿에 포함된 모든 리소스를 생성할 수 있는 권한이 있는 IAM 역할과 대상 AWS 계정을 반영하는 자격 증명을 수임해야 합니다. 역할을 수임하는 GitHub 작업의 예는 GitHub 작업에 대한 "자격 AWS 증명 구성" 작업 설명서의 역할 수임을 참조하세요.
- 권한 - 프로비저닝 코드는 필요에 따라 계정으로 인증하고(예: AWS 계정에 인증) 리소스 프로비저닝 권한 부여를 제공해야 합니다(예: 역할 제공).
- 서비스 프로비저닝 - 개발자가 자체 관리형 프로비저닝을 사용하는 서비스 인스턴스를 환경에 배포하면는 환경과 연결된 리포지토리에 PR을 AWS Proton 제출하여 서비스 인스턴스에 대한 인프라를 프로비저닝합니다. 개발자는 리포지토리를 볼 수 없으며 변경할 수도 없습니다.

> **Note**
>
> 서비스를 만드는 개발자는 프로비저닝 방법에 상관없이 동일한 프로세스를 사용하며, 이러한 프로세스를 통해 차이점을 파악할 수 있습니다. 하지만 자체 관리형 프로비저닝을 사용하는 개발자의 경우 프로비저닝을 시작하기 전에 누군가(자신이 아닐 수도 있음)가 인프라 저장소의 PR을 병합할 때까지 기다려야 하기 때문에 개발자의 응답 속도가 느려질 수 있습니다.

- 파이프라인이 있는 서비스 - 자체 관리형 프로비저닝이 있는 환경에 대한 서비스 템플릿에는 Terraform HCL로 작성된 파이프라인 정의(예: AWS CodePipeline 파이프라인)가 포함될 수 있습니다. 가 이러한 파이프라인을 프로비저닝 AWS Proton 할 수 있도록 관리자는 연결된 파이프라인 리포지토리를 제공합니다 AWS Proton. 파이프라인을 프로비저닝할 때 리포지토리의 고객 소유 자동화 작업은 파이프라인을 프로비저닝할 권한이 있는 IAM 역할과 대상 AWS 계정을 반영하는 자격 증명을 수임해야 합니다. 파이프라인 리포지토리 및 역할은 각 개별 환경에 사용되는 것과 별개입니다.

연결된 리포지토리는 AWS 계정 수준에서 한 번만 AWS Proton 별도로에 제공되며 모든 파이프라인을 프로비저닝하고 관리하는 데 사용됩니다. 해당 역할에는 파이프라인에 필요한 파이프라인 및 기타 리소스를 생성할 수 있는 권한이 있어야 합니다.

다음 절차는 AWS Proton에 파이프라인 리포지토리 및 역할을 제공하는 방법을 보여줍니다.

AWS Proton console

파이프라인 역할을 제공하려면

1. AWS Proton 콘솔의 탐색 창에서 설정 > 계정 설정을 선택한 다음 구성을 선택합니다.
2. CI/CD 파이프라인 리포지토리 섹션을 사용하여 새 리포지토리 링크 또는 기존 리포지토리 링크를 구성할 수 있습니다.

AWS Proton API

파이프라인 역할을 제공하려면

1. UpdateAccountSettings API 작업을 사용합니다.
2. `pipelineProvisioningRepository` 파라미터에 파이프라인 리포지토리의 공급자, 이름, 브랜치를 입력합니다.

AWS CLI

파이프라인 역할을 제공하려면

다음 명령을 실행합니다.

```
$ aws proton update-account-settings \
    --pipeline-provisioning-repository \
        "provider=GITHUB,name=my-pipeline-repo-name,branch=my-branch"
```

- 자체 관리형 프로비저닝 리소스 삭제 — Terraform 모듈에는 리소스 정의 외에도 Terraform 작업에 필요한 구성 요소가 포함될 수 있습니다. 따라서 AWS Proton 는 환경 또는 서비스 인스턴스에 대한 모든 Terraform 파일을 삭제할 수 없습니다. 대신는 삭제할 파일을 AWS Proton 표시하고 PR 메타데이터에서 플래그를 업데이트했습니다. 자동화는 해당 플래그를 읽고 이를 사용하여 terraform destroy 명령을 트리거할 수 있습니다.

AWS Proton 용어

환경 템플릿

여러 애플리케이션 또는 리소스에서 사용되는 VPC 또는 클러스터와 같은 공유 인프라를 정의합니다.

환경 템플릿 번들

AWS Proton에서 환경 템플릿을 만들고 등록하기 위해 업로드하는 파일 모음입니다. 환경 템플릿 번들에는 다음이 포함됩니다.

1. 인프라를 코드 입력 파라미터로 정의하는 스키마 파일입니다.
2. 여러 애플리케이션이나 리소스에서 사용하는 VPC 또는 클러스터와 같은 공유 인프라를 정의하는 코드형 인프라(IaC) 파일입니다.
3. IaC 파일을 나열하는 매니페스트 파일.

환경

여러 애플리케이션 또는 리소스에서 사용되는 VPC 또는 클러스터와 같은 공유 인프라를 프로비저닝합니다.

서비스 템플릿

환경에 애플리케이션 또는 마이크로서비스를 배포하고 유지 관리하는 데 필요한 인프라 유형을 정의합니다.

서비스 템플릿 번들

AWS Proton에서 서비스 템플릿을 만들고 등록하기 위해 업로드하는 파일 모음입니다. 서비스 템플릿 번들에는 다음이 포함됩니다.

1. 코드형 인프라(IaC) 입력 파라미터를 정의하는 스키마 파일입니다.
2. 환경에서 애플리케이션 또는 마이크로서비스를 배포하고 유지 관리하는 데 필요한 인프라를 정의하는 IaC 파일입니다.
3. IaC 파일을 나열하는 매니페스트 파일.
4. 선택 사항
   a. 서비스 파이프라인 인프라를 정의하는 IaC 파일.
   b. IaC 파일을 나열하는 매니페스트 파일.

서비스

환경에 애플리케이션 또는 마이크로서비스를 배포하고 유지 관리하는 데 필요한 프로비저닝하는 인프라.

서비스 인스턴스

환경에 애플리케이션 또는 마이크로서비스를 지원하는 프로비저닝하는 인프라.

서비스 파이프라인

파이프라인을 지원하는 프로비저닝된 인프라.

템플릿 버전

템플릿의 메이저 또는 마이너 버전. 자세한 내용은 버전이 지정된 템플릿 단원을 참조하세요.

입력 파라미터

스키마 파일에 정의되고 IaC(코드형 인프라) 파일에 사용되므로 IaC 파일을 다양한 사용 사례에 반복적으로 사용할 수 있습니다.

스키마 파일

코드형 인프라 파일 입력 파라미터를 정의합니다.

스펙 파일

스키마 파일에 정의된 대로 코드형 인프라 파일 입력 파라미터 값을 지정합니다.

매니페스트 파일

코드형 인프라 파일을 나열합니다.

용 템플릿 작성 및 번들 생성 AWS Proton

AWS Proton 는 코드형 인프라(IaC) 파일을 기반으로 리소스를 프로비저닝합니다. 재사용 가능한 IaC 파일로 인프라를 설명합니다. 다양한 환경 및 응용 프로그램에서 파일을 재사용할 수 있도록 하려면 파일을 템플릿으로 작성하고 입력 파라미터를 정의한 다음 이러한 파라미터를 IaC 정의에 사용합니다. 나중에 프로비저닝 리소스(환경, 서비스 인스턴스 또는 구성 요소)를 생성할 때는 입력 값을 템플릿과 결합하여 프로비저닝할 준비가 된 IaC 파일을 생성하는 렌더링 엔진을 AWS Proton 사용합니다.

관리자는 대부분의 템플릿을 템플릿 번들로 작성한 다음 업로드하여에 등록합니다 AWS Proton. 이 페이지의 나머지 부분에서는 이러한 AWS Proton 템플릿 번들에 대해 설명합니다. 직접 정의된 구성 요소는 예외입니다. 개발자가 직접 만든 구성 요소는 IaC 템플릿 파일을 직접 제공합니다. 구성 요소에 대한 자세한 내용은 구성 요소를 참조하세요.

주제

- 템플릿 번들
- AWS Proton 파라미터
- AWS Proton 코드형 인프라 파일
- 스키마 파일
- 에 대한 템플릿 파일 정리 AWS Proton
- 템플릿 번들 고려 사항

템플릿 번들

관리자는 템플릿을 생성하고에 등록합니다 AWS Proton. 이러한 템플릿을 사용하여 환경과 서비스를 만들 수 있습니다. 서비스를 생성하면가 서비스 인스턴스를 AWS Proton 프로비저닝하고 선택한 환경에 배포합니다. 자세한 내용은 AWS Proton 플랫폼 팀용 단원을 참조하십시오.

에서 템플릿을 생성하고 등록하려면 및 환경 또는 서비스를 프로비저닝 AWS Proton 해야 하는 코드형 인프라(IaC) 파일이 포함된 템플릿 번들을 업로드 AWS Proton합니다.

템플릿 번들에는 다음이 포함됩니다.

- IaC 파일을 나열하는 매니페스트 YAML 파일이 포함된 코드형 인프라(IaC)파일.
- IAC 파일 입력 파라미터 정의를 위한 스키마 YAML 파일.

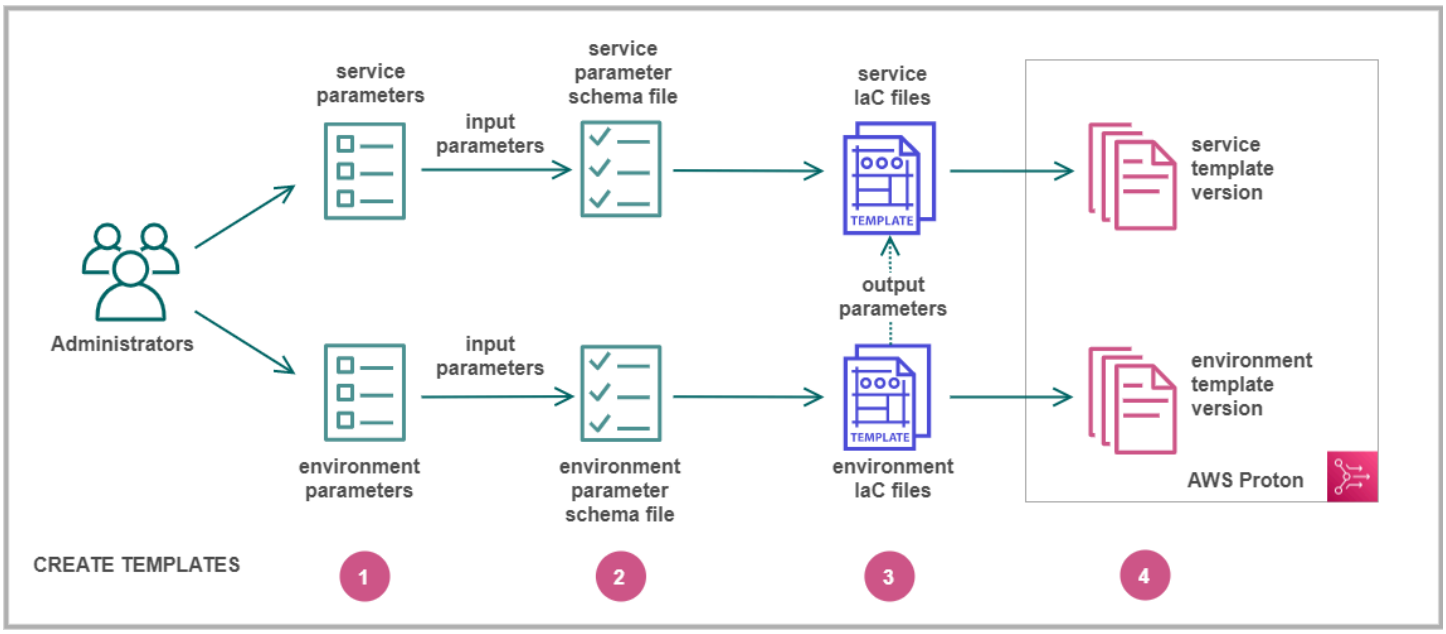
CloudFormation 환경 템플릿 번들에는 IaC 파일 하나가 포함되어 있습니다.

CloudFormation 서비스 템플릿 번들에는 서비스 인스턴스 정의를 위한 IaC 파일 하나와 파이프라인 정의를 위한 선택적 IaC 파일이 하나 포함되어 있습니다.

Terraform 환경 및 서비스 템플릿 번들에는 각각 여러 IaC 파일이 포함될 수 있습니다.

AWS Proton 에는 입력 파라미터 스키마 파일이 필요합니다. AWS CloudFormation 를 사용하여 IaC 파일을 생성할 때 Jinja 구문을 사용하여 입력 파라미터를 참조합니다.는 IaC 파일의 파라미터를 참조하는 데 사용할 수 있는 파라미터 네임스페이스를 AWS Proton 제공합니다.

다음 다이어그램은 템플릿을 생성하기 위해 수행할 수 있는 단계의 예를 보여줍니다 AWS Proton.



1 은

입력 파라미터를 식별합니다.

2 은

스키마 파일을 생성하여 입력 파라미터를 정의합니다.

3 은

입력 파라미터를 참조하는 IaC 파일을 생성합니다. 환경 IaC 파일 출력을 서비스 IaC 파일의 입력으로 참조할 수 있습니다.

4

에 템플릿 버전을 등록 AWS Proton 하고 템플릿 번들을 업로드합니다.

AWS Proton 파라미터

IaC(코드형 인프라) 파일에서 파라미터를 정의하고 사용하여 유연하고 재사용 가능하게 만들 수 있습니다. 네임 AWS Proton 스페이스에서 파라미터의 이름을 참조하여 IaC 파일의 파라미터 값을 읽습니다. 파라미터 값은 리소스 프로비저닝 중에 생성하는 렌더링된 IaC 파일에 AWS Proton 주입됩니다. AWS CloudFormation IaC 파라미터를 처리하려면 Jinja를 AWS Proton 사용합니다. Terraform IaC 파라미터를 처리하기 위해는 Terraform 파라미터 값 파일을 AWS Proton 생성하고 HCL에 내장된 파라미터화 기능에 의존합니다.

를 사용하면 코드가 가져올 수 있는 입력 파일을 CodeBuild 프로비저닝 AWS Proton 생성합니다. 파일은 템플릿 매니페스트의 속성에 따라 JSON 또는 HCL 파일입니다. 자세한 내용은 the section called "CodeBuild 프로비저닝 파라미터" 단원을 참조하세요.

환경, 서비스, 구성 요소 IaC 파일의 파라미터 또는 프로비저닝 코드에서 다음 요구 사항을 참조할 수 있습니다.

- 각 파라미터 이름의 길이는 100자를 초과할 수 없습니다.
- 파라미터 네임스페이스와 리소스 이름을 합친 길이는 리소스 이름의 문자 제한을 초과하지 않습니다.

AWS Proton 이러한 할당량을 초과하면 프로비저닝이 실패합니다.

파라미터 유형

다음 파라미터 유형을 AWS Proton IaC 파일에서 참조할 수 있습니다.

입력 파라미터

환경 및 서비스 인스턴스는 환경 또는 서비스 템플릿에 연결하는 스키마 파일에 정의한 입력 파라미터를 사용할 수 있습니다. 리소스의 IaC 파일에서 리소스의 입력 파라미터를 참조할 수 있습니다. 구성 요소 IaC 파일은 구성 요소가 연결된 서비스 인스턴스의 입력 파라미터를 참조할 수 있습니다.

AWS Proton 는 스키마 파일에 대해 입력 파라미터 이름을 확인하고 IaC 파일에서 참조되는 파라미터와 일치시켜 리소스 프로비저닝 중에 사양 파일에 제공하는 입력 값을 주입합니다.

출력 파라미터

모든 IaC 파일에서 출력을 정의할 수 있습니다. 출력은 예를 들어 템플릿에서 제공하는 리소스 중 하나의 이름, ID 또는 ARN일 수도 있고 템플릿의 입력 중 하나를 전달하는 방법일 수도 있습니다. 다른 리소스의 IaC 파일에서 이러한 출력을 참조할 수 있습니다.

클라우드포메이션 IaC 파일에서 `Outputs:` 블록의 출력 파라미터를 정의합니다. Terraform IaC 파일에서 `output` 명령문을 사용하여 각 출력 파라미터를 정의합니다.

리소스 파라미터

AWS Proton 는 AWS Proton 리소스 파라미터를 자동으로 생성합니다. 이러한 파라미터는 AWS Proton 리소스 객체의 속성을 노출합니다. 리소스 파라미터의 예는 `environment.name`입니다.

IaC 파일에서 AWS Proton 파라미터 사용

IaC 파일에서 파라미터 값을 읽으려면 파라미터 네임스페이스에서 AWS Proton 파라미터 이름을 참조합니다. AWS CloudFormation IaC 파일의 경우 Jinja 구문을 사용하고 파라미터를 중괄호와 따옴표 쌍으로 묶습니다.

다음 표는 지원되는 각 템플릿 언어에 대한 참조 구문을 예제와 함께 보여 줍니다.

템플릿 언어	구문	예: "VPC"라는 이름의 환경 입력
CloudFormation	`"{{ parameter-name }}"`	`"{{ environment.inputs.VPC }}"`
Terraform	`var.parameter-name`	`var.environment.inputs.VPC` 생성된 테라폼 변수 정의

> **Note**
>
> IaC 파일에서 CloudFormation 동적 파라미터를 사용하는 경우 Jinja의 오해 오류를 방지하려면 파라미터를 이스케이프 처리해야 합니다. 자세한 내용은 문제 해결 AWS Proton 섹션을 참조하세요.

다음 표에는 모든 AWS Proton 리소스 파라미터의 네임스페이스 이름이 나열되어 있습니다. 각 템플릿 파일 형식은 파라미터 네임스페이스의 다른 하위 집합을 사용할 수 있습니다.

템플릿 파일.	파라미터 유형	파라미터 이름	설명
환경	리소스	`environment.` **`name`**	환경 이름
	입력	`environment.inputs.` *`input-name`*	스키마로 정의된 환경 입력
서비스	리소스	`environment.` **`name`** `environment.` **`account_id`**	환경 이름 및 AWS 계정 ID
	output	`environment.outputs.` *`output-name`*	환경 IaC 파일 출력
	리소스	`service.`**`branch_name`** `service.`**`name`** `service.`**`repository_connection_arn`** `service.`**`repository_id`**	서비스 이름 및 코드 리포지토리
	리소스	`service_instance.` **`name`**	서비스 인스턴스 이름
	입력	`service_instance.inputs.` *`input-name`*	스키마 정의 서비스 인스턴스 입력
	리소스	`service_instance.components.default.` **`name`**	첨부된 기본 구성 요소 이름
	output	`service_instance.components.default.outputs.` *`output-name`*	첨부된 기본 구성 요소 IaC 파일 출력
파이프라인	리소스	`service_instance.environment.` **`name`** `service_instance.environment.` **`account_id`**	서비스 인스턴스 환경 이름 및 AWS 계정 ID

템플릿 파일.	파라미터 유형	파라미터 이름	설명
	output	`service_instance.environment.outputs.` *output-name*	서비스 인스턴스 환경 IaC 파일 출력
	입력	`pipeline.inputs.` *input-name*	스키마 정의 파이프라인 입력
	리소스	`service.`**branch_name** `service.`**name** `service.`**repository_connection_arn** `service.`**repository_id**	서비스 이름 및 코드 리포지토리
	입력	`service_instance.inputs.` *input-name*	스키마 정의 서비스 인스턴스 입력
	수집	`{% for service_instance in` **service_instances** `%}...{% endfor %}`	반복해서 사용할 수 있는 서비스 인스턴스 모음
구성 요소	리소스	`environment.` **name** `environment.` **account_id**	환경 이름 및 AWS 계정 계정 ID
	output	`environment.outputs.` *output-name*	환경 IaC 파일 출력
	리소스	`service.`**branch_name** `service.`**name** `service.`**repository_connection_arn** `service.`**repository_id**	서비스 이름 및 코드 리포지토리(첨부된 구성 요소)

템플릿 파일.	파라미터 유형	파라미터 이름	설명
	리소스	`service_instance.` **`name`**	서비스 인스턴스 이름(연결된 구성 요소)
	입력	`service_instance.inputs.` *`input-name`*	스키마로 정의된 서비스 인스턴스 입력(연결된 구성 요소)
	리소스	`component.` **`name`**	구성 요소 이름

자세한 내용과 예제는 다양한 리소스 유형 및 템플릿 언어에 대한 IaC 템플릿 파일의 파라미터에 대한 하위 항목을 참조하세요.

주제

- 환경 CloudFormation IaC 파일 파라미터 세부 정보 및 예제
- CloudFormation IaC 파일 파라미터 세부 정보 및 예제
- 구성 요소 CloudFormation IaC 파일 파라미터 세부 정보 및 예제
- CloudFormation IaC 파일용 파라미터 필터
- CodeBuild 프로비저닝 파라미터 세부 정보 및 예제
- Terraform 코드형 인프라(IaC) 파일 파라미터 세부 정보 및 예제

환경 CloudFormation IaC 파일 파라미터 세부 정보 및 예제

환경 인프라의 파라미터를 코드형 인프라(IaC) 파일로 정의하고 참조할 수 있습니다. AWS Proton 파라미터, 파라미터 유형, 파라미터 네임스페이스 및 IaC 파일에서 파라미터를 사용하는 방법에 대한 자세한 설명은 섹션을 참조하세요the section called “파라미터”.

환경 파라미터를 정의하세요.

환경 IaC 파일의 입력 및 출력 파라미터를 모두 정의할 수 있습니다.

- 입력 파라미터 — 스키마 파일에 환경 입력 파라미터를 정의합니다.

  다음 목록에는 일반적인 사용 사례에 대한 환경 입력 파라미터의 예가 포함되어 있습니다.

  - VPC CIDR 값
  - 로드 밸런서 설정
  - 데이터베이스 설정
  - 상태 확인 타임아웃

  관리자는 환경을 생성할 때 입력 파라미터 값을 제공할 수 있습니다.
  - 콘솔을 사용하여가 AWS Proton 제공하는 스키마 기반 양식을 작성합니다.
  - CLI를 사용하여 값이 포함된 사양을 제공하세요.
- 출력 파라미터 - 환경 IaC 파일에서 환경 출력을 정의합니다. 그런 다음 다른 리소스의 IaC 파일에서 이러한 출력을 참조할 수 있습니다.

환경 IaC 파일에서 파라미터 값을 읽습니다.

환경 IaC 파일에서 환경과 관련된 파라미터를 읽을 수 있습니다. IaC 파일에서 파라미터 값을 읽으려면 AWS Proton 파라미터 네임스페이스에 있는 파라미터 이름을 참조하세요.

- 입력 파라미터 - `environment.inputs.`*`input-name`* 참조를 통해 환경 입력 값을 읽습니다.
- 리소스 파라미터 -와 같은 이름을 참조하여 AWS Proton 리소스 파라미터를 읽습니다`environment.name`.

> **Note**
>
> 환경 IaC 파일에는 다른 리소스의 출력 파라미터를 사용할 수 없습니다.

파라미터가 포함된 예제 환경 및 서비스 IaC 파일

다음 예제는 환경 IaC 파일의 파라미터 정의 및 참조를 보여줍니다. 그런 다음 예제는 환경 IaC 파일에 정의된 환경 출력 파라미터를 서비스 IaC 파일에서 참조할 수 있는 방법을 보여줍니다.

Example환경 CloudFormation IaC 파일

이 예제에서는 다음 사항에 유의합니다.

- `environment.inputs.` 네임스페이스는 환경 입력 파라미터를 나타냅니다.
- EC2 Systems Manager (SSM) `StoreInputValue` 파라미터는 환경 입력을 연결합니다.

- MyEnvParameterValue 출력에는 출력 파라미터와 동일한 입력 파라미터 연결이 표시됩니다. 세 개의 추가 출력 파라미터도 입력 파라미터를 개별적으로 노출합니다.
- 6개의 추가 출력 파라미터는 환경이 제공하는 리소스를 노출합니다.

```
Resources:
  StoreInputValue:
    Type: AWS::SSM::Parameter
    Properties:
      Type: String
      Value: "{{ environment.inputs.my_sample_input }}
 {{ environment.inputs.my_other_sample_input}}
 {{ environment.inputs.another_optional_input }}"
              # input parameter references

# These output values are available to service infrastructure as code files as outputs,
 when given the
# the 'environment.outputs' namespace, for example,
 service_instance.environment.outputs.ClusterName.
Outputs:
  MyEnvParameterValue:                                    # output definition
    Value: !GetAtt StoreInputValue.Value
  MySampleInputValue:                                     # output definition
    Value: "{{ environment.inputs.my_sample_input }}"         #   input parameter
 reference
  MyOtherSampleInputValue:                                # output definition
    Value: "{{ environment.inputs.my_other_sample_input }}"   #   input parameter
 reference
  AnotherOptionalInputValue:                              # output definition
    Value: "{{ environment.inputs.another_optional_input }}"  #   input parameter
 reference
  ClusterName:                                            # output definition
    Description: The name of the ECS cluster
    Value: !Ref 'ECSCluster'                              #   provisioned resource
  ECSTaskExecutionRole:                                   # output definition
    Description: The ARN of the ECS role
    Value: !GetAtt 'ECSTaskExecutionRole.Arn'             #   provisioned resource
  VpcId:                                                  # output definition
    Description: The ID of the VPC that this stack is deployed in
    Value: !Ref 'VPC'                                     #   provisioned resource
  PublicSubnetOne:                                        # output definition
    Description: Public subnet one
    Value: !Ref 'PublicSubnetOne'                         #   provisioned resource
```

```
  PublicSubnetTwo:                                          # output definition
    Description: Public subnet two
    Value: !Ref 'PublicSubnetTwo'                           #   provisioned resource
  ContainerSecurityGroup:                                   # output definition
    Description: A security group used to allow Fargate containers to receive traffic
    Value: !Ref 'ContainerSecurityGroup'                    #   provisioned resource
```

Example서비스 클라우드포메이션 IAC 파일

`environment.outputs.` 네임스페이스는 환경 IaC 파일의 환경 출력을 나타냅니다. 예를 들어 이름 `environment.outputs.ClusterName`은 `ClusterName` 환경 출력 파라미터의 값을 읽습니다.

```
AWSTemplateFormatVersion: '2010-09-09'
Description: Deploy a service on AWS Fargate, hosted in a public subnet, and accessible
 via a public load balancer.
Mappings:
  TaskSize:
    x-small:
      cpu: 256
      memory: 512
    small:
      cpu: 512
      memory: 1024
    medium:
      cpu: 1024
      memory: 2048
    large:
      cpu: 2048
      memory: 4096
    x-large:
      cpu: 4096
      memory: 8192
Resources:
  # A log group for storing the stdout logs from this service's containers
  LogGroup:
    Type: AWS::Logs::LogGroup
    Properties:
      LogGroupName: '{{service_instance.name}}' # resource parameter

  # The task definition. This is a simple metadata description of what
  # container to run, and what resource requirements it has.
  TaskDefinition:
    Type: AWS::ECS::TaskDefinition
```

```
    Properties:
      Family: '{{service_instance.name}}' # resource parameter
      Cpu: !FindInMap [TaskSize, {{service_instance.inputs.task_size}}, cpu] # input
parameter
      Memory: !FindInMap [TaskSize, {{service_instance.inputs.task_size}}, memory]
      NetworkMode: awsvpc
      RequiresCompatibilities:
        - FARGATE
      ExecutionRoleArn: '{{environment.outputs.ECSTaskExecutionRole}}'  # output
reference to an environment infrastructure code file
      TaskRoleArn: !Ref "AWS::NoValue"
      ContainerDefinitions:
        - Name: '{{service_instance.name}}'  # resource parameter
          Cpu: !FindInMap [TaskSize, {{service_instance.inputs.task_size}}, cpu]
          Memory: !FindInMap [TaskSize, {{service_instance.inputs.task_size}}, memory]
          Image: '{{service_instance.inputs.image}}'
          PortMappings:
            - ContainerPort: '{{service_instance.inputs.port}}' # input parameter
          LogConfiguration:
            LogDriver: 'awslogs'
            Options:
              awslogs-group: '{{service_instance.name}}' # resource parameter
              awslogs-region: !Ref 'AWS::Region'
              awslogs-stream-prefix: '{{service_instance.name}}' # resource parameter

  # The service_instance. The service is a resource which allows you to run multiple
  # copies of a type of task, and gather up their logs and metrics, as well
  # as monitor the number of running tasks and replace any that have crashed
  Service:
    Type: AWS::ECS::Service
    DependsOn: LoadBalancerRule
    Properties:
      ServiceName: '{{service_instance.name}}'  # resource parameter
      Cluster: '{{environment.outputs.ClusterName}}' # output reference to an
environment infrastructure as code file
      LaunchType: FARGATE
      DeploymentConfiguration:
        MaximumPercent: 200
        MinimumHealthyPercent: 75
      DesiredCount: '{{service_instance.inputs.desired_count}}'# input parameter
      NetworkConfiguration:
        AwsvpcConfiguration:
          AssignPublicIp: ENABLED
          SecurityGroups:
```

```
            - '{{environment.outputs.ContainerSecurityGroup}}' # output reference to an
 environment infrastructure as code file
          Subnets:
            - '{{environment.outputs.PublicSubnetOne}}' # output reference to an
 environment infrastructure as code file
            - '{{environment.outputs.PublicSubnetTwo}}' # output reference to an
 environment infrastructure as code file
      TaskDefinition: !Ref 'TaskDefinition'
      LoadBalancers:
        - ContainerName: '{{service_instance.name}}'  # resource parameter
          ContainerPort: '{{service_instance.inputs.port}}' # input parameter
          TargetGroupArn: !Ref 'TargetGroup'
[...]
```

CloudFormation IaC 파일 파라미터 세부 정보 및 예제

서비스 및 파이프라인 인프라에서 파라미터를 코드형 인프라(IaC) 파일로 정의하고 참조할 수 있습니다. AWS Proton 파라미터, 파라미터 유형, 파라미터 네임스페이스 및 IaC 파일의 파라미터 사용 방법에 대한 자세한 설명은 the section called “파라미터”을 참조하세요.

서비스 파라미터를 정의하세요.

서비스 IaC 파일의 입력 및 출력 파라미터를 모두 정의할 수 있습니다.

- 입력 파라미터 — 스키마 파일에 서비스 인스턴스 입력 파라미터를 정의합니다.

  다음 목록에는 일반적인 사용 사례에 대한 서비스 입력 파라미터의 예가 나와 있습니다.

  - 포트
  - 태스크 크기
  - 이미지
  - 원하는 개수
  - Docker 파일
  - 유닛 테스트 명령

  서비스를 생성할 때 입력 파라미터 값을 제공합니다.

  - 콘솔을 사용하여가 AWS Proton 제공하는 스키마 기반 양식을 작성합니다.
  - CLI를 사용하여 값이 포함된 사양을 제공하세요.
- 출력 파라미터 — 서비스 IaC 파일에서 서비스 인스턴스 출력을 정의합니다. 그런 다음 다른 리소스의 IaC 파일에서 이러한 출력을 참조할 수 있습니다.

서비스 IaC 파일에서 파라미터 값 읽기

서비스 IaC 파일에서 서비스 및 기타 리소스와 관련된 파라미터를 읽을 수 있습니다. 파라미터 네임스페이스에서 파라미터 이름을 참조하여 AWS Proton 파라미터 값을 읽습니다.

- 입력 파라미터 - `service_instance.inputs.`*`input-name`*를 참조하여 서비스 인스턴스 입력 값을 읽습니다.
- 리소스 파라미터 - `service.name`, `service_instance.name`및와 같은 이름을 참조하여 AWS Proton 리소스 파라미터를 읽습니다`environment.name`.
- 출력 파라미터 - `environment.outputs.`*`output-name`* 또는 `service_instance.components.default.outputs.`*`output-name`*를 참조하여 다른 리소스의 출력을 읽습니다.

파라미터가 포함된 서비스 IaC 파일 예시

다음 예시는 서비스 클라우드포메이션 IaC 파일의 스니펫입니다. `environment.outputs.` 네임스페이스는 환경 IaC 파일의 환경 출력을 나타냅니다. `service_instance.inputs.` 네임스페이스는 서비스 인스턴스 입력 파라미터를 나타냅니다. `service_instance.name` 속성은 AWS Proton 리소스 파라미터를 나타냅니다.

```
Resources:
  StoreServiceInstanceInputValue:
    Type: AWS::SSM::Parameter
    Properties:
      Type: String
      Value: "{{ service.name }} {{ service_instance.name }}
 {{ service_instance.inputs.my_sample_service_instance_required_input }}
 {{ service_instance.inputs.my_sample_service_instance_optional_input }}
 {{ environment.outputs.MySampleInputValue }}
 {{ environment.outputs.MyOtherSampleInputValue }}"
              #  resource parameter references               # input parameter
 references
                                          # output references to an environment
 infrastructure as code file
Outputs:
  MyServiceInstanceParameter:                                                  #
 output definition
    Value: !Ref StoreServiceInstanceInputValue
  MyServiceInstanceRequiredInputValue:                                         #
 output definition
```

```
    Value: "{{ service_instance.inputs.my_sample_service_instance_required_input }}"  #
input parameter reference
 MyServiceInstanceOptionalInputValue:                                                  #
output definition
    Value: "{{ service_instance.inputs.my_sample_service_instance_optional_input }}"  #
input parameter reference
 MyServiceInstancesEnvironmentSampleOutputValue:                                       #
output definition
    Value: "{{ environment.outputs.MySampleInputValue }}"                             #
output reference to an environment IaC file
 MyServiceInstancesEnvironmentOtherSampleOutputValue:                                  #
output definition
    Value: "{{ environment.outputs.MyOtherSampleInputValue }}"                        #
output reference to an environment IaC file
```

구성 요소 CloudFormation IaC 파일 파라미터 세부 정보 및 예제

구성요소 코드형 인프라(IaC) 파일에서 파라미터를 정의하고 참조할 수 있습니다. AWS Proton 파라미터, 파라미터 유형, 파라미터 네임스페이스 및 IaC 파일에서 파라미터를 사용하는 방법에 대한 자세한 설명은 섹션을 참조하세요the section called “파라미터”. 구성 요소에 대한 자세한 내용은 구성 요소를 참조하세요.

구성 요소 출력 파라미터를 정의합니다.

구성 요소 IaC 파일에서 출력 파라미터를 정의할 수 있습니다. 다른 리소스의 서비스 IaC 파일에서 이러한 출력을 참조할 수 있습니다.

> **Note**
>
> 구성 요소 IaC 파일의 입력은 정의할 수 없습니다. 연결된 구성 요소는 연결된 서비스 인스턴스에서 입력을 가져올 수 있습니다. 분리된 구성 요소에는 입력이 없습니다.

구성 요소 IaC 파일에서 파라미터 값 읽기

구성 요소와 관련된 파라미터 및 구성 요소 IaC 파일에서 다른 리소스를 읽을 수 있습니다. 파라미터 네임스페이스에서 파라미터 이름을 참조하여 AWS Proton 파라미터 값을 읽습니다.

- 입력 파라미터 - `service_instance.inputs.`*`input-name`*를 참조하여 첨부된 서비스 인스턴스 입력 값을 읽습니다.

- 리소스 파라미터 - `component.name`, `service.name`, `service_instance.name`, 등의 이름을 참조하여 AWS Proton 리소스 파라미터를 읽습니다`environment.name`.
- 출력 파라미터 - `environment.outputs.`*`output-name`* 참조를 통해 환경 출력을 읽습니다.

파라미터가 포함된 구성 요소 및 서비스 IaC 파일 예시

다음 예에서는 Simple Storage Service(S3) 버킷과 관련 액세스 정책을 프로비저닝하고 두 리소스의 리소스 이름(ARN) 을 구성 요소 출력으로 노출하는 구성 요소를 보여줍니다. 서비스 IaC 템플릿은 구성 요소 출력을 Elastic Container Service(ECS) 작업의 컨테이너 환경 변수로 추가하여 컨테이너에서 실행 중인 코드에서 출력을 사용할 수 있도록 하고, 작업 역할에 버킷 액세스 정책을 추가합니다. 버킷 이름은 환경, 서비스, 서비스 인스턴스 및 구성 요소의 이름을 기반으로 합니다. 즉, 버킷은 특정 서비스 인스턴스를 확장하는 구성 요소 템플릿의 특정 인스턴스와 결합됩니다. 개발자는 이 구성 요소 템플릿을 기반으로 여러 사용자 지정 구성 요소를 생성하여 다양한 서비스 인스턴스 및 기능 요구 사항에 맞게 S3 버킷을 프로비저닝할 수 있습니다.

이 예제는 Jinja `{{ ... }}` 구문을 사용하여 서비스 IaC 파일의 구성 요소 및 기타 리소스 파라미터를 참조하는 방법을 보여줍니다. 구성 요소가 서비스 인스턴스에 연결된 경우에만 `{% if ... %}` 명령문을 사용하여 명령문 블록을 추가할 수 있습니다. `proton_cfn_*` 키워드는 출력 파라미터 값을 삭제하고 형식을 지정하는 데 사용할 수 있는 필터입니다. 필터에 대한 자세한 내용은 the section called “CloudFormation 파라미터 필터” 단원을 참조하세요.

관리자는 서비스 IaC 템플릿 파일을 작성합니다.

Example구성 요소를 사용한 서비스 CloudFormation IaC 파일

```
# service/instance_infrastructure/cloudformation.yaml

Resources:
  TaskDefinition:
    Type: AWS::ECS::TaskDefinition
    Properties:
      TaskRoleArn: !Ref TaskRole
      ContainerDefinitions:
        - Name: '{{service_instance.name}}'
          # ...
          {% if service_instance.components.default.outputs | length > 0 %}
          Environment:
            {{ service_instance.components.default.outputs |
                proton_cfn_ecs_task_definition_formatted_env_vars }}
          {% endif %}
```

```
  # ...

  TaskRole:
    Type: AWS::IAM::Role
    Properties:
      # ...
      ManagedPolicyArns:
        - !Ref BaseTaskRoleManagedPolicy
        {{ service_instance.components.default.outputs
            | proton_cfn_iam_policy_arns }}

  # Basic permissions for the task
  BaseTaskRoleManagedPolicy:
    Type: AWS::IAM::ManagedPolicy
    Properties:
      # ...
```

개발자는 구성 요소 IaC 템플릿 파일을 작성합니다.

Example컴포넌트 CloudFormation IaC 파일

```
# cloudformation.yaml

# A component that defines an S3 bucket and a policy for accessing the bucket.
Resources:
  S3Bucket:
    Type: AWS::S3::Bucket
    Properties:
      BucketName: '{{environment.name}}-{{service.name}}-{{service_instance.name}}-
{{component.name}}'
  S3BucketAccessPolicy:
    Type: AWS::IAM::ManagedPolicy
    Properties:
      PolicyDocument:
        Version: "2012-10-17"
        Statement:
          - Effect: Allow
            Action:
              - 's3:Get*'
              - 's3:List*'
              - 's3:PutObject'
            Resource: !GetAtt S3Bucket.Arn
Outputs:
```

```
  BucketName:
    Description: "Bucket to access"
    Value: !GetAtt S3Bucket.Arn
  BucketAccessPolicyArn:
    Value: !Ref S3BucketAccessPolicy
```

가 서비스 인스턴스에 대한 CloudFormation 템플릿을 AWS Proton 렌더링하고 모든 파라미터를 실제 값으로 대체하는 경우 템플릿은 다음 파일과 같을 수 있습니다.

Example서비스 인스턴스 CloudFormation 렌더링 IaC 파일

```
Resources:
  TaskDefinition:
    Type: AWS::ECS::TaskDefinition
    Properties:
      TaskRoleArn: !Ref TaskRole
      ContainerDefinitions:
        - Name: '{{service_instance.name}}'
          # ...
          Environment:
            - Name: BucketName
              Value: arn:aws:s3:us-
east-1:123456789012:environment_name-service_name-service_instance_name-component_name
            - Name: BucketAccessPolicyArn
              Value: arn:aws:iam::123456789012:policy/cfn-generated-policy-name
  # ...

  TaskRole:
    Type: AWS::IAM::Role
    Properties:
      # ...
      ManagedPolicyArns:
        - !Ref BaseTaskRoleManagedPolicy
        - arn:aws:iam::123456789012:policy/cfn-generated-policy-name

  # Basic permissions for the task
  BaseTaskRoleManagedPolicy:
    Type: AWS::IAM::ManagedPolicy
    Properties:
      # ...
```

CloudFormation IaC 파일용 파라미터 필터

AWS CloudFormation IaC 파일의 AWS Proton 파라미터를 참조할 때 필터라고 하는 Jinja 한정자를 사용하여 파라미터 값을 렌더링된 템플릿에 삽입하기 전에 검증, 필터링 및 형식을 지정할 수 있습니다. 구성 요소 생성 및 첨부는 개발자가 수행하므로 서비스 인스턴스 템플릿에서 구성 요소 출력을 사용하는 관리자는 구성 요소 출력 파라미터의 존재 여부와 유효성을 확인하고자 할 수 있으므로 필터 검증은 구성 요소 출력 파라미터를 참조할 때 특히 유용합니다. 하지만 Jinja IaC 파일에서는 필터를 사용할 수 있습니다.

다음 섹션에서는 사용 가능한 파라미터 필터를 설명하고 정의하며 example. AWS Proton defines를 제공합니다. `default`은 Jinja 내장 필터입니다.

ECS 작업의 환경 속성 형식 지정

선언

```
dict # proton_cfn_ecs_task_definition_formatted_env_vars (raw: boolean = True) # YAML
 list of dicts
```

설명

이 필터는 Elastic Container Service(ECS) 태스크 정의 `ContainerDefinition`섹션의 환경 속성에 사용될 출력 목록의 형식을 지정합니다.

`raw`을 `False`로 설정하여 파라미터 값의 유효성도 확인합니다. 이 경우 값은 정규 표현식 `^[a-zA-Z0-9_-]*$`과 일치해야 합니다. 값이 이 검증에 실패하면 템플릿 렌더링이 실패합니다.

예제

다음과 같은 사용자 지정 구성 요소 템플릿 사용:

```
Resources:
  # ...
Outputs:
  Output1:
    Description: "Example component output 1"
    Value: hello
  Output2:
    Description: "Example component output 2"
    Value: world
```

그리고 다음 서비스 템플릿:

```
Resources:
  TaskDefinition:
    Type: AWS::ECS::TaskDefinition
    Properties:
      # ...
      ContainerDefinitions:
        - Name: MyServiceName
          # ...
          Environment:
            {{ service_instance.components.default.outputs
              | proton_cfn_ecs_task_definition_formatted_env_vars }}
```

렌더링된 서비스 템플릿은 다음과 같습니다.

```
Resources:
  TaskDefinition:
    Type: AWS::ECS::TaskDefinition
    Properties:
      # ...
      ContainerDefinitions:
        - Name: MyServiceName
          # ...
          Environment:
            - Name: Output1
              Value: hello
            - Name: Output2
              Value: world
```

Lambda 함수의 환경 속성 형식 지정

선언

```
dict # proton_cfn_lambda_function_formatted_env_vars (raw: boolean = True) # YAML dict
```

설명

이 필터는 AWS Lambda 함수 정의의 `Properties` 섹션에 있는 환경 속성에서 사용할 출력 목록의 형식을 지정합니다.

`raw`을 `False`로 설정하여 파라미터 값의 유효성도 확인합니다. 이 경우 값은 정규 표현식 `^[a-zA-Z0-9_-]*$`과 일치해야 합니다. 값이 이 검증에 실패하면 템플릿 렌더링이 실패합니다.

예제

다음과 같은 사용자 지정 구성 요소 템플릿 사용:

```
Resources:
  # ...
Outputs:
  Output1:
    Description: "Example component output 1"
    Value: hello
  Output2:
    Description: "Example component output 2"
    Value: world
```

그리고 다음 서비스 템플릿:

```
Resources:
  Lambda:
    Type: AWS::Lambda::Function
    Properties:
      Environment:
        Variables:
          {{ service_instance.components.default.outputs
            | proton_cfn_lambda_function_formatted_env_vars }}
```

렌더링된 서비스 템플릿은 다음과 같습니다.

```
Resources:
  Lambda:
    Type: AWS::Lambda::Function
    Properties:
      Environment:
        Variables:
          Output1: hello
          Output2: world
```

IAM 정책 ARN을 추출하여 IAM 역할에 포함시킵니다.

선언

```
dict # proton_cfn_iam_policy_arns # YAML list
```

설명

이 필터는 AWS Identity and Access Management (IAM) 역할 정의의 `Properties` 섹션에 있는 ManagedPolicyArns 속성에서 사용할 출력 목록의 형식을 지정합니다. 필터는 정규 표현식 `^arn:[a-zA-Z-]+:iam::\d{12}:policy/`을 사용하여 출력 파라미터 목록에서 유효한 IAM 정책 ARN을 추출합니다. 이 필터를 사용하여 출력 파라미터 값의 정책을 서비스 템플릿의 IAM 역할 정의에 추가할 수 있습니다.

예제

다음과 같은 사용자 지정 구성 요소 템플릿 사용:

```
Resources:
  # ...
  ExamplePolicy1:
    Type: AWS::IAM::ManagedPolicy
    Properties:
      # ...
  ExamplePolicy2:
    Type: AWS::IAM::ManagedPolicy
    Properties:
      # ...

  # ...

Outputs:
  Output1:
    Description: "Example component output 1"
    Value: hello
  Output2:
    Description: "Example component output 2"
    Value: world
  PolicyArn1:
    Description: "ARN of policy 1"
    Value: !Ref ExamplePolicy1
  PolicyArn2:
    Description: "ARN of policy 2"
    Value: !Ref ExamplePolicy2
```

그리고 다음 서비스 템플릿:

```
Resources:
```

```
  # ...

  TaskRole:
    Type: AWS::IAM::Role
    Properties:
      # ...
      ManagedPolicyArns:
        - !Ref BaseTaskRoleManagedPolicy
        {{ service_instance.components.default.outputs
            | proton_cfn_iam_policy_arns }}

  # Basic permissions for the task
  BaseTaskRoleManagedPolicy:
    Type: AWS::IAM::ManagedPolicy
    Properties:
      # ...
```

렌더링된 서비스 템플릿은 다음과 같습니다.

```
Resources:

  # ...

  TaskRole:
    Type: AWS::IAM::Role
    Properties:
      # ...
      ManagedPolicyArns:
        - !Ref BaseTaskRoleManagedPolicy
        - arn:aws:iam::123456789012:policy/cfn-generated-policy-name-1
        - arn:aws:iam::123456789012:policy/cfn-generated-policy-name-2

  # Basic permissions for the task
  BaseTaskRoleManagedPolicy:
    Type: AWS::IAM::ManagedPolicy
    Properties:
      # ...
```

속성 값 삭제

선언

```
string # proton_cfn_sanitize # string
```

설명

범용 필터입니다. 이를 사용하여 파라미터 값의 안전성을 검증할 수 있습니다. 필터는 값이 정규 표현식 ^[a-zA-Z0-9_-]*$과 일치하는지 또는 유효한 리소스 이름(ARN) 인지 확인합니다. 값이 이 검증에 실패하면 템플릿 렌더링이 실패합니다.

예제

다음과 같은 사용자 지정 구성 요소 템플릿 사용:

```
Resources:
  # ...
Outputs:
  Output1:
    Description: "Example of valid output"
    Value: "This-is_valid_37"
  Output2:
    Description: "Example incorrect output"
    Value: "this::is::incorrect"
  SomeArn:
    Description: "Example ARN"
    Value: arn:aws:some-service::123456789012:some-resource/resource-name
```

- 서비스 템플릿의 다음 참조:

  ```
  # ...
    {{ service_instance.components.default.outputs.Output1
      | proton_cfn_sanitize }}
  ```

  다음과 같이 렌더링됩니다.

  ```
  # ...
    This-is_valid_37
  ```

- 서비스 템플릿의 다음 참조:

  ```
  # ...
    {{ service_instance.components.default.outputs.Output2
  ```

```
    | proton_cfn_sanitize }}
```

다음과 같은 렌더링 오류가 발생한 결과:

```
Illegal character(s) detected in "this::is::incorrect". Must match regex ^[a-zA-
Z0-9_-]*$ or be a valid ARN
```

- 서비스 템플릿의 다음 참조:

```
# ...
  {{ service_instance.components.default.outputs.SomeArn
    | proton_cfn_sanitize }}
```

다음과 같이 렌더링됩니다.

```
# ...
  arn:aws:some-service::123456789012:some-resource/resource-name
```

존재하지 않는 참조에 대한 기본값 제공

설명

`default` 필터는 네임스페이스 참조가 없는 경우 기본값을 제공합니다. 이를 사용하여 참조하는 파라미터가 누락된 경우에도 오류 없이 렌더링할 수 있는 강력한 템플릿을 작성할 수 있습니다.

예제

서비스 템플릿의 다음 참조를 사용하면 서비스 인스턴스에 직접 정의된 (기본) 구성 요소가 첨부되어 있지 않거나 연결된 구성 요소에 이름이 지정된 `test` 출력이 없는 경우 템플릿 렌더링이 실패합니다.

```
# ...
  {{ service_instance.components.default.outputs.test }}
```

이 문제를 방지하려면 `default` 필터를 추가합니다.

```
# ...
  {{ service_instance.components.default.outputs.test | default("[optional-value]") }}
```

CodeBuild 프로비저닝 파라미터 세부 정보 및 예제

템플릿에서 CodeBuild 기반 AWS Proton 리소스에 대한 파라미터를 정의하고 프로비저닝 코드에서 이러한 파라미터를 참조할 수 있습니다. AWS Proton 파라미터, 파라미터 유형, 파라미터 네임스페이스 및 IaC 파일에서 파라미터를 사용하는 방법에 대한 자세한 설명은 섹션을 참조하세요the section called “파라미터”.

> **Note**
>
> CodeBuild 프로비저닝을 환경 및 서비스와 함께 사용할 수 있습니다. 현재로서는 이 방법으로 구성 요소를 프로비저닝할 수 없습니다.

입력 파라미터

환경 또는 서비스와 같은 AWS Proton 리소스를 생성할 때 템플릿의 스키마 파일에 정의된 입력 파라미터 값을 제공합니다. 생성한 리소스가를 사용하는 경우 CodeBuild 프로비저닝는 이러한 입력 값을 입력 파일로 AWS Proton 렌더링합니다. 프로비저닝 코드는 이 파일에서 파라미터 값을 가져오고 가져올 수 있습니다.

CodeBuild 템플릿의 예는 the section called “CodeBuild 번들”을 참조하세요. 매니페스트 파일에 대한 자세한 내용은 the section called “매니페스트 및 마무리” 단원을 참조하세요.

다음 예제는 서비스 인스턴스의 CodeBuild 기반 프로비저닝 중에 생성된 JSON 입력 파일입니다.

예: CodeBuild 프로비저닝과 AWS CDK 함께 사용

```
{
  "service_instance": {
    "name": "my-service-staging",
    "inputs": {
      "port": "8080",
      "task_size": "medium"
    }
  },
  "service": {
    "name": "my-service"
  },
  "environment": {
    "account_id": "123456789012",
    "name": "my-env-staging",
```

```
    "outputs": {
      "vpc-id": "hdh2323423"
    }
  }
}
```

출력 파라미터

리소스 프로비저닝 출력을에 다시 전달하기 위해 AWS Proton프로비저닝 코드는 템플릿의 스키마 파일에 정의된 출력 파라미터 값을 `proton-outputs.json` 사용하여 라는 JSON 파일을 생성할 수 있습니다. ??? 예를 들어 `cdk deploy` 명령에는 프로비저닝 출력이 있는 JSON 파일을 생성 AWS CDK 하도록에 지시하는 `--outputs-file` 인수가 있습니다. 리소스가를 사용하는 경우 CodeBuild 템플릿 매니페스트에 다음 명령을 AWS CDK지정합니다.

```
aws proton notify-resource-deployment-status-change
```

AWS Proton 는이 JSON 파일을 찾습니다. 프로비저닝 코드가 성공적으로 완료된 후 파일이 있는 경우는 파일에서 출력 파라미터 값을 AWS Proton 읽습니다.

Terraform 코드형 인프라(IaC) 파일 파라미터 세부 정보 및 예제

템플릿 번들의 `variable.tf` 파일에 Terraform 입력 변수를 포함할 수 있습니다. 스키마를 생성하여 관리형 변수를 생성할 AWS Proton 수도 있습니다.는 스키마 파일`.tf files`에서 변수를 AWS Proton 생성합니다. 자세한 내용은 the section called “Terraform IaC 파일” 단원을 참조하십시오.

인프라에서 스키마 정의 AWS Proton 변수를 참조하려면 Terraform IaC 테이블의 파라미터 및 네임스페이스에 AWS Proton 표시된 네임스페이스를 `.tf files`사용합니다. IaC 예를 들어 `var.environment.inputs.vpc_cidr`를 사용할 수 있습니다. 따옴표 안에 이러한 변수를 대괄호로 묶고 첫 번째 중괄호 앞에 달러 기호를 추가합니다(예: “`${var.environment.inputs.vpc_cidr}`”).

다음 예제에서는 네임스페이스를 사용하여 환경에 AWS Proton 파라미터를 포함하는 방법을 보여줍니다`.tf file`.

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 3.0"
    }
```

```
  }
  // This tells terraform to store the state file in s3 at the location
  // s3://terraform-state-bucket/tf-os-sample/terraform.tfstate
  backend "s3" {
    bucket = "terraform-state-bucket"
    key    = "tf-os-sample/terraform.tfstate"
    region = "us-east-1"
  }
}

// Configure the AWS Provider
provider "aws" {
  region = "us-east-1"
  default_tags {
    tags = var.proton_tags
  }
}

resource "aws_ssm_parameter" "my_ssm_parameter" {
  name  = "my_ssm_parameter"
  type  = "String"
  // Use the Proton environment.inputs. namespace
  value = var.environment.inputs.ssm_parameter_value
}
```

AWS Proton 코드형 인프라 파일

템플릿 번들의 기본 부분은 프로비저닝하려는 인프라 리소스 및 속성을 정의하는 코드형 인프라(IaC) 파일입니다. 코드형 인프라 엔진 AWS CloudFormation 은 이러한 유형의 파일을 사용하여 인프라 리소스를 프로비저닝합니다.

> **Note**
>
> IaC 파일은 템플릿 번들과 독립적으로 직접 정의된 구성 요소에 대한 직접 입력으로 사용할 수도 있습니다. 구성 요소에 대한 자세한 내용은 구성 요소를 참조하세요.

AWS Proton 는 현재 두 가지 유형의 IaC 파일을 지원합니다.

- CloudFormation 파일 — AWS관리형 프로비저닝에 사용됩니다. AWS Proton 는 파라미터화를 위해 CloudFormation 템플릿 파일 형식 위에 Jinja를 사용합니다.

- Terraform HCL - 자체 관리형 프로비저닝에 사용됩니다. HCL은 기본적으로 파라미터화를 지원합니다.

프로비저닝 방법 조합을 사용하여 AWS Proton 리소스를 프로비저닝할 수 없습니다. 단, 둘 중 하나만 사용해야 합니다. 자체 AWS관리형 프로비저닝 환경에는 관리형 프로비저닝 서비스를 배포할 수 없으며, 그 반대의 경우도 마찬가지입니다.

자세한 내용은 the section called “프로비저닝 방법”, 환경, 서비스 및 구성 요소 부분을 참조하세요.

CloudFormation IaC 파일

에서 코드 파일로 AWS CloudFormation 인프라를 사용하는 방법을 알아봅니다 AWS Proton. CloudFormation 는 AWS 리소스를 모델링하고 설정하는 데 도움이 되는 코드형 인프라(IaC) 서비스입니다. parametrization. AWS Proton expands 파라미터에 대한 CloudFormation 템플릿 파일 형식 위에 Jinja를 사용하여 템플릿에서 인프라 리소스를 정의하고 전체 CloudFormation 템플릿을 렌더링합니다. CloudFormation은 정의된 리소스를 CloudFormation 스택으로 프로비저닝합니다. 자세한 정보는 CloudFormation 사용 설명서의 CloudFormation(이)란 무엇입니까? 를 참조하세요.

AWS Proton 는 CloudFormation IaC에 대한 AWS관리형 프로비저닝을 지원합니다.

기존 인프라를 코드 파일로 사용하여 시작하세요.

와 함께 사용할 기존 코드형 인프라(IaC) 파일을 조정할 수 있습니다 AWS Proton.

다음 CloudFormation 예제인 예제 1과 예제 2는 기존 CloudFormation IaC 파일을 나타냅니다. CloudFormation은 이러한 파일을 사용하여 두 개의 서로 다른 CloudFormation 스택을 생성할 수 있습니다.

예제 1에서 CloudFormation IaC 파일은 컨테이너 애플리케이션 간에 공유할 인프라를 프로비저닝하도록 구성되어 있습니다. 이 예시에서는 동일한 IaC 파일을 사용하여 프로비저닝된 인프라 세트를 여러 개 생성할 수 있도록 입력 파라미터가 추가되었습니다. 각 세트는 서로 다른 VPC 및 서브넷 CIDR 값 세트와 함께 서로 다른 이름을 가질 수 있습니다. 관리자 또는 개발자는 IaC 파일을 사용하여 CloudFormation으로 인프라 리소스를 프로비저닝할 때 이러한 파라미터에 값을 제공합니다. 편의를 위해 이 예제에서는 이러한 입력 파라미터가 주석으로 표시되고 여러 번 참조됩니다. 출력은 템플릿 끝에 정의됩니다. 다른 CloudFormation IaC 파일에서 참조할 수 있습니다.

예제 2에서 CloudFormation IaC 파일은 예제 1에서 프로비저닝된 인프라에 애플리케이션을 배포하도록 구성되어 있습니다. 편의를 위해 파라미터가 주석 처리되어 있습니다.

예제 1: CloudFormation IaC 파일

```
AWSTemplateFormatVersion: '2010-09-09'
Description: AWS Fargate cluster running containers in a public subnet. Only supports
             public facing load balancer, and public service discovery namespaces.
Parameters:
   VpcCIDR:       # input parameter
        Description: CIDR for VPC
        Type: String
        Default: "10.0.0.0/16"
   SubnetOneCIDR: # input parameter
        Description: CIDR for SubnetOne
        Type: String
        Default: "10.0.0.0/24"
   SubnetTwoCIDR: # input parameters
        Description: CIDR for SubnetTwo
        Type: String
        Default: "10.0.1.0/24"
Resources:
  VPC:
    Type: AWS::EC2::VPC
    Properties:
      EnableDnsSupport: true
      EnableDnsHostnames: true
      CidrBlock:
        Ref: 'VpcCIDR'

  # Two public subnets, where containers will have public IP addresses
  PublicSubnetOne:
    Type: AWS::EC2::Subnet
    Properties:
      AvailabilityZone:
         Fn::Select:
         - 0
         - Fn::GetAZs: {Ref: 'AWS::Region'}
      VpcId: !Ref 'VPC'
      CidrBlock:
         Ref: 'SubnetOneCIDR'
      MapPublicIpOnLaunch: true

  PublicSubnetTwo:
    Type: AWS::EC2::Subnet
    Properties:
      AvailabilityZone:
```

```
        Fn::Select:
        - 1
        - Fn::GetAZs: {Ref: 'AWS::Region'}
      VpcId: !Ref 'VPC'
      CidrBlock:
        Ref: 'SubnetTwoCIDR'
      MapPublicIpOnLaunch: true

  # Setup networking resources for the public subnets. Containers
  # in the public subnets have public IP addresses and the routing table
  # sends network traffic via the internet gateway.
  InternetGateway:
    Type: AWS::EC2::InternetGateway
  GatewayAttachement:
    Type: AWS::EC2::VPCGatewayAttachment
    Properties:
      VpcId: !Ref 'VPC'
      InternetGatewayId: !Ref 'InternetGateway'
  PublicRouteTable:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref 'VPC'
  PublicRoute:
    Type: AWS::EC2::Route
    DependsOn: GatewayAttachement
    Properties:
      RouteTableId: !Ref 'PublicRouteTable'
      DestinationCidrBlock: '0.0.0.0/0'
      GatewayId: !Ref 'InternetGateway'
  PublicSubnetOneRouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      SubnetId: !Ref PublicSubnetOne
      RouteTableId: !Ref PublicRouteTable
  PublicSubnetTwoRouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      SubnetId: !Ref PublicSubnetTwo
      RouteTableId: !Ref PublicRouteTable

  # ECS Resources
  ECSCluster:
    Type: AWS::ECS::Cluster
```

```
  # A security group for the containers we will run in Fargate.
  # Rules are added to this security group based on what ingress you
  # add for the cluster.
  ContainerSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Access to the Fargate containers
      VpcId: !Ref 'VPC'

  # This is a role which is used by the ECS tasks themselves.
  ECSTaskExecutionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
        - Effect: Allow
          Principal:
            Service: [ecs-tasks.amazonaws.com]
          Action: ['sts:AssumeRole']
      Path: /
      ManagedPolicyArns:
        - 'arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy'

# These output values will be available to other templates to use.
Outputs:
  ClusterName:                                          # output
    Description: The name of the ECS cluster
    Value: !Ref 'ECSCluster'
    Export:
      Name:
        Fn::Sub: "${AWS::StackName}-ECSCluster"
  ECSTaskExecutionRole:                                  # output
    Description: The ARN of the ECS role
    Value: !GetAtt 'ECSTaskExecutionRole.Arn'
    Export:
      Name:
        Fn::Sub: "${AWS::StackName}-ECSTaskExecutionRole"
  VpcId:                                                 # output
    Description: The ID of the VPC that this stack is deployed in
    Value: !Ref 'VPC'
    Export:
      Name:
        Fn::Sub: "${AWS::StackName}-VPC"
  PublicSubnetOne:                                       # output
```

```
    Description: Public subnet one
    Value: !Ref 'PublicSubnetOne'
    Export:
      Name:
        Fn::Sub: "${AWS::StackName}-PublicSubnetOne"
  PublicSubnetTwo:                                          # output
    Description: Public subnet two
    Value: !Ref 'PublicSubnetTwo'
    Export:
      Name:
        Fn::Sub: "${AWS::StackName}-PublicSubnetTwo"
  ContainerSecurityGroup:                                   # output
    Description: A security group used to allow Fargate containers to receive traffic
    Value: !Ref 'ContainerSecurityGroup'
    Export:
      Name:
        Fn::Sub: "${AWS::StackName}-ContainerSecurityGroup"
```

예제 2: CloudFormation IaC 파일

```
AWSTemplateFormatVersion: '2010-09-09'
Description: Deploy a service on AWS Fargate, hosted in a public subnet, and accessible
 via a public load balancer.
Parameters:
    ContainerPortInput:  # input parameter
        Description: The port to route traffic to
        Type: Number
        Default: 80
    TaskCountInput:      # input parameter
        Description: The default number of Fargate tasks you want running
        Type: Number
        Default: 1
    TaskSizeInput:       # input parameter
        Description: The size of the task you want to run
        Type: String
        Default: x-small
    ContainerImageInput: # input parameter
        Description: The name/url of the container image
        Type: String
        Default: "public.ecr.aws/z9d2n7e1/nginx:1.19.5"
    TaskNameInput:       # input parameter
        Description: Name for your task
        Type: String
```

```
        Default: "my-fargate-instance"
    StackName:           # input parameter
        Description: Name of the environment stack to deploy to
        Type: String
        Default: "my-fargate-environment"
Mappings:
  TaskSizeMap:
    x-small:
      cpu: 256
      memory: 512
    small:
      cpu: 512
      memory: 1024
    medium:
      cpu: 1024
      memory: 2048
    large:
      cpu: 2048
      memory: 4096
    x-large:
      cpu: 4096
      memory: 8192
Resources:
  # A log group for storing the stdout logs from this service's containers
  LogGroup:
    Type: AWS::Logs::LogGroup
    Properties:
      LogGroupName:
        Ref: 'TaskNameInput' # input parameter

  # The task definition. This is a simple metadata description of what
  # container to run, and what resource requirements it has.
  TaskDefinition:
    Type: AWS::ECS::TaskDefinition
    Properties:
      Family: !Ref 'TaskNameInput'
      Cpu: !FindInMap [TaskSizeMap, !Ref 'TaskSizeInput', cpu]
      Memory: !FindInMap [TaskSizeMap, !Ref 'TaskSizeInput', memory]
      NetworkMode: awsvpc
      RequiresCompatibilities:
        - FARGATE
      ExecutionRoleArn:
        Fn::ImportValue:
```

```
          !Sub "${StackName}-ECSTaskExecutionRole"    # output parameter from another
CloudFormation template
             awslogs-region: !Ref 'AWS::Region'
             awslogs-stream-prefix: !Ref 'TaskNameInput'


 # The service_instance. The service is a resource which allows you to run multiple
 # copies of a type of task, and gather up their logs and metrics, as well
 # as monitor the number of running tasks and replace any that have crashed
 Service:
   Type: AWS::ECS::Service
   DependsOn: LoadBalancerRule
   Properties:
     ServiceName: !Ref 'TaskNameInput'
     Cluster:
       Fn::ImportValue:
         !Sub "${StackName}-ECSCluster"  # output parameter from another
CloudFormation template
     LaunchType: FARGATE
     DeploymentConfiguration:
       MaximumPercent: 200
       MinimumHealthyPercent: 75
     DesiredCount: !Ref 'TaskCountInput'
     NetworkConfiguration:
       AwsvpcConfiguration:
         AssignPublicIp: ENABLED
         SecurityGroups:
           - Fn::ImportValue:
               !Sub "${StackName}-ContainerSecurityGroup"    # output parameter from
another CloudFormation template
         Subnets:
           - Fn::ImportValue:r CloudFormation template
     TaskRoleArn: !Ref "AWS::NoValue"
     ContainerDefinitions:
       - Name: !Ref 'TaskNameInput'
         Cpu: !FindInMap [TaskSizeMap, !Ref 'TaskSizeInput', cpu]
         Memory: !FindInMap [TaskSizeMap, !Ref 'TaskSizeInput', memory]
         Image: !Ref 'ContainerImageInput'             # input parameter
         PortMappings:
           - ContainerPort: !Ref 'ContainerPortInput'  # input parameter

         LogConfiguration:
           LogDriver: 'awslogs'
           Options:
```

```
              awslogs-group: !Ref 'TaskNameInput'
                !Sub "${StackName}-PublicSubnetOne"    # output parameter from another
CloudFormation template
           - Fn::ImportValue:
                !Sub "${StackName}-PublicSubnetTwo"    # output parameter from another
CloudFormation template
     TaskDefinition: !Ref 'TaskDefinition'
     LoadBalancers:
       - ContainerName: !Ref 'TaskNameInput'
         ContainerPort: !Ref 'ContainerPortInput'  # input parameter
         TargetGroupArn: !Ref 'TargetGroup'

 # A target group. This is used for keeping track of all the tasks, and
 # what IP addresses / port numbers they have. You can query it yourself,
 # to use the addresses yourself, but most often this target group is just
 # connected to an application load balancer, or network load balancer, so
 # it can automatically distribute traffic across all the targets.
 TargetGroup:
   Type: AWS::ElasticLoadBalancingV2::TargetGroup
   Properties:
     HealthCheckIntervalSeconds: 6
     HealthCheckPath: /
     HealthCheckProtocol: HTTP
     HealthCheckTimeoutSeconds: 5
     HealthyThresholdCount: 2
     TargetType: ip
     Name: !Ref 'TaskNameInput'
     Port: !Ref 'ContainerPortInput'
     Protocol: HTTP
     UnhealthyThresholdCount: 2
     VpcId:
       Fn::ImportValue:
         !Sub "${StackName}-VPC"    # output parameter from another CloudFormation
template

 # Create a rule on the load balancer for routing traffic to the target group
 LoadBalancerRule:
   Type: AWS::ElasticLoadBalancingV2::ListenerRule
   Properties:
     Actions:
       - TargetGroupArn: !Ref 'TargetGroup'
         Type: 'forward'
     Conditions:
       - Field: path-pattern
```

```
          Values:
            - '*'
      ListenerArn: !Ref PublicLoadBalancerListener
      Priority: 1

  # Enable autoscaling for this service
  ScalableTarget:
    Type: AWS::ApplicationAutoScaling::ScalableTarget
    DependsOn: Service
    Properties:
      ServiceNamespace: 'ecs'
      ScalableDimension: 'ecs:service:DesiredCount'
      ResourceId:
        Fn::Join:
          - '/'
          - - service
            - Fn::ImportValue:
                !Sub "${StackName}-ECSCluster"
            - !Ref 'TaskNameInput'
      MinCapacity: 1
      MaxCapacity: 10
      RoleARN: !Sub arn:aws:iam::${AWS::AccountId}:role/
aws-service-role/ecs.application-autoscaling.amazonaws.com/
AWSServiceRoleForApplicationAutoScaling_ECSService

  # Create scaling policies for the service
  ScaleDownPolicy:
    Type: AWS::ApplicationAutoScaling::ScalingPolicy
    DependsOn: ScalableTarget
    Properties:
      PolicyName:
        Fn::Join:
          - '/'
          - - scale
            - !Ref 'TaskNameInput'
            - down
      PolicyType: StepScaling
      ResourceId:
        Fn::Join:
          - '/'
          - - service
            - Fn::ImportValue:
                !Sub "${StackName}-ECSCluster"  # output parameter from another
 CloudFormation template
```

```
            - !Ref 'TaskNameInput'
      ScalableDimension: 'ecs:service:DesiredCount'
      ServiceNamespace: 'ecs'
      StepScalingPolicyConfiguration:
        AdjustmentType: 'ChangeInCapacity'
        StepAdjustments:
          - MetricIntervalUpperBound: 0
            ScalingAdjustment: -1
        MetricAggregationType: 'Average'
        Cooldown: 60

  ScaleUpPolicy:
    Type: AWS::ApplicationAutoScaling::ScalingPolicy
    DependsOn: ScalableTarget
    Properties:
      PolicyName:
        Fn::Join:
          - '/'
          - - scale
            - !Ref 'TaskNameInput'
            - up
      PolicyType: StepScaling
      ResourceId:
        Fn::Join:
          - '/'
          - - service
            - Fn::ImportValue:
                !Sub "${StackName}-ECSCluster"
            - !Ref 'TaskNameInput'
      ScalableDimension: 'ecs:service:DesiredCount'
      ServiceNamespace: 'ecs'
      StepScalingPolicyConfiguration:
        AdjustmentType: 'ChangeInCapacity'
        StepAdjustments:
          - MetricIntervalLowerBound: 0
            MetricIntervalUpperBound: 15
            ScalingAdjustment: 1
          - MetricIntervalLowerBound: 15
            MetricIntervalUpperBound: 25
            ScalingAdjustment: 2
          - MetricIntervalLowerBound: 25
            ScalingAdjustment: 3
        MetricAggregationType: 'Average'
        Cooldown: 60
```

```
# Create alarms to trigger these policies
LowCpuUsageAlarm:
  Type: AWS::CloudWatch::Alarm
  Properties:
    AlarmName:
      Fn::Join:
        - '-'
        - - low-cpu
          - !Ref 'TaskNameInput'
    AlarmDescription:
      Fn::Join:
        - ' '
        - - "Low CPU utilization for service"
          - !Ref 'TaskNameInput'
    MetricName: CPUUtilization
    Namespace: AWS/ECS
    Dimensions:
      - Name: ServiceName
        Value: !Ref 'TaskNameInput'
      - Name: ClusterName
        Value:
          Fn::ImportValue:
            !Sub "${StackName}-ECSCluster"
    Statistic: Average
    Period: 60
    EvaluationPeriods: 1
    Threshold: 20
    ComparisonOperator: LessThanOrEqualToThreshold
    AlarmActions:
      - !Ref ScaleDownPolicy

HighCpuUsageAlarm:
  Type: AWS::CloudWatch::Alarm
  Properties:
    AlarmName:
      Fn::Join:
        - '-'
        - - high-cpu
          - !Ref 'TaskNameInput'
    AlarmDescription:
      Fn::Join:
        - ' '
        - - "High CPU utilization for service"
```

```
            - !Ref 'TaskNameInput'
      MetricName: CPUUtilization
      Namespace: AWS/ECS
      Dimensions:
        - Name: ServiceName
          Value: !Ref 'TaskNameInput'
        - Name: ClusterName
          Value:
            Fn::ImportValue:
              !Sub "${StackName}-ECSCluster"
      Statistic: Average
      Period: 60
      EvaluationPeriods: 1
      Threshold: 70
      ComparisonOperator: GreaterThanOrEqualToThreshold
      AlarmActions:
        - !Ref ScaleUpPolicy

  EcsSecurityGroupIngressFromPublicALB:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      Description: Ingress from the public ALB
      GroupId:
        Fn::ImportValue:
          !Sub "${StackName}-ContainerSecurityGroup"
      IpProtocol: -1
      SourceSecurityGroupId: !Ref 'PublicLoadBalancerSG'

  # Public load balancer, hosted in public subnets that is accessible
  # to the public, and is intended to route traffic to one or more public
  # facing services. This is used for accepting traffic from the public
  # internet and directing it to public facing microservices
  PublicLoadBalancerSG:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Access to the public facing load balancer
      VpcId:
        Fn::ImportValue:
          !Sub "${StackName}-VPC"
      SecurityGroupIngress:
          # Allow access to ALB from anywhere on the internet
          - CidrIp: 0.0.0.0/0
            IpProtocol: -1
```

```
  PublicLoadBalancer:
    Type: AWS::ElasticLoadBalancingV2::LoadBalancer
    Properties:
      Scheme: internet-facing
      LoadBalancerAttributes:
      - Key: idle_timeout.timeout_seconds
        Value: '30'
      Subnets:
        # The load balancer is placed into the public subnets, so that traffic
        # from the internet can reach the load balancer directly via the internet
 gateway
        - Fn::ImportValue:
            !Sub "${StackName}-PublicSubnetOne"
        - Fn::ImportValue:
            !Sub "${StackName}-PublicSubnetTwo"
      SecurityGroups: [!Ref 'PublicLoadBalancerSG']

  PublicLoadBalancerListener:
    Type: AWS::ElasticLoadBalancingV2::Listener
    DependsOn:
      - PublicLoadBalancer
    Properties:
      DefaultActions:
        - TargetGroupArn: !Ref 'TargetGroup'
          Type: 'forward'
      LoadBalancerArn: !Ref 'PublicLoadBalancer'
      Port: 80
      Protocol: HTTP
# These output values will be available to other templates to use.
Outputs:
  ServiceEndpoint:        # output
    Description: The URL to access the service
    Value: !Sub "http://${PublicLoadBalancer.DNSName}"
```

이러한 파일을와 함께 사용하도록 조정할 수 있습니다 AWS Proton.

에 코드형 인프라 가져오기 AWS Proton

약간 수정하면 가 환경을 배포하는 데 사용하는 환경 템플릿 번들의 코드형 인프라(IaC) 파일로 예제 1을 사용할 수 있습니다(예제 3 참조).IaC AWS Proton

CloudFormation 파라미터를 사용하는 대신 Jinja 구문을 사용하여 Open API 기반 스키마 파일에 정의한 파라미터를 참조하세요. 이러한 입력 파라미터는 편의를 위해 주석 처리되며 IaC 파일에서 여러 번

참조됩니다. 이렇게 하면가 파라미터 값을 감사하고 확인할 AWS Proton 수 있습니다. 또한 한 IaC 파일의 출력 파라미터 값을 다른 IaC 파일의 파라미터와 일치시키고 삽입할 수 있습니다.

관리자는 입력 파라미터에 AWS Proton `environment.inputs.` 네임스페이스를 추가할 수 있습니다. 서비스 IaC 파일의 환경 IaC 파일 출력을 참조하는 경우 출력에 `environment.outputs.` 네임스페이스를 추가할 수 있습니다 (예: `environment.outputs.ClusterName`). 마지막으로 중괄호와 따옴표로 묶습니다.

이러한 수정을 통해에서 CloudFormation IaC 파일을 사용할 수 있습니다 AWS Proton.

예제 3: 코드 파일로서의 AWS Proton 환경 인프라

```
AWSTemplateFormatVersion: '2010-09-09'
Description: AWS Fargate cluster running containers in a public subnet. Only supports
             public facing load balancer, and public service discovery prefixes.
Mappings:
  # The VPC and subnet configuration is passed in via the environment spec.
  SubnetConfig:
    VPC:
      CIDR: '{{ environment.inputs.vpc_cidr}}'        # input parameter
    PublicOne:
      CIDR: '{{ environment.inputs.subnet_one_cidr}}' # input parameter
    PublicTwo:
      CIDR: '{{ environment.inputs.subnet_two_cidr}}' # input parameter
Resources:
  VPC:
    Type: AWS::EC2::VPC
    Properties:
      EnableDnsSupport: true
      EnableDnsHostnames: true
      CidrBlock: !FindInMap ['SubnetConfig', 'VPC', 'CIDR']

  # Two public subnets, where containers will have public IP addresses
  PublicSubnetOne:
    Type: AWS::EC2::Subnet
    Properties:
      AvailabilityZone:
         Fn::Select:
         - 0
         - Fn::GetAZs: {Ref: 'AWS::Region'}
      VpcId: !Ref 'VPC'
      CidrBlock: !FindInMap ['SubnetConfig', 'PublicOne', 'CIDR']
      MapPublicIpOnLaunch: true
```

```
PublicSubnetTwo:
  Type: AWS::EC2::Subnet
  Properties:
    AvailabilityZone:
       Fn::Select:
       - 1
       - Fn::GetAZs: {Ref: 'AWS::Region'}
    VpcId: !Ref 'VPC'
    CidrBlock: !FindInMap ['SubnetConfig', 'PublicTwo', 'CIDR']
    MapPublicIpOnLaunch: true

# Setup networking resources for the public subnets. Containers
# in the public subnets have public IP addresses and the routing table
# sends network traffic via the internet gateway.
InternetGateway:
  Type: AWS::EC2::InternetGateway
GatewayAttachement:
  Type: AWS::EC2::VPCGatewayAttachment
  Properties:
    VpcId: !Ref 'VPC'
    InternetGatewayId: !Ref 'InternetGateway'
PublicRouteTable:
  Type: AWS::EC2::RouteTable
  Properties:
    VpcId: !Ref 'VPC'
PublicRoute:
  Type: AWS::EC2::Route
  DependsOn: GatewayAttachement
  Properties:
    RouteTableId: !Ref 'PublicRouteTable'
    DestinationCidrBlock: '0.0.0.0/0'
    GatewayId: !Ref 'InternetGateway'
PublicSubnetOneRouteTableAssociation:
  Type: AWS::EC2::SubnetRouteTableAssociation
  Properties:
    SubnetId: !Ref PublicSubnetOne
    RouteTableId: !Ref PublicRouteTable
PublicSubnetTwoRouteTableAssociation:
  Type: AWS::EC2::SubnetRouteTableAssociation
  Properties:
    SubnetId: !Ref PublicSubnetTwo
    RouteTableId: !Ref PublicRouteTable
```

```
  # ECS Resources
  ECSCluster:
    Type: AWS::ECS::Cluster

  # A security group for the containers we will run in Fargate.
  # Rules are added to this security group based on what ingress you
  # add for the cluster.
  ContainerSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Access to the Fargate containers
      VpcId: !Ref 'VPC'

  # This is a role which is used by the ECS tasks themselves.
  ECSTaskExecutionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
        - Effect: Allow
          Principal:
            Service: [ecs-tasks.amazonaws.com]
          Action: ['sts:AssumeRole']
      Path: /
      ManagedPolicyArns:
        - 'arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy'

# These output values are available to service infrastructure as code files as outputs,
 when given the
# the 'service_instance.environment.outputs.' namespace, for example,
 service_instance.environment.outputs.ClusterName.

Outputs:
  ClusterName:                                          # output
    Description: The name of the ECS cluster
    Value: !Ref 'ECSCluster'
  ECSTaskExecutionRole:                                 # output
    Description: The ARN of the ECS role
    Value: !GetAtt 'ECSTaskExecutionRole.Arn'
  VpcId:                                                # output
    Description: The ID of the VPC that this stack is deployed in
    Value: !Ref 'VPC'
  PublicSubnetOne:                                      # output
    Description: Public subnet one
```

```
    Value: !Ref 'PublicSubnetOne'
  PublicSubnetTwo:                                    # output
    Description: Public subnet two
    Value: !Ref 'PublicSubnetTwo'
  ContainerSecurityGroup:                             # output
    Description: A security group used to allow Fargate containers to receive traffic
    Value: !Ref 'ContainerSecurityGroup'
```

예제 1과 예제 3 IaC 파일은 약간 다른 CloudFormation 스택을 생성합니다. 스택 템플릿 파일에서는 파라미터가 다르게 표시됩니다. 예제 1 CloudFormation 스택 템플릿 파일은 스택 템플릿 보기에 파라미터 레이블(키)을 표시합니다. 예제 3 AWS Proton CloudFormation 인프라 스택 템플릿 파일에는 파라미터 값이 표시됩니다. AWS Proton 입력 파라미터는 콘솔 CloudFormation 스택 파라미터 보기에 표시되지 않습니다.

예제 4에서 AWS Proton 서비스 IaC 파일은 예제 2에 해당합니다.

예제 4: AWS Proton 서비스 인스턴스 IaC 파일

```
AWSTemplateFormatVersion: '2010-09-09'
Description: Deploy a service on AWS Fargate, hosted in a public subnet, and accessible
 via a public load balancer.
Mappings:
  TaskSize:
    x-small:
      cpu: 256
      memory: 512
    small:
      cpu: 512
      memory: 1024
    medium:
      cpu: 1024
      memory: 2048
    large:
      cpu: 2048
      memory: 4096
    x-large:
      cpu: 4096
      memory: 8192
Resources:
  # A log group for storing the stdout logs from this service's containers
  LogGroup:
    Type: AWS::Logs::LogGroup
    Properties:
```

```
      LogGroupName: '{{service_instance.name}}' # resource parameter

  # The task definition. This is a simple metadata description of what
  # container to run, and what resource requirements it has.
  TaskDefinition:
    Type: AWS::ECS::TaskDefinition
    Properties:
      Family: '{{service_instance.name}}'
      Cpu: !FindInMap [TaskSize, {{service_instance.inputs.task_size}}, cpu] # input
parameter
      Memory: !FindInMap [TaskSize, {{service_instance.inputs.task_size}}, memory]
      NetworkMode: awsvpc
      RequiresCompatibilities:
        - FARGATE
      ExecutionRoleArn: '{{environment.outputs.ECSTaskExecutionRole}}' # output from an
environment infrastructure as code file
      TaskRoleArn: !Ref "AWS::NoValue"
      ContainerDefinitions:
        - Name: '{{service_instance.name}}'
          Cpu: !FindInMap [TaskSize, {{service_instance.inputs.task_size}}, cpu]
          Memory: !FindInMap [TaskSize, {{service_instance.inputs.task_size}}, memory]
          Image: '{{service_instance.inputs.image}}'
          PortMappings:
            - ContainerPort: '{{service_instance.inputs.port}}' # input parameter
          LogConfiguration:
            LogDriver: 'awslogs'
            Options:
              awslogs-group: '{{service_instance.name}}'
              awslogs-region: !Ref 'AWS::Region'
              awslogs-stream-prefix: '{{service_instance.name}}'

  # The service_instance. The service is a resource which allows you to run multiple
  # copies of a type of task, and gather up their logs and metrics, as well
  # as monitor the number of running tasks and replace any that have crashed
  Service:
    Type: AWS::ECS::Service
    DependsOn: LoadBalancerRule
    Properties:
      ServiceName: '{{service_instance.name}}'
      Cluster: '{{environment.outputs.ClusterName}}' # output from an environment
infrastructure as code file
      LaunchType: FARGATE
      DeploymentConfiguration:
        MaximumPercent: 200
```

```
        MinimumHealthyPercent: 75
      DesiredCount: '{{service_instance.inputs.desired_count}}'       # input parameter
      NetworkConfiguration:
        AwsvpcConfiguration:
          AssignPublicIp: ENABLED
          SecurityGroups:
            - '{{environment.outputs.ContainerSecurityGroup}}' # output from an
environment infrastructure as code file
          Subnets:
            - '{{environment.outputs.PublicSubnetOne}}'        # output from an
environment infrastructure as code file
            - '{{environment.outputs.PublicSubnetTwo}}'
      TaskDefinition: !Ref 'TaskDefinition'
      LoadBalancers:
        - ContainerName: '{{service_instance.name}}'
          ContainerPort: '{{service_instance.inputs.port}}'
          TargetGroupArn: !Ref 'TargetGroup'

  # A target group. This is used for keeping track of all the tasks, and
  # what IP addresses / port numbers they have. You can query it yourself,
  # to use the addresses yourself, but most often this target group is just
  # connected to an application load balancer, or network load balancer, so
  # it can automatically distribute traffic across all the targets.
  TargetGroup:
    Type: AWS::ElasticLoadBalancingV2::TargetGroup
    Properties:
      HealthCheckIntervalSeconds: 6
      HealthCheckPath: /
      HealthCheckProtocol: HTTP
      HealthCheckTimeoutSeconds: 5
      HealthyThresholdCount: 2
      TargetType: ip
      Name: '{{service_instance.name}}'
      Port: '{{service_instance.inputs.port}}'
      Protocol: HTTP
      UnhealthyThresholdCount: 2
      VpcId: '{{environment.outputs.VpcId}}' # output from an environment
infrastructure as code file

  # Create a rule on the load balancer for routing traffic to the target group
  LoadBalancerRule:
    Type: AWS::ElasticLoadBalancingV2::ListenerRule
    Properties:
      Actions:
```

```
        - TargetGroupArn: !Ref 'TargetGroup'
          Type: 'forward'
      Conditions:
        - Field: path-pattern
          Values:
            - '*'
      ListenerArn: !Ref PublicLoadBalancerListener
      Priority: 1

  # Enable autoscaling for this service
  ScalableTarget:
    Type: AWS::ApplicationAutoScaling::ScalableTarget
    DependsOn: Service
    Properties:
      ServiceNamespace: 'ecs'
      ScalableDimension: 'ecs:service:DesiredCount'
      ResourceId:
        Fn::Join:
          - '/'
          - - service
            - '{{environment.outputs.ClusterName}}' # output from an environment
 infrastructure as code file
            - '{{service_instance.name}}'
      MinCapacity: 1
      MaxCapacity: 10
      RoleARN: !Sub arn:aws:iam::${AWS::AccountId}:role/
aws-service-role/ecs.application-autoscaling.amazonaws.com/
AWSServiceRoleForApplicationAutoScaling_ECSService

  # Create scaling policies for the service
  ScaleDownPolicy:
    Type: AWS::ApplicationAutoScaling::ScalingPolicy
    DependsOn: ScalableTarget
    Properties:
      PolicyName:
        Fn::Join:
          - '/'
          - - scale
            - '{{service_instance.name}}'
            - down
      PolicyType: StepScaling
      ResourceId:
        Fn::Join:
          - '/'
```

```
          - - service
            - '{{environment.outputs.ClusterName}}'
            - '{{service_instance.name}}'
      ScalableDimension: 'ecs:service:DesiredCount'
      ServiceNamespace: 'ecs'
      StepScalingPolicyConfiguration:
        AdjustmentType: 'ChangeInCapacity'
        StepAdjustments:
          - MetricIntervalUpperBound: 0
            ScalingAdjustment: -1
        MetricAggregationType: 'Average'
        Cooldown: 60

  ScaleUpPolicy:
    Type: AWS::ApplicationAutoScaling::ScalingPolicy
    DependsOn: ScalableTarget
    Properties:
      PolicyName:
        Fn::Join:
          - '/'
          - - scale
            - '{{service_instance.name}}'
            - up
      PolicyType: StepScaling
      ResourceId:
        Fn::Join:
          - '/'
          - - service
            - '{{environment.outputs.ClusterName}}'
            - '{{service_instance.name}}'
      ScalableDimension: 'ecs:service:DesiredCount'
      ServiceNamespace: 'ecs'
      StepScalingPolicyConfiguration:
        AdjustmentType: 'ChangeInCapacity'
        StepAdjustments:
          - MetricIntervalLowerBound: 0
            MetricIntervalUpperBound: 15
            ScalingAdjustment: 1
          - MetricIntervalLowerBound: 15
            MetricIntervalUpperBound: 25
            ScalingAdjustment: 2
          - MetricIntervalLowerBound: 25
            ScalingAdjustment: 3
        MetricAggregationType: 'Average'
```

```
      Cooldown: 60

# Create alarms to trigger these policies
LowCpuUsageAlarm:
  Type: AWS::CloudWatch::Alarm
  Properties:
    AlarmName:
      Fn::Join:
        - '-'
        - - low-cpu
          - '{{service_instance.name}}'
    AlarmDescription:
      Fn::Join:
        - ' '
        - - "Low CPU utilization for service"
          - '{{service_instance.name}}'
    MetricName: CPUUtilization
    Namespace: AWS/ECS
    Dimensions:
      - Name: ServiceName
        Value: '{{service_instance.name}}'
      - Name: ClusterName
        Value:
          '{{environment.outputs.ClusterName}}'
    Statistic: Average
    Period: 60
    EvaluationPeriods: 1
    Threshold: 20
    ComparisonOperator: LessThanOrEqualToThreshold
    AlarmActions:
      - !Ref ScaleDownPolicy

HighCpuUsageAlarm:
  Type: AWS::CloudWatch::Alarm
  Properties:
    AlarmName:
      Fn::Join:
        - '-'
        - - high-cpu
          - '{{service_instance.name}}'
    AlarmDescription:
      Fn::Join:
        - ' '
        - - "High CPU utilization for service"
```

```
            - '{{service_instance.name}}'
      MetricName: CPUUtilization
      Namespace: AWS/ECS
      Dimensions:
        - Name: ServiceName
          Value: '{{service_instance.name}}'
        - Name: ClusterName
          Value:
            '{{environment.outputs.ClusterName}}'
      Statistic: Average
      Period: 60
      EvaluationPeriods: 1
      Threshold: 70
      ComparisonOperator: GreaterThanOrEqualToThreshold
      AlarmActions:
        - !Ref ScaleUpPolicy

  EcsSecurityGroupIngressFromPublicALB:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      Description: Ingress from the public ALB
      GroupId: '{{environment.outputs.ContainerSecurityGroup}}'
      IpProtocol: -1
      SourceSecurityGroupId: !Ref 'PublicLoadBalancerSG'

  # Public load balancer, hosted in public subnets that is accessible
  # to the public, and is intended to route traffic to one or more public
  # facing services. This is used for accepting traffic from the public
  # internet and directing it to public facing microservices
  PublicLoadBalancerSG:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Access to the public facing load balancer
      VpcId: '{{environment.outputs.VpcId}}'
      SecurityGroupIngress:
          # Allow access to ALB from anywhere on the internet
          - CidrIp: 0.0.0.0/0
            IpProtocol: -1

  PublicLoadBalancer:
    Type: AWS::ElasticLoadBalancingV2::LoadBalancer
    Properties:
      Scheme: internet-facing
      LoadBalancerAttributes:
```

```
      - Key: idle_timeout.timeout_seconds
        Value: '30'
      Subnets:
        # The load balancer is placed into the public subnets, so that traffic
        # from the internet can reach the load balancer directly via the internet
 gateway
        - '{{environment.outputs.PublicSubnetOne}}'
        - '{{environment.outputs.PublicSubnetTwo}}'
      SecurityGroups: [!Ref 'PublicLoadBalancerSG']

  PublicLoadBalancerListener:
    Type: AWS::ElasticLoadBalancingV2::Listener
    DependsOn:
      - PublicLoadBalancer
    Properties:
      DefaultActions:
        - TargetGroupArn: !Ref 'TargetGroup'
          Type: 'forward'
      LoadBalancerArn: !Ref 'PublicLoadBalancer'
      Port: 80
      Protocol: HTTP
Outputs:
  ServiceEndpoint:         # output
    Description: The URL to access the service
    Value: !Sub "http://${PublicLoadBalancer.DNSName}"
```

예제 5에서 AWS Proton 파이프라인 IaC 파일은 예제 4에서 프로비저닝된 서비스 인스턴스를 지원하도록 파이프라인 인프라를 프로비저닝합니다.

예제 5: AWS Proton 서비스 파이프라인 IaC 파일

```
Resources:
  ECRRepo:
    Type: AWS::ECR::Repository
    DeletionPolicy: Retain
  BuildProject:
    Type: AWS::CodeBuild::Project
    Properties:
      Artifacts:
        Type: CODEPIPELINE
      Environment:
        ComputeType: BUILD_GENERAL1_SMALL
        Image: aws/codebuild/amazonlinux2-x86_64-standard:3.0
```

```
        PrivilegedMode: true
        Type: LINUX_CONTAINER
        EnvironmentVariables:
        - Name: repo_name
          Type: PLAINTEXT
          Value: !Ref ECRRepo
        - Name: service_name
          Type: PLAINTEXT
          Value: '{{ service.name }}'    # resource parameter
      ServiceRole:
        Fn::GetAtt:
          - PublishRole
          - Arn
      Source:
        BuildSpec:
          Fn::Join:
            - ""
            - - >-
                {
                  "version": "0.2",
                  "phases": {
                    "install": {
                      "runtime-versions": {
                        "docker": 18
                      },
                      "commands": [
                        "pip3 install --upgrade --user awscli",
                        "echo
 'f6bd1536a743ab170b35c94ed4c7c4479763356bd543af5d391122f4af852460  yq_linux_amd64' >
 yq_linux_amd64.sha",
                        "wget https://github.com/mikefarah/yq/releases/download/3.4.0/
yq_linux_amd64",
                        "sha256sum -c yq_linux_amd64.sha",
                        "mv yq_linux_amd64 /usr/bin/yq",
                        "chmod +x /usr/bin/yq"
                      ]
                    },
                    "pre_build": {
                      "commands": [
                        "cd $CODEBUILD_SRC_DIR",
                        "$(aws ecr get-login --no-include-email --region
 $AWS_DEFAULT_REGION)",
                        "{{ pipeline.inputs.unit_test_command }}",    # input parameter
                      ]
```

```
                    },
                    "build": {
                      "commands": [
                        "IMAGE_REPO_NAME=$repo_name",
                        "IMAGE_TAG=$CODEBUILD_BUILD_NUMBER",
                        "IMAGE_ID=
              - Ref: AWS::AccountId
              - >-
                .dkr.ecr.$AWS_DEFAULT_REGION.amazonaws.com/$IMAGE_REPO_NAME:
$IMAGE_TAG",
                        "docker build -t $IMAGE_REPO_NAME:$IMAGE_TAG -f
 {{ pipeline.inputs.dockerfile }} .",     # input parameter
                        "docker tag $IMAGE_REPO_NAME:$IMAGE_TAG $IMAGE_ID;",
                        "docker push $IMAGE_ID"
                      ]
                    },
                    "post_build": {
                      "commands": [
                        "aws proton --region $AWS_DEFAULT_REGION get-service --name
 $service_name | jq -r .service.spec > service.yaml",
                        "yq w service.yaml 'instances[*].spec.image' \"$IMAGE_ID\" >
 rendered_service.yaml"
                      ]
                    }
                  },
                  "artifacts": {
                    "files": [
                      "rendered_service.yaml"
                    ]
                  }
                }
        Type: CODEPIPELINE
      EncryptionKey:
        Fn::GetAtt:
          - PipelineArtifactsBucketEncryptionKey
          - Arn
{% for service_instance in service_instances %}
  Deploy{{loop.index}}Project:
    Type: AWS::CodeBuild::Project
    Properties:
      Artifacts:
        Type: CODEPIPELINE
      Environment:
        ComputeType: BUILD_GENERAL1_SMALL
```

```
        Image: aws/codebuild/amazonlinux2-x86_64-standard:3.0
        PrivilegedMode: false
        Type: LINUX_CONTAINER
        EnvironmentVariables:
        - Name: service_name
          Type: PLAINTEXT
          Value:  '{{service.name}}'          # resource parameter
        - Name: service_instance_name
          Type: PLAINTEXT
          Value: '{{service_instance.name}}'  # resource parameter
      ServiceRole:
        Fn::GetAtt:
          - DeploymentRole
          - Arn
      Source:
        BuildSpec: >-
          {
            "version": "0.2",
            "phases": {
              "build": {
                "commands": [
                  "pip3 install --upgrade --user awscli",
                  "aws proton --region $AWS_DEFAULT_REGION update-service-instance
 --deployment-type CURRENT_VERSION --name $service_instance_name --service-name
 $service_name --spec file://rendered_service.yaml",
                  "aws proton --region $AWS_DEFAULT_REGION wait service-instance-
deployed --name $service_instance_name --service-name $service_name"
                ]
              }
            }
          }
        Type: CODEPIPELINE
      EncryptionKey:
        Fn::GetAtt:
          - PipelineArtifactsBucketEncryptionKey
          - Arn
{% endfor %}
  # This role is used to build and publish an image to ECR
  PublishRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Action: sts:AssumeRole
```

```
            Effect: Allow
            Principal:
              Service: codebuild.amazonaws.com
        Version: "2012-10-17"
  PublishRoleDefaultPolicy:
    Type: AWS::IAM::Policy
    Properties:
      PolicyDocument:
        Statement:
          - Action:
              - logs:CreateLogGroup
              - logs:CreateLogStream
              - logs:PutLogEvents
            Effect: Allow
            Resource:
              - Fn::Join:
                  - ""
                  - - "arn:"
                    - Ref: AWS::Partition
                    - ":logs:"
                    - Ref: AWS::Region
                    - ":"
                    - Ref: AWS::AccountId
                    - :log-group:/aws/codebuild/
                    - Ref: BuildProject
              - Fn::Join:
                  - ""
                  - - "arn:"
                    - Ref: AWS::Partition
                    - ":logs:"
                    - Ref: AWS::Region
                    - ":"
                    - Ref: AWS::AccountId
                    - :log-group:/aws/codebuild/
                    - Ref: BuildProject
                    - :*
          - Action:
              - codebuild:CreateReportGroup
              - codebuild:CreateReport
              - codebuild:UpdateReport
              - codebuild:BatchPutTestCases
            Effect: Allow
            Resource:
              Fn::Join:
```

```
            - ""
            - - "arn:"
              - Ref: AWS::Partition
              - ":codebuild:"
              - Ref: AWS::Region
              - ":"
              - Ref: AWS::AccountId
              - :report-group/
              - Ref: BuildProject
              - -*
        - Action:
            - ecr:GetAuthorizationToken
          Effect: Allow
          Resource: "*"
        - Action:
            - ecr:BatchCheckLayerAvailability
            - ecr:CompleteLayerUpload
            - ecr:GetAuthorizationToken
            - ecr:InitiateLayerUpload
            - ecr:PutImage
            - ecr:UploadLayerPart
          Effect: Allow
          Resource:
            Fn::GetAtt:
              - ECRRepo
              - Arn
        - Action:
            - proton:GetService
          Effect: Allow
          Resource: "*"
        - Action:
            - s3:GetObject*
            - s3:GetBucket*
            - s3:List*
            - s3:DeleteObject*
            - s3:PutObject*
            - s3:Abort*
          Effect: Allow
          Resource:
            - Fn::GetAtt:
                - PipelineArtifactsBucket
                - Arn
            - Fn::Join:
                - ""
```

```
                - - Fn::GetAtt:
                      - PipelineArtifactsBucket
                      - Arn
                  - /*
          - Action:
              - kms:Decrypt
              - kms:DescribeKey
              - kms:Encrypt
              - kms:ReEncrypt*
              - kms:GenerateDataKey*
            Effect: Allow
            Resource:
              Fn::GetAtt:
                - PipelineArtifactsBucketEncryptionKey
                - Arn
          - Action:
              - kms:Decrypt
              - kms:Encrypt
              - kms:ReEncrypt*
              - kms:GenerateDataKey*
            Effect: Allow
            Resource:
              Fn::GetAtt:
                - PipelineArtifactsBucketEncryptionKey
                - Arn
        Version: "2012-10-17"
      PolicyName: PublishRoleDefaultPolicy
      Roles:
        - Ref: PublishRole

  DeploymentRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Action: sts:AssumeRole
            Effect: Allow
            Principal:
              Service: codebuild.amazonaws.com
        Version: "2012-10-17"
  DeploymentRoleDefaultPolicy:
    Type: AWS::IAM::Policy
    Properties:
      PolicyDocument:
```

```
        Statement:
          - Action:
              - logs:CreateLogGroup
              - logs:CreateLogStream
              - logs:PutLogEvents
            Effect: Allow
            Resource:
              - Fn::Join:
                  - ""
                  - - "arn:"
                    - Ref: AWS::Partition
                    - ":logs:"
                    - Ref: AWS::Region
                    - ":"
                    - Ref: AWS::AccountId
                    - :log-group:/aws/codebuild/Deploy*Project*
              - Fn::Join:
                  - ""
                  - - "arn:"
                    - Ref: AWS::Partition
                    - ":logs:"
                    - Ref: AWS::Region
                    - ":"
                    - Ref: AWS::AccountId
                    - :log-group:/aws/codebuild/Deploy*Project:*
          - Action:
              - codebuild:CreateReportGroup
              - codebuild:CreateReport
              - codebuild:UpdateReport
              - codebuild:BatchPutTestCases
            Effect: Allow
            Resource:
              Fn::Join:
                - ""
                - - "arn:"
                  - Ref: AWS::Partition
                  - ":codebuild:"
                  - Ref: AWS::Region
                  - ":"
                  - Ref: AWS::AccountId
                  - :report-group/Deploy*Project
                  - -*
          - Action:
              - proton:UpdateServiceInstance
```

```
              - proton:GetServiceInstance
            Effect: Allow
            Resource: "*"
          - Action:
              - s3:GetObject*
              - s3:GetBucket*
              - s3:List*
            Effect: Allow
            Resource:
              - Fn::GetAtt:
                  - PipelineArtifactsBucket
                  - Arn
              - Fn::Join:
                  - ""
                  - - Fn::GetAtt:
                        - PipelineArtifactsBucket
                        - Arn
                    - /*
          - Action:
              - kms:Decrypt
              - kms:DescribeKey
            Effect: Allow
            Resource:
              Fn::GetAtt:
                - PipelineArtifactsBucketEncryptionKey
                - Arn
          - Action:
              - kms:Decrypt
              - kms:Encrypt
              - kms:ReEncrypt*
              - kms:GenerateDataKey*
            Effect: Allow
            Resource:
              Fn::GetAtt:
                - PipelineArtifactsBucketEncryptionKey
                - Arn
        Version: "2012-10-17"
      PolicyName: DeploymentRoleDefaultPolicy
      Roles:
        - Ref: DeploymentRole
  PipelineArtifactsBucketEncryptionKey:
    Type: AWS::KMS::Key
    Properties:
      KeyPolicy:
```

```
        Statement:
          - Action:
              - kms:Create*
              - kms:Describe*
              - kms:Enable*
              - kms:List*
              - kms:Put*
              - kms:Update*
              - kms:Revoke*
              - kms:Disable*
              - kms:Get*
              - kms:Delete*
              - kms:ScheduleKeyDeletion
              - kms:CancelKeyDeletion
              - kms:GenerateDataKey
              - kms:TagResource
              - kms:UntagResource
            Effect: Allow
            Principal:
              AWS:
                Fn::Join:
                  - ""
                  - - "arn:"
                    - Ref: AWS::Partition
                    - ":iam::"
                    - Ref: AWS::AccountId
                    - :root
            Resource: "*"
          - Action:
              - kms:Decrypt
              - kms:DescribeKey
              - kms:Encrypt
              - kms:ReEncrypt*
              - kms:GenerateDataKey*
            Effect: Allow
            Principal:
              AWS:
                Fn::GetAtt:
                  - PipelineRole
                  - Arn
            Resource: "*"
          - Action:
              - kms:Decrypt
              - kms:DescribeKey
```

```
            - kms:Encrypt
            - kms:ReEncrypt*
            - kms:GenerateDataKey*
          Effect: Allow
          Principal:
            AWS:
              Fn::GetAtt:
                - PublishRole
                - Arn
          Resource: "*"
        - Action:
            - kms:Decrypt
            - kms:Encrypt
            - kms:ReEncrypt*
            - kms:GenerateDataKey*
          Effect: Allow
          Principal:
            AWS:
              Fn::GetAtt:
                - PublishRole
                - Arn
          Resource: "*"
        - Action:
            - kms:Decrypt
            - kms:DescribeKey
          Effect: Allow
          Principal:
            AWS:
              Fn::GetAtt:
                - DeploymentRole
                - Arn
          Resource: "*"
        - Action:
            - kms:Decrypt
            - kms:Encrypt
            - kms:ReEncrypt*
            - kms:GenerateDataKey*
          Effect: Allow
          Principal:
            AWS:
              Fn::GetAtt:
                - DeploymentRole
                - Arn
          Resource: "*"
```

```
        Version: "2012-10-17"
    UpdateReplacePolicy: Delete
    DeletionPolicy: Delete
  PipelineArtifactsBucket:
    Type: AWS::S3::Bucket
    Properties:
      VersioningConfiguration:
        Status: Enabled
      BucketEncryption:
        ServerSideEncryptionConfiguration:
          - ServerSideEncryptionByDefault:
              KMSMasterKeyID:
                Fn::GetAtt:
                  - PipelineArtifactsBucketEncryptionKey
                  - Arn
              SSEAlgorithm: aws:kms
      PublicAccessBlockConfiguration:
        BlockPublicAcls: true
        BlockPublicPolicy: true
        IgnorePublicAcls: true
        RestrictPublicBuckets: true
    UpdateReplacePolicy: Retain
    DeletionPolicy: Retain
  PipelineArtifactsBucketEncryptionKeyAlias:
    Type: AWS::KMS::Alias
    Properties:
      AliasName: 'alias/codepipeline-encryption-key-{{ service.name }}'
      TargetKeyId:
        Fn::GetAtt:
          - PipelineArtifactsBucketEncryptionKey
          - Arn
    UpdateReplacePolicy: Delete
    DeletionPolicy: Delete
  PipelineRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Action: sts:AssumeRole
            Effect: Allow
            Principal:
              Service: codepipeline.amazonaws.com
        Version: "2012-10-17"
  PipelineRoleDefaultPolicy:
```

```
    Type: AWS::IAM::Policy
    Properties:
      PolicyDocument:
        Statement:
          - Action:
              - s3:GetObject*
              - s3:GetBucket*
              - s3:List*
              - s3:DeleteObject*
              - s3:PutObject*
              - s3:Abort*
            Effect: Allow
            Resource:
              - Fn::GetAtt:
                  - PipelineArtifactsBucket
                  - Arn
              - Fn::Join:
                  - ""
                  - - Fn::GetAtt:
                        - PipelineArtifactsBucket
                        - Arn
                    - /*
          - Action:
              - kms:Decrypt
              - kms:DescribeKey
              - kms:Encrypt
              - kms:ReEncrypt*
              - kms:GenerateDataKey*
            Effect: Allow
            Resource:
              Fn::GetAtt:
                - PipelineArtifactsBucketEncryptionKey
                - Arn
          - Action: codestar-connections:*
            Effect: Allow
            Resource: "*"
          - Action: sts:AssumeRole
            Effect: Allow
            Resource:
              Fn::GetAtt:
                - PipelineBuildCodePipelineActionRole
                - Arn
          - Action: sts:AssumeRole
            Effect: Allow
```

```
            Resource:
              Fn::GetAtt:
                - PipelineDeployCodePipelineActionRole
                - Arn
        Version: "2012-10-17"
      PolicyName: PipelineRoleDefaultPolicy
      Roles:
        - Ref: PipelineRole
  Pipeline:
    Type: AWS::CodePipeline::Pipeline
    Properties:
      RoleArn:
        Fn::GetAtt:
          - PipelineRole
          - Arn
      Stages:
        - Actions:
            - ActionTypeId:
                Category: Source
                Owner: AWS
                Provider: CodeStarSourceConnection
                Version: "1"
              Configuration:
                ConnectionArn: '{{ service.repository_connection_arn }}'
                FullRepositoryId: '{{ service.repository_id }}'
                BranchName: '{{ service.branch_name }}'
              Name: Checkout
              OutputArtifacts:
                - Name: Artifact_Source_Checkout
              RunOrder: 1
          Name: Source
        - Actions:
            - ActionTypeId:
                Category: Build
                Owner: AWS
                Provider: CodeBuild
                Version: "1"
              Configuration:
                ProjectName:
                  Ref: BuildProject
              InputArtifacts:
                - Name: Artifact_Source_Checkout
              Name: Build
              OutputArtifacts:
```

```
                - Name: BuildOutput
              RoleArn:
                Fn::GetAtt:
                  - PipelineBuildCodePipelineActionRole
                  - Arn
              RunOrder: 1
          Name: Build {%- for service_instance in service_instances %}
        - Actions:
            - ActionTypeId:
                Category: Build
                Owner: AWS
                Provider: CodeBuild
                Version: "1"
              Configuration:
                ProjectName:
                  Ref: Deploy{{loop.index}}Project
              InputArtifacts:
                - Name: BuildOutput
              Name: Deploy
              RoleArn:
                Fn::GetAtt:
                  - PipelineDeployCodePipelineActionRole
                  - Arn
              RunOrder: 1
          Name: 'Deploy{{service_instance.name}}'
{%- endfor %}
      ArtifactStore:
        EncryptionKey:
          Id:
            Fn::GetAtt:
              - PipelineArtifactsBucketEncryptionKey
              - Arn
          Type: KMS
        Location:
          Ref: PipelineArtifactsBucket
        Type: S3
    DependsOn:
      - PipelineRoleDefaultPolicy
      - PipelineRole
  PipelineBuildCodePipelineActionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
```

```
        - Action: sts:AssumeRole
          Effect: Allow
          Principal:
            AWS:
              Fn::Join:
                - ""
                - - "arn:"
                  - Ref: AWS::Partition
                  - ":iam::"
                  - Ref: AWS::AccountId
                  - :root
      Version: "2012-10-17"
PipelineBuildCodePipelineActionRoleDefaultPolicy:
  Type: AWS::IAM::Policy
  Properties:
    PolicyDocument:
      Statement:
        - Action:
            - codebuild:BatchGetBuilds
            - codebuild:StartBuild
            - codebuild:StopBuild
          Effect: Allow
          Resource:
            Fn::GetAtt:
              - BuildProject
              - Arn
      Version: "2012-10-17"
    PolicyName: PipelineBuildCodePipelineActionRoleDefaultPolicy
    Roles:
      - Ref: PipelineBuildCodePipelineActionRole
PipelineDeployCodePipelineActionRole:
  Type: AWS::IAM::Role
  Properties:
    AssumeRolePolicyDocument:
      Statement:
        - Action: sts:AssumeRole
          Effect: Allow
          Principal:
            AWS:
              Fn::Join:
                - ""
                - - "arn:"
                  - Ref: AWS::Partition
                  - ":iam::"
```

```
                    - Ref: AWS::AccountId
                    - :root
        Version: "2012-10-17"
  PipelineDeployCodePipelineActionRoleDefaultPolicy:
    Type: AWS::IAM::Policy
    Properties:
      PolicyDocument:
        Statement:
          - Action:
              - codebuild:BatchGetBuilds
              - codebuild:StartBuild
              - codebuild:StopBuild
            Effect: Allow
            Resource:
              Fn::Join:
                - ""
                - - "arn:"
                  - Ref: AWS::Partition
                  - ":codebuild:"
                  - Ref: AWS::Region
                  - ":"
                  - Ref: AWS::AccountId
                  - ":project/Deploy*"
        Version: "2012-10-17"
      PolicyName: PipelineDeployCodePipelineActionRoleDefaultPolicy
      Roles:
        - Ref: PipelineDeployCodePipelineActionRole
Outputs:
  PipelineEndpoint:
    Description: The URL to access the pipeline
    Value: !Sub "https://${AWS::Region}.console.aws.amazon.com/codesuite/codepipeline/
pipelines/${Pipeline}/view?region=${AWS::Region}"

                ]
              }
            }
          }
        Type: CODEPIPELINE
      EncryptionKey:
        Fn::GetAtt:
          - PipelineArtifactsBucketEncryptionKey
          - Arn
{% endfor %}
  # This role is used to build and publish an image to ECR
```

```
  PublishRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Action: sts:AssumeRole
            Effect: Allow
            Principal:
              Service: codebuild.amazonaws.com
        Version: "2012-10-17"
  PublishRoleDefaultPolicy:
    Type: AWS::IAM::Policy
    Properties:
      PolicyDocument:
        Statement:
          - Action:
              - logs:CreateLogGroup
              - logs:CreateLogStream
              - logs:PutLogEvents
            Effect: Allow
            Resource:
              - Fn::Join:
                  - ""
                  - - "arn:"
                    - Ref: AWS::Partition
                    - ":logs:"
                    - Ref: AWS::Region
                    - ":"
                    - Ref: AWS::AccountId
                    - :log-group:/aws/codebuild/
                    - Ref: BuildProject
              - Fn::Join:
                  - ""
                  - - "arn:"
                    - Ref: AWS::Partition
                    - ":logs:"
                    - Ref: AWS::Region
                    - ":"
                    - Ref: AWS::AccountId
                    - :log-group:/aws/codebuild/
                    - Ref: BuildProject
                    - :*
          - Action:
              - codebuild:CreateReportGroup
```

```
            - codebuild:CreateReport
            - codebuild:UpdateReport
            - codebuild:BatchPutTestCases
          Effect: Allow
          Resource:
            Fn::Join:
              - ""
              - - "arn:"
                - Ref: AWS::Partition
                - ":codebuild:"
                - Ref: AWS::Region
                - ":"
                - Ref: AWS::AccountId
                - :report-group/
                - Ref: BuildProject
                - -*
        - Action:
            - ecr:GetAuthorizationToken
          Effect: Allow
          Resource: "*"
        - Action:
            - ecr:BatchCheckLayerAvailability
            - ecr:CompleteLayerUpload
            - ecr:GetAuthorizationToken
            - ecr:InitiateLayerUpload
            - ecr:PutImage
            - ecr:UploadLayerPart
          Effect: Allow
          Resource:
            Fn::GetAtt:
              - ECRRepo
              - Arn
        - Action:
            - proton:GetService
          Effect: Allow
          Resource: "*"
        - Action:
            - s3:GetObject*
            - s3:GetBucket*
            - s3:List*
            - s3:DeleteObject*
            - s3:PutObject*
            - s3:Abort*
          Effect: Allow
```

```
            Resource:
              - Fn::GetAtt:
                  - PipelineArtifactsBucket
                  - Arn
              - Fn::Join:
                  - ""
                  - - Fn::GetAtt:
                        - PipelineArtifactsBucket
                        - Arn
                    - /*
          - Action:
              - kms:Decrypt
              - kms:DescribeKey
              - kms:Encrypt
              - kms:ReEncrypt*
              - kms:GenerateDataKey*
            Effect: Allow
            Resource:
              Fn::GetAtt:
                - PipelineArtifactsBucketEncryptionKey
                - Arn
          - Action:
              - kms:Decrypt
              - kms:Encrypt
              - kms:ReEncrypt*
              - kms:GenerateDataKey*
            Effect: Allow
            Resource:
              Fn::GetAtt:
                - PipelineArtifactsBucketEncryptionKey
                - Arn
        Version: "2012-10-17"
      PolicyName: PublishRoleDefaultPolicy
      Roles:
        - Ref: PublishRole

  DeploymentRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Action: sts:AssumeRole
            Effect: Allow
            Principal:
```

```
            Service: codebuild.amazonaws.com
      Version: "2012-10-17"
  DeploymentRoleDefaultPolicy:
    Type: AWS::IAM::Policy
    Properties:
      PolicyDocument:
        Statement:
          - Action:
              - logs:CreateLogGroup
              - logs:CreateLogStream
              - logs:PutLogEvents
            Effect: Allow
            Resource:
              - Fn::Join:
                  - ""
                  - - "arn:"
                    - Ref: AWS::Partition
                    - ":logs:"
                    - Ref: AWS::Region
                    - ":"
                    - Ref: AWS::AccountId
                    - :log-group:/aws/codebuild/Deploy*Project*
              - Fn::Join:
                  - ""
                  - - "arn:"
                    - Ref: AWS::Partition
                    - ":logs:"
                    - Ref: AWS::Region
                    - ":"
                    - Ref: AWS::AccountId
                    - :log-group:/aws/codebuild/Deploy*Project:*
          - Action:
              - codebuild:CreateReportGroup
              - codebuild:CreateReport
              - codebuild:UpdateReport
              - codebuild:BatchPutTestCases
            Effect: Allow
            Resource:
              Fn::Join:
                - ""
                - - "arn:"
                  - Ref: AWS::Partition
                  - ":codebuild:"
                  - Ref: AWS::Region
```

```
                - ":"
                - Ref: AWS::AccountId
                - :report-group/Deploy*Project
                - -*
          - Action:
              - proton:UpdateServiceInstance
              - proton:GetServiceInstance
            Effect: Allow
            Resource: "*"
          - Action:
              - s3:GetObject*
              - s3:GetBucket*
              - s3:List*
            Effect: Allow
            Resource:
              - Fn::GetAtt:
                  - PipelineArtifactsBucket
                  - Arn
              - Fn::Join:
                  - ""
                  - - Fn::GetAtt:
                        - PipelineArtifactsBucket
                        - Arn
                    - /*
          - Action:
              - kms:Decrypt
              - kms:DescribeKey
            Effect: Allow
            Resource:
              Fn::GetAtt:
                - PipelineArtifactsBucketEncryptionKey
                - Arn
          - Action:
              - kms:Decrypt
              - kms:Encrypt
              - kms:ReEncrypt*
              - kms:GenerateDataKey*
            Effect: Allow
            Resource:
              Fn::GetAtt:
                - PipelineArtifactsBucketEncryptionKey
                - Arn
        Version: "2012-10-17"
      PolicyName: DeploymentRoleDefaultPolicy
```

```
      Roles:
        - Ref: DeploymentRole
  PipelineArtifactsBucketEncryptionKey:
    Type: AWS::KMS::Key
    Properties:
      KeyPolicy:
        Statement:
          - Action:
              - kms:Create*
              - kms:Describe*
              - kms:Enable*
              - kms:List*
              - kms:Put*
              - kms:Update*
              - kms:Revoke*
              - kms:Disable*
              - kms:Get*
              - kms:Delete*
              - kms:ScheduleKeyDeletion
              - kms:CancelKeyDeletion
              - kms:GenerateDataKey
              - kms:TagResource
              - kms:UntagResource
            Effect: Allow
            Principal:
              AWS:
                Fn::Join:
                  - ""
                  - - "arn:"
                    - Ref: AWS::Partition
                    - ":iam::"
                    - Ref: AWS::AccountId
                    - :root
            Resource: "*"
          - Action:
              - kms:Decrypt
              - kms:DescribeKey
              - kms:Encrypt
              - kms:ReEncrypt*
              - kms:GenerateDataKey*
            Effect: Allow
            Principal:
              AWS:
                Fn::GetAtt:
```

```
            - PipelineRole
            - Arn
      Resource: "*"
    - Action:
        - kms:Decrypt
        - kms:DescribeKey
        - kms:Encrypt
        - kms:ReEncrypt*
        - kms:GenerateDataKey*
      Effect: Allow
      Principal:
        AWS:
          Fn::GetAtt:
            - PublishRole
            - Arn
      Resource: "*"
    - Action:
        - kms:Decrypt
        - kms:Encrypt
        - kms:ReEncrypt*
        - kms:GenerateDataKey*
      Effect: Allow
      Principal:
        AWS:
          Fn::GetAtt:
            - PublishRole
            - Arn
      Resource: "*"
    - Action:
        - kms:Decrypt
        - kms:DescribeKey
      Effect: Allow
      Principal:
        AWS:
          Fn::GetAtt:
            - DeploymentRole
            - Arn
      Resource: "*"
    - Action:
        - kms:Decrypt
        - kms:Encrypt
        - kms:ReEncrypt*
        - kms:GenerateDataKey*
      Effect: Allow
```

```
            Principal:
              AWS:
                Fn::GetAtt:
                  - DeploymentRole
                  - Arn
            Resource: "*"
        Version: "2012-10-17"
    UpdateReplacePolicy: Delete
    DeletionPolicy: Delete
  PipelineArtifactsBucket:
    Type: AWS::S3::Bucket
    Properties:
      BucketEncryption:
        ServerSideEncryptionConfiguration:
          - ServerSideEncryptionByDefault:
              KMSMasterKeyID:
                Fn::GetAtt:
                  - PipelineArtifactsBucketEncryptionKey
                  - Arn
              SSEAlgorithm: aws:kms
      PublicAccessBlockConfiguration:
        BlockPublicAcls: true
        BlockPublicPolicy: true
        IgnorePublicAcls: true
        RestrictPublicBuckets: true
    UpdateReplacePolicy: Retain
    DeletionPolicy: Retain
  PipelineArtifactsBucketEncryptionKeyAlias:
    Type: AWS::KMS::Alias
    Properties:
      AliasName: 'alias/codepipeline-encryption-key-{{ service.name }}'     # resource
parameter
      TargetKeyId:
        Fn::GetAtt:
          - PipelineArtifactsBucketEncryptionKey
          - Arn
    UpdateReplacePolicy: Delete
    DeletionPolicy: Delete
  PipelineRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Action: sts:AssumeRole
```

```
          Effect: Allow
          Principal:
            Service: codepipeline.amazonaws.com
      Version: "2012-10-17"
  PipelineRoleDefaultPolicy:
    Type: AWS::IAM::Policy
    Properties:
      PolicyDocument:
        Statement:
          - Action:
              - s3:GetObject*
              - s3:GetBucket*
              - s3:List*
              - s3:DeleteObject*
              - s3:PutObject*
              - s3:Abort*
            Effect: Allow
            Resource:
              - Fn::GetAtt:
                  - PipelineArtifactsBucket
                  - Arn
              - Fn::Join:
                  - ""
                  - - Fn::GetAtt:
                        - PipelineArtifactsBucket
                        - Arn
                    - /*
          - Action:
              - kms:Decrypt
              - kms:DescribeKey
              - kms:Encrypt
              - kms:ReEncrypt*
              - kms:GenerateDataKey*
            Effect: Allow
            Resource:
              Fn::GetAtt:
                - PipelineArtifactsBucketEncryptionKey
                - Arn
          - Action: codestar-connections:*
            Effect: Allow
            Resource: "*"
          - Action: sts:AssumeRole
            Effect: Allow
            Resource:
```

```
              Fn::GetAtt:
                - PipelineBuildCodePipelineActionRole
                - Arn
          - Action: sts:AssumeRole
            Effect: Allow
            Resource:
              Fn::GetAtt:
                - PipelineDeployCodePipelineActionRole
                - Arn
        Version: "2012-10-17"
      PolicyName: PipelineRoleDefaultPolicy
      Roles:
        - Ref: PipelineRole
 Pipeline:
   Type: AWS::CodePipeline::Pipeline
   Properties:
     RoleArn:
       Fn::GetAtt:
         - PipelineRole
         - Arn
     Stages:
       - Actions:
           - ActionTypeId:
               Category: Source
               Owner: AWS
               Provider: CodeStarSourceConnection
               Version: "1"
             Configuration:
               ConnectionArn: '{{ service.repository_connection_arn }}'   # resource
parameter
               FullRepositoryId: '{{ service.repository_id }}'            # resource
parameter
               BranchName: '{{ service.branch_name }}'                    # resource
parameter
             Name: Checkout
             OutputArtifacts:
               - Name: Artifact_Source_Checkout
             RunOrder: 1
         Name: Source
       - Actions:
           - ActionTypeId:
               Category: Build
               Owner: AWS
               Provider: CodeBuild
```

```
                Version: "1"
              Configuration:
                ProjectName:
                  Ref: BuildProject
              InputArtifacts:
                - Name: Artifact_Source_Checkout
              Name: Build
              OutputArtifacts:
                - Name: BuildOutput
              RoleArn:
                Fn::GetAtt:
                  - PipelineBuildCodePipelineActionRole
                  - Arn
              RunOrder: 1
          Name: Build {%- for service_instance in service_instances %}
        - Actions:
            - ActionTypeId:
                Category: Build
                Owner: AWS
                Provider: CodeBuild
                Version: "1"
              Configuration:
                ProjectName:
                  Ref: Deploy{{loop.index}}Project
              InputArtifacts:
                - Name: BuildOutput
              Name: Deploy
              RoleArn:
                Fn::GetAtt:
                  - PipelineDeployCodePipelineActionRole
                  - Arn
              RunOrder: 1
          Name: 'Deploy{{service_instance.name}}'         # resource parameter
{%- endfor %}
      ArtifactStore:
        EncryptionKey:
          Id:
            Fn::GetAtt:
              - PipelineArtifactsBucketEncryptionKey
              - Arn
          Type: KMS
        Location:
          Ref: PipelineArtifactsBucket
        Type: S3
```

```
    DependsOn:
      - PipelineRoleDefaultPolicy
      - PipelineRole
  PipelineBuildCodePipelineActionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Action: sts:AssumeRole
            Effect: Allow
            Principal:
              AWS:
                Fn::Join:
                  - ""
                  - - "arn:"
                    - Ref: AWS::Partition
                    - ":iam::"
                    - Ref: AWS::AccountId
                    - :root
        Version: "2012-10-17"
  PipelineBuildCodePipelineActionRoleDefaultPolicy:
    Type: AWS::IAM::Policy
    Properties:
      PolicyDocument:
        Statement:
          - Action:
              - codebuild:BatchGetBuilds
              - codebuild:StartBuild
              - codebuild:StopBuild
            Effect: Allow
            Resource:
              Fn::GetAtt:
                - BuildProject
                - Arn
        Version: "2012-10-17"
      PolicyName: PipelineBuildCodePipelineActionRoleDefaultPolicy
      Roles:
        - Ref: PipelineBuildCodePipelineActionRole
  PipelineDeployCodePipelineActionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Action: sts:AssumeRole
```

```
            Effect: Allow
            Principal:
              AWS:
                Fn::Join:
                  - ""
                  - - "arn:"
                    - Ref: AWS::Partition
                    - ":iam::"
                    - Ref: AWS::AccountId
                    - :root
        Version: "2012-10-17"
  PipelineDeployCodePipelineActionRoleDefaultPolicy:
    Type: AWS::IAM::Policy
    Properties:
      PolicyDocument:
        Statement:
          - Action:
              - codebuild:BatchGetBuilds
              - codebuild:StartBuild
              - codebuild:StopBuild
            Effect: Allow
            Resource:
              Fn::Join:
                - ""
                - - "arn:"
                  - Ref: AWS::Partition
                  - ":codebuild:"
                  - Ref: AWS::Region
                  - ":"
                  - Ref: AWS::AccountId
                  - ":project/Deploy*"
        Version: "2012-10-17"
      PolicyName: PipelineDeployCodePipelineActionRoleDefaultPolicy
      Roles:
        - Ref: PipelineDeployCodePipelineActionRole
Outputs:
  PipelineEndpoint:
    Description: The URL to access the pipeline
    Value: !Sub "https://${AWS::Region}.console.aws.amazon.com/codesuite/codepipeline/
pipelines/${Pipeline}/view?region=${AWS::Region}"
```

CodeBuild 프로비저닝 템플릿 번들

CodeBuild 프로비저닝을 사용하면 IaC 템플릿을 사용하여 IaC 파일을 렌더링하고 IaC 프로비저닝 엔진을 사용하여 실행하는 대신 셸 명령을 AWS Proton 간단히 실행합니다. 이를 위해 AWS Proton 는 환경 계정에서 환경에 대한 AWS CodeBuild 프로젝트를 생성하고 각 AWS Proton 리소스 생성 또는 업데이트에 대해 명령을 실행하는 작업을 시작합니다. 템플릿 번들을 작성할 때는 인프라 프로비저닝 및 프로비저닝 해제 명령과 이러한 명령에 필요할 수 있는 모든 프로그램, 스크립트 및 기타 파일을 지정하는 매니페스트를 제공합니다. 명령은 AWS Proton 이 제공하는 입력을 읽을 수 있으며, 인프라를 프로비저닝 또는 프로비저닝 해제하고 출력 값을 생성할 수 있습니다.

또한 매니페스트는가 코드가 입력하고 입력 값을 가져올 수 있는 입력 파일을 AWS Proton 렌더링하는 방법을 지정합니다. JSON 또는 HCL로 렌더링할 수 있습니다. 파라미터에 대한 자세한 내용은 the section called “CodeBuild 프로비저닝 파라미터”를 참조하세요. 매니페스트 파일에 대한 자세한 내용은 the section called “매니페스트 및 마무리” 단원을 참조하세요.

> **Note**
>
> CodeBuild 프로비저닝을 환경 및 서비스와 함께 사용할 수 있습니다. 현재로서는 이 방법으로 구성 요소를 프로비저닝할 수 없습니다.

예: CodeBuild 프로비저닝과 AWS CDK 함께 사용

CodeBuild 프로비저닝을 사용하는 예로를 사용하여 AWS 리소스를 AWS Cloud Development Kit (AWS CDK) 프로비저닝(배포) 및 프로비저닝 해제(파기)하는 코드와 CDK를 설치하고 CDK 코드를 실행하는 매니페스트를 포함할 수 있습니다.

다음 섹션에는 AWS CDK를 사용하여 환경을 프로비저닝하는 CodeBuild 프로비저닝 템플릿 번들에 포함할 수 있는 예제 파일이 나열되어 있습니다.

매니페스트

다음 매니페스트 파일은 CodeBuild 프로비저닝을 지정하며를 설치하고 사용하는 데 필요한 명령 AWS CDK, 출력 파일 처리 및 출력을에 다시 보고하는 작업을 포함합니다 AWS Proton.

Example infrastructure/manifest.yaml

```
infrastructure:
  templates:
    - rendering_engine: codebuild
      settings:
```

```
        image: aws/codebuild/amazonlinux2-x86_64-standard:4.0
        runtimes:
          nodejs: 16
        provision:
          - npm install
          - npm run build
          - npm run cdk bootstrap
          - npm run cdk deploy -- --require-approval never --outputs-file proton-
outputs.json
          - jq 'to_entries | map_values(.value) | add | to_entries | map({key:.key,
 valueString:.value})' < proton-outputs.json > outputs.json
          - aws proton notify-resource-deployment-status-change --resource-arn
 $RESOURCE_ARN --status IN_PROGRESS --outputs file://./outputs.json
        deprovision:
          - npm install
          - npm run build
          - npm run cdk destroy
        project_properties:
          VpcConfig:
            VpcId: "{{ environment.inputs.codebuild_vpc_id }}"
            Subnets: "{{ environment.inputs.codebuild_subnets }}"
            SecurityGroupIds: "{{ environment.inputs.codebuild_security_groups }}"
```

스키마

다음 스키마 파일은 환경 파라미터를 정의합니다. AWS CDK 코드는 배포 중에 이러한 파라미터의 값을 참조할 수 있습니다.

Example schema/schema.yaml

```
schema:
  format:
    openapi: "3.0.0"
  environment_input_type: "MyEnvironmentInputType"
  types:
    MyEnvironmentInputType:
      type: object
      description: "Input properties for my environment"
      properties:
        my_sample_input:
          type: string
          description: "This is a sample input"
          default: "hello world"
```

```
        my_other_sample_input:
          type: string
          description: "Another sample input"
      required:
        - my_other_sample_input
```

AWS CDK 파일

다음 파일은 Node.js CDK 프로젝트의 예제입니다.

Example infrastructure/package.json

```
{
  "name": "ProtonEnvironment",
  "version": "0.1.0",
  "bin": {
    "ProtonEnvironmente": "bin/ProtonEnvironment.js"
  },
  "scripts": {
    "build": "tsc",
    "watch": "tsc -w",
    "test": "jest",
    "cdk": "cdk"
  },
  "devDependencies": {
    "@types/jest": "^28.1.7",
    "@types/node": "18.7.6",
    "jest": "^28.1.3",
    "ts-jest": "^28.0.8",
    "aws-cdk": "2.37.1",
    "ts-node": "^10.9.1",
    "typescript": "~4.7.4"
  },
  "dependencies": {
    "aws-cdk-lib": "2.37.1",
    "constructs": "^10.1.77",
    "source-map-support": "^0.5.21"
  }
}
```

Example infrastructure/tsconfig.json

```
{
```

```
  "compilerOptions": {
    "target": "ES2018",
    "module": "commonjs",
    "lib": [
      "es2018"
    ],
    "declaration": true,
    "strict": true,
    "noImplicitAny": true,
    "strictNullChecks": true,
    "noImplicitThis": true,
    "alwaysStrict": true,
    "noUnusedLocals": false,
    "noUnusedParameters": false,
    "noImplicitReturns": true,
    "noFallthroughCasesInSwitch": false,
    "inlineSourceMap": true,
    "inlineSources": true,
    "experimentalDecorators": true,
    "strictPropertyInitialization": false,
    "resolveJsonModule": true,
    "esModuleInterop": true,
    "typeRoots": [
      "./node_modules/@types"
    ]
  },
  "exclude": [
    "node_modules",
    "cdk.out"
  ]
}
```

Example infrastructure/cdk.json

```
{
  "app": "npx ts-node --prefer-ts-exts bin/ProtonEnvironment.ts",
  "outputsFile": "proton-outputs.json",
  "watch": {
    "include": [
      "**"
    ],
    "exclude": [
      "README.md",
```

```
      "cdk*.json",
      "**/*.d.ts",
      "**/*.js",
      "tsconfig.json",
      "package*.json",
      "yarn.lock",
      "node_modules",
      "test"
    ]
  },
  "context": {
    "@aws-cdk/aws-apigateway:usagePlanKeyOrderInsensitiveId": true,
    "@aws-cdk/core:stackRelativeExports": true,
    "@aws-cdk/aws-rds:lowercaseDbIdentifier": true,
    "@aws-cdk/aws-lambda:recognizeVersionProps": true,
    "@aws-cdk/aws-cloudfront:defaultSecurityPolicyTLSv1.2_2021": true,
    "@aws-cdk-containers/ecs-service-extensions:enableDefaultLogDriver": true,
    "@aws-cdk/aws-ec2:uniqueImdsv2TemplateName": true,
    "@aws-cdk/core:target-partitions": [
      "aws",
      "aws-cn"
    ]
  }
}
```

Example infrastructure/bin/ProtonEnvironment.ts

```
#!/usr/bin/env node
import 'source-map-support/register';
import * as cdk from 'aws-cdk-lib';
import { ProtonEnvironmentStack } from '../lib/ProtonEnvironmentStack';

const app = new cdk.App();
new ProtonEnvironmentStack(app, 'ProtonEnvironmentStack', {});
```

Example infrastructure/lib/ProtonEnvironmentStack.ts

```
import { Stack, StackProps } from 'aws-cdk-lib';
import { Construct } from 'constructs';
import * as cdk from 'aws-cdk-lib';
import * as ssm from 'aws-cdk-lib/aws-ssm';
import input from '../proton-inputs.json';
```

```
export class ProtonEnvironmentStack extends Stack {
  constructor(scope: Construct, id: string, props?: StackProps) {
    super(scope, id, { ...props, stackName: process.env.STACK_NAME });

    const ssmParam = new ssm.StringParameter(this, "ssmParam", {
      stringValue: input.environment.inputs.my_sample_input,
      parameterName: `${process.env.STACK_NAME}-Param`,
      tier: ssm.ParameterTier.STANDARD
    })

    new cdk.CfnOutput(this, 'ssmParamOutput', {
      value: ssmParam.parameterName,
      description: 'The name of the ssm parameter',
      exportName: `${process.env.STACK_NAME}-Param`
    });
  }
}
```

렌더링된 입력 파일

CodeBuild 기반 프로비저닝 템플릿을 사용하여 환경을 만들면 AWS Proton 은 사용자가 제공한 입력 파라미터 값이 포함된 입력 파일을 렌더링합니다. 코드에서 이러한 값을 참조할 수 있습니다. 다음 파일은 렌더링된 입력 파일의 예제입니다.

Example infrastructure/proton-inputs.json

```
{
  "environment": {
    "name": "myenv",
    "inputs": {
      "my_sample_input": "10.0.0.0/16",
      "my_other_sample_input": "11.0.0.0/16"
    }
  }
}
```

Terraform IaC 파일

Terraform 코드형 인프라(IaC) 파일을와 함께 사용하는 방법을 알아봅니다 AWS Proton. Terraform은 HashiCorp에서 개발한 널리 사용되는 오픈 소스 IaC 엔진입니다. Terraform 모듈은 HashiCorp의 HCL 언어로 개발되었으며 Web Services를 비롯한 여러 백엔드 인프라 제공업체를 지원합니다.

AWS Proton 는 Terraform IaC에 대한 자체 관리형 프로비저닝을 지원합니다.

풀 리퀘스트에 응답하고 인프라 프로비저닝을 구현하는 프로비저닝 리포지토리의 전체 예시는 GitHub 의 AWS Proton용 Terraform 오픈 소스 GitHub 작업 자동화 템플릿을 참조하세요.

Terraform IAC 템플릿 번들 파일에서 자체 관리형 프로비저닝이 작동하는 방식:

1. Terraform 템플릿 번들에서 환경을 생성하면가 콘솔 또는 `spec file` 입력 파라미터로 `.tf` 파일을 AWS Proton 컴파일합니다.
2. 컴파일된 IaC 파일을 AWS Proton에 등록한 리포지토리에 병합하기 위해 풀 리퀘스트를 생성합니다.
3. 요청이 승인되면는 사용자가 제공한 프로비저닝 상태를 AWS Proton 기다립니다.
4. 요청이 거부되면 환경 생성이 취소됩니다.
5. 풀 리퀘스트 제한 시간이 초과되면 환경 생성이 완료되지 않습니다.

AWS Proton Terraform IaC 고려 사항 포함:

- AWS Proton 는 Terraform 프로비저닝을 관리하지 않습니다.
- 프로비저닝 리포지토리를에 등록해야 합니다 AWS Proton.이 리포지토리에 대한 풀 요청을 AWS Proton 수행합니다.
- 프로비저닝 리포지토리 AWS Proton 에 연결하려면 CodeStar 연결을 생성해야 합니다.
- AWS Proton 컴파일된 IaC 파일에서를 프로비저닝하려면 AWS Proton 풀 요청에 응답해야 합니다. AWS Proton 는 환경 및 서비스 생성 및 업데이트 작업 후 풀 요청을 수행합니다. 자세한 내용은 AWS Proton 환경 및 AWS Proton 서비스 섹션을 참조하세요.
- AWS Proton 컴파일된 IaC 파일에서 파이프라인을 프로비저닝하려면 CI/CD 파이프라인 리포지토리를 생성해야 합니다.
- 풀 요청 기반 프로비저닝 자동화에는 AWS Proton 프로비저닝된 AWS Proton 리소스 상태 변경을 알리는 단계가 포함되어야 합니다. AWS Proton NotifyResourceDeploymentStatusChange API를 사용할 수 있습니다.
- CloudFormation IaC 파일에서 생성된 서비스, 파이프라인 및 구성 요소를 Terraform IaC 파일로 만든 환경에 배포할 수 없습니다.
- Terraform IaC 파일에서 생성된 서비스, 파이프라인 및 구성 요소를 CloudFormation IaC 파일로 만든 환경에 배포할 수 없습니다.

Terraform IaC 파일을 준비 AWS Proton할 때 다음 예제와 같이 입력 변수에 네임스페이스를 연결합니다. 자세한 내용은 파라미터를 참조하세요.

예제 1: AWS Proton environment Terraform IaC 파일

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 3.0"
    }
  }
  // This tells terraform to store the state file in s3 at the location
  // s3://terraform-state-bucket/tf-os-sample/terraform.tfstate
  backend "s3" {
    bucket = "terraform-state-bucket"
    key    = "tf-os-sample/terraform.tfstate"
    region = "us-east-1"
  }
}

// Configure the AWS Provider
provider "aws" {
  region = "us-east-1"
  default_tags {
    tags = var.proton_tags
  }
}

resource "aws_ssm_parameter" "my_ssm_parameter" {
  name  = "my_ssm_parameter"
  type  = "String"
  // Use the Proton environment.inputs. namespace
  value = var.environment.inputs.ssm_parameter_value
}
```

컴파일된 코드형 인프라

환경 또는 서비스를 생성할 때는 콘솔 또는 `spec file` 입력을 사용하여 코드 파일로 인프라를 AWS Proton 컴파일합니다. 다음 예시와 같이 입력에 대한 `proton.`*`resource-type`*`.variables.tf` 및 `proton.auto.tfvars.json` 파일을 생성하여 Terraform에서 사용할 수 있습니다. 이러한 파일은 환경 또는 서비스 인스턴스 이름과 일치하는 폴더의 지정된 리포지토리에 있습니다.

이 예제에서는가 변수 정의 및 변수 값에 태그를 AWS Proton 포함하는 방법과 이러한 AWS Proton 태그를 프로비저닝된 리소스에 전파하는 방법을 보여줍니다. 자세한 내용은 the section called “프로비저닝된 리소스에 태그 전파” 단원을 참조하십시오.

예 2: 이름이 "dev"인 환경을 위해 컴파일된 IaC 파일.

dev/environment.tf:

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 3.0"
    }
  }
  // This tells terraform to store the state file in s3 at the location
  // s3://terraform-state-bucket/tf-os-sample/terraform.tfstate
  backend "s3" {
    bucket = "terraform-state-bucket"
    key    = "tf-os-sample/terraform.tfstate"
    region = "us-east-1"
  }
}

// Configure the AWS Provider
provider "aws" {
  region = "us-east-1"
  default_tags {
    tags = var.proton_tags
  }
}

resource "aws_ssm_parameter" "my_ssm_parameter" {
  name  = "my_ssm_parameter"
  type  = "String"
  // Use the Proton environment.inputs. namespace
  value = var.environment.inputs.ssm_parameter_value
}
```

dev/proton.environment.variables.tf:

```
variable "environment" {
  type = object({
```

```
    inputs = map(string)
    name = string
  })
}

variable "proton_tags" {
  type = map(string)
  default = null
}
```

dev/proton.auto.tfvars.json:

```
{
  "environment": {
    "name": "dev",
    "inputs": {
      "ssm_parameter_value": "MyNewParamValue"
    }
  }

  "proton_tags" : {
    "proton:account" : "123456789012",
    "proton:template" : "arn:aws:proton:us-east-1:123456789012:environment-template/
fargate-env",
    "proton:environment" : "arn:aws:proton:us-east-1:123456789012:environment/dev"
  }
}
```

리포지토리 경로

AWS Proton 는 환경 또는 서비스 생성 작업의 콘솔 또는 사양 입력을 사용하여 컴파일된 IaC 파일을 찾을 리포지토리와 경로를 찾습니다. 입력 값은 네임스페이스 입력 파라미터에 전달됩니다.

AWS Proton 는 두 개의 리포지토리 경로 레이아웃을 지원합니다. 다음 예제에서는 두 환경의 네임스페이스 리소스 파라미터로 경로의 이름을 지정합니다. 각 환경에는 두 서비스의 서비스 인스턴스가 있으며, 이 중 하나의 서비스 인스턴스에는 구성 요소가 직접 정의되어 있습니다.

리소스 유형	이름 파라미터	=	리소스 이름
환경	`environment.name`	=	`"env-prod"`

리소스 유형	이름 파라미터	=	리소스 이름
환경	environment.name		"env-staged"
서비스	service.name		"service-one"
서비스 인스턴스	service_instance.name		"instance-one-prod"
서비스 인스턴스	service_instance.name		"instance-one-staged"
서비스	service.name		"service-two"
서비스 인스턴스	service_instance.name		"instance-two-prod"
구성 요소	service_instance.components.default.name		"component-prod"
서비스 인스턴스	service_instance.name		"instance-two-staged"
구성 요소	service_instance.components.default.name		"component-staged"

Layout 1

가 environments 폴더가 있는 지정된 리포지토리를 AWS Proton 찾으면 컴파일된 IaC 파일을 포함하고 로 이름이 지정된 폴더가 생성됩니다environment.name.

에서 서비스 인스턴스 호환 환경 이름과 일치하는 environments 폴더 이름이 포함된 폴더가 있는 지정된 리포지토리를 AWS Proton 찾으면 컴파일된 인스턴스 IaC 파일을 포함하고 로 이름이 지정된 폴더가 생성됩니다service_instance.name.

```
/repo
    /environments
        /env-prod                           # environment folder
            main.tf
            proton.environment.variables.tf
            proton.auto.tfvars.json

            /service-one-instance-one-prod    # instance folder
                main.tf
                proton.service_instance.variables.tf
                proton.auto.tfvars.json

            /service-two-instance-two-prod    # instance folder
                main.tf
                proton.service_instance.variables.tf
                proton.auto.tfvars.json

            /component-prod                   # component folder
                main.tf
                proton.component.variables.tf
                proton.auto.tfvars.json

        /env-staged                           # environment folder
            main.tf
            proton.variables.tf
            proton.auto.tfvars.json

            /service-one-instance-one-staged  # instance folder
                main.tf
                proton.service_instance.variables.tf
                proton.auto.tfvars.json

            /service-two-instance-two-staged  # instance folder
                main.tf
                proton.service_instance.variables.tf
                proton.auto.tfvars.json

            /component-staged                 # component folder
                main.tf
```

```
                proton.component.variables.tf
                proton.auto.tfvars.json
```

Layout 2

가 environments 폴더 없이 지정된 리포지토리를 AWS Proton 찾으면 컴파일된 환경 IaC 파일을 찾을 수 있는 environment.name 폴더가 생성됩니다.

가 서비스 인스턴스 호환 환경 이름과 일치하는 폴더 이름을 가진 지정된 리포지토리를 AWS Proton 찾으면 컴파일된 인스턴스 IaC 파일을 찾을 service_instance.name 폴더를 생성합니다.

```
/repo
    /env-prod                           # environment folder
        main.tf
        proton.environment.variables.tf
        proton.auto.tfvars.json

        /service-one-instance-one-prod    # instance folder
            main.tf
            proton.service_instance.variables.tf
            proton.auto.tfvars.json

        /service-two-instance-two-prod    # instance folder
            main.tf
            proton.service_instance.variables.tf
            proton.auto.tfvars.json

        /component-prod                   # component folder
            main.tf
            proton.component.variables.tf
            proton.auto.tfvars.json

    /env-staged                           # environment folder
        main.tf
        proton.variables.tf
        proton.auto.tfvars.json

        /service-one-instance-one-staged  # instance folder
            main.tf
            proton.service_instance.variables.tf
            proton.auto.tfvars.json
```

```
        /service-two-instance-two-staged  # instance folder
            main.tf
            proton.service_instance.variables.tf
            proton.auto.tfvars.json

        /component-staged                 # component folder
            main.tf
            proton.component.variables.tf
            proton.auto.tfvars.json
```

스키마 파일

관리자는 Open API Data Models(스키마) 섹션을 사용하여 템플릿 번들에 대한 파라미터 스키마 YAML 파일을 정의할 때 AWS Proton 에서 스키마에 정의한 요구 사항을 기준으로 파라미터 값 입력을 검증할 수 있습니다.

형식 및 사용 가능한 키워드에 대한 자세한 내용은 OpenAPI의 스키마 개체 단원을 참조하세요.

환경 템플릿 번들에 대한 스키마 요구 사항

스키마는 OpenAPI의 YAML 형식의 데이터 모델 (스키마) 섹션을 따라야 합니다. 또한 환경 템플릿 번들의 일부여야 합니다.

환경 스키마의 경우 Open API의 데이터 모델 (스키마) 섹션을 사용하고 있음을 확인하기 위해 형식이 지정된 헤더를 포함해야 합니다. 다음 환경 스키마 예제에서 이러한 헤더는 처음 세 줄에 표시됩니다.

`environment_input_type`은 반드시 사용자가 제공한 이름과 함께 포함되고 정의되어야 합니다. 다음 예제에서는 5번 줄에 정의되어 있습니다. 이 파라미터를 정의하여 AWS Proton 환경 리소스와 연결합니다.

Open API 스키마 모델을 따르려면 `types`을 포함해야 합니다. 다음 예제에서는 6번 줄입니다.

`types` 다음으로 `environment_input_type` 유형을 정의해야 합니다. 환경의 입력 파라미터를 `environment_input_type`의 속성으로 정의합니다. 환경 인프라에 스키마와 연결된 코드 (IaC) 파일로 나열된 하나 이상의 파라미터와 일치하는 이름을 가진 속성을 하나 이상 포함해야 합니다.

환경을 생성하고 사용자 지정 파라미터 값을 제공하면는 스키마 파일을 AWS Proton 사용하여 일치시키고, 검증하고, 연결된 CloudFormation IaC 파일의 중괄호로 묶인 파라미터에 삽입합니다. 각 속성(파라미터)에 대해 `name`와 `type`를 입력합니다. 선택적으로 `description`, `default` 및 `pattern`도 제공합니다.

다음 예제 표준 환경 템플릿 스키마의 정의된 파라미터에는 `default` 키워드 및 기본값과 함께 `vpc_cidr`, `subnet_one_cidr` 및 `subnet_two_cidr`이 포함됩니다. 이 환경 템플릿 번들 스키마를 사용하여 환경을 만들 때 기본값을 그대로 사용하거나 사용자 고유의 값을 제공할 수 있습니다. 파라미터가 기본값이 없고 `required` 속성(파라미터)으로 나열되어 있는 경우 환경을 만들 때 파라미터에 대한 값을 제공해야 합니다.

두 번째 예제 표준 환경 템플릿 스키마는 `required` 파라미터 `my_other_sample_input`을 나열합니다.

두 가지 환경 템플릿 유형에 대한 스키마를 생성할 수 있습니다. 자세한 내용은 템플릿 등록 및 게시 단원을 참조하세요.

- 표준 환경 템플릿

  다음 예제에서는 환경 입력 유형이 설명과 입력 속성으로 정의됩니다. 이 스키마 예제는 예제 3에 표시된 AWS Proton CloudFormation IaC 파일과 함께 사용할 수 있습니다.

  표준 환경 템플릿의 예제 스키마:

```
schema:                             # required
  format:                           # required
    openapi: "3.0.0"                # required
  # required              defined by administrator
  environment_input_type: "PublicEnvironmentInput"
  types:                            # required
    # defined by administrator
    PublicEnvironmentInput:
      type: object
      description: "Input properties for my environment"
      properties:
        vpc_cidr:                    # parameter
          type: string
          description: "This CIDR range for your VPC"
          default: 10.0.0.0/16
          pattern: ([0-9]{1,3}\.){3}[0-9]{1,3}($|/(16|24))
        subnet_one_cidr:             # parameter
          type: string
          description: "The CIDR range for subnet one"
          default: 10.0.0.0/24
          pattern: ([0-9]{1,3}\.){3}[0-9]{1,3}($|/(16|24))
        subnet_two_cidr:             # parameter
          type: string
```

```
          description: "The CIDR range for subnet one"
          default: 10.0.1.0/24
          pattern: ([0-9]{1,3}\.){3}[0-9]{1,3}($|/(16|24))
```

`required` 파라미터가 포함된 표준 환경 템플릿의 스키마 예시:

```
schema:                           # required
  format:                         # required
    openapi: "3.0.0"              # required
  # required              defined by administrator
  environment_input_type: "MyEnvironmentInputType"
  types:                          # required
    # defined by administrator
    MyEnvironmentInputType:
      type: object
      description: "Input properties for my environment"
      properties:
        my_sample_input:          # parameter
          type: string
          description: "This is a sample input"
          default: "hello world"
        my_other_sample_input:    # parameter
          type: string
          description: "Another sample input"
        another_optional_input:   # parameter
          type: string
          description: "Another optional input"
          default: "!"
      required:
        - my_other_sample_input
```

- 고객 관리형 환경 템플릿

  다음 예제의 스키마에는 고객 관리형 인프라를 프로비저닝하는 데 사용한 IaC의 출력을 복제하는 출력 목록만 포함되어 있습니다. 출력 값 유형은 문자열로만 정의해야 합니다(목록, 배열 또는 기타 유형은 정의하지 않아야 함). 예를 들어, 다음 코드 스니펫은 외부 CloudFormation 템플릿의 출력 섹션을 보여줍니다. 예제 1 에 표시된 템플릿에서 가져온 것입니다. 예제 4에서 생성된 AWS Proton Fargate 서비스에 대한 외부 고객 관리형 인프라를 생성하는 데 사용할 수 있습니다.

> **Important**
>
> 관리자는 프로비저닝 및 관리형 인프라와 모든 출력 파라미터가 연결된 고객 관리형 환경 템플릿과 호환되는지 확인해야 합니다. 이러한 변경 사항은 표시되지 않으므로는 사용자를 대신하여 변경 사항을 AWS Proton 설명할 수 없습니다 AWS Proton. 불일치로 인해 장애가 발생합니다.

고객 관리형 환경 템플릿에 대한 CloudFormation IAC 파일 출력 예시:

```
// Cloudformation Template Outputs
[...]
Outputs:
  ClusterName:
    Description: The name of the ECS cluster
    Value: !Ref 'ECSCluster'
  ECSTaskExecutionRole:
    Description: The ARN of the ECS role
    Value: !GetAtt 'ECSTaskExecutionRole.Arn'
  VpcId:
    Description: The ID of the VPC that this stack is deployed in
    Value: !Ref 'VPC'
[...]
```

해당 AWS Proton 고객 관리형 환경 템플릿 번들의 스키마는 다음 예제에 나와 있습니다. 각 출력 값은 문자열로 정의됩니다.

고객 관리형 환경 템플릿의 예제 스키마:

```
schema:                             # required
  format:                           # required
    openapi: "3.0.0"                # required
  # required              defined by administrator
  environment_input_type: "EnvironmentOutput"
  types:                            # required
    # defined by administrator
    EnvironmentOutput:
      type: object
      description: "Outputs of the environment"
      properties:
```

```
        ClusterName:               # parameter
          type: string
          description: "The name of the ECS cluster"
        ECSTaskExecutionRole:      # parameter
          type: string
          description: "The ARN of the ECS role"
        VpcId:                     # parameter
          type: string
          description: "The ID of the VPC that this stack is deployed in"
[...]
```

서비스 템플릿 번들에 대한 스키마 요구 사항

스키마는 다음 예시와 같이 YAML 형식의 OpenAPI의 데이터 모델(스키마) 섹션을 따라야 합니다. 서비스 템플릿 번들에 스키마 파일을 제공해야 합니다.

다음 서비스 스키마 예제에는 형식이 지정된 헤더를 포함해야 합니다. 다음 예제에서는 처음 세 줄에 해당합니다. 이는 Open API의 데이터 모델 (스키마) 섹션을 사용하고 있음을 확인하기 위한 것입니다.

`service_input_type`은 반드시 사용자가 제공한 이름과 함께 포함되고 정의되어야 합니다. 다음 예제에서는 5번 줄입니다. 이렇게 하면 파라미터가 AWS Proton 서비스 리소스와 연결됩니다.

서비스 AWS Proton 파이프라인은 콘솔 또는 CLI를 사용하여 서비스를 생성할 때 기본적으로 포함됩니다. 서비스에 대한 서비스 파이프라인을 포함할 때는 제공한 이름과 `pipeline_input_type`를 포함해야 합니다. 다음 예제에서는 7번 줄입니다. AWS Proton 서비스 파이프라인을 포함하지 않는 경우 이 파라미터를 포함하지 마십시오. 자세한 내용은 템플릿 등록 및 게시 단원을 참조하십시오.

Open API 스키마 모델을 따르려면 `types`을 포함해야 합니다. 다음 예제에서는 9번째 줄에 있습니다.

`types` 다음으로 `service_input_type` 유형을 정의해야 합니다. 서비스의 입력 파라미터를 `service_input_type`의 속성으로 정의합니다. 스키마와 연결된 서비스 코드형 인프라(IaC) 파일에 나열된 하나 이상의 파라미터와 일치하는 이름의 속성을 하나 이상 포함해야 합니다.

서비스 파이프라인을 정의하려면 `service_input_type` 정의 아래에 `pipeline_input_type`를 정의해야 합니다. 스키마와 연결된 파이프라인 코드형 인프라(IaC) 파일에 나열된 하나 이상의 파라미터와 일치하는 이름의 속성을 하나 이상 포함해야 합니다. AWS Proton 서비스 파이프라인을 포함하지 않는 경우이 정의를 포함하지 마십시오.

관리자 또는 개발자가 서비스를 생성하고 사용자 지정 파라미터 값을 제공하는 경우는 스키마 파일을 AWS Proton 사용하여 일치하는지 확인하고 연결된 CloudFormation IaC 파일의 중괄호로 묶인 파라미

터에 삽입합니다. 각 속성(파라미터)에 대해 name와 type를 입력합니다. 선택적으로 description, default 및 pattern도 제공합니다.

예제 스키마에 정의된 파라미터에는 키워드 및 기본값과 port, desired_count, task_size, image 및 default이 포함됩니다. 이 서비스 템플릿 번들 스키마를 사용하여 서비스를 만들 때 기본값을 사용하거나 고유한 값을 제공할 수 있습니다. 파라미터 unique_name도 예제에 포함되어 있으며 기본값이 없습니다. required 속성(파라미터)으로 나열됩니다. 관리자 또는 개발자는 서비스를 생성할 때 required 파라미터 값을 제공해야 합니다.

서비스 파이프라인이 포함된 서비스 템플릿을 생성하려면 스키마에 pipeline_input_type를 포함합니다.

서비스 파이프라인이 포함된 서비스에 대한 AWS Proton 서비스 스키마 파일의 예입니다.

이 스키마 예제는 예제 4 및 예제 5에 표시된 AWS Proton IaC 파일과 함께 사용할 수 있습니다. ??? 서비스 파이프라인이 포함됩니다.

```
schema:                             # required
  format:                           # required
    openapi: "3.0.0"                # required
  # required           defined by administrator
  service_input_type: "LoadBalancedServiceInput"
  # only include if including AWS Proton service pipeline, defined by administrator
  pipeline_input_type: "PipelineInputs"

  types:                            # required
    # defined by administrator
    LoadBalancedServiceInput:
      type: object
      description: "Input properties for a loadbalanced Fargate service"
      properties:
        port:                       # parameter
          type: number
          description: "The port to route traffic to"
          default: 80
          minimum: 0
          maximum: 65535
        desired_count:              # parameter
          type: number
          description: "The default number of Fargate tasks you want running"
          default: 1
          minimum: 1
```

```
        task_size:                # parameter
          type: string
          description: "The size of the task you want to run"
          enum: ["x-small", "small", "medium", "large", "x-large"]
          default: "x-small"
        image:                    # parameter
          type: string
          description: "The name/url of the container image"
          default: "public.ecr.aws/z9d2n7e1/nginx:1.19.5"
          minLength: 1
          maxLength: 200
        unique_name:              # parameter
          type: string
          description: "The unique name of your service identifier. This will be used
 to name your log group, task definition and ECS service"
          minLength: 1
          maxLength: 100
      required:
        - unique_name
    # defined by administrator
    PipelineInputs:
      type: object
      description: "Pipeline input properties"
      properties:
        dockerfile:               # parameter
          type: string
          description: "The location of the Dockerfile to build"
          default: "Dockerfile"
          minLength: 1
          maxLength: 100
        unit_test_command:        # parameter
          type: string
          description: "The command to run to unit test the application code"
          default: "echo 'add your unit test command here'"
          minLength: 1
          maxLength: 200
```

서비스 파이프라인 없이 서비스 템플릿을 만들려면 다음 예시와 같이 스키마에 `pipeline_input_type`를 포함하지 마십시오.

서비스 파이프라인이 포함되지 않은 서비스에 대한 AWS Proton 서비스 스키마 파일의 예

```
schema:                           # required
```

```
  format:                           # required
    openapi: "3.0.0"                # required
  # required            defined by administrator
  service_input_type: "MyServiceInstanceInputType"

  types:                            # required
    # defined by administrator
    MyServiceInstanceInputType:
      type: object
      description: "Service instance input properties"
      required:
        - my_sample_service_instance_required_input
      properties:
        my_sample_service_instance_optional_input:   # parameter
          type: string
          description: "This is a sample input"
          default: "hello world"
        my_sample_service_instance_required_input:   # parameter
          type: string
          description: "Another sample input"
```

에 대한 템플릿 파일 정리 AWS Proton

환경 및 서비스 코드형 인프라(IaC) 파일과 해당 스키마 파일을 준비한 후에는 디렉토리로 구성해야 합니다. 또한 매니페스트 YAML 파일을 만들어야 합니다. 매니페스트 파일에는 디렉토리의 IaC 파일, 렌더링 엔진 및 이 템플릿의 IaC를 개발하는 데 사용된 템플릿 언어가 나열됩니다.

> **Note**
>
> 매니페스트 파일은 템플릿 번들과 매니페스트독립적으로 직접 정의된 구성 요소에 대한 직접 입력으로 사용할 수도 있습니다. 이 경우 클라우드포메이션과 테라폼 모두에 대해 항상 단일 IaC 템플릿 파일을 지정합니다. 구성 요소에 대한 자세한 내용은 구성 요소를 참조하세요.

매니페스트 파일은 다음 예제와 일치해야 합니다.

CloudFormation 매니페스트 파일 형식:

CloudFormation을 사용하면 하나의 파일을 나열할 수 있습니다.

```
infrastructure:
```

```
  templates:
    - file: "cloudformation.yaml"
      rendering_engine: jinja
      template_language: cloudformation
```

Terraform 매니페스트 파일 형식:

Terraform을 사용하면 단일 파일을 명시적으로 나열하거나 와일드카드 *를 사용하여 디렉터리의 각 파일을 나열할 수 있습니다.

> **Note**
>
> 와일드카드에는 이름이 `.tf`로 끝나는 파일만 포함됩니다. 다른 파일은 무시됩니다.

```
infrastructure:
  templates:
    - file: "*"
      rendering_engine: hcl
      template_language: terraform
```

CodeBuild 기반 프로비저닝 매니페스트 파일 형식:

CodeBuild 기반 프로비저닝을 사용하면 프로비저닝 및 디프로비저닝 쉘 명령을 지정할 수 있습니다.

> **Note**
>
> 번들에는 매니페스트 외에도 명령이 의존하는 모든 파일이 포함되어야 합니다.

다음 예제 매니페스트는 CodeBuild 기반 프로비저닝을 사용하여 AWS Cloud Development Kit (AWS CDK) ()를 사용하여 리소스를 프로비저닝(배포) 및 프로비저닝 해제(파기)합니다AWS CDK. 템플릿 번들에는 CDK 코드도 포함되어야 합니다.

프로비저닝 중에 AWS Proton은 템플릿의 스키마에 `proton-input.json` 이름으로 정의한 입력 파라미터 값이 포함된 입력 파일을 생성합니다.

```
infrastructure:
  templates:
```

```
    - rendering_engine: codebuild
      settings:
        image: aws/codebuild/amazonlinux2-x86_64-standard:4.0
        runtimes:
          nodejs: 16
        provision:
          - npm install
          - npm run build
          - npm run cdk bootstrap
          - npm run cdk deploy -- --require-approval never --outputs-file proton-
outputs.json
          - jq 'to_entries | map_values(.value) | add | to_entries | map({key:.key,
 valueString:.value})' < proton-outputs.json > outputs.json
          - aws proton notify-resource-deployment-status-change --resource-arn
 $RESOURCE_ARN --status IN_PROGRESS --outputs file://./outputs.json
        deprovision:
          - npm install
          - npm run build
          - npm run cdk destroy
        project_properties:
          VpcConfig:
            VpcId: "{{ environment.inputs.codebuild_vpc_id }}"
            Subnets: "{{ environment.inputs.codebuild_subnets }}"
            SecurityGroupIds: "{{ environment.inputs.codebuild_security_groups }}"
```

환경 또는 서비스 템플릿 번들에 대한 디렉터리 및 매니페스트 파일을 설정한 후 디렉터리를 tar 볼에 압축하여가 검색할 AWS Proton 수 있는 Amazon Simple Storage Service(Amazon S3) 버킷 또는 템플릿 동기화 Git 리포지토리에 업로드합니다.

등록한 환경 또는 서비스 템플릿의 마이너 버전을 생성할 때 S3 버킷에 있는 환경 또는 서비스 템플릿 번들 tar 볼의 경로를 AWS Proton제공합니다. 새 템플릿 마이너 버전으로 AWS Proton 저장합니다. 환경 또는 서비스를 생성하거나 업데이트할 새 템플릿 마이너 버전을 선택할 수 있습니다 AWS Proton.

환경 템플릿 번들 마무리

생성하는 환경 템플릿 번들에는 두 가지 유형이 있습니다 AWS Proton.

- 표준 환경 템플릿용 환경 템플릿 번들을 만들려면 다음 환경 템플릿 번들 디렉터리 구조와 같이 스키마, 코드형 인프라(IaC) 파일 및 매니페스트 파일을 디렉터리에 구성하세요.
- 고객 관리형 환경 템플릿용 환경 템플릿 번들을 만들려면 스키마 파일과 디렉토리만 제공하세요. 인프라 디렉터리와 파일을 포함하지 마십시오. 인프라 디렉터리와 파일이 포함된 경우 오류가 AWS Proton 발생합니다.

자세한 내용은 템플릿 등록 및 게시 단원을 참조하십시오.

CloudFormation 환경 템플릿 번들 디렉토리 구조:

```
/schema
  schema.yaml
/infrastructure
  manifest.yaml
  cloudformation.yaml
```

Terraform 환경 템플릿 번들 디렉토리 구조:

```
/schema
  schema.yaml
/infrastructure
  manifest.yaml
  environment.tf
```

서비스 템플릿 번들 마무리

서비스 템플릿 번들을 생성하려면 서비스 템플릿 번들 디렉터리 구조 예제와 같이 스키마, 코드형 인프라(IaC) 파일 및 매니페스트 파일을 디렉터리로 구성해야 합니다.

템플릿 번들에 서비스 파이프라인을 포함하지 않는 경우에는 이 템플릿 번들에 연결할 서비스 템플릿을 만들 때 파이프라인 디렉토리와 파일을 포함하지 말고 `"pipelineProvisioning": "CUSTOMER_MANAGED"`을 설정합니다.

> **Note**
>
> 서비스 그룹을 생성한 후에는 `pipelineProvisioning`을 변경할 수 없습니다.

자세한 내용은 템플릿 등록 및 게시 단원을 참조하세요.

CloudFormation 서비스 템플릿 번들 디렉토리 구조:

```
/schema
  schema.yaml
/instance_infrastructure
  manifest.yaml
```

```
  cloudformation.yaml
/pipeline_infrastructure
  manifest.yaml
  cloudformation.yaml
```

Terraform 서비스 템플릿 번들 디렉토리 구조:

```
/schema
  schema.yaml
/instance_infrastructure
  manifest.yaml
  instance.tf
/pipeline_infrastructure
  manifest.yaml
  pipeline.tf
```

템플릿 번들 고려 사항

- 코드형 인프라(IaC) 파일

  AWS Proton 는 템플릿에서 올바른 파일 형식을 감사합니다. 그러나 AWS Proton 는 템플릿 개발, 종속성 및 로직 오류를 확인하지 않습니다. 예를 들어 서비스 또는 환경 템플릿의 일부로 CloudFormation IaC 파일에 Amazon S3 버킷 생성을 지정했다고 가정합니다. 서비스는 이러한 템플릿을 기반으로 생성됩니다. 이제 어느 시점에서 서비스를 삭제하고 싶다고 가정해 보겠습니다. 지정된 S3 버킷이 비어 있지 않고 CloudFormation IaC 파일이 `Retain`에서 해당 버킷을 로 표시하지 않으면 서비스 삭제 작업에 `DeletionPolicy` AWS Proton 실패합니다.
- 번들 파일 크기 제한 및 형식
  - 번들 파일 크기, 개수 및 이름 크기 제한은 AWS Proton 할당량에서 확인할 수 있습니다.
  - 템플릿 번들 파일 디렉토리는 타르 볼에 압축되어 S3(Simple Storage Service) 버킷에 위치합니다.
  - 번들의 각 파일은 유효한 형식의 YAML 파일이어야 합니다.
- S3 버킷 템플릿 번들 암호화

  S3 버킷의 템플릿 번들에 저장된 민감한 데이터를 암호화하려면 SSE-S3 또는 SSE-KMS 키를 사용하여가 이를 AWS Proton 검색할 수 있도록 허용합니다.

AWS Proton 템플릿

템플릿 라이브러리에 AWS Proton 템플릿 번들을 추가하려면 템플릿 마이너 버전을 생성하여에 등록합니다 AWS Proton. 템플릿 생성 시 S3 버킷의 이름과 템플릿 번들 경로를 제공합니다. 템플릿이 게시되면 플랫폼 팀 구성원과 개발자가 템플릿을 선택할 수 있습니다. 선택한 후 템플릿을 AWS Proton 사용하여 인프라 및 애플리케이션을 생성하고 프로비저닝합니다.

관리자는 환경 템플릿을 생성하고 등록할 수 있습니다 AWS Proton. 그런 다음 이 환경 템플릿을 사용하여 여러 환경을 배포할 수 있습니다. 예를 들어, "dev", "스테이징" 및 "prod" 환경을 배포하는 데 사용할 수 있습니다. "개발" 환경에는 프라이빗 서브넷이 있는 VPC와 모든 리소스에 대한 제한적인 액세스 정책이 포함될 수 있습니다. 환경 출력은 서비스의 입력으로 사용할 수 있습니다.

환경 템플릿을 만들고 등록하여 서로 다른 두 가지 유형의 환경을 만들 수 있습니다. 사용자와 개발자 모두 AWS Proton 를 사용하여 두 유형 모두에 서비스를 배포할 수 있습니다.

- 가 환경 인프라를 프로비저닝하고 관리하는 표준 환경을 생성하는 데 AWS Proton 사용하는 표준 환경 템플릿을 등록하고 게시합니다.
- 가 기존 프로비저닝된 인프라에 연결하는 고객 관리형 환경을 생성하는 데 AWS Proton 사용하는 고객 관리형 환경 템플릿을 등록하고 게시합니다. AWS Proton 는 기존 프로비저닝된 인프라를 관리하지 않습니다.

에 서비스 템플릿을 생성하고 등록 AWS Proton 하여 환경에 서비스를 배포할 수 있습니다. 서비스를 배포하려면 먼저 AWS Proton 환경을 생성해야 합니다.

다음 목록은를 사용하여 템플릿을 생성하고 관리하는 방법을 설명합니다 AWS Proton.

- (선택 사항) AWS Proton API 호출 및 IAM 서비스 역할에 대한 개발자 액세스를 제어하는 AWS Proton IAM 역할을 준비합니다. 자세한 내용은 the section called “IAM 역할” 단원을 참조하십시오.
- 템플릿 번들을 작성합니다. 자세한 내용은 템플릿 번들 단원을 참조하십시오.
- 템플릿 번들을 Amazon S3 버킷에 구성, 압축 및 저장한 AWS Proton 후에 템플릿을 생성하고 등록합니다. 이 작업은 콘솔에서 또는 AWS CLI를 사용하여 수행할 수 있습니다.
- 템플릿을 테스트하고 사용하여에 등록한 후 AWS Proton 프로비저닝된 리소스를 생성하고 관리합니다 AWS Proton.
- 템플릿 수명 주기 내내 템플릿의 메이저 버전과 마이너 버전을 만들고 관리할 수 있습니다.

템플릿 버전을 수동으로 관리하거나 템플릿 동기화 구성을 사용하여 관리할 수 있습니다.

- AWS Proton 콘솔 및를 사용하여 새 마이너 또는 메이저 버전을 AWS CLI 생성합니다.
- 정의한 리포지토리에서 템플릿 번들에 대한 변경을 감지하면에서 새 마이너 또는 메이저 버전을 자동으로 생성할 수 있는 템플릿 동기화 구성을 생성합니다. AWS Proton

자세한 내용은 AWS Proton 서비스 API 참조를 참조하세요.

주제

- 버전이 지정된 템플릿
- 템플릿 등록 및 게시
- 템플릿 데이터 보기
- 템플릿 업데이트
- 템플릿 삭제
- 템플릿 동기화 구성
- 서비스 동기화 구성

버전이 지정된 템플릿

관리자 또는 플랫폼 팀의 구성원은 인프라 리소스를 프로비저닝하는 데 사용되는 버전이 지정된 템플릿 라이브러리를 정의, 생성 및 관리합니다. 템플릿 버전에는 마이너 버전과 메이저 버전의 두 가지 유형이 있습니다.

- 마이너 버전 — 이전 버전과 호환되는 스키마가 있는 템플릿에 대한 변경. 이러한 변경을 위해 개발자는 새 템플릿 버전으로 업데이트할 때 새 정보를 제공할 필요가 없습니다.

  마이너 버전을 변경하려고 하면 AWS Proton 는 새 버전의 스키마가 템플릿의 이전 마이너 버전과 이전 버전과 호환되는지 여부를 확인하기 위해 최선을 다합니다. 새 스키마가 이전 버전과 호환되지 않는 경우는 새 마이너 버전 등록에 AWS Proton 실패합니다.

> **Note**
>
> 호환성은 전적으로 schema를 기반으로 결정됩니다. AWS Proton 는 템플릿 번들 코드형 인프라(IaC) 파일이 이전 마이너 버전과 호환되는지 확인하지 않습니다. 예를 들어 AWS Proton 는 새 IaC 파일이 이전 마이너 버전의 템플릿으로 프로비저닝된 인프라에서 실행 중인 애플리케이션에 대한 주요 변경 사항을 유발하는지 확인하지 않습니다.

- 메이저 버전 - 이전 버전과 호환되지 않을 수 있는 템플릿 변경 내용. 이러한 변경에는 일반적으로 개발자의 새로운 입력이 필요하며 템플릿 스키마 변경이 수반되는 경우가 많습니다.

  팀의 운영 모델에 따라 이전 버전과 호환되는 변경 내용을 메이저 버전으로 지정할 수도 있습니다.

는 템플릿 버전 요청이 마이너 버전인지 메이저 버전인지 AWS Proton 를 결정하는 방법은 템플릿 변경 사항을 추적하는 방법에 따라 달라집니다.

- 새 템플릿 버전 생성을 명시적으로 요청하는 경우 메이저 버전 번호를 지정하여 메이저 버전을 요청하고 메이저 버전 번호를 지정하지 않음으로써 마이너 버전을 요청합니다.
- 템플릿 동기화를 사용하는 경우(따라서 명시적 템플릿 버전 요청을 하지 않는 경우)는 기존 YAML 파일에서 발생하는 템플릿 변경에 대해 새 마이너 버전을 생성하려고 AWS Proton 시도합니다.는 새 템플릿 변경에 대해 새 디렉터리를 생성할 때 메이저 버전을 AWS Proton 생성합니다(예: v1에서 v2로 이동).

> **Note**
>
> 에서 변경 사항이 이전 버전과 호환되지 않는다고 AWS Proton 판단하더라도 템플릿 동기화를 기반으로 한 새 마이너 버전 등록은 여전히 실패합니다.

템플릿의 새 버전을 게시할 때 가장 높은 메이저 버전과 마이너 버전인 경우 해당 버전은 권장 버전이 됩니다. 새 AWS Proton 리소스는 새 권장 버전을 사용하여 생성되며, 관리자에게 새 버전을 사용하고 오래된 버전을 사용하는 기존 AWS Proton 리소스를 업데이트하라는 AWS Proton 메시지를 표시합니다.

템플릿 등록 및 게시

다음 섹션에 설명된 AWS Proton대로 환경 및 서비스 템플릿을에 등록하고 게시할 수 있습니다.

콘솔 또는를 사용하여 템플릿의 새 버전을 생성할 수 있습니다 AWS CLI.

또는 콘솔 또는를 사용하여 템플릿을 AWS CLI 생성하고 이에 대한 템플릿 동기화 구성을 구성할 수 있습니다. 이 구성을 사용하면 정의한 등록된 git 리포지토리에 있는 템플릿 번들에서 AWS Proton 동기화할 수 있습니다. 템플릿 중 하나를 변경하는 템플릿 리포지토리로 커밋이 푸시될 때마다(해당 버전이 아직 없는 경우) 새 템플릿 마이너 또는 메이저 버전이 생성됩니다. 템플릿 동기화 구성 사전 요구 사항 및 요구 사항에 대한 자세한 내용은 템플릿 동기화 구성을 참조하세요.

환경 템플릿 등록 및 게시

다음 유형의 환경 템플릿을 등록하고 게시할 수 있습니다.

- 가 환경 인프라를 배포하고 관리하는 데 AWS Proton 사용하는 표준 환경 템플릿을 등록하고 게시합니다.
- 가 관리하는 기존 프로비저닝된 인프라에 연결하는 데 AWS Proton 사용하는 고객 관리형 환경 템플릿을 등록하고 게시합니다. AWS Proton 기존 프로비저닝된 인프라를 관리하지 않습니다.

> **Important**
>
> 관리자는 프로비저닝 및 관리형 인프라와 모든 출력 파라미터가 연결된 고객 관리형 환경 템플릿과 호환되는지 확인합니다. 이러한 변경 사항은 표시되지 않으므로는 사용자를 대신하여 변경 사항을 AWS Proton 설명할 수 없습니다 AWS Proton. 불일치로 인해 장애가 발생합니다.

콘솔 또는를 사용하여 환경 템플릿을 등록하고 게시 AWS CLI 할 수 있습니다.

AWS Management Console

콘솔을 사용하여 새 환경 템플릿을 등록할 수 있습니다.

1. AWS Proton 콘솔에서 환경 템플릿을 선택합니다.
2. 환경 템플릿 생성을 선택합니다.
3. 환경 템플릿 생성 페이지의 템플릿 옵션 단원에서 사용 가능한 두 가지 템플릿 옵션 중 하나를 선택합니다.
   - 새 환경을 프로비저닝하기 위한 템플릿을 생성합니다.
   - 관리하는 프로비저닝된 인프라를 사용할 템플릿을 생성합니다.
4. 새 환경을 프로비저닝을 위한 템플릿 생성을 선택한 경우 템플릿 번들 소스 단원에서 사용 가능한 세 가지 템플릿 번들 소스 옵션 중 하나를 선택합니다. 템플릿 동기화 요구 사항 및 사전 요구 사항에 대한 자세한 내용은 템플릿 동기화 구성을 참조하세요.
   - 샘플 템플릿 번들 중 하나 사용.
   - 자체 템플릿 번들 사용.
   - Git에서 템플릿 동기화.

5. 템플릿 번들 경로를 제공하세요.

   a. 샘플 템플릿 번들 중 하나 사용을 선택했다면

      샘플 템플릿 번들 단원에서 샘플 템플릿 번들을 선택합니다.

   b. 소스 코드 단원에서 Git에서 템플릿 동기화 선택한 경우,

      i. 템플릿 동기화 구성을 위한 리포지토리를 선택합니다.

      ii. 동기화할 리포지토리 브랜치의 이름을 입력합니다.

      iii. (선택 사항) 템플릿 번들 검색을 제한하려면 디렉터리 이름을 입력합니다.

   c. 그렇지 않으면 S3 번들 위치 단원에서 템플릿 번들 경로를 제공하세요.

6. 템플릿 세부 정보 단원에서.

   a. 템플릿 이름을 입력합니다.

   b. (선택 사항) 템플릿 표시 이름을 입력합니다.

   c. (선택 사항) 환경 템플릿에 대한 템플릿 설명을 입력합니다.

7. (선택 사항) 암호화 설정 단원에서 암호화 설정 사용자 지정(고급) 확인란을 선택하여 고유한 암호화 키를 제공하세요.

8. (선택 사항) 태그 단원에서 새 태그 추가 를 선택하고 키와 값을 입력하여 고객 관리형 태그를 생성합니다.

9. 환경 템플릿 생성을 선택합니다.

   이제 새 환경 템플릿의 상태 및 세부 정보를 표시하는 새 페이지가 열립니다. 이러한 세부 정보에는 AWS 및 고객 관리형 태그 목록이 포함됩니다.는 AWS Proton 리소스를 생성할 때 AWS 관리형 태그를 AWS Proton 자동으로 생성합니다. 자세한 내용은 AWS Proton 리소스 및 태그 지정 단원을 참조하십시오.

10. 새 환경 템플릿 상태의 상태는 초안 상태에서 시작됩니다. 사용자 및 `proton:CreateEnvironment` 권한이 있는 다른 사용자가 보고 액세스할 수 있습니다. 다음 단계에 따라 다른 사람이 템플릿을 사용할 수 있도록 합니다.

11. 템플릿 버전 단원에서 방금 만든 템플릿의 마이너 버전 왼쪽에 있는 라디오 버튼 (1.0) 을 선택합니다. 또는 정보 알림에서 게시를 선택하고 다음 단계를 건너뛸 수도 있습니다.

12. 템플릿 버전 단원에서 게시를 선택합니다.

13. 템플릿 상태가 게시됨으로 변경됩니다. 템플릿의 최신 버전이므로 권장 버전입니다.

14. 탐색 창에서 환경 템플릿을 선택하여 환경 템플릿 및 세부 정보 목록을 확인합니다.

콘솔을 사용하여 환경 템플릿의 새 메이저 버전 및 마이너 버전을 등록할 수 있습니다.

자세한 내용은 버전이 지정된 템플릿 단원을 참조하세요.

1. AWS Proton 콘솔에서 환경 템플릿을 선택합니다.
2. 환경 템플릿 목록에서 메이저 또는 마이너 버전을 만들 환경 템플릿의 이름을 선택합니다.
3. 환경 템플릿 세부 정보 보기의 템플릿 버전 단원에서 새 버전 생성을 선택합니다.
4. 새 환경 템플릿 버전 생성 페이지의 템플릿 번들 옵션 단원에서 사용 가능한 두 가지 템플릿 번들 소스 옵션 중 하나를 선택합니다.
   - 샘플 템플릿 번들 중 하나 사용.
   - 자체 템플릿 번들 사용.
5. 선택된 템플릿 번들 경로를 제공하세요.
   - 샘플 템플릿 번들 중 하나 사용을 선택한 경우 샘플 템플릿 번들 단원에서 샘플 템플릿 번들을 선택합니다.
   - 자체 템플릿 번들 사용을 선택한 경우 S3 번들 위치 단원에서 템플릿 번들 경로를 선택합니다.
6. 템플릿 세부 정보 단원에서.
   a. (선택 사항) 템플릿 표시 이름을 입력합니다.
   b. (선택 사항) 서비스 템플릿에 대한 템플리 설명을 입력합니다.
7. 템플릿 세부 정보 단원에서 다음 옵션 중 하나를 선택합니다.
   - 마이너 버전을 만들려면 새 메이저 버전 생성 체크 확인란을 비워 두십시오.
   - 메이저 버전을 만들려면 새 메이저 버전 생성 체크 확인란을 선택하세요.
8. 콘솔 단계를 계속 진행하여 새 마이너 또는 메이저 버전을 만들고 새 버전 생성을 선택합니다.

AWS CLI

CLI를 사용하여 다음 단계에 표시된 대로 새 환경 템플릿을 등록하고 게시합니다.

1. 지역, 이름, 표시 이름(선택 사항) 및 설명(선택 사항)을 지정하여 표준 또는 고객 관리형 환경 템플릿을 생성합니다.
   a. 표준 환경 템플릿을 생성합니다.

다음 명령을 실행합니다.

```
$ aws proton create-environment-template \
    --name "simple-env" \
    --display-name "Fargate" \
    --description "VPC with public access"
```

응답:

```
{
    "environmentTemplate": {
        "arn": "arn:aws:proton:region-id:123456789012:environment-template/
simple-env",
        "createdAt": "2020-11-11T23:02:45.336000+00:00",
        "description": "VPC with public access",
        "displayName": "VPC",
        "lastModifiedAt": "2020-11-11T23:02:45.336000+00:00",
        "name": "simple-env"
    }
}
```

b. provisioning 파라미터에 CUSTOMER_MANAGED값을 추가하여 고객 관리형 환경 템플릿을 생성합니다.

다음 명령을 실행합니다.

```
$ aws proton create-environment-template \
    --name "simple-env" \
    --display-name "Fargate" \
    --description "VPC with public access" \
    --provisioning "CUSTOMER_MANAGED"
```

응답:

```
{
    "environmentTemplate": {
        "arn": "arn:aws:proton:region-id:123456789012:environment-template/
simple-env",
        "createdAt": "2020-11-11T23:02:45.336000+00:00",
        "description": "VPC with public access",
        "displayName": "VPC",
```

```
        "lastModifiedAt": "2020-11-11T23:02:45.336000+00:00",
        "name": "simple-env",
        "provisioning": "CUSTOMER_MANAGED"
    }
}
```

2. 환경 템플릿의 메이저 버전 1의 마이너 버전 0을 생성합니다.

   이 단계와 나머지 단계는 표준 및 고객 관리형 환경 템플릿 모두에서 동일합니다.

   템플릿 이름, 메이저 버전, 환경 템플릿 번들이 포함된 버킷의 S3 버킷 이름 및 키를 포함하세요.

   다음 명령을 실행합니다.

```
$ aws proton create-environment-template-version \
    --template-name "simple-env" \
    --description "Version 1" \
    --source s3="{bucket=your_s3_bucket, key=your_s3_key}"
```

   응답:

```
{
    "environmentTemplateVersion": {
        "arn": "arn:aws:proton:region-id:123456789012:environment-template/
simple-env:1.0",
        "createdAt": "2020-11-11T23:02:47.763000+00:00",
        "description": "Version 1",
        "lastModifiedAt": "2020-11-11T23:02:47.763000+00:00",
        "majorVersion": "1",
        "minorVersion": "0",
        "status": "REGISTRATION_IN_PROGRESS",
        "templateName": "simple-env"
    }
}
```

3. get 명령을 사용하여 등록 상태를 확인합니다.

   다음 명령을 실행합니다.

```
$ aws proton get-environment-template-version \
    --template-name "simple-env" \
```

```
    --major-version "1" \
    --minor-version "0"
```

응답:

```
{
    "environment": {
        "arn": "arn:aws:proton:region-id:123456789012:environment-template/
simple-env:1.0",
        "createdAt": "2020-11-11T23:02:47.763000+00:00",
        "description": "Version 1",
        "lastModifiedAt": "2020-11-11T23:02:47.763000+00:00",
        "majorVersion": "1",
        "minorVersion": "0",
        "recommendedMinorVersion": "0",
        "schema": "schema:\n  format:\n    openapi: \"3.0.0\"\n
 environment_input_type: \"MyEnvironmentInputType\"\n  types:\n
 MyEnvironmentInputType:\n      type: object\n      description: \"Input
 properties for my environment\"\n      properties:\n        my_sample_input:\n
          type: string\n          description: \"This is a sample input\"\n
      default: \"hello world\"\n        my_other_sample_input:\n          type:
 string\n          description: \"Another sample input\"\n      required:\n
   - my_other_sample_input\n",
        "status": "DRAFT",
        "statusMessage": "",
        "templateName": "simple-env"
    }
}
```

4. 템플릿 이름과 메이저 및 마이너 버전을 제공하여 환경 템플릿의 메이저 버전 1의 마이너 버전 0을 게시합니다. 이 버전이 Recommended 버전입니다.

   다음 명령을 실행합니다.

```
$ aws proton update-environment-template-version \
    --template-name "simple-env" \
    --major-version "1" \
    --minor-version "0" \
    --status "PUBLISHED"
```

   응답:

```
{
    "environmentTemplateVersion": {
        "arn": "arn:aws:proton:region-id:123456789012:environment-template/
simple-env:1.0",
        "createdAt": "2020-11-11T23:02:47.763000+00:00",
        "description": "Version 1",
        "lastModifiedAt": "2020-11-11T23:02:54.610000+00:00",
        "majorVersion": "1",
        "minorVersion": "0",
        "recommendedMinorVersion": "0",
        "schema": "schema:\n  format:\n    openapi: \"3.0.0\"\n
 environment_input_type: \"MyEnvironmentInputType\"\n  types:\n
 MyEnvironmentInputType:\n      type: object\n      description: \"Input
 properties for my environment\"\n      properties:\n        my_sample_input:\n
          type: string\n          description: \"This is a sample input\"\n
      default: \"hello world\"\n        my_other_sample_input:\n          type:
 string\n          description: \"Another sample input\"\n      required:\n
   - my_other_sample_input\n",
        "status": "PUBLISHED",
        "statusMessage": "",
        "templateName": "simple-env"
    }
}
```


를 사용하여 새 템플릿을 생성한 후 AWS 및 고객 관리형 태그 목록을 볼 AWS CLI수 있습니다.는 AWS 관리형 태그를 AWS Proton 자동으로 생성합니다. AWS CLI를 사용하여 고객 관리 태그를 수정하고 생성할 수도 있습니다. 자세한 내용은 AWS Proton 리소스 및 태그 지정 단원을 참조하십시오.

다음 명령을 실행합니다.

```
$ aws proton list-tags-for-resource \
    --resource-arn "arn:aws:proton:region-id:123456789012:environment-
template/simple-env"
```

서비스 템플릿 등록 및 게시

서비스 템플릿 버전을 만들 때 호환되는 환경 템플릿의 목록을 지정합니다. 이렇게 하면 개발자가 서비스 템플릿을 선택할 때 서비스를 배포할 환경을 선택할 수 있습니다.

서비스 템플릿에서 서비스를 만들거나 서비스 템플릿을 게시하기 전에 나열된 호환 가능한 환경 템플릿에서 환경이 배포되었는지 확인하세요.

제거된 호환 환경 템플릿으로 구축된 환경에 서비스를 배포한 경우 서비스를 새 메이저 버전으로 업데이트할 수 없습니다.

서비스 템플릿 버전에 호환되는 환경 템플릿을 추가하거나 제거하려면 새 메이저 버전을 만들어야 합니다.

콘솔 또는를 사용하여 서비스 템플릿을 등록하고 게시 AWS CLI 할 수 있습니다.

AWS Management Console

콘솔을 사용하여 새 서비스 템플릿을 등록하고 게시할 수 있습니다.

1. AWS Proton 콘솔에서 서비스 템플릿을 선택합니다.
2. 서비스 템플릿 생성을 선택합니다.
3. 서비스 템플릿 생성 페이지의 템플릿 번들 소스 옵션 단원에서 사용 가능한 두 가지 템플릿 옵션 중 하나를 선택합니다.
   - 자체 템플릿 번들 사용.
   - Git에서 템플릿 동기화.
4. 템플릿 번들 경로를 제공하세요.
   a. 소스 코드 리포지토리 단원에서 Git에서 템플릿 동기화를 선택한 경우,
      i. 템플릿 동기화 구성을 위한 리포지토리를 선택합니다.
      ii. 동기화할 리포지토리 브랜치의 이름을 입력합니다.
      iii. (선택 사항) 템플릿 번들 검색을 제한하려면 디렉터리 이름을 입력합니다.
   b. 그렇지 않으면 S3 번들 위치 단원에서 템플릿 번들 경로를 제공하세요.
5. 템플릿 세부 정보 단원에서.
   a. 템플릿 이름을 입력합니다.
   b. (선택 사항) 템플릿 표시 이름을 입력합니다.
   c. (선택 사항) 서비스 템플릿에 대한 템플리 설명을 입력합니다.
6. 호환 가능한 환경 템플릿 단원에서 호환 가능한 환경 템플릿 목록에서 선택합니다.

7. (선택 사항) 암호화 설정 단원에서 암호화 설정 사용자 지정(고급)을 선택하여 고유한 암호화 키를 제공하세요.
8. (선택 사항) 파이프라인 단원에서,

   서비스 템플릿에 서비스 파이프라인 정의를 포함하지 않는 경우 페이지 하단에 있는 파이프라인 - 선택 사항 확인란의 선택을 취소합니다. 대상 그룹을 생성한 후에는 이 설정을 변경할 수 없습니다. 자세한 내용은 템플릿 번들 단원을 참조하세요.
9. (선택 사항) 지원되는 구성 요소 소스 섹션의 구성 요소 소스에서 직접 정의를 선택하여 직접 정의된 구성 요소를 서비스 인스턴스에 연결할 수 있도록 합니다.
10. (선택 사항) 태그 단원에서 새 태그 추가 를 선택하고 키와 값을 입력하여 고객 관리형 태그를 생성합니다.
11. 서비스 템플릿 생성을 선택합니다.

    이제 새 서비스 템플릿의 상태와 세부 정보가 표시되는 새 페이지가 열립니다. 이러한 세부 정보에는 AWS 및 고객 관리형 태그 목록이 포함됩니다.는 AWS Proton 리소스를 생성할 때 AWS 관리형 태그를 AWS Proton 자동으로 생성합니다. 자세한 내용은 AWS Proton 리소스 및 태그 지정 단원을 참조하십시오.
12. 새 서비스 템플릿 상태의 상태는 초안 상태에서 시작됩니다. 사용자 및 `proton:CreateService` 권한이 있는 다른 사용자가 보고 액세스할 수 있습니다. 다음 단계에 따라 다른 사람이 템플릿을 사용할 수 있도록 합니다.
13. 템플릿 버전 단원에서 방금 만든 템플릿의 마이너 버전 왼쪽에 있는 라디오 버튼 (1.0) 을 선택합니다. 또는 정보 알림에서 게시를 선택하고 다음 단계를 건너뛸 수도 있습니다.
14. 템플릿 버전 단원에서 게시를 선택합니다.
15. 템플릿 상태가 게시됨으로 변경됩니다. 템플릿의 최신 버전이므로 권장 버전입니다.
16. 탐색 창에서 서비스 템플릿을 선택하여 서비스 템플릿 및 세부 정보 목록을 확인합니다.

콘솔을 사용하여 서비스 템플릿의 새 메이저 버전 및 마이너 버전을 등록할 수 있습니다.

자세한 내용은 버전이 지정된 템플릿 단원을 참조하세요.

1. AWS Proton 콘솔에서 서비스 템플릿을 선택합니다.
2. 서비스 템플릿 목록에서 메이저 또는 마이너 버전을 만들 서비스 템플릿의 이름을 선택합니다.
3. 서비스 템플릿 세부 정보 보기의 템플릿 버전 단원에서 새 버전 생성을 선택합니다.
4. 새 서비스 템플릿 버전 생성 페이지의 번들 소스 단원에서 자체 템플릿 번들 사용을 선택합니다.

5. S3 번들 위치 단원에서 템플릿 번들 경로를 선택합니다.
6. 템플릿 세부 정보 단원에서.
   a. (선택 사항) 템플릿 표시 이름을 입력합니다.
   b. (선택 사항) 서비스 템플릿에 대한 템플리 설명을 입력합니다.
7. 템플릿 세부 정보 단원에서 다음 옵션 중 하나를 선택합니다.
   - 마이너 버전을 만들려면 새 메이저 버전 생성 체크 확인란을 비워 두십시오.
   - 메이저 버전을 만들려면 새 메이저 버전 생성 체크 확인란을 선택하세요.
8. 콘솔 단계를 계속 진행하여 새 마이너 또는 메이저 버전을 만들고 새 버전 생성을 선택합니다.

AWS CLI

서비스 파이프라인 없이 서비스를 배포하는 서비스 템플릿을 만들려면 `create-service-template` 명령에 파라미터와 값 `--pipeline-provisioning "CUSTOMER_MANAGED"`을 추가합니다. 템플릿 번들 생성 및 서비스 템플릿 번들에 대한 스키마 요구 사항의 설명에 따라 템플릿 번들을 구성하세요.

> **Note**
> 서비스 그룹을 생성한 후에는 `pipelineProvisioning`을 변경할 수 없습니다.

1. 다음 단계에 표시된 대로 CLI를 사용하여 서비스 파이프라인 유무에 관계없이 새 서비스 템플릿을 등록하고 게시할 수 있습니다.
   a. CLI를 사용하여 서비스 파이프라인으로 서비스 템플릿을 생성합니다.

      이름, 표시 이름 (선택 사항) 및 설명 (선택 사항) 을 지정합니다.

      다음 명령을 실행합니다.

```
$ aws proton create-service-template \
    --name "fargate-service" \
    --display-name "Fargate" \
    --description "Fargate-based Service"
```

      응답:

```
{
    "serviceTemplate": {
        "arn": "arn:aws:proton:region-id:123456789012:service-template/
fargate-service",
        "createdAt": "2020-11-11T23:02:55.551000+00:00",
        "description": "Fargate-based Service",
        "displayName": "Fargate",
        "lastModifiedAt": "2020-11-11T23:02:55.551000+00:00",
        "name": "fargate-service"
    }
}
```

b. 서비스 파이프라인 없이 서비스 템플릿을 생성합니다.

`--pipeline-provisioning`를 추가합니다.

다음 명령을 실행합니다.

```
$ aws proton create-service-template \
    --name "fargate-service" \
    --display-name "Fargate" \
    --description "Fargate-based Service" \
    --pipeline-provisioning "CUSTOMER_MANAGED"
```

응답:

```
{
    "serviceTemplate": {
        "arn": "arn:aws:proton:region-id:123456789012:service-template/
fargate-service",
        "createdAt": "2020-11-11T23:02:55.551000+00:00",
        "description": "Fargate-based Service",
        "displayName": "Fargate",
        "lastModifiedAt": "2020-11-11T23:02:55.551000+00:00",
        "name": "fargate-service",
        "pipelineProvisioning": "CUSTOMER_MANAGED"
    }
}
```

2. 서비스 템플릿의 메이저 버전 1의 마이너 버전 0을 생성합니다.

   템플릿 이름, 호환되는 환경 템플릿, 주요 버전, 서비스 템플릿 번들이 포함된 버킷의 S3 버킷 이름 및 키를 포함합니다.

   다음 명령을 실행합니다.

```
$ aws proton create-service-template-version \
    --template-name "fargate-service" \
    --description "Version 1" \
    --source s3="{bucket=your_s3_bucket, key=your_s3_key}" \
    --compatible-environment-templates '[{"templateName":"simple-
env","majorVersion":"1"}]'
```

   응답:

```
{
    "serviceTemplateMinorVersion": {
        "arn": "arn:aws:proton:region-id:123456789012:service-template/fargate-
service:1.0",
        "compatibleEnvironmentTemplates": [
            {
                "majorVersion": "1",
                "templateName": "simple-env"
            }
        ],
        "createdAt": "2020-11-11T23:02:57.912000+00:00",
        "description": "Version 1",
        "lastModifiedAt": "2020-11-11T23:02:57.912000+00:00",
        "majorVersion": "1",
        "minorVersion": "0",
        "status": "REGISTRATION_IN_PROGRESS",
        "templateName": "fargate-service"
    }
}
```

3. get 명령을 사용하여 등록 상태를 확인합니다.

   다음 명령을 실행합니다.

```
$ aws proton get-service-template-version \
```

```
    --template-name "fargate-service" \
    --major-version "1" \
    --minor-version "0"
```

응답:

```
{
    "serviceTemplateMinorVersion": {
        "arn": "arn:aws:proton:us-east-1:123456789012:service-template/fargate-
service:1.0",
        "compatibleEnvironmentTemplates": [
            {
                "majorVersion": "1",
                "templateName": "simple-env"
            }
        ],
        "createdAt": "2020-11-11T23:02:57.912000+00:00",
        "description": "Version 1",
        "lastModifiedAt": "2020-11-11T23:02:57.912000+00:00",
        "majorVersion": "1",
        "minorVersion": "0",
        "schema": "schema:\n  format:\n    openapi: \"3.0.0\"\n
 pipeline_input_type: \"MyPipelineInputType\"\n  service_input_type:
 \"MyServiceInstanceInputType\"\n\n  types:\n    MyPipelineInputType:\n
    type: object\n      description: \"Pipeline input properties\"\n
 required:\n        - my_sample_pipeline_required_input\n      properties:\n
        my_sample_pipeline_optional_input:\n          type: string\n
   description: \"This is a sample input\"\n          default: \"hello world
\"\n        my_sample_pipeline_required_input:\n          type: string\n
     description: \"Another sample input\"\n\n    MyServiceInstanceInputType:
\n      type: object\n      description: \"Service instance input properties
\"\n      required:\n        - my_sample_service_instance_required_input\n
    properties:\n        my_sample_service_instance_optional_input:\n
   type: string\n          description: \"This is a sample input\"\n
 default: \"hello world\"\n        my_sample_service_instance_required_input:\n
         type: string\n          description: \"Another sample input\"",
        "status": "DRAFT",
        "statusMessage": "",
        "templateName": "fargate-service"
    }
}
```

4. update 명령을 사용하여 상태를 "PUBLISHED"로 변경하여 서비스 템플릿을 게시합니다.

다음 명령을 실행합니다.

```
$ aws proton update-service-template-version \
    --template-name "fargate-service" \
    --description "Version 1" \
    --major-version "1" \
    --minor-version "0" \
    --status "PUBLISHED"
```

응답:


```
{
    "serviceTemplateVersion": {
        "arn": "arn:aws:proton:region-id:123456789012:service-template/fargate-
service:1.0",
        "compatibleEnvironmentTemplates": [
            {
                "majorVersion": "1",
                "templateName": "simple-env"
            }
        ],
        "createdAt": "2020-11-11T23:02:57.912000+00:00",
        "description": "Version 1",
        "lastModifiedAt": "2020-11-11T23:02:57.912000+00:00",
        "majorVersion": "1",
        "minorVersion": "0",
        "recommendedMinorVersion": "0",
        "schema": "schema:\n  format:\n    openapi: \"3.0.0\"\n
 pipeline_input_type: \"MyPipelineInputType\"\n  service_input_type:
 \"MyServiceInstanceInputType\"\n\n  types:\n    MyPipelineInputType:\n
    type: object\n      description: \"Pipeline input properties\"\n
 required:\n        - my_sample_pipeline_required_input\n      properties:\n
        my_sample_pipeline_optional_input:\n          type: string\n
 description: \"This is a sample input\"\n          default: \"hello pipeline
\"\n        my_sample_pipeline_required_input:\n          type: string\n
     description: \"Another sample input\"\n\n    MyServiceInstanceInputType:
\n      type: object\n      description: \"Service instance input properties
\"\n      required:\n        - my_sample_service_instance_required_input\n
    properties:\n        my_sample_service_instance_optional_input:\n
   type: string\n          description: \"This is a sample input\"\n
 default: \"hello world\"\n        my_sample_service_instance_required_input:\n
         type: string\n          description: \"Another sample input\"\n",
```

```
        "status": "PUBLISHED",
        "statusMessage": "",
        "templateName": "fargate-service"
    }
}
```

5. get 명령을 사용하여 서비스 템플릿 세부 데이터를 검색하여가 버전 1.0을 게시 AWS Proton 했는지 확인합니다.

   다음 명령을 실행합니다.

```
$ aws proton get-service-template-version \
    --template-name "fargate-service" \
    --major-version "1" \
    --minor-version "0"
```

   응답:

```
{
    "serviceTemplateMinorVersion": {
        "arn": "arn:aws:proton:us-east-1:123456789012:service-template/fargate-
service:1.0",
        "compatibleEnvironmentTemplates": [
            {
                "majorVersion": "1",
                "templateName": "simple-env"
            }
        ],
        "createdAt": "2020-11-11T23:02:57.912000+00:00",
        "description": "Version 1",
        "lastModifiedAt": "2020-11-11T23:03:04.767000+00:00",
        "majorVersion": "1",
        "minorVersion": "0",
        "schema": "schema:\n  format:\n    openapi: \"3.0.0\"\n
 pipeline_input_type: \"MyPipelineInputType\"\n  service_input_type:
 \"MyServiceInstanceInputType\"\n\n  types:\n    MyPipelineInputType:\n
    type: object\n      description: \"Pipeline input properties\"\n
 required:\n        - my_sample_pipeline_required_input\n      properties:\n
        my_sample_pipeline_optional_input:\n          type: string\n
   description: \"This is a sample input\"\n          default: \"hello world
\"\n        my_sample_pipeline_required_input:\n          type: string\n
     description: \"Another sample input\"\n\n    MyServiceInstanceInputType:
\n      type: object\n      description: \"Service instance input properties
```

```
\"\n      required:\n        - my_sample_service_instance_required_input\n
    properties:\n        my_sample_service_instance_optional_input:\n
   type: string\n          description: \"This is a sample input\"\n
 default: \"hello world\"\n        my_sample_service_instance_required_input:\n
         type: string\n          description: \"Another sample input\"",
        "status": "PUBLISHED",
        "statusMessage": "",
        "templateName": "fargate-service"
    }
}
```

템플릿 데이터 보기

AWS Proton 콘솔 및 AWS CLI을 사용하여 세부 정보가 포함된 템플릿 목록을 보고 세부 데이터가 포함된 개별 템플릿을 볼 수 있습니다.

고객 관리형 환경 템플릿 데이터에는 `CUSTOMER_MANAGED` 값이 있는 `provisioned` 파라미터가 포함됩니다.

서비스 템플릿에 서비스 파이프라인이 포함되지 않은 경우 서비스 템플릿 데이터에는 `pipelineProvisioning` 파라미터가 `CUSTOMER_MANAGED` 값과 함께 포함됩니다.

자세한 내용은 템플릿 등록 및 게시 단원을 참조하십시오.

콘솔 또는를 사용하여 템플릿 데이터를 AWS CLI 나열하고 볼 수 있습니다.

AWS Management Console

콘솔을 사용하여 템플릿을 나열하고 볼 수 있습니다.

1. 템플릿 목록을 보려면 (환경 또는 서비스) 템플릿을 선택합니다.
2. 세부 데이터를 보려면 템플릿 이름을 선택합니다.

   템플릿의 세부 정보 데이터, 템플릿의 메이저 및 마이너 버전 목록, 템플릿 버전 및 템플릿 태그를 사용하여 배포된 AWS Proton 리소스 목록을 봅니다.

   권장 메이저 버전과 마이너 버전은 권장으로 표시되어 있습니다.

AWS CLI

AWS CLI 를 사용하여 템플릿을 나열하고 봅니다.

다음 명령을 실행합니다.

```
$ aws proton get-environment-template-version \
    --template-name "simple-env" \
    --major-version "1" \
    --minor-version "0"
```

응답:

```
{
    "environmentTemplateVersion": {
        "arn": "arn:aws:proton:region-id:123456789012:environment-template/simple-
env:1.0",
        "createdAt": "2020-11-10T18:35:08.293000+00:00",
        "description": "Version 1",
        "lastModifiedAt": "2020-11-10T18:35:11.162000+00:00",
        "majorVersion": "1",
        "minorVersion": "0",
        "recommendedMinorVersion": "0",
        "schema": "schema:\n  format:\n    openapi: \"3.0.0\"\n
 environment_input_type: \"MyEnvironmentInputType\"\n  types:\n
 MyEnvironmentInputType:\n      type: object\n      description: \"Input properties
 for my environment\"\n      properties:\n        my_sample_input:\n
  type: string\n          description: \"This is a sample input\"\n
 default: \"hello world\"\n        my_other_sample_input:\n          type: string
\n          description: \"Another sample input\"\n      required:\n        -
 my_other_sample_input\n",
        "status": "DRAFT",
        "statusMessage": "",
        "templateName": "simple-env"
    }
}
```

다음 명령을 실행합니다.

```
$ aws proton list-environment-templates
```

응답:

```
{
    "templates": [
        {
```

```
            "arn": "arn:aws:proton:region-id:123456789012:environment-template/
simple-env-3",
            "createdAt": "2020-11-10T18:35:05.763000+00:00",
            "description": "VPC with Public Access",
            "displayName": "VPC",
            "lastModifiedAt": "2020-11-10T18:35:05.763000+00:00",
            "name": "simple-env-3",
            "recommendedVersion": "1.0"
        },
        {
            "arn": "arn:aws:proton:region-id:123456789012:environment-template/
simple-env-1",
            "createdAt": "2020-11-10T00:14:06.881000+00:00",
            "description": "Some SSM Parameters",
            "displayName": "simple-env-1",
            "lastModifiedAt": "2020-11-10T00:14:06.881000+00:00",
            "name": "simple-env-1",
            "recommendedVersion": "1.0"
        }
    ]
}
```

서비스 템플릿의 마이너 버전을 봅니다.

다음 명령을 실행합니다.

```
$ aws proton get-service-template-version \
    --template-name "fargate-service" \
    --major-version "1" \
    --minor-version "0"
```

응답:

```
{
    "serviceTemplateMinorVersion": {
        "arn": "arn:aws:proton:us-east-1:123456789012:service-template/fargate-
service:1.0",
        "compatibleEnvironmentTemplates": [
            {
                "majorVersion": "1",
                "templateName": "simple-env"
            }
        ],
```

```
        "createdAt": "2020-11-11T23:02:57.912000+00:00",
        "description": "Version 1",
        "lastModifiedAt": "2020-11-11T23:02:57.912000+00:00",
        "majorVersion": "1",
        "minorVersion": "0",
        "schema": "schema:\n  format:\n    openapi: \"3.0.0\"\n
 pipeline_input_type: \"MyPipelineInputType\"\n  service_input_type:
 \"MyServiceInstanceInputType\"\n\n  types:\n    MyPipelineInputType:\n
    type: object\n      description: \"Pipeline input properties\"\n
 required:\n        - my_sample_pipeline_required_input\n      properties:\n
       my_sample_pipeline_optional_input:\n          type: string\n
 description: \"This is a sample input\"\n          default: \"hello world\"\n
  my_sample_pipeline_required_input:\n          type: string\n          description:
 \"Another sample input\"\n\n    MyServiceInstanceInputType:\n      type: object
\n      description: \"Service instance input properties\"\n      required:\n
       - my_sample_service_instance_required_input\n      properties:\n
 my_sample_service_instance_optional_input:\n          type: string\n
 description: \"This is a sample input\"\n          default: \"hello world\"\n
     my_sample_service_instance_required_input:\n          type: string\n
 description: \"Another sample input\"",
        "status": "DRAFT",
        "statusMessage": "",
        "templateName": "fargate-service"
    }
}
```

다음 예제 명령 및 응답에 표시된 것처럼 서비스 파이프라인이 없는 서비스 템플릿을 봅니다.

다음 명령을 실행합니다.

```
$ aws proton get-service-template \
    --name "simple-svc-template-cli"
```

응답:

```
{
    "serviceTemplate": {
        "arn": "arn:aws:proton:region-id:123456789012:service-template/simple-svc-
template-cli",
        "createdAt": "2021-02-18T15:38:57.949000+00:00",
        "displayName": "simple-svc-template-cli",
        "lastModifiedAt": "2021-02-18T15:38:57.949000+00:00",
        "status": "DRAFT",
```

```
        "name": "simple-svc-template-cli",
        "pipelineProvisioning": "CUSTOMER_MANAGED"
    }
}
```

템플릿 업데이트

다음 목록에 설명된 대로 템플릿을 업데이트할 수 있습니다.

- 콘솔 또는 AWS CLI을 사용할 때 템플릿의 `description` 또는 `display name`를 편집합니다. 템플릿의 `name`을 편집할 수는 없습니다.
- 콘솔 또는 AWS CLI을 사용할 때 템플릿 마이너 버전의 상태를 업데이트하세요. 상태를 `DRAFT`에서 `PUBLISHED`로 변경할 수만 있습니다.
- AWS CLI를 사용할 때 템플릿의 마이너 버전 또는 메이저 버전에 대한 표시 이름과 설명을 편집합니다.

AWS Management Console

다음 단계에 설명된 대로 콘솔을 사용하여 템플릿 설명과 표시 이름을 편집합니다.

템플릿 목록에서.

1. AWS Proton 콘솔에서 (환경 또는 서비스) 템플릿을 선택합니다.
2. 템플릿 목록에서 설명이나 표시 이름을 업데이트하려는 템플릿의 왼쪽에 있는 라디오 버튼을 선택합니다.
3. 작업을 선택하고 편집을 선택합니다.
4. (환경 또는 서비스) 템플릿 편집 페이지의 템플릿 세부 정보 단원에서 양식에 편집 내용을 입력하고 변경 내용 저장을 선택합니다.

다음 설명에 따라 콘솔을 사용하여 템플릿의 마이너 버전 상태를 변경하여 템플릿을 게시하세요. 상태를 `DRAFT`에서 `PUBLISHED`로 변경할 수만 있습니다.

(환경 또는 서비스) 템플릿 세부 정보 페이지에서.

1. AWS Proton 콘솔에서 (환경 또는 서비스) 템플릿을 선택합니다.
2. 템플릿 목록에서 마이너 버전의 상태를 업데이트하려는 템플릿의 이름을 초안에서 게시됨으로 선택합니다.

3. (환경 또는 서비스) 템플릿 세부 정보 페이지의 템플릿 버전 단원에서 게시하려는 마이너 버전의 왼쪽에 있는 라디오 버튼을 선택합니다.
4. 템플릿 버전 단원에서 게시를 선택합니다. 상태가 초안에서 게시됨으로 바뀝니다.

AWS CLI

다음 예제 명령 및 응답은 환경 템플릿의 설명을 편집하는 방법을 보여줍니다.

다음 명령을 실행합니다.

```
$ aws proton update-environment-template \
    --name "simple-env" \
    --description "A single VPC with public access"
```

응답:

```
{
    "environmentTemplate": {
        "arn": "arn:aws:proton:region-id:123456789012:environment-template/simple-
env",
        "createdAt": "2020-11-28T22:02:10.651000+00:00",
        "description": "A single VPC with public access",
        "displayName": "simple-env",
        "lastModifiedAt": "2020-11-29T16:11:18.956000+00:00",
        "majorVersion": "1",
        "minorVersion": "0",
        "recommendedMinorVersion": "0",
        "schema": "schema:\n  format:\n    openapi: \"3.0.0\"\n
 environment_input_type: \"MyEnvironmentInputType\"\n  types:\n
 MyEnvironmentInputType:\n      type: object\n      description: \"Input properties
 for my environment\"\n      properties:\n        my_sample_input:\n
  type: string\n          description: \"This is a sample input\"\n
 default: \"hello world\"\n        my_other_sample_input:\n          type: string
\n          description: \"Another sample input\"\n      required:\n        -
 my_other_sample_input\n",
        "status": "PUBLISHED",
        "statusMessage": "",
        "templateName": "simple-env"
    }
}
```

AWS CLI 를 사용하여 서비스 템플릿을 업데이트할 수도 있습니다. 서비스 템플릿의 마이너 버전 상태 업데이트 예는 서비스 템플릿 등록 및 게시, 5단계를 참조하세요.

템플릿 삭제

템플릿은 콘솔 및 AWS CLI를 사용하여 삭제할 수 있습니다.

해당 버전에 배포된 환경이 없는 경우 환경 템플릿의 마이너 버전을 삭제할 수 있습니다.

해당 버전에 배포된 서비스 인스턴스 또는 파이프라인이 없는 경우 서비스 템플릿의 마이너 버전을 삭제할 수 있습니다. 파이프라인을 서비스 인스턴스와 다른 템플릿 버전에 배포할 수 있습니다. 예를 들어 서비스 인스턴스가 1.0에서 버전 1.1로 업데이트되고 파이프라인이 여전히 버전 1.0으로 배포된 경우 서비스 템플릿 1.0을 삭제할 수 없습니다.

AWS Management Console

콘솔을 사용하여 전체 템플릿 또는 템플릿의 개별 마이너 및 메이저 버전을 삭제할 수 있습니다.

콘솔을 사용하여 다음과 같이 템플릿을 삭제합니다.

> **Note**
>
> 콘솔을 사용하여 템플릿 삭제할 때.
>
> - 템플릿 전체를 삭제하면 템플릿의 메이저 버전과 마이너 버전도 삭제됩니다.

(환경 또는 서비스) 템플릿 목록에서.

1. AWS Proton 콘솔에서 (환경 또는 서비스) 템플릿을 선택합니다.
2. 템플릿 목록에서 삭제할 템플릿의 왼쪽에 있는 라디오 버튼을 선택합니다.

   버전에 배포된 AWS Proton 리소스가 없는 경우에만 전체 템플릿을 삭제할 수 있습니다.
3. 작업을 선택한 다음 삭제를 선택하여 전체 템플릿을 삭제합니다.
4. 삭제 작업을 확인하라는 모달이 표시됩니다.
5. 지침을 따르고 예, 삭제를 선택합니다.

(환경 또는 서비스) 템플릿 세부 정보 페이지에서.

1. AWS Proton 콘솔에서 (환경 또는 서비스) 템플릿을 선택합니다.
2. 템플릿 목록에서 템플릿의 전체 메이저 또는 마이너 버전을 완전히 삭제 또는 삭제할 템플릿의 이름을 선택합니다.
3. 전체 템플릿 삭제하기.

   버전에 배포된 AWS Proton 리소스가 없는 경우에만 전체 템플릿을 삭제할 수 있습니다.

   a. 페이지 오른쪽 상단의 삭제를 선택합니다.
   b. 삭제 작업을 확인하라는 모달이 표시됩니다.
   c. 지침을 따르고 예, 삭제를 선택합니다.
4. 템플릿의 메이저 버전 또는 마이너 버전을 삭제하기.

   해당 버전에 배포된 AWS Proton 리소스가 없는 경우에만 템플릿의 마이너 버전을 삭제할 수 있습니다.

   a. 템플릿 버전 단원에서 삭제할 버전의 왼쪽에 있는 라디오 버튼을 선택합니다.
   b. 템플릿 버전 단원에서 삭제를 선택합니다.
   c. 삭제 작업을 확인하라는 모달이 표시됩니다.
   d. 지침을 따르고 예, 삭제를 선택합니다.

AWS CLI

AWS CLI 템플릿 삭제 작업에는 템플릿의 다른 버전 삭제가 포함되지 않습니다. 를 사용하는 경우 다음 조건에서 템플릿을 AWS CLI삭제합니다.

- 템플릿의 마이너 버전이나 메이저 버전이 없는 경우 전체 템플릿을 삭제하세요.
- 마지막 남은 마이너 버전을 삭제하면 메이저 버전도 삭제됩니다.
- 해당 버전에 배포된 AWS Proton 리소스가 없는 경우 템플릿의 마이너 버전을 삭제합니다.
- 템플릿의 다른 마이너 버전이 없고 해당 버전에 배포된 AWS Proton 리소스가 없는 경우 템플릿의 권장 마이너 버전을 삭제합니다.

다음 예제 명령 및 응답은를 사용하여 템플릿을 삭제 AWS CLI 하는 방법을 보여줍니다.

다음 명령을 실행합니다.

```
$ aws proton delete-environment-template-version \
    --template-name "simple-env" \
    --major-version "1" \
    --minor-version "0"
```

응답:

```
{
    "environmentTemplateVersion": {
        "arn": "arn:aws:proton:region-id:123456789012:environment-template/simple-
env:1.0",
        "createdAt": "2020-11-11T23:02:47.763000+00:00",
        "description": "Version 1",
        "lastModifiedAt": "2020-11-11T23:02:54.610000+00:00",
        "majorVersion": "1",
        "minorVersion": "0",
        "status": "PUBLISHED",
        "statusMessage": "",
        "templateName": "simple-env"
    }
}
```

다음 명령을 실행합니다.

```
$ aws proton delete-environment-template \
    --name "simple-env"
```

응답:

```
{
    "environmentTemplate": {
        "arn": "arn:aws:proton:region-id:123456789012:environment-template/simple-
env",
        "createdAt": "2020-11-11T23:02:45.336000+00:00",
        "description": "VPC with Public Access",
        "displayName": "VPC",
        "lastModifiedAt": "2020-11-12T00:23:22.339000+00:00",
        "name": "simple-env",
        "recommendedVersion": "1.0"
    }
}
```

다음 명령을 실행합니다.

```
$ aws proton delete-service-template-version \
    --template-name "fargate-service" \
    --major-version "1" \
    --minor-version "0"
```

응답:

```
{
    "serviceTemplateVersion": {
        "arn": "arn:aws:proton:region-id:123456789012:service-template/fargate-
service:1.0",
        "compatibleEnvironmentTemplates": [{"majorVersion": "1", "templateName":
 "simple-env"}],
        "createdAt": "2020-11-28T22:07:05.798000+00:00",
        "lastModifiedAt": "2020-11-28T22:19:05.368000+00:00",
        "majorVersion": "1",
        "minorVersion": "0",
        "status": "PUBLISHED",
        "statusMessage": "",
        "templateName": "fargate-service"
    }
}
```

템플릿 동기화 구성

정의한 등록된 git 리포지토리에 있는 템플릿 번들에서 AWS Proton 동기화하도록 템플릿을 구성하는 방법을 알아봅니다. 커밋이 리포지토리로 푸시되면 AWS Proton 은 리포지토리 템플릿 번들의 변경 사항을 확인합니다. 템플릿 번들 변경을 감지하면 해당 버전이 아직 존재하지 않는 경우 템플릿의 새 마이너 또는 메이저 버전이 생성됩니다.는 AWS Proton 현재 GitHub, GitHub Enterprise 및 BitBucket을 지원합니다.

커밋을 동기화된 템플릿 번들로 푸시하기

템플릿 중 하나로 추적되는 브랜치에 커밋을 푸시하면 AWS Proton 은 리포지토리를 복제하고 동기화하는 데 필요한 템플릿을 결정합니다. 디렉토리의 파일을 스캔하여 `{template-name}/{major-version}/`의 규칙과 일치하는 디렉터리를 찾습니다.

가 리포지토리 및 브랜치와 연결된 템플릿과 메이저 버전을 AWS Proton 결정하면 모든 템플릿을 병렬로 동기화하려고 시도하기 시작합니다.

특정 템플릿에 동기화할 때마다 AWS Proton 먼저는 마지막으로 동기화가 성공한 이후 템플릿 디렉터리의 내용이 변경되었는지 확인합니다. 콘텐츠가 변경되지 않으면 중복 번들 등록을 AWS Proton 건너뜁니다. 이렇게 하면 템플릿 번들의 콘텐츠가 변경될 경우 새 템플릿 마이너 버전이 생성됩니다. 템플릿 번들의 내용이 변경되면 번들이에 등록됩니다 AWS Proton.

템플릿 번들이 등록된 후는 등록이 완료될 때까지 등록 상태를 AWS Proton 모니터링합니다.

특정 템플릿 마이너 버전과 메이저 버전은 한 번에 한 번만 동기화할 수 있습니다. 동기화가 진행되는 동안 푸시되었을 수 있는 모든 커밋은 일괄 처리됩니다. 일괄 처리된 커밋은 이전 동기화 시도가 완료된 후에 동기화됩니다.

서비스 템플릿 동기화

AWS Proton 는 git 리포지토리에서 환경 및 서비스 템플릿을 모두 동기화할 수 있습니다. 서비스 템플릿을 동기화하려면 템플릿 번들의 각 메이저 버전 디렉터리에 이름이 지정된 `.template-registration.yaml` 추가 파일을 추가합니다. 이 파일에는 커밋 후 서비스 템플릿 버전을 생성할 AWS Proton 때 필요한 호환 환경 및 지원되는 구성 요소 소스 등의 추가 세부 정보가 포함되어 있습니다.

파일의 전체 경로는 *`service-template-name`*`/`*`major-version`*`/.template-registration.yaml`입니다. 자세한 내용은 the section called “서비스 템플릿 동기화” 단원을 참조하세요.

템플릿 동기화 구성 고려 사항

템플릿 동기화 구성 사용에 대한 다음 고려 사항을 검토하세요.

- 리포지토리는 250MB를 넘지 않아야 합니다.
- 템플릿 동기화를 구성하려면 먼저 리포지토리를 AWS Proton에 연결합니다. 자세한 내용은 the section called “리포지토리 링크 생성” 단원을 참조하십시오.
- 동기화된 템플릿에서 새 템플릿 버전을 만들면 DRAFT 상태가 됩니다.
- 다음 중 하나에 해당하는 경우 템플릿의 새 마이너 버전이 생성됩니다.
  - 템플릿 번들 컨텐츠는 마지막으로 동기화된 템플릿 마이너 버전의 컨텐츠와 다릅니다.
  - 이전에 마지막으로 동기화된 템플릿 마이너 버전이 삭제되었습니다.
- 동기화를 일시 중지할 수는 없습니다.

- 새 마이너 버전과 메이저 버전 모두 자동으로 동기화됩니다.
- 템플릿 동기화 구성으로는 새 최상위 템플릿을 만들 수 없습니다.
- 템플릿 동기화 구성을 사용하면 여러 리포지토리의 템플릿 하나에 동기화할 수 없습니다.
- 브랜치 대신 태그를 사용할 수는 없습니다.
- 서비스 템플릿을 만들 때는 호환되는 환경 템플릿을 지정합니다.
- 환경 템플릿을 만든 다음 동일한 커밋에서 서비스 템플릿과 호환되는 환경으로 추가할 수 있습니다.
- 단일 템플릿 메이저 버전과의 동기화는 한 번에 하나씩 실행됩니다. 동기화 중에 새 커밋이 감지되면 일괄 처리되어 활성 동기화가 끝날 때 적용됩니다. 다른 템플릿 메이저 버전과의 동기화는 병렬로 이루어집니다.
- 템플릿이 동기화되는 브랜치를 변경하면 이전 브랜치에서 진행 중인 모든 동기화가 먼저 완료됩니다. 그러면 새 브랜치부터 동기화가 시작됩니다.
- 템플릿을 동기화하는 데 사용할 리포지토리를 변경하면 이전 리포지토리에서 진행 중인 모든 동기화가 실패하거나 완료될 수 있습니다. 동기화 단계에 따라 다릅니다.

자세한 내용은 AWS Proton 서비스 API 참조를 참조하세요.

주제

템플릿 동기화 구성을 생성합니다.

를 사용하여 템플릿 동기화 구성을 생성하는 방법을 알아봅니다 AWS Proton.

템플릿 동기화 구성 전제 조건을 생성하기:

- AWS Proton와 리포지토리 연결했습니다..
- 템플릿 번들은 리포지토리에 있습니다.

리포지토리 링크는 다음 구성 요소로 이루어져 있습니다.

- 리포지토리에 액세스하고 알림을 구독할 수 있는 AWS Proton 권한을 부여하는 CodeConnections 연결입니다.
- 서비스 연결 역할 리포지토리를 연결하면 서비스 연결 역할이 자동으로 만들어집니다.

첫 번째 템플릿 동기화 구성을 생성하기 전에 다음 디렉토리 레이아웃과 같이 템플릿 번들을 저장소에 푸시하세요.

```
/templates/                                                  # subdirectory (optional)
/templates/my-env-template/                                  # template name
/templates/my-env-template/v1/                               # template version
/templates/my-env-template/v1/infrastructure/                # template bundle
/templates/my-env-template/v1/schema/
```

첫 번째 템플릿 동기화 구성을 만든 후 업데이트된 템플릿 번들을 새 버전 에 추가하는 커밋을 푸시하면 새 템플릿 버전이 자동으로 생성됩니다(예: `/my-env-template/v2/` 아래).

```
/templates/                                                  # subdirectory (optional)
/templates/my-env-template/                                  # template name
/templates/my-env-template/v1/                               # template version
/templates/my-env-template/v1/infrastructure/                # template bundle
/templates/my-env-template/v1/schema/
/templates/my-env-template/v2/
/templates/my-env-template/v2/infrastructure/
/templates/my-env-template/v2/schema/
```

단일 커밋에 하나 이상의 동기화 구성 템플릿에 대한 새 템플릿 번들 버전을 포함할 수 있습니다.는 커밋에 포함된 각 새 템플릿 번들 버전에 대해 새 템플릿 버전을 AWS Proton 생성합니다.

템플릿 동기화 구성을 생성한 후에도 콘솔에서 또는 S3 버킷에서 템플릿 번들을 업로드 AWS CLI 하여를 사용하여 새 버전의 템플릿을 수동으로 생성할 수 있습니다. 템플릿 동기화는 리포지토리에서 로 한 방향으로만 작동합니다 AWS Proton. 수동으로 생성한 템플릿 버전은 동기화되지 않습니다.

템플릿 동기화 구성을 설정한 후는 리포지토리의 AWS Proton 변경 사항을 수신합니다. 변경 사항이 푸시될 때마다 템플릿과 이름이 같은 디렉토리를 찾습니다. 그런 다음 해당 디렉터리 내부에서 메이저 버전과 같은 디렉터리를 찾습니다.는 템플릿 번들을 해당 템플릿 메이저 버전에 AWS Proton 등록합니다. 새 버전은 항상 DRAFT 상태를 유지합니다. 콘솔 또는 를 사용하여 새 버전을 게시할 수 있습니다 AWS CLI.

예를 들어 my-env-template이라는 템플릿이 다음과 같은 레이아웃으로 브랜치 main의 my-repo/templates에서 동기화되도록 구성되어 있다고 가정해 보겠습니다.

```
/code
/code/service.go
README.md
/templates/
/templates/my-env-template/
/templates/my-env-template/v1/
/templates/my-env-template/v1/infrastructure/
/templates/my-env-template/v1/schema/
/templates/my-env-template/v2/
/templates/my-env-template/v2/infrastructure/
/templates/my-env-template/v2/schema/
```

AWS Proton 는의 내용을 /templates/my-env-template/v1/에 동기화my-env-template:1하고의 내용을 /templates/my-env-template/v2/에 동기화합니다my-env-template:2. 아직 존재하지 않는 경우 다음과 같은 메이저 버전이 생성됩니다.

AWS Proton 에서 템플릿 이름과 일치하는 첫 번째 디렉터리를 찾았습니다. 템플릿 동기화 구성을 생성하거나 편집할 subdirectoryPath 때를 지정하여 디렉터리 AWS Proton 검색을 제한할 수 있습니다. 예를 들어, subdirectoryPath에 대해 /production-templates/로 지정할 수 있습니다.

콘솔 또는 CLI를 사용하여 서비스 동기화 구성을 만들 수 있습니다.

AWS Management Console

콘솔을 사용하여 템플릿 및 템플릿 동기화 구성을 생성합니다.

1. AWS Proton 콘솔에서 환경 템플릿을 선택합니다.
2. 환경 템플릿 생성을 선택합니다.
3. 환경 템플릿 생성 페이지의 템플릿 옵션 단원에서 새 환경을 프로비저닝하기 위한 템플릿 생성을 선택합니다.
4. 템플릿 번들 소스 단원에서 Git에서 템플릿 동기화를 선택합니다.
5. 소스 코드 리포지토리에서,
   a. 리포지토리의 경우 템플릿 번들이 포함된 연결된 리포지토리를 선택합니다.
   b. 브랜치의 경우 동기화할 리포지토리 브랜치를 선택합니다.

c. (선택 사항) 템플릿 번들 디렉터리의 경우 디텍터리의 이름을 입력하여 템플릿 번들 검색 범위를 좁힙니다.

6. 템플릿 세부 정보 단원에서.

   a. 템플릿 이름을 입력합니다.

   b. (선택 사항) 템플릿 표시 이름을 입력합니다.

   c. (선택 사항) 환경 템플릿에 대한 템플릿 설명을 입력합니다.

7. (선택 사항) 암호화 설정 단원에서 암호화 설정 사용자 지정(고급) 확인란을 선택하여 고유한 암호화 키를 제공하세요.

8. (선택 사항) 태그 단원에서 새 태그 추가를 선택하고 키와 값을 입력하여 고객 관리형 태그를 생성합니다.

9. 환경 템플릿 생성을 선택합니다.

   이제 새 환경 템플릿의 상태 및 세부 정보를 표시하는 새 페이지가 열립니다. 이러한 세부 정보에는 AWS 관리형 및 고객 관리형 태그 목록이 포함됩니다.는 AWS Proton 리소스를 생성할 때 AWS 관리형 태그를 AWS Proton 자동으로 생성합니다. 자세한 내용은 AWS Proton 리소스 및 태그 지정 단원을 참조하십시오.

10. 템플릿 세부 정보 페이지에서 동기화 탭을 선택하여 템플릿 동기화 구성 세부 정보 데이터를 확인합니다.

11. 템플릿 버전 탭을 선택하면 상태 세부 정보가 포함된 템플릿 버전을 볼 수 있습니다.

12. 새 환경 템플릿 상태의 상태는 초안 상태에서 시작됩니다. 사용자 및 `proton:CreateEnvironment` 권한이 있는 다른 사용자가 보고 액세스할 수 있습니다. 다음 단계에 따라 다른 사람이 템플릿을 사용할 수 있도록 합니다.

13. 템플릿 버전 단원에서 방금 만든 템플릿의 마이너 버전 왼쪽에 있는 라디오 버튼 (1.0) 을 선택합니다. 또는 정보 알림에서 게시를 선택하고 다음 단계를 건너뛸 수도 있습니다.

14. 템플릿 버전 단원에서 게시를 선택합니다.

15. 템플릿 상태가 게시됨으로 변경됩니다. 템플릿의 최신 및 권장 버전입니다.

16. 탐색 창에서 환경 템플릿을 선택하여 환경 템플릿 및 세부 정보 목록을 확인합니다.

서비스 템플릿과 템플릿 동기화 구성을 만드는 절차는 비슷합니다.

AWS CLI

AWS CLI를 사용하여 템플릿 및 템플릿 동기화 구성을 생성합니다.

1. 템플릿 생성 이 예시에서는 환경 템플릿이 생성됩니다.

   다음 명령을 실행합니다.

```
$ aws proton create-environment-template \
    --name "env-template"
```

   응답은 다음과 같습니다.

```
{
    "environmentTemplate": {
        "arn": "arn:aws:proton:us-east-1:123456789012:environment-template/env-template",
        "createdAt": "2021-11-07T23:32:43.045000+00:00",
        "displayName": "env-template",
        "lastModifiedAt": "2021-11-07T23:32:43.045000+00:00",
        "name": "env-template",
        "status": "DRAFT",
        "templateName": "env-template"
    }
}
```

2. 다음을 AWS CLI 제공하여와 템플릿 동기화 구성을 생성합니다.

   - 동기화할 템플릿입니다. 템플릿 동기화 구성을 만든 후에도 콘솔에서 또는 AWS CLI를 사용하여 수동으로 새 버전을 만들 수 있습니다.
   - 템플릿 이름입니다.
   - 템플릿 유형.
   - 동기화하려는 링크된 리포지토리
   - 연결된 저장소 제공자.
   - 템플릿 번들이 있는 브랜치입니다.
   - (선택 사항) 템플릿 번들이 포함된 디렉터리의 경로입니다. 기본적으로는 템플릿 이름과 일치하는 첫 번째 디렉터리를 AWS Proton 찾습니다.

   다음 명령을 실행합니다.

```
$ aws proton create-template-sync-config \
    --template-name "env-template" \
    --template-type "ENVIRONMENT" \
    --repository-name "myrepos/templates" \
    --repository-provider "GITHUB" \
    --branch "main" \
    --subdirectory "env-template/"
```

응답은 다음과 같습니다.

```
{
    "templateSyncConfigDetails": {
        "branch": "main",
        "repositoryName": "myrepos/templates",
        "repositoryProvider": "GITHUB",
        "subdirectory": "templates",
        "templateName": "env-template",
        "templateType": "ENVIRONMENT"
    }
}
```

3. 템플릿 버전을 게시하려면 템플릿 등록 및 게시을 참조하세요.

서비스 템플릿 동기화

위 예제는 환경 템플릿을 동기화하는 방법을 보여줍니다. 서비스 템플릿도 비슷합니다. 서비스 템플릿을 동기화하려면 템플릿 번들의 각 메이저 버전 디렉터리에 이름이 지정된 `.template-registration.yaml` 추가 파일을 추가합니다. 이 파일에는 커밋 후 서비스 템플릿 버전을 생성할 때 AWS Proton 필요한 추가 세부 정보가 포함되어 있습니다. AWS Proton 콘솔 또는 API를 사용하여 서비스 템플릿 버전을 명시적으로 생성하는 경우 이러한 세부 정보를 입력으로 제공하면이 파일이 템플릿 동기화를 위해 이러한 입력을 대체합니다.

```
./templates/                                                    # subdirectory (optional)
 /templates/my-svc-template/                                    # service template name
 /templates/my-svc-template/v1/                                 # service template version
 /templates/my-svc-template/v1/.template-registration.yaml      # service template version
 properties
 /templates/my-svc-template/v1/instance_infrastructure/         # template bundle
 /templates/my-svc-template/v1/schema/
```

`.template-registration.yaml` 파일에는 다음 코드가 포함되어 있습니다.

- 호환되는 환경[필수] - 이러한 환경 템플릿 및 메이저 버전을 기반으로 하는 환경은 이 서비스 템플릿 버전을 기반으로 하는 서비스와 호환됩니다.
- 지원되는 구성 요소 소스[선택 사항] - 이러한 소스를 사용하는 구성 요소는 이 서비스 템플릿 버전을 기반으로 하는 서비스와 호환됩니다. 지정하지 않으면 구성 요소를 이러한 서비스에 연결할 수 없습니다. 구성 요소에 대한 자세한 내용은 구성 요소를 참조하세요.

파일의 YAML 구문은 다음과 같습니다.

```
compatible_environments:
  - env-templ-name:major-version
  - ...
supported_component_sources:
  - DIRECTLY_DEFINED
```

하나 이상의 환경 템플릿/메이저 버전 조합을 지정합니다. `supported_component_sources` 지정은 선택 사항이며 지원되는 값은 `DIRECTLY_DEFINED`뿐입니다.

Example.template-registration.yaml

이 예제에서 서비스 템플릿 버전은 `my-env-template` 환경 템플릿의 메이저 버전 1 및 2와 호환됩니다. 또한 `another-env-template` 환경 템플릿의 주요 버전 1 및 3과도 호환됩니다. 파일은 `supported_component_sources`을 지정하지 않으므로 이 서비스 템플릿 버전을 기반으로 하는 서비스에 구성 요소를 연결할 수 없습니다.

```
compatible_environments:
  - my-env-template:1
  - my-env-template:2
  - another-env-template:1
  - another-env-template:3
```

> **Note**
>
> 이전에는에서 호환되는 환경을 지정하기 `.compatible-envs`위해 다른 파일를 AWS Proton 정의했습니다. AWS Proton 는 이전 버전과의 호환성을 위해 해당 파일과 형식을 지원합니다. 확장이 불가능하고 구성 요소와 같은 최신 기능을 지원할 수 없으므로 더 이상 사용하지 않는 것이 좋습니다.

템플릿 동기화 구성 세부 정보 보기

콘솔 또는 CLI를 사용하여 템플릿 동기화 구성 세부 데이터를 확인합니다.

AWS Management Console

콘솔을 사용하여 템플릿 동기화 구성 세부 정보를 확인합니다.

1. 탐색 창에서 (환경 또는 서비스) 템플릿을 선택합니다.
2. 세부 데이터를 보려면 템플릿 동기화 구성을 만든 템플릿의 이름을 선택합니다.
3. 템플릿의 세부 정보 페이지에서 동기화 탭을 선택하여 템플릿 동기화 구성 세부 정보 데이터를 확인합니다.

AWS CLI

AWS CLI 를 사용하여 동기화된 템플릿을 봅니다.

다음 명령을 실행합니다.

```
$ aws proton get-template-sync-config \
    --template-name "svc-template" \
    --template-type "SERVICE"
```

응답은 다음과 같습니다.

```
{
    "templateSyncConfigDetails": {
        "branch": "main",
        "repositoryProvider": "GITHUB",
        "repositoryName": "myrepos/myrepo",
        "subdirectory": "svc-template",
        "templateName": "svc-template",
        "templateType": "SERVICE"
    }
}
```

AWS CLI 를 사용하여 템플릿 동기화 상태를 가져옵니다.

`template-version`에 템플릿 메이저 버전을 입력합니다.

다음 명령을 실행합니다.

```
$ aws proton get-template-sync-status \
    --template-name "env-template" \
    --template-type "ENVIRONMENT" \
    --template-version "1"
```

템플릿 동기화 구성 편집

`template-name`및 `template-type`를 제외한 모든 템플릿 동기화 구성 파라미터를 편집할 수 있습니다.

콘솔 또는 CLI를 사용하여 템플릿 동기화 구성을 편집하는 방법을 알아보세요.

AWS Management Console

콘솔을 사용하여 템플릿 동기화 구성 브랜치를 편집합니다.

템플릿 목록에서.

1. AWS Proton 콘솔에서 (환경 또는 서비스) 템플릿을 선택합니다.
2. 템플릿 목록에서 편집하려는 템플릿 동기화 구성이 포함된 템플릿의 이름을 선택합니다.
3. 템플릿 세부 정보 페이지에서 템플릿 동기화 탭을 선택합니다.
4. 템플릿 동기화 세부 정보 단원에서 편집을 선택합니다.
5. 편집 페이지의 소스 코드 리포지토리 단원에서 브랜치트을 대해 브랜치를 선택한 다음 구성 저장을 선택합니다.

AWS CLI

다음 예제 명령 및 응답은 CLI를 사용하여 템플릿 동기화 구성 **branch**를 편집하는 방법을 보여줍니다.

다음 명령을 실행합니다.

```
$ aws proton update-template-sync-config \
    --template-name "env-template" \
    --template-type "ENVIRONMENT" \
    --repository-provider "GITHUB" \
```

```
    --repository-name "myrepos/templates" \
    --branch "fargate" \
    --subdirectory "env-template"
```

응답은 다음과 같습니다.

```
{
    "templateSyncConfigDetails": {
        "branch": "fargate",
        "repositoryProvider": "GITHUB",
        "repositoryName": "myrepos/myrepo",
        "subdirectory": "templates",
        "templateName": "env-template",
        "templateType": "ENVIRONMENT"
    }
}
```

마찬가지로 AWS CLI 를 사용하여 동기화된 서비스 템플릿을 업데이트할 수 있습니다.

템플릿 동기화 구성 삭제

콘솔 또는 CLI를 사용하여 템플릿 동기화 구성을 삭제합니다.

AWS Management Console

콘솔을 사용하여 템플릿 동기화 구성을 삭제합니다.

1. 템플릿 세부 정보 페이지에서 동기화 탭을 선택합니다.
2. 동기화 세부 정보 단원에서 연결 해제를 선택합니다.

AWS CLI

다음 예제 명령 및 응답은를 사용하여 동기화된 템플릿 구성을 삭제 AWS CLI 하는 방법을 보여줍니다.

다음 명령을 실행합니다.

```
$ aws proton delete-template-sync-config \
    --template-name "env-template" \
```

```
    --template-type "ENVIRONMENT"
```

응답은 다음과 같습니다.

```
{
    "templateSyncConfig": {
        "templateName": "env-template",
        "templateType": "ENVIRONMENT"
    }
}
```

서비스 동기화 구성

서비스 동기화를 사용하면 Git을 사용하여 AWS Proton 서비스를 구성하고 배포할 수 있습니다. 서비스 동기화를 사용하여 Git 리포지토리에 정의된 구성으로 AWS Proton 서비스에 대한 초기 배포 및 업데이트를 관리할 수 있습니다. Git을 통해 버전 추적 및 풀 요청과 같은 기능을 사용하여 서비스를 구성, 관리 및 배포할 수 있습니다. 서비스 동기화는 AWS Proton 및 Git을 결합하여 AWS Proton 템플릿을 통해 정의되고 관리되는 표준화된 인프라를 프로비저닝하는 데 도움이 됩니다. Git 리포지토리의 서비스 정의를 관리하고 도구 전환을 줄입니다. Git을 단독으로 사용하는 것에 비해에서 템플릿 및 배포 AWS Proton 를 표준화하면 인프라 관리에 소요되는 시간을 줄일 수 있습니다. AWS Proton 또한 개발자와 플랫폼 팀 모두에게 더 높은 투명성과 감사 가능성을 제공합니다.

AWS Proton OPS 파일

`proton-ops` 파일은가 서비스 인스턴스를 업데이트하는 데 사용되는 사양 파일을 AWS Proton 찾는 위치를 정의합니다. 또한 서비스 인스턴스를 업데이트할 순서와 한 인스턴스에서 다른 인스턴스로 변경 사항을 프로모션할 시기를 정의합니다.

이 `proton-ops` 파일은 연결된 저장소에 있는 사양 파일 또는 여러 사양 파일을 사용하여 서비스 인스턴스를 동기화하는 것을 지원합니다. 다음 예와 같이 `proton-ops` 파일에 동기화 블록을 정의하여 이 작업을 수행할 수 있습니다.

예시 ./configuration/proton-ops.yaml:

```
sync:
  services:
      frontend-svc:
          alpha:
              branch: dev
```

```
            spec: ./frontend-svc/test/frontend-spec.yaml
        beta:
            branch: dev
            spec: ./frontend-svc/test/frontend-spec.yaml
        gamma:
            branch: pre-prod
            spec: ./frontend-svc/pre-prod/frontend-spec.yaml
        prod-one:
            branch: prod
            spec: ./frontend-svc/prod/frontend-spec-second.yaml
        prod-two:
            branch: prod
            spec: ./frontend-svc/prod/frontend-spec-second.yaml
        prod-three:
            branch: prod
            spec: ./frontend-svc/prod/frontend-spec-second.yaml
```

위 예시에서 `frontend-svc`는 서비스 이름이고, `alpha`, `beta` , `gamma`, `prod-one`, `prod-two` 및 `prod-three`은 는 인스턴스입니다.

spec 파일은 `proton-ops`파일 내에 정의된 모든 인스턴스 또는 인스턴스의 하위 집합일 수 있습니다. 하지만 최소한 브랜치 내에 정의된 인스턴스와 동기화 대상 사양이 있어야 합니다. 인스턴스가 특정 브랜치 및 spec 파일 위치와 함께 `proton-ops` 파일에 정의되어 있지 않으면 서비스 동기화에서 해당 인스턴스를 만들거나 업데이트하지 않습니다.

다음 예는 spec 파일의 모양을 보여줍니다. `proton-ops` 파일은 이러한 spec 파일에서 동기화된다는 점을 기억합니다.

예제 **`./frontend-svc/test/frontend-spec.yaml`**:

```
proton: "ServiceSpec"
instances:
- name: "alpha"
  environment: "frontend-env"
  spec:
    port: 80
    desired_count: 1
    task_size: "x-small"
    image: "public.ecr.aws/z9d2n7e1/nginx:1.21.0"
- name: "beta"
  environment: "frontend-env"
  spec:
```

```
    port: 80
    desired_count: 1
    task_size: "x-small"
    image: "public.ecr.aws/z9d2n7e1/nginx:1.21.0"
```

예제 **./frontend-svc/pre-prod/frontend-spec.yaml**:

```
proton: "ServiceSpec"
instances:
- name: "gamma"
  environment: "frontend-env"
  spec:
    port: 80
    desired_count: 1
    task_size: "x-small"
    image: "public.ecr.aws/z9d2n7e1/nginx:1.21.0"
```

예제 **./frontend-svc/prod/frontend-spec-second.yaml**:

```
proton: "ServiceSpec"
instances:
- name: "prod-one"
  environment: "frontend-env"
  spec:
    port: 80
    desired_count: 1
    task_size: "x-small"
    image: "public.ecr.aws/z9d2n7e1/nginx:1.21.0"
- name: "prod-two"
  environment: "frontend-env"
  spec:
    port: 80
    desired_count: 1
    task_size: "x-small"
    image: "public.ecr.aws/z9d2n7e1/nginx:1.21.0"
- name: "prod-three"
  environment: "frontend-env"
  spec:
    port: 80
    desired_count: 1
    task_size: "x-small"
    image: "public.ecr.aws/z9d2n7e1/nginx:1.21.0"
```

인스턴스가 동기화되지 않고 동기화를 시도할 때 계속 문제가 발생하는 경우 GetServiceInstanceSyncStatusAPI를 직접 호출하면 문제를 해결하는 데 도움이 될 수 있습니다.

> **Note**
>
> 서비스 동기화를 사용하는 고객은 여전히 AWS Proton 제한으로 제한됩니다.

Blockers

서비스 동기화를 사용하여 AWS Proton 서비스를 동기화하면 서비스 사양을 업데이트하고 Git 리포지토리에서 서비스 인스턴스를 생성하고 업데이트할 수 있습니다. 그러나 AWS Management Console 또는를 통해 서비스 또는 인스턴스를 수동으로 업데이트해야 하는 경우가 있을 수 있습니다 AWS CLI.

AWS Proton 는 서비스 인스턴스 업데이트 또는 서비스 인스턴스 삭제 AWS CLI와 같이 AWS Management Console 또는를 통해 수행한 수동 변경 사항을 덮어쓰지 않도록 합니다. 이를 위해 AWS Proton 은 수동 변경이 감지되면 서비스 동기화를 비활성화하여 서비스 동기화 차단기를 자동으로 생성합니다.

서비스와 관련된 모든 차단기를 가져오려면 서비스와 관련된 각 `serviceInstance`에 대해 다음을 순서대로 수행해야 합니다.

- `serviceName`만 사용하여 `getServiceSyncBlockerSummary` API를 직접 호출합니다.
- `serviceName`와 `serviceInstanceName`를 사용하여 `getServiceSyncBlockerSummary` API를 직접 호출합니다.

그러면 가장 최근의 차단기 목록과 이와 관련된 상태가 반환됩니다. ACTIVE로 표시된 차단기가 있는 경우 각각의 경우에 대해 `blockerId` 및 `resolvedReason`가 있는 `UpdateServiceSyncBlocker` API를 직접 호출하여 해결해야 합니다.

서비스 인스턴스를 수동으로 업데이트하거나 생성하는 경우는 서비스 인스턴스에서 서비스 동기화 블로커를 AWS Proton 생성합니다.는 AWS Proton 다른 모든 서비스 인스턴스를 동기화하기 위해 계속 진행하지만 블로커가 해결될 때까지이 서비스 인스턴스의 동기화를 비활성화합니다. 서비스에서 서비스 인스턴스를 삭제하면가 서비스에서 서비스 동기화 차단기를 AWS Proton 생성합니다. 이렇게 하면 AWS Proton 가 블로커가 해결될 때까지 서비스 인스턴스를 동기화할 수 없습니다.

모든 활성 차단기를 설정한 후에는 활성 차단기 각각에 `blockerId` 및 `resolvedReason`이 있는 `UpdateServiceSyncBlocker` API를 직접 호출하여 차단기를 해결해야 합니다.

를 사용하여 로 이동하여 AWS Proton 서비스 동기화 탭을 선택하여 서비스 동기화가 비활성화되었는지 확인할 AWS Management Console수 있습니다. 서비스 또는 서비스 인스턴스가 차단된 경우 활성화 버튼이 나타납니다. 서비스 동기화를 활성화하려면 활성화를 선택합니다.

주제

서비스 동기화 구성 생성

콘솔 또는를 사용하여 서비스 동기화 구성을 생성할 수 있습니다 AWS CLI.

AWS Management Console

1. 서비스 템플릿 선택 페이지에서 템플릿을 선택하고 구성을 선택합니다.
2. 서비스 구성 페이지의 서비스 세부 정보 단원에서 새 서비스 이름을 입력합니다.
3. (선택 사항) 서비스 설명을 입력합니다.
4. 애플리케이션 소스 코드 리포지토리 섹션에서 연결된 Git 리포지토리 선택을 선택하여 이미 연결한 리포지토리를 선택합니다 AWS Proton. 연결된 리포지토리가 아직 없는 경우 다른 Git 리포지토리 연결을 선택하고 리포지토리 링크 만들기의 지침을 따릅니다.
5. 리포지토리의 경우 목록에서 소스 코드 리포지토리의 이름을 선택합니다.
6. 브랜치 경우 목록에서 소스 코드의 리포지토리 브랜치 이름을 선택합니다.
7. (선택 사항) 태그 단원에서 새 태그 추가 를 선택하고 키와 값을 입력하여 고객 관리형 태그를 생성합니다.
8. 다음을 선택합니다.
9. 서비스 인스턴스 구성 페이지의 서비스 정의 소스 단원에서 Git에서 서비스 동기화를 선택합니다.
10. 서비스 정의 파일 단원에서 AWS Proton 가 `proton-ops` 파일을 생성하려면 AWS Proton 이 파일을 생성하도록 하기를 선택합니다. 이 옵션을 사용하면가 지정한 위치에 `spec` 및 `proton-ops` 파일을 AWS Proton 생성합니다. 자체 OPS 파일을 생성하려면 자체 파일을 제공합니다를 선택합니다.

11. 서비스 정의 리포지토리 섹션에서 연결된 Git 리포지토리 선택을 선택하여 이미 연결한 리포지토리를 선택합니다 AWS Proton.
12. 리포지토리 이름의 경우 목록에서 소스 코드 리포지토리의 이름을 선택합니다.
13. **proton-ops** 파일 브랜치의 경우가 OPS 및 사양 파일을 AWS Proton 넣을 목록에서 브랜치의 이름을 선택합니다.
14. 서비스 인스턴스 섹션의 각 필드는 `proton-ops` 파일의 값을 기반으로 자동으로 채워집니다.
15. 다음을 선택하고 입력 내용을 검토합니다.
16. 생성(Create)을 선택합니다.

AWS CLI

를 사용하여 서비스 동기화 구성 생성 AWS CLI

- 다음 명령을 실행합니다.

```
$ aws proton create-service-sync-config \
    --resource "service-arn" \
    --repository-provider "GITHUB" \
    --repository "example/proton-sync-service" \
    --ops-file-branch "main" \
    --proton-ops-file "./configuration/custom-proton-ops.yaml" (optional)
```

응답은 다음과 같습니다.

```
{
    "serviceSyncConfig": {
        "branch": "main",
        "filePath": "./configuration/custom-proton-ops.yaml",
        "repositoryName": "example/proton-sync-service",
        "repositoryProvider": "GITHUB",
        "serviceName": "service name"
    }
}
```

서비스 동기화의 구성 세부 정보 보기

콘솔 또는 AWS CLI를 사용하여 서비스 동기화에 대한 구성 세부 정보 데이터를 볼 수 있습니다.

AWS Management Console

콘솔을 사용하여 서비스 동기화의 구성 세부 정보를 봅니다.

1. 탐색 창에서 서비스를 선택합니다.
2. 세부 데이터를 보려면 서비스 동기화 구성을 만든 서비스의 이름을 선택합니다.
3. 서비스의 세부 정보 페이지에서 서비스 동기화 탭을 선택하여 서비스 동기화에 대한 구성 세부 데이터를 확인합니다.

AWS CLI

AWS CLI 를 사용하여 동기화된 서비스를 가져옵니다.

다음 명령을 실행합니다.

```
$ aws proton get-service-sync-config \
    --service-name "service name"
```

응답은 다음과 같습니다.

```
{
    "serviceSyncConfig": {
        "branch": "main",
        "filePath": "./configuration/custom-proton-ops.yaml",
        "repositoryName": "example/proton-sync-service",
        "repositoryProvider": "GITHUB",
        "serviceName": "service name"
    }
}
```

AWS CLI 를 사용하여 서비스 동기화 상태를 가져옵니다.

다음 명령을 실행합니다.

```
$ aws proton get-service-sync-status \
    --service-name "service name"
```

서비스 동기화 구성 편집

콘솔 또는를 사용하여 서비스 동기화 구성을 편집할 수 있습니다 AWS CLI.

AWS Management Console

콘솔을 사용하여 서비스 동기화 구성을 편집합니다.

1. 탐색 창에서 서비스를 선택합니다.
2. 세부 데이터를 보려면 서비스 동기화 구성을 만든 서비스의 이름을 선택합니다.
3. 서비스 세부 정보 페이지에서 서비스 동기화 탭을 선택합니다.
4. 서비스 동기화 단원에서 편집을 선택합니다.
5. 편집 페이지에서 편집하려는 정보를 업데이트한 다음 저장을 선택합니다.

AWS CLI

다음 예제 명령 및 응답은 AWS CLI를 사용하여 서비스 동기화 구성을 편집하는 방법을 보여줍니다.

다음 명령을 실행합니다.

```
$ aws proton update-service-sync-config \
    --service-name "service name" \
    --repository-provider "GITHUB" \
    --repository "example/proton-sync-service" \
    --ops-file-branch "main" \
    --ops-file "./configuration/custom-proton-ops.yaml"
```

응답은 다음과 같습니다.

```
{
    "serviceSyncConfig": {
        "branch": "main",
        "filePath": "./configuration/custom-proton-ops.yaml",
        "repositoryName": "example/proton-sync-service",
        "repositoryProvider": "GITHUB",
        "serviceName": "service name"
    }
}
```

서비스 동기화 구성 삭제

콘솔 또는를 사용하여 서비스 동기화 구성을 삭제할 수 있습니다 AWS CLI.

AWS Management Console

콘솔을 사용하여 서비스 동기화 구성 삭제

1. 서비스 세부 정보 페이지에서 서비스 동기화 탭을 선택합니다.
2. 서비스 동기화 세부 정보 단원에서 연결 해제를 선택하여 리포지토리 연결을 끊습니다. 리포지토리 연결이 끊긴 후에는 더 이상 해당 리포지토리에서 서비스를 동기화하지 않습니다.

AWS CLI

다음 예제 명령 및 응답은를 사용하여 서비스 동기화 구성을 삭제 AWS CLI 하는 방법을 보여줍니다.

다음 명령을 실행합니다.

```
$ aws proton delete-service-sync-config \
    --service-name "service name"
```

응답은 다음과 같습니다.

```
{
    "serviceSyncConfig": {
        "branch": "main",
        "filePath": "./configuration/custom-proton-ops.yaml",
        "repositoryName": "example/proton-sync-service",
        "repositoryProvider": "GITHUB",
        "serviceName": "service name"
    }
}
```

> **Note**
>
> 서비스 동기화는 서비스 인스턴스를 삭제하지 않습니다. 구성만 삭제됩니다.

AWS Proton 환경

의 경우 AWS Proton환경은 서비스가 배포되는 AWS Proton 공유 리소스 및 정책 세트를 나타냅니다. 서비스 AWS Proton 인스턴스 간에 공유될 것으로 예상되는 모든 리소스를 포함할 수 있습니다. 이러한 리소스에는 VPC, 클러스터, 공유 로드 밸런서 또는 API 게이트웨이가 포함될 수 있습니다. 서비스를 배포하려면 먼저 AWS Proton 환경을 생성해야 합니다.

이 단원에서는 만들기, 보기, 업데이트 및 삭제 작업을 사용하여 환경을 관리하는 방법에 대해 설명합니다. >추가 정보는 AWS Proton 서비스 API 참조를 참조하세요.

주제

- IAM 역할
- 환경 생성
- 환경 데이터 보기
- 환경 업데이트
- 환경을 삭제합니다.
- 환경 계정 연결
- 고객 관리형 환경
- CodeBuild 프로비저닝 역할 생성

IAM 역할

를 사용하면 소유하고 관리하는 AWS 리소스에 대한 IAM 역할과 AWS KMS 키를 AWS Proton제공할 수 있습니다. 이는 나중에 개발자가 소유하고 관리하는 리소스에 적용되어 사용됩니다. 개발자 팀의 AWS Proton API 액세스를 제어하는 IAM 역할을 생성합니다.

AWS Proton 서비스 역할

새 환경을 만들 때는 관련 IAM 서비스 역할을 제공합니다. 역할에는 환경 템플릿과 서비스 템플릿 모두에 정의된 모든 프로비저닝된 인프라를 업데이트하는 데 필요한 모든 권한이 포함됩니다. 역할 예는 AWS Proton 를 사용하여 프로비저닝하기 위한 서비스 역할 CloudFormation을 참조하세요. 환경 계정 연결 및 환경 계정을 사용하는 경우 선택한 환경 계정에서 역할을 생성합니다. 자세한 정보는 한 계정에서 환경을 만들고 다른 계정에서 프로비저닝합니다. 및 환경 계정 연결를 참조하세요.

이 서비스 역할을 제공하는 방법과 역할을 맡는 사람은 환경의 프로비전 방법에 따라 다릅니다.

- AWS관리형 프로비저닝 - 환경을 생성하는 동안 AWS Proton직접 또는 계정 연결을 통해 간접적으로에 역할을 제공합니다.는 환경 및 서비스 인프라를 프로비저닝하기 위해 관련 계정의 역할을 수 AWS Proton 임합니다.
- 자체 관리형 프로비저닝 — 풀 리퀘스트 (PR) 가 프로비저닝 작업을 트리거할 때 적절한 자격 증명을 사용하여 적절한 역할을 맡도록 프로비저닝 자동화를 구성하는 것은 사용자의 책임입니다. 역할을 수임하는 GitHub 작업의 예는 GitHub 작업에 대한 "자 AWS 격 증명 구성" 작업 설명서의 역할 수임을 참조하세요.

프로비저닝 방법에 대한 자세한 내용은 the section called “프로비저닝 방법”을 참조하세요.

환경 생성

AWS Proton 환경을 만드는 방법을 알아봅니다.

다음 두 가지 방법 중 하나로 AWS Proton 환경을 생성할 수 있습니다.

- 표준 환경 템플릿을 사용하여 표준 환경을 생성, 관리 및 프로비저닝합니다. 환경의 인프라를 AWS Proton 프로비저닝합니다.
- 고객 관리형 환경 템플릿을 사용하여 고객 관리형 인프라에 AWS Proton 연결합니다. 외부에서 자체 공유 리소스를 프로비저닝한 다음 AWS Proton에서 사용할 AWS Proton 수 있는 프로비저닝 출력을 제공합니다.

환경을 만들 때 여러 프로비저닝 접근 방식 중 하나를 선택할 수 있습니다.

- AWS 관리형 프로비저닝 - 단일 계정에서 환경을 생성, 관리 및 프로비저닝합니다. 환경을 AWS Proton 프로비저닝합니다.

  이 방법은 클라우드포메이션 인프라 코드 (IaC) 템플릿만 지원합니다.
- AWS 다른 계정에 대한 관리형 프로비저닝 - 단일 관리 계정에서 환경 계정 연결이 있는 다른 계정에 프로비저닝된 환경을 생성하고 관리합니다.는 다른 계정에 환경을 AWS Proton 프로비저닝합니다. 자세한 내용은 한 계정에서 환경을 만들고 다른 계정에서 프로비저닝합니다. 및 환경 계정 연결 섹션을 참조하세요.

  이 방법은 CloudFormation 인프라 코드 (IaC) 템플릿만 지원합니다.
- 자체 관리형 프로비저닝 - 자체 프로비저닝 인프라를 사용하여 연결된 리포지토리에 프로비저닝 풀 요청을 AWS Proton 제출합니다.

이 방법은 Terraform IaC 템플릿만 지원합니다.

- CodeBuild 프로비저닝 - 사용자가 제공하는 셸 명령을 실행하는 AWS CodeBuild 데 AWS Proton 사용합니다. 명령은가 AWS Proton 제공하는 입력을 읽을 수 있으며 인프라 프로비저닝 또는 프로비저닝 해제와 출력 값 생성을 담당합니다. 이 방법을 위한 템플릿 번들에는 매니페스트 파일의 명령과 이러한 명령에 필요할 수 있는 프로그램, 스크립트 또는 기타 파일이 포함되어 있습니다.

  CodeBuild 프로비저닝을 사용하는 예로를 사용하여 AWS 리소스를 AWS Cloud Development Kit (AWS CDK) 프로비저닝하는 코드와 CDK를 설치하고 CDK 코드를 실행하는 매니페스트를 포함할 수 있습니다.

  자세한 내용은 the section called “CodeBuild 번들” 단원을 참조하십시오.

  > **Note**
  >
  > CodeBuild 프로비저닝을 환경 및 서비스와 함께 사용할 수 있습니다. 현재로서는 이 방법으로 구성 요소를 프로비저닝할 수 없습니다.

AWS 관리형 프로비저닝(동일한 계정과 다른 계정 모두에서) AWS Proton 을 사용하면가 리소스를 직접 호출하여 프로비저닝합니다.

자체 관리형 프로비저닝을 사용하면 AWS Proton 는 풀 요청을 생성하여 IaC 엔진이 리소스를 프로비저닝하는 데 사용하는 컴파일된 IaC 파일을 제공합니다.

자세한 내용은 the section called “프로비저닝 방법”, the section called “템플릿 번들” 및 the section called “환경 스키마 요구 사항”을 참조하세요.

주제

- 동일한 계정에서 표준 환경을 만들고 프로비저닝합니다.
- 한 계정에서 환경을 만들고 다른 계정에서 프로비저닝합니다.
- 자체 관리형 프로비저닝을 사용하여 환경을 만들고 프로비저닝합니다.

동일한 계정에서 표준 환경을 만들고 프로비저닝합니다.

콘솔 또는를 사용하여 단일 계정에서 환경을 AWS CLI 생성하고 프로비저닝합니다. 프로비저닝은에서 관리합니다 AWS.

AWS Management Console

콘솔을 사용하여 단일 계정으로 환경을 만들고 프로비저닝할 수 있습니다.

1. AWS Proton 콘솔에서 환경을 선택합니다.
2. 환경 생성을 선택합니다.
3. 환경 템플릿 선택 페이지에서 템플릿을 선택하고 구성을 선택합니다.
4. 환경 구성 페이지의 프로비저닝 단원에서 AWS 관리형 프로비저닝을 선택합니다.
5. 배포 계정 단원에서 이 AWS 계정를 선택합니다.
6. 환경 구성 페이지의 환경 설정 단원에서 환경 이름을 입력합니다.
7. (선택 사항) 환경에 대한 설명을 입력합니다.
8. 환경 역할 단원에서 AWS Proton 서비스 역할 설정의 일부로 만든 AWS Proton 서비스 역할을 선택합니다.
9. (선택 사항) 구성 요소 역할 단원에서 직접 정의된 구성 요소를 환경에서 실행할 수 있게 하고 해당 구성 요소가 프로비저닝할 수 있는 리소스의 범위를 좁히는 서비스 역할을 선택합니다. 자세한 내용은 구성 요소을 참조하세요.
10. (선택 사항) 태그 단원에서 새 태그 추가를 선택하고 키와 값을 입력하여 고객 관리형 태그를 생성합니다.
11. 다음을 선택합니다.
12. 환경 사용자 지정 설정 구성 페이지에서 `required` 파라미터 값을 입력해야 합니다. `optional` 파라미터 값을 입력하거나 지정된 경우 기본값을 사용할 수 있습니다.
13. 다음을 선택하고 입력 내용을 검토합니다.
14. 생성(Create)을 선택합니다.

    환경 세부 정보 및 상태는 물론 환경에 대한 AWS 관리 태그와 고객 관리 태그를 확인하세요.
15. 탐색 창에서 환경을 선택합니다.

    새 페이지에는 환경 목록이 상태 및 기타 환경 세부 정보와 함께 표시됩니다.

AWS CLI

AWS CLI 를 사용하여 단일 계정에서 환경을 생성하고 프로비저닝합니다.

환경을 만들려면 AWS Proton 서비스 역할 ARN, 사양 파일 경로, 환경 이름, 환경 템플릿 ARN, 메이저 및 마이너 버전, 설명 (선택 사항) 을 지정합니다.

다음 예제는 환경 템플릿 스키마 파일에 정의된 두 입력의 값을 지정하는 YAML 형식이 지정된 사양 파일을 보여줍니다. `get-environment-template-minor-version` 명령을 사용하여 환경 템플릿 스키마를 볼 수 있습니다.

```
proton: EnvironmentSpec
spec:
  my_sample_input: "the first"
  my_other_sample_input: "the second"
```

다음 명령을 실행하여 환경을 생성합니다.

```
$ aws proton create-environment \
    --name "MySimpleEnv" \
    --template-name simple-env \
    --template-major-version 1 \
    --proton-service-role-arn "arn:aws:iam::123456789012:role/AWS ProtonServiceRole"
 \
    --spec "file://env-spec.yaml"
```

응답:

```
{
    "environment": {
        "arn": "arn:aws:proton:region-id:123456789012:environment/MySimpleEnv",
        "createdAt": "2020-11-11T23:03:05.405000+00:00",
        "deploymentStatus": "IN_PROGRESS",
        "lastDeploymentAttemptedAt": "2020-11-11T23:03:05.405000+00:00",
        "name": "MySimpleEnv",
        "protonServiceRoleArn": "arn:aws:iam::123456789012:role/ProtonServiceRole",
        "templateName": "simple-env"
    }
}
```

새 환경을 생성한 후 다음 예제 명령과 같이 AWS 및 고객 관리형 태그 목록을 볼 수 있습니다. AWS Proton 는 AWS 관리형 태그를 자동으로 생성합니다. AWS CLI를 사용하여 고객 관리 태그를 수정하고 생성할 수도 있습니다. 자세한 내용은 AWS Proton 리소스 및 태그 지정 단원을 참조하십시오.

명령:

```
$ aws proton list-tags-for-resource \
```

```
    --resource-arn "arn:aws:proton:region-id:123456789012:environment/MySimpleEnv"
```

한 계정에서 환경을 만들고 다른 계정에서 프로비저닝합니다.

콘솔 또는를 사용하여 다른 계정의 환경 인프라를 프로비저닝하는 관리 계정에서 표준 환경을 AWS CLI 생성합니다. 프로비저닝은 AWS에서 관리합니다.

콘솔 또는 CLI를 사용하기 전에 다음 단계를 완료합니다.

1. 관리 및 환경 계정의 AWS 계정 IDs를 식별하고 나중에 사용할 수 있도록 복사합니다.
2. 환경 계정에서 생성할 환경에 대한 최소 권한이 있는 AWS Proton 서비스 역할을 생성합니다. 자세한 내용은 AWS Proton 를 사용하여 프로비저닝하기 위한 서비스 역할 CloudFormation 단원을 참조하십시오.

AWS Management Console

콘솔을 사용하여 한 계정에서 환경을 만들고 다른 계정에서 프로비저닝합니다.

1. 환경 계정에서 환경 계정 연결을 생성하고 이를 사용하여 관리 계정에 연결하라는 요청을 보내십시오.
   a. AWS Proton 콘솔의 탐색 창에서 환경 계정 연결을 선택합니다.
   b. 환경 계정 연결 페이지에서 연결 요청을 선택합니다.

      > **Note**
      >
      > 환경 계정 연결 페이지 제목에 나열된 계정 ID가 미리 식별된 환경 계정 ID와 일치하는지 확인하세요.

   c. 연결 요청 페이지의 환경 역할 단원에서 기존 서비스 역할과 환경용으로 만든 서비스 역할의 이름을 선택합니다.
   d. 관리 계정에 연결 섹션에서 AWS Proton 환경의 관리 계정 ID와 환경 이름을 입력합니다. 나중에 사용할 수 있도록 이름을 복사하세요.
   e. 페이지 오른쪽 하단에서 연결 요청을 선택합니다.
   f. 관리 계정으로 전송된 환경 연결 테이블에 요청이 보류 중으로 표시되고 관리 계정의 요청을 수락하는 방법이 모달에 표시됩니다.

2. 관리 계정에서 환경 계정의 연결 요청을 수락합니다.

   a. 관리 계정에 로그인하고 AWS Proton 콘솔에서 환경 계정 연결을 선택합니다.

   b. 환경 계정 연결 페이지의 환경 계정 연결 요청 테이블에서 미리 식별된 환경 계정 ID와 일치하는 환경 계정 ID를 가진 환경 계정 연결을 선택합니다.

   > **Note**
   >
   > 환경 계정 연결 페이지 제목에 나열된 계정 ID가 미리 식별된 관리 계정 ID와 일치하는지 확인하세요.

   c. 수락을 선택합니다. 상태가 [보류 중] 에서 [연결됨] 으로 변경됩니다.

3. 관리 계정에서 환경을 생성합니다.

   a. 탐색 창에서 환경 템플릿을 선택합니다.

   b. 환경 템플릿 페이지에서 환경 템플릿 생성을 선택합니다.

   c. 환경 템플릿 선택 페이지에서 환경 템플릿을 선택합니다.

   d. 환경 구성 페이지의 프로비저닝 단원에서 AWS 관리형 프로비저닝을 선택합니다.

   e. 배포 계정 섹션에서 다른 AWS 계정을 선택합니다.

   f. 환경 세부 정보 단원에서 환경 계정 연결 및 환경 이름을 선택합니다.

   g. 다음을 선택합니다.

   h. 양식을 작성하고 검토 및 만들기 페이지가 표시될 때까지 다음을 선택합니다.

   i. 검토 후 환경 만들기를 선택합니다.

AWS CLI

AWS CLI 를 사용하여 한 계정에서 환경을 생성하고 다른 계정에서 프로비저닝합니다.

환경 계정에서 다음 명령을 실행하여 환경 계정 연결을 만들고 연결을 요청합니다.

```
$ aws proton create-environment-account-connection \
    --environment-name "simple-env-connected" \
    --role-arn "arn:aws:iam::222222222222:role/service-role/env-account-proton-
service-role" \
    --management-account-id "111111111111"
```

응답:

```
{
    "environmentAccountConnection": {
        "arn": "arn:aws:proton:region-id:222222222222:environment-account-
connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "environmentAccountId": "222222222222",
        "environmentName": "simple-env-connected",
        "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "lastModifiedAt": "2021-04-28T23:13:50.847000+00:00",
        "managementAccountId": "111111111111",
        "requestedAt": "2021-04-28T23:13:50.847000+00:00",
        "roleArn": "arn:aws:iam::222222222222:role/service-role/env-account-proton-
service-role",
        "status": "PENDING"
    }
}
```

관리 계정에서 다음 명령을 실행하여 환경 계정 연결 요청을 수락합니다.

```
$ aws proton accept-environment-account-connection \
    --id "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
```

응답:

```
{
    "environmentAccountConnection": {
        "arn": "arn:aws:proton:region-id:222222222222:environment-account-
connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "environmentAccountId": "222222222222",
        "environmentName": "simple-env-connected",
        "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "lastModifiedAt": "2021-04-28T23:15:33.486000+00:00",
        "managementAccountId": "111111111111",
        "requestedAt": "2021-04-28T23:13:50.847000+00:00",
        "roleArn": "arn:aws:iam::222222222222:role/service-role/env-account-proton-
service-role",
        "status": "CONNECTED"
    }
}
```

다음 명령을 실행하여 환경 계정 연결을 확인합니다.

```
$ aws proton get-environment-account-connection \
```

```
    --id "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
```

응답:

```
{
    "environmentAccountConnection": {
        "arn": "arn:aws:proton:region-id:222222222222:environment-account-
connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "environmentAccountId": "222222222222",
        "environmentName": "simple-env-connected",
        "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "lastModifiedAt": "2021-04-28T23:15:33.486000+00:00",
        "managementAccountId": "111111111111",
        "requestedAt": "2021-04-28T23:13:50.847000+00:00",
        "roleArn": "arn:aws:iam::222222222222:role/service-role/env-account-proton-
service-role",
        "status": "CONNECTED"
    }
}
```

관리 계정에서 다음 명령을 실행하여 환경을 만듭니다.

```
$ aws proton create-environment \
    --name "simple-env-connected" \
    --template-name simple-env-template \
    --template-major-version "1" \
    --template-minor-version "1" \
    --spec "file://simple-env-template/specs/original.yaml" \
    --environment-account-connection-id "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
```

응답:

```
{
    "environment": {
        "arn": "arn:aws:proton:region-id:111111111111:environment/simple-env-
connected",
        "createdAt": "2021-04-28T23:02:57.944000+00:00",
        "deploymentStatus": "IN_PROGRESS",
        "environmentAccountConnectionId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "environmentAccountId": "222222222222",
        "lastDeploymentAttemptedAt": "2021-04-28T23:02:57.944000+00:00",
        "name": "simple-env-connected",
```

```
        "templateName": "simple-env-template"
    }
}
```

자체 관리형 프로비저닝을 사용하여 환경을 만들고 프로비저닝합니다.

자체 관리형 프로비저닝을 사용하는 경우는 자체 프로비저닝 인프라를 사용하여 연결된 리포지토리에 프로비저닝 풀 요청을 AWS Proton 제출합니다. 풀 요청은 AWS 서비스를 호출하는 자체 워크플로를 시작하여 인프라를 프로비저닝합니다.

자체 관리형 프로비저닝 고려 사항:

- 환경을 만들기 전에 자체 관리형 프로비저닝을 위한 리포지토리 리소스 디렉터리를 설정합니다. 자세한 내용은 AWS Proton 코드형 인프라 파일 단원을 참조하십시오.
- 환경을 생성한 후는 인프라 프로비저닝 상태에 대한 비동기 알림을 수신할 때까지 AWS Proton 기다립니다. 프로비저닝 코드는 API를 AWS Proton `NotifyResourceStateChange` 사용하여 이러한 비동기 알림을 전송해야 합니다 AWS Proton.

콘솔에서 또는 AWS CLI와 함께 자체 관리형 프로비저닝을 사용할 수 있습니다. 다음 예시는 Terraform에서 자체 관리형 프로비저닝을 사용하는 방법을 보여줍니다.

AWS Management Console

콘솔을 사용하여 자체 관리형 프로비저닝을 사용하여 Terraform 환경을 만들 수 있습니다.

1. AWS Proton 콘솔에서 환경을 선택합니다.
2. 환경 생성을 선택합니다.
3. 환경 템플릿 선택 페이지에서 Terraform 템플릿을 선택하고 구성을 선택합니다.
4. 환경 구성 페이지의 프로비저닝 단원에서 자체 관리형 프로비저닝을 선택합니다.
5. 프로비전 리포지토리 세부 정보 단원에서,

    a. 프로비저닝 리포지토리를 아직 연결 AWS Proton하지 않은 경우 새 리포지토리를 선택하고 리포지토리 공급자 중 하나를 선택한 다음 CodeStar 연결에서 연결 중 하나를 선택합니다.

> **Note**
>
> 관련 리포지토리 공급자 계정에 아직 연결되지 않은 경우 새 CodeStar 연결 추가를 선택합니다. 그런 다음 연결을 만든 다음 CodeStar 연결 메뉴 옆에 있는 새로 고침 버튼을 선택합니다. 이제 메뉴에서 새 연결을 선택할 수 있을 것입니다.

리포지토리를 이미 연결한 경우 기존 리포지토리를 AWS Proton선택합니다.

b. 리포지토리 이름에서 퍼블릭 리포지토리를 선택합니다. 드롭다운 메뉴에는 기존 리포지토리 또는 새 리포지토리의 공급자 계정에 있는 리포지토리 목록이 표시됩니다.

c. 브랜치 이름에서 리포지토리 브랜치 중의 하나를 선택합니다.

6. 환경 설정 단원에서 환경 이름을 입력합니다.
7. (선택 사항) 환경에 대한 설명을 입력합니다.
8. (선택 사항) 태그 단원에서 새 태그 추가를 선택하고 키와 값을 입력하여 고객 관리형 태그를 생성합니다.
9. 다음을 선택합니다.
10. 환경 사용자 지정 설정 구성 페이지에서 `required` 파라미터 값을 입력해야 합니다. `optional` 파라미터 값을 입력하거나 지정된 경우 기본값을 사용할 수 있습니다.
11. 다음을 선택하고 입력 내용을 검토합니다.
12. 생성을 선택하여 풀 리퀘스트를 전송합니다.

    - 풀 리퀘스트를 승인하면 배포가 진행 중인 것입니다.
    - 풀 리퀘스트를 거부하면 환경 생성이 취소됩니다.
    - 풀 리퀘스트 제한 시간이 초과되면 환경 생성이 완료되지 않습니다.

13. 환경 세부 정보 및 상태와 환경에 대한 AWS 관리형 태그 및 고객 관리형 태그를 확인합니다.
14. 탐색 창에서 환경을 선택합니다.

    새 페이지에는 환경 목록이 상태 및 기타 환경 세부 정보와 함께 표시됩니다.

AWS CLI

자체 관리형 프로비저닝을 사용하여 환경을 만들 때는 `provisioningRepository`파라미터를 추가하고 `ProtonServiceRoleArn` 및 `environmentAccountConnectionId` 파라미터는 생략합니다.

AWS CLI 를 사용하여 자체 관리형 프로비저닝으로 Terraform 환경을 생성합니다.

1. 환경을 만들고 검토 및 승인을 위해 리포지토리에 풀 리퀘스트를 보내세요.

   다음 예제에서는 환경 템플릿 스키마 파일을 기반으로 두 입력의 값을 정의하는 YAML 형식이 지정된 사양 파일을 보여줍니다. `get-environment-template-minor-version` 명령을 사용하여 환경 템플릿 스키마를 볼 수 있습니다.

   Spec:

   ```
   proton: EnvironmentSpec
   spec:
     ssm_parameter_value: "test"
   ```

   다음 명령을 실행하여 환경을 생성합니다.

   ```
   $ aws proton create-environment \
       --name "pr-environment" \
       --template-name "pr-env-template" \
       --template-major-version "1" \
       --provisioning-repository="branch=main,name=myrepos/env-
   repo,provider=GITHUB" \
       --spec "file://env-spec.yaml"
   ```

   응답:>

   ```
   {
       "environment": {
           "arn": "arn:aws:proton:region-id:123456789012:environment/pr-
   environment",
           "createdAt": "2021-11-18T17:06:58.679000+00:00",
           "deploymentStatus": "IN_PROGRESS",
           "lastDeploymentAttemptedAt": "2021-11-18T17:06:58.679000+00:00",
           "name": "pr-environment",
           "provisioningRepository": {
               "arn": "arn:aws:proton:region-id:123456789012:repository/
   github:myrepos/env-repo",
               "branch": "main",
               "name": "myrepos/env-repo",
               "provider": "GITHUB"
           },
           "templateName": "pr-env-template"
   ```

```
    }
```

2. 요청을 검토합니다.

   - 요청을 승인하면 프로비저닝이 진행 중인 것입니다.
   - 요청을 거부하면 환경 생성이 취소됩니다.
   - 풀 리퀘스트 제한 시간이 초과되면 환경 생성이 완료되지 않습니다.

3. 에 프로비저닝 상태를 비동기적으로 제공합니다 AWS Proton. 다음 예제에서는 성공적인 프로비저닝 AWS Proton 을 알립니다.

```
$ aws proton notify-resource-deployment-status-change \
    --resource-arn "arn:aws:proton:region-id:123456789012:environment/pr-environment" \
    --status "SUCCEEDED"
```

환경 데이터 보기

AWS Proton 콘솔 또는를 사용하여 환경 세부 정보를 볼 수 있습니다 AWS CLI.

AWS Management Console

AWS Proton 콘솔을 사용하여 세부 정보가 있는 환경 목록과 세부 데이터가 포함된 개별 환경 목록을 볼 수 있습니다.

1. 환경 목록을 보려면 탐색 창에서 환경을 선택합니다.
2. 세부 데이터를 보려면 환경 이름을 선택합니다.

   환경 세부 데이터를 확인합니다.

AWS CLI

the AWS CLI get을 사용하거나 환경 세부 정보를 나열합니다.

다음 명령을 실행합니다.

```
$ aws proton get-environment \
    --name "MySimpleEnv"
```

응답:

```
{
    "environment": {
        "arn": "arn:aws:proton:region-id:123456789012:environment/MySimpleEnv",
        "createdAt": "2020-11-11T23:03:05.405000+00:00",
        "deploymentStatus": "SUCCEEDED",
        "lastDeploymentAttemptedAt": "2020-11-11T23:03:05.405000+00:00",
        "lastDeploymentSucceededAt": "2020-11-11T23:03:05.405000+00:00",
        "name": "MySimpleEnv",
        "protonServiceRoleArn": "arn:aws:iam::123456789012:role/ProtonServiceRole",
        "spec": "proton: EnvironmentSpec\nspec:\n  my_sample_input: \"the first\"\n
 my_other_sample_input: \"the second\"\n",
        "templateMajorVersion": "1",
        "templateMinorVersion": "0",
        "templateName": "simple-env"
    }
}
```

환경 업데이트

AWS Proton 환경이 환경 계정 연결과 연결된 경우 환경 계정 연결을 업데이트하거나 연결하기 위한 `protonServiceRoleArn` 파라미터를 업데이트하거나 포함하지 마십시오.

다음 두 가지 모두에 해당하는 경우에만 새 환경 계정 연결로 업데이트할 수 있습니다.

- 환경 계정 연결이 현재 환경 계정 연결이 만들어진 환경 계정과 동일한 환경 계정에서 만들어졌습니다.
- >환경 계정 연결이 현재 환경과 연결되어 있습니다.

환경이 환경 계정 연결과 연결되어 있지 않은 경우 `environmentAccountConnectionId` 파라미터를 업데이트하거나 포함하지 마십시오.

`environmentAccountConnectionId` 또는 `protonServiceRoleArn` 파라미터 및 값을 업데이트할 수 있습니다. 둘 다 업데이트할 수는 없습니다.

자체 관리형 프로비저닝을 사용하는 환경에서는 `provisioning-repository` 파라미터를 업데이트하거나 `environmentAccountConnectionId` 및 `protonServiceRoleArn` 파라미터를 생략하지 마십시오.

다음 목록에 설명된 대로 환경을 업데이트하는 데는 네 가지 모드가 있습니다. 를 사용할 때 AWS CLIdeployment-type 필드는 모드를 정의합니다. 콘솔을 사용할 때 이러한 모드는 작업에서 드롭다운되는 편집, 업데이트, 사소한 업데이트 및 주요 업데이트 작업에 매핑됩니다.

`NONE`

이 모드에서는 배포가 이루어지지 않습니다. 요청된 메타데이터 파라미터만 업데이트됩니다.

`CURRENT_VERSION`

이 모드에서는 사용자가 제공한 새 사양으로 환경이 배포되고 업데이트됩니다. 요청된 파라미터만 업데이트됩니다. 이 `deployment-type`을 사용할 때 마이너 버전 또는 메이저 버전 파라미터를 포함하지 마십시오.

`MINOR_VERSION`

이 모드에서는 환경이 기본적으로 현재 사용 중인 메이저 버전의 권장되는 최신 마이너 버전으로 배포 및 업데이트됩니다. 현재 사용 중인 메이저 버전과 다른 마이너 버전을 지정할 수도 있습니다.

`MAJOR_VERSION`

이 모드에서는 기본적으로 현재 템플릿의 게시된 권장(최신) 메이저 버전 및 마이너 버전으로 환경이 배포 및 업데이트됩니다. 사용 중인 메이저 버전보다 상위의 다른 메이저 버전과 마이너 버전을 지정할 수도 있습니다(선택 사항).

주제

AWS 관리형 프로비저닝 환경 업데이트

표준 프로비저닝은 CloudFormation을 사용하여 프로비저닝하는 환경에서만 지원됩니다.

콘솔 또는 AWS CLI 를 사용하여 환경을 업데이트합니다.

AWS Management Console

다음 단계에 따라 콘솔을 사용하여 환경을 업데이트합니다.

1. 다음 2단계 중 하나를 선택합니다.
   a. 환경 목록에서.
      i. AWS Proton 콘솔에서 환경을 선택합니다.
      ii. 환경 목록에서 업데이트하려는 환경의 왼쪽에 있는 라디오 버튼을 선택합니다.
   b. 콘솔의 작업자 환경 세부 정보 페이지
      i. AWS Proton 콘솔에서 환경을 선택합니다.
      ii. 환경 목록에서 업데이트할 환경의 이름을 선택합니다.
2. 다음 4단계 중 하나를 선택하여 환경을 업데이트합니다.
   a. 환경 배포가 필요 없는 편집을 하려면
      i. 설명을 변경하는 경우를 예로 들 수 있습니다.

         편집을 선택합니다.
      ii. 양식을 작성하고 다음 선택합니다.
      iii. 편집 내용을 검토하고 업데이트를 선택합니다.
   b. 메타데이터 입력만 업데이트하려면
      i. 작업을 선택한 다음 업데이트를 선택합니다.
      ii. 양식을 작성하고 편집을 선택합니다.
      iii. 양식을 작성하고 검토 페이지가 표시될 때까지 다음을 선택합니다.
      iv. 업데이트 내용을 검토하고 업데이트를 선택합니다.
   c. 환경 템플릿의 새 마이너 버전으로 업데이트하기.
      i. 작업을 선택한 다음 업데이트 마이너를 선택합니다.
      ii. 양식을 작성하고 다음 선택합니다.
      iii. 양식을 작성하고 검토 페이지가 표시될 때까지 다음을 선택합니다.
      iv. 업데이트 내용을 검토하고 업데이트를 선택합니다.

d. 환경 템플릿의 새 메이저 버전으로 업데이트하기.

i. 작업을 선택한 다음 업데이트 메이저를 선택합니다.
ii. 양식을 작성하고 다음 선택합니다.
iii. 양식을 작성하고 검토 페이지가 표시될 때까지 다음을 선택합니다.
iv. 업데이트 내용을 검토하고 업데이트를 선택합니다.

AWS CLI

AWS Proton AWS CLI 를 사용하여 환경을 새 마이너 버전으로 업데이트합니다.

다음 명령을 실행하여 환경을 업데이트합니다.

```
$ aws proton update-environment \
        --name "MySimpleEnv" \
        --deployment-type "MINOR_VERSION" \
        --template-major-version "1" \
        --template-minor-version "1" \
        --proton-service-role-arn arn:aws:iam::123456789012:role/service-
role/ProtonServiceRole \
        --spec "file:///spec.yaml"
```

응답:

```
{
    "environment": {
        "arn": "arn:aws:proton:region-id:123456789012:environment/MySimpleEnv",
        "createdAt": "2021-04-02T17:29:55.472000+00:00",
        "deploymentStatus": "IN_PROGRESS",
        "lastDeploymentAttemptedAt": "2021-04-02T17:48:26.307000+00:00",
        "lastDeploymentSucceededAt": "2021-04-02T17:29:55.472000+00:00",
        "name": "MySimpleEnv",
        "protonServiceRoleArn": "arn:aws:iam::123456789012:role/service-role/
ProtonServiceRole",
        "templateMajorVersion": "1",
        "templateMinorVersion": "0",
        "templateName": "simple-env"
    }
}
```

다음 명령을 실행하여 상태를 가져와 확인합니다.

```
$ aws proton get-environment \
        --name "MySimpleEnv"
```

응답:

```
{
    "environment": {
        "arn": "arn:aws:proton:region-id:123456789012:environment/MySimpleEnv",
        "createdAt": "2021-04-02T17:29:55.472000+00:00",
        "deploymentStatus": "SUCCEEDED",
        "environmentName": "MySimpleEnv",
        "lastDeploymentAttemptedAt": "2021-04-02T17:48:26.307000+00:00",
        "lastDeploymentSucceededAt": "2021-04-02T17:48:26.307000+00:00",
        "protonServiceRoleArn": "arn:aws:iam::123456789012:role/service-role/
ProtonServiceRole",
        "spec": "proton: EnvironmentSpec\n\nspec:\n  my_sample_input: hello\n
 my_other_sample_input: everybody\n",
        "templateMajorVersion": "1",
        "templateMinorVersion": "1",
        "templateName": "simple-env"
    }
}
```

자체 관리형 프로비저닝 환경 업데이트

자체 관리형 프로비저닝은 Terraform으로 프로비저닝하는 환경에서만 지원됩니다.

콘솔 또는 AWS CLI 를 사용하여 환경을 업데이트합니다.

AWS Management Console

다음 단계에 따라 콘솔을 사용하여 환경을 업데이트합니다.

1. 다음 2단계 중 하나를 선택합니다.

   a. 환경 목록에서.

      i. AWS Proton 콘솔에서 환경을 선택합니다.

      ii. 환경 목록에서 업데이트하려는 환경 템플릿의 왼쪽에 있는 라디오 버튼을 선택합니다.

b. 콘솔의 작업자 환경 세부 정보 페이지

i. AWS Proton 콘솔에서 환경을 선택합니다.

ii. 환경 목록에서 업데이트할 환경의 이름을 선택합니다.

2. 다음 4단계 중 하나를 선택하여 환경을 업데이트합니다.

a. 환경 배포가 필요 없는 편집을 하려면

i. 설명을 변경하는 경우를 예로 들 수 있습니다.

편집을 선택합니다.

ii. 양식을 작성하고 다음 선택합니다.

iii. 편집 내용을 검토하고 업데이트를 선택합니다.

b. 메타데이터 입력만 업데이트하려면

i. 작업을 선택한 다음 업데이트를 선택합니다.

ii. 양식을 작성하고 편집을 선택합니다.

iii. 양식을 작성하고 검토 페이지가 표시될 때까지 다음을 선택합니다.

iv. 업데이트 내용을 검토하고 업데이트를 선택합니다.

c. 환경 템플릿의 새 마이너 버전으로 업데이트하기.

i. 작업을 선택한 다음 업데이트 마이너를 선택합니다.

ii. 양식을 작성하고 다음 선택합니다.

iii. 양식을 작성하고 검토 페이지가 표시될 때까지 다음을 선택합니다.

iv. 업데이트 내용을 검토하고 업데이트를 선택합니다.

d. 환경 템플릿의 새 메이저 버전으로 업데이트하기.

i. 작업을 선택한 다음 업데이트 메이저를 선택합니다.

ii. 양식을 작성하고 다음 선택합니다.

iii. 양식을 작성하고 검토 페이지가 표시될 때까지 다음을 선택합니다.

iv. 업데이트 내용을 검토하고 업데이트를 선택합니다.

AWS CLI

AWS CLI 를 사용하여 자체 관리형 프로비저닝을 사용하여 Terraform 환경을 새 마이너 버전으로 업데이트합니다.

1. 다음 명령을 실행하여 환경을 업데이트합니다.

```
$ aws proton update-environment \
    --name "pr-environment" \
    --deployment-type "MINOR_VERSION" \
    --template-major-version "1" \
    --template-minor-version "1" \
    --provisioning-repository "branch=main,name=myrepos/env-
repo,provider=GITHUB" \
    --spec "file://env-spec-mod.yaml"
```

응답:

```
{
    "environment": {
        "arn": "arn:aws:proton:region-id:123456789012:environment/pr-
environment",
        "createdAt": "2021-11-18T21:09:15.745000+00:00",
        "deploymentStatus": "IN_PROGRESS",
        "lastDeploymentAttemptedAt": "2021-11-18T21:25:41.998000+00:00",
        "lastDeploymentSucceededAt": "2021-11-18T21:09:15.745000+00:00",
        "name": "pr-environment",
        "provisioningRepository": {
            "arn": "arn:aws:proton:region-id:123456789012:repository/
github:myrepos/env-repo",
            "branch": "main",
            "name": "myrepos/env-repo",
            "provider": "GITHUB"
        },
        "templateMajorVersion": "1",
        "templateMinorVersion": "0",
        "templateName": "pr-env-template"
    }
}
```

2. 다음 명령을 실행하여 상태를 가져와 확인합니다.

```
$ aws proton get-environment \
    --name "pr-environment"
```

응답:

```
{
    "environment": {
        "arn": "arn:aws:proton:region-id:123456789012:environment/pr-
environment",
        "createdAt": "2021-11-18T21:09:15.745000+00:00",
        "deploymentStatus": "SUCCEEDED",
        "lastDeploymentAttemptedAt": "2021-11-18T21:25:41.998000+00:00",
        "lastDeploymentSucceededAt": "2021-11-18T21:25:41.998000+00:00",
        "name": "pr-environment",
        "provisioningRepository": {
            "arn": "arn:aws:proton:region-id:123456789012:repository/
github:myrepos/env-repo",
            "branch": "main",
            "name": "myrepos/env-repo",
            "provider": "GITHUB"
        },
        "spec": "proton: EnvironmentSpec\nspec:\n   ssm_parameter_value: \"test
\"\n ssm_another_parameter_value: \"update\"\n",
        "templateMajorVersion": "1",
        "templateMinorVersion": "1",
        "templateName": "pr-env-template"
    }
}
```

3. 에서 전송한 풀 요청을 검토합니다 AWS Proton.

   - 요청을 승인하면 프로비저닝이 진행 중인 것입니다.
   - 요청을 거부하면 환경 생성이 취소됩니다.
   - 풀 리퀘스트 제한 시간이 초과되면 환경 생성이 완료되지 않습니다.

4. 프로비저닝 상태를에 제공합니다 AWS Proton.

```
$ aws proton notify-resource-deployment-status-change \
    --resource-arn "arn:aws:proton:region-id:123456789012:environment/pr-
environment" \
    --status "SUCCEEDED"
```

진행 중인 환경 배포 취소

deploymentStatus가 IN_PROGRESS. AWS Proton attempts에 있는 경우 환경 업데이트 배포 취소를 시도하여 배포를 취소할 수 있습니다. 취소 성공은 보장되지 않습니다.

업데이트 배포를 취소하면는 다음 단계에 나열된 대로 배포 취소를 AWS Proton 시도합니다.

AWS관리형 프로비저닝을 사용하면 다음을 AWS Proton 수행합니다.

- 배포 상태를 CANCELLING로 설정합니다.
- IN_PROGRESS가 되면 진행 중인 배포를 중지하고 배포에 의해 생성된 모든 새 리소스를 삭제합니다.
- 배포 상태를 CANCELLED로 설정합니다.
- 리소스 상태를 배포가 시작되기 전의 상태로 되돌립니다.

자체 관리형 프로비저닝을 사용하면에서 다음을 AWS Proton 수행합니다.

- 변경 사항을 리포지토리에 병합하지 못하도록 풀 리퀘스트를 닫으려고 시도합니다.
- 풀 리퀘스트가 성공적으로 종료되면 배포 상태를 CANCELLED로 설정합니다.

환경 배포를 취소하는 방법에 대한 지침은 AWS Proton API 참조의 CancelEnvironmentDeployment 항목을 참조하세요.

콘솔 또는 CLI를 사용하여 진행 중인 환경을 취소할 수 있습니다.

AWS Management Console

콘솔을 사용하여 다음 단계에 따라 환경 업데이트 배포를 취소할 수 있습니다.

1. AWS Proton 콘솔의 왼쪽 탐색 창에서 환경을 선택합니다.
2. 환경 목록에서 취소하려는 배포 업데이트가 있는 환경의 이름을 선택합니다.
3. 업데이트 배포 상태가 진행 중이면 환경 세부 정보 페이지에서 작업을 선택한 다음 배포 취소를 선택합니다.
4. 취소할 것인지 확인하는 모달이 표시됩니다. 배포 취소를 선택합니다.
5. 업데이트 배포 상태가 취소 중으로 설정된 다음 취소가 완료되면 취소됨으로 설정됩니다.

AWS CLI

AWS Proton AWS CLI 를 사용하여 새 마이너 버전 2에 대한 IN_PROGRESS 환경 업데이트 배포를 취소합니다.

이 예제에 사용된 템플릿에는 업데이트 배포가 성공하기 전에 취소가 시작되도록 대기 조건이 포함되어 있습니다.

다음 명령을 실행하여 업데이트를 취소합니다.

```
$ aws proton cancel-environment-deployment \
        --environment-name "MySimpleEnv"
```

응답:

```
{
    "environment": {
        "arn": "arn:aws:proton:region-id:123456789012:environment/MySimpleEnv",
        "createdAt": "2021-04-02T17:29:55.472000+00:00",
        "deploymentStatus": "CANCELLING",
        "lastDeploymentAttemptedAt": "2021-04-02T18:15:10.243000+00:00",
        "lastDeploymentSucceededAt": "2021-04-02T17:48:26.307000+00:00",
        "name": "MySimpleEnv",
        "protonServiceRoleArn": "arn:aws:iam::123456789012:role/service-role/
ProtonServiceRole",
        "spec": "proton: EnvironmentSpec\n\nspec:\n  my_sample_input: hello\n
 my_other_sample_input: everybody\n",
        "templateMajorVersion": "1",
        "templateMinorVersion": "1",
        "templateName": "simple-env"
    }
}
```

다음 명령을 실행하여 상태를 가져와 확인합니다.

```
$ aws proton get-environment \
        --name "MySimpleEnv"
```

응답:

```
{
    "environment": {
```

```
        "arn": "arn:aws:proton:region-id:123456789012:environment/MySimpleEnv",
        "createdAt": "2021-04-02T17:29:55.472000+00:00",
        "deploymentStatus": "CANCELLED",
        "deploymentStatusMessage": "User initiated cancellation.",
        "lastDeploymentAttemptedAt": "2021-04-02T18:15:10.243000+00:00",
        "lastDeploymentSucceededAt": "2021-04-02T17:48:26.307000+00:00",
        "name": "MySimpleEnv",
        "protonServiceRoleArn": "arn:aws:iam::123456789012:role/service-role/
ProtonServiceRole",
        "spec": "proton: EnvironmentSpec\n\nspec:\n  my_sample_input: hello\n
 my_other_sample_input: everybody\n",
        "templateMajorVersion": "1",
        "templateMinorVersion": "1",
        "templateName": "simple-env"
    }
}
```


환경을 삭제합니다.

AWS Proton 콘솔 또는를 사용하여 AWS Proton 환경을 삭제할 수 있습니다 AWS CLI.

> **Note**
>
> 관련 구성 요소가 있는 환경은 삭제할 수 없습니다. 이러한 환경을 삭제하려면 먼저 해당 환경에서 실행 중인 모든 구성 요소를 삭제해야 합니다. 구성 요소에 대한 자세한 내용은 구성 요소를 참조하세요.

AWS Management Console

다음 두 옵션에 설명된 대로 콘솔을 사용하여 환경을 삭제합니다.

환경 목록에서.

1. AWS Proton 콘솔에서 환경을 선택합니다.
2. 환경 목록에서 삭제하려는 환경 왼쪽에 있는 라디오 버튼을 선택합니다.
3. 작업을 선택한 후 삭제를 선택합니다.
4. 삭제 작업을 확인하라는 모달이 표시됩니다.
5. 지침을 따르고 예, 삭제를 선택합니다.

환경 세부 정보 페이지에서.

1. AWS Proton 콘솔에서 환경을 선택합니다.
2. 환경 목록에서 삭제할 환경의 이름을 선택합니다.
3. 환경 세부 정보 페이지에서 작업을 선택한 다음 삭제를 선택합니다.
4. 삭제할 것인지 확인하는 모달이 표시됩니다.
5. 지침을 따르고 예, 삭제를 선택합니다.

AWS CLI

AWS CLI 를 사용하여 환경을 삭제합니다.

서비스 또는 서비스 인스턴스가 환경에 배포된 경우에는 환경을 삭제하지 마십시오.

다음 명령을 실행합니다.

```
$ aws proton delete-environment \
    --name "MySimpleEnv"
```

응답:

```
{
    "environment": {
        "arn": "arn:aws:proton:region-id:123456789012:environment/MySimpleEnv",
        "createdAt": "2021-04-02T17:29:55.472000+00:00",
        "deploymentStatus": "DELETE_IN_PROGRESS",
        "lastDeploymentAttemptedAt": "2021-04-02T17:48:26.307000+00:00",
        "lastDeploymentSucceededAt": "2021-04-02T17:48:26.307000+00:00",
        "name": "MySimpleEnv",
        "protonServiceRoleArn": "arn:aws:iam::123456789012:role/ProtonServiceRole",
        "templateMajorVersion": "1",
        "templateMinorVersion": "1",
        "templateName": "simple-env"
    }
}
```

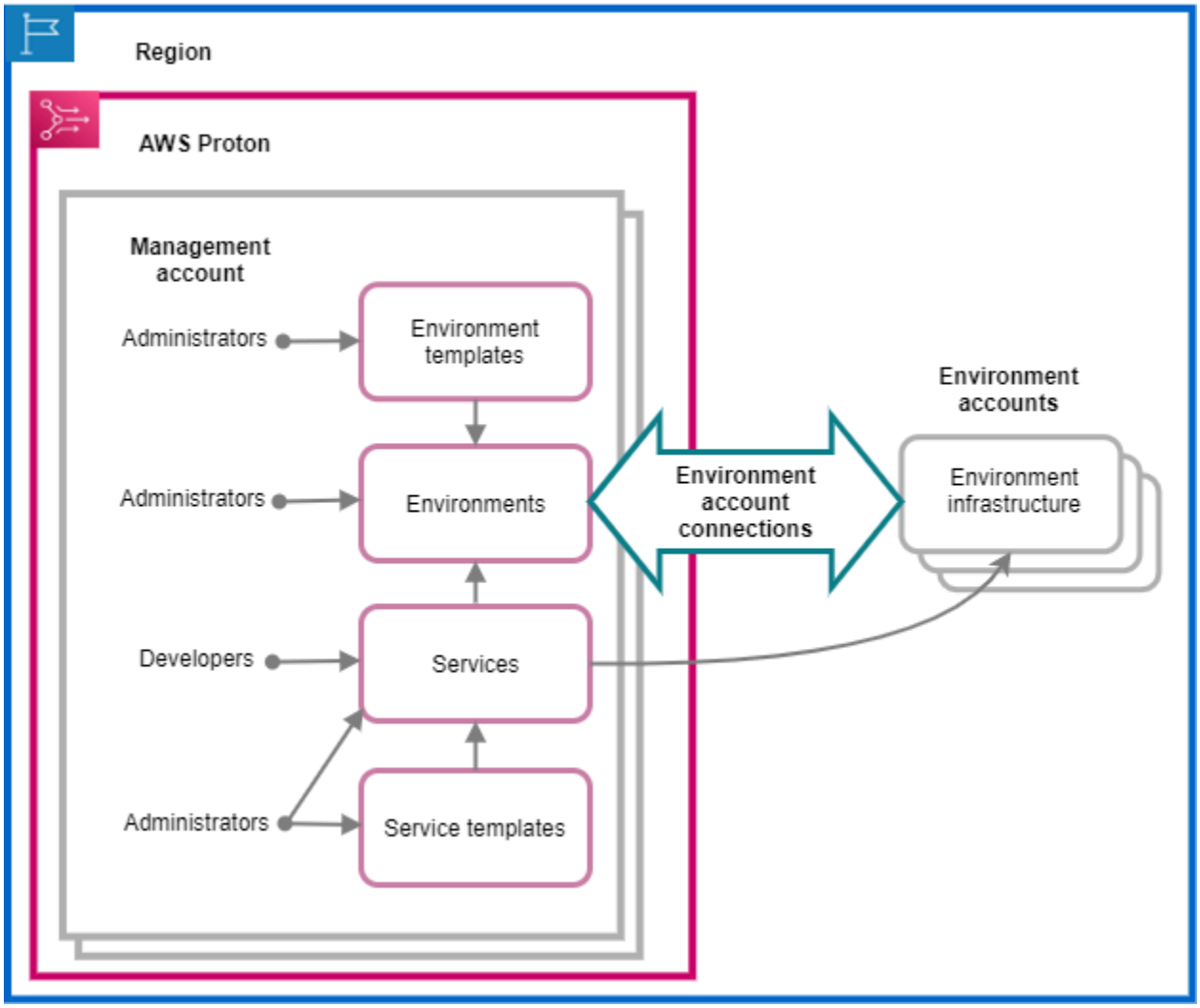
환경 계정 연결

개요

한 계정에서 AWS Proton 환경을 생성 및 관리하고 다른 계정에서 인프라 리소스를 프로비저닝하는 방법을 알아봅니다. 이를 통해 규모에 맞게 가시성과 효율성을 개선할 수 있습니다. 환경 계정 연결은 CloudFormation 인프라를 코드로 사용하는 표준 프로비저닝만 지원합니다.

> **Note**
>
> 이 항목의 정보는 AWS 관리형 프로비저닝으로 구성된 환경과 관련이 있습니다. 자체 관리형 프로비저닝으로 구성된 환경에서는 인프라를 직접 프로비저닝하지 AWS Proton 않습니다. 대신 리포지토리로 풀 리퀘스트(PR)를 보내 프로비저닝합니다. 자동화 코드가 올바른 ID와 역할을 맡도록 하는 것은 사용자의 책임입니다.
> 프로비저닝 방법에 대한 자세한 내용은 the section called “프로비저닝 방법”을 참조하세요.

용어



AWS Proton 환경 계정 연결을 사용하면 한 계정에서 환경을 생성하고 AWS Proton 다른 계정에서 인프라를 프로비저닝할 수 있습니다.

관리 계정

관리자로서 다른 AWS Proton 환경 계정의 인프라 리소스를 프로비저닝하는 환경을 생성하는 단일 계정입니다.

환경 계정

다른 계정에서 AWS Proton 환경을 만들 때 환경 인프라가 프로비저닝되는 계정입니다.

환경 계정 연결

관리 계정과 환경 계정 간의 안전한 양방향 연결. 다음 표에 설명된 대로 이 권한을 부여합니다.

특정 지역의 환경 계정에서 환경 계정 연결을 만들면 같은 지역에 있는 관리 계정만 환경 계정 연결을 보고 사용할 수 있습니다. 즉, 관리 계정에서 생성된 AWS Proton 환경과 환경 계정에서 프로비저닝된 환경 인프라가 동일한 리전에 있어야 합니다.

환경 계정 연결 고려 사항

- 환경 계정에서 프로비저닝하려는 각 환경에 대한 환경 계정 연결이 필요합니다.
- 환경 계정 연결 할당량에 대한 자세한 내용은 AWS Proton 할당량을 참조하세요.

태그 지정

환경 계정에서 콘솔 또는 AWS CLI 를 사용하여 환경 계정 연결 고객 관리형 태그를 보고 관리합니다. AWS 관리형 태그는 환경 계정 연결에 대해 생성되지 않습니다. 자세한 내용은 태그 지정 단원을 참조하십시오.

한 계정에서 환경을 만들고 다른 계정에서 인프라를 프로비저닝합니다.

단일 관리 계정으로 환경을 만들고 프로비전하려면 만들려는 환경에 대한 환경 계정을 설정하세요.

환경 계정에서 시작하여 연결을 생성하세요.

환경 계정에서 환경 인프라 리소스를 프로비저닝하는 데 필요한 권한으로만 범위가 지정된 AWS Proton 서비스 역할을 생성합니다. 자세한 내용은 AWS Proton 를 사용하여 프로비저닝하기 위한 서비스 역할 CloudFormation 단원을 참조하십시오.

그런 다음 환경 계정 연결 요청을 생성하여 관리 계정으로 전송하세요. 요청이 수락되면는 연결된 환경 계정에서 환경 리소스 프로비저닝을 허용하는 연결된 IAM 역할을 사용할 수 AWS Proton 있습니다.

관리 계정에서 환경 계정 연결을 수락하거나 거부합니다.

관리 계정에서 환경 계정 연결 요청을 수락하거나 거부합니다. 관리 계정에서 환경 계정 연결을 삭제할 수 없습니다.

요청을 수락하면는 연결된 환경 계정에서 리소스 프로비저닝을 허용하는 연결된 IAM 역할을 사용할 AWS Proton 수 있습니다.

환경 인프라 리소스는 관련 환경 계정에 프로비저닝됩니다. AWS Proton APIs 관리 계정에서 환경 및 인프라 리소스에 액세스하고 관리하는 데만 사용할 수 있습니다. 자세한 내용은 한 계정에서 환경을 만들고 다른 계정에서 프로비저닝합니다. 및 환경 업데이트 섹션을 참조하세요.

요청을 거부한 후에는 거부된 환경 계정 연결을 수락하거나 사용할 수 없습니다.

> **Note**
>
> 환경에 연결된 환경 계정 연결은 거부할 수 없습니다. 환경 계정 연결을 거부하려면 먼저 관련 환경을 삭제해야 합니다.

환경 계정에서 프로비저닝된 인프라 리소스에 액세스하세요.

환경 계정에서 프로비저닝된 인프라 리소스를 보고 액세스할 수 있습니다. 예를 들어 필요한 경우 CloudFormation API 작업을 사용하여 스택을 모니터링하고 정리할 수 있습니다. AWS Proton API 작업을 사용하여 인프라 리소스를 프로비저닝하는 데 사용된 AWS Proton 환경에 액세스하거나 관리할 수 없습니다.

환경 계정에서 환경 계정에서 만든 환경 계정 연결을 삭제할 수 있습니다. 수락하거나 거부할 수 없습니다. 환경에서 사용 AWS Proton 중인 환경 계정 연결을 삭제하면 환경 계정 및 명명된 환경에 대해 새 환경 연결이 수락될 때까지는 환경 인프라 리소스를 관리할 수 AWS Proton 없습니다. 환경 연결 없이 남아 있는 프로비저닝된 리소스를 정리하는 것은 사용자의 책임입니다.

콘솔 또는 CLI를 사용하여 환경 계정 연결을 관리합니다.

콘솔 또는 CLI를 사용하여 환경 계정 연결을 만들고 관리할 수 있습니다.

AWS Management Console

콘솔을 사용하여 다음 단계에 따라 환경 계정 연결을 만들고 관리 계정으로 요청을 보냅니다.

1. 관리 계정에서 만들려는 환경의 이름을 정하거나 환경 계정 연결이 필요한 기존 환경의 이름을 선택합니다.
2. 환경 계정의 AWS Proton 콘솔의 탐색 창에서 환경 계정 연결을 선택합니다.
3. 환경 계정 연결 페이지에서 연결 요청을 선택합니다.

> **Note**
>
> 환경 계정 연결 페이지 제목에 나열된 계정 ID를 확인합니다. 지정된 환경에 프로비저닝하려는 환경 계정의 계정 ID와 일치하는지 확인하세요.

4. 연결 요청 페이지에서:
   a. 관리 계정에 연결 단원에서 1단계에서 입력한 관리 계정 ID 및 환경 이름을 입력합니다.
   b. 환경 역할 섹션에서 새 서비스 역할을 선택하면가 AWS Proton 자동으로 새 역할을 생성합니다. 또는 기존 서비스 역할과 이전에 만든 서비스 역할의 이름을 선택합니다.

   > **Note**
   >
   > 에서 AWS Proton 자동으로 생성하는 역할에는 광범위한 권한이 있습니다. 환경 인프라 리소스를 프로비저닝하는 데 필요한 권한으로 역할의 범위를 좁히는 것이 좋습니다. 자세한 내용은 AWS Proton 를 사용하여 프로비저닝하기 위한 서비스 역할 CloudFormation을 참조하세요.

   c. (선택 사항) 태그 단원에서 새 태그 추가를 선택하여 환경 계정 연결을 위한 고객 관리 태그를 생성합니다.
   d. 연결 요청을 선택합니다.
5. 관리 계정으로 전송된 환경 연결 테이블에 요청이 보류 중으로 표시되고 관리 계정의 요청을 수락하는 방법을 알려주는 모달이 표시됩니다.

환경 계정 연결 요청을 수락하거나 거부합니다.

1. AWS Proton 콘솔 관리 계정의 탐색 창에서 환경 계정 연결을 선택합니다.

2. 환경 계정 연결 페이지의 환경 계정 연결 요청 테이블에서 수락 또는 거부할 환경 연결 요청을 선택합니다.

> **Note**
>
> 환경 계정 연결 페이지 제목에 나열된 계정 ID를 확인합니다. 거부할 환경 계정 연결과 연결된 관리 계정의 계정 ID와 일치하는지 확인합니다. 이 환경 계정 연결을 거부한 후에는 거부된 환경 계정 연결을 수락하거나 사용할 수 없습니다.

3. 거부 또는 수락을 선택합니다.

- 거부를 선택한 경우 상태가 보류에서 거부로 바뀝니다.
- 수락을 선택한 경우 상태가 보류에서 연결됨으로 바뀝니다.

환경 계정 연결을 삭제합니다.

1. 환경 계정의 AWS Proton 콘솔의 탐색 창에서 환경 계정 연결을 선택합니다.

> **Note**
>
> 환경 계정 연결 페이지 제목에 나열된 계정 ID를 확인합니다. 거부할 환경 계정 연결과 연결된 관리 계정의 계정 ID와 일치하는지 확인합니다. 이 환경 계정 연결을 삭제한 후에는가 환경 계정의 환경 인프라 리소스를 관리할 AWS Proton 수 없습니다. 환경 계정에 대한 새 환경 계정 연결과 명명된 환경이 관리 계정에서 수락된 후에만 관리할 수 있습니다.

2. 환경 계정 연결 페이지의 관리 계정에 연결 요청 전송 단원에서 삭제를 선택합니다.
3. 삭제할 것인지 확인하는 모달이 표시됩니다. 삭제를 선택합니다.

AWS CLI

관리 계정에서 만들려는 환경의 이름을 정하거나 환경 계정 연결이 필요한 기존 환경의 이름을 선택합니다.

환경 계정에서 환경 계정 연결을 생성하세요.

다음 명령을 실행합니다.

```
$ aws proton create-environment-account-connection \
```

```
    --environment-name "simple-env-connected" \
    --role-arn "arn:aws:iam::222222222222:role/service-role/env-account-proton-
service-role" \
    --management-account-id "111111111111"
```

응답:

```
{
    "environmentAccountConnection": {
        "arn": "arn:aws:proton:region-id:222222222222:environment-account-
connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "environmentAccountId": "222222222222",
        "environmentName": "simple-env-connected",
        "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "lastModifiedAt": "2021-04-28T23:13:50.847000+00:00",
        "managementAccountId": "111111111111",
        "requestedAt": "2021-04-28T23:13:50.847000+00:00",
        "roleArn": "arn:aws:iam::222222222222:role/service-role/env-account-proton-
service-role",
        "status": "PENDING"
    }
}
```

다음 명령 및 응답에 표시된 대로 관리 계정의 환경 계정 연결을 수락하거나 거부합니다.

> **Note**
>
> 이 환경 계정 연결을 거부하면 거부된 환경 계정 연결을 수락하거나 사용할 수 없습니다.

거부를 지정하면 상태가 보류 중에서 거부됨으로 바뀝니다.

수락을 지정하면 상태가 보류 중에서 연결됨으로 바뀝니다.

다음 명령을 실행하여 환경 계정 연결을 수락합니다.

```
$ aws proton accept-environment-account-connection \
    --id "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
```

응답:

```
{
    "environmentAccountConnection": {
        "arn": "arn:aws:proton:region-id:222222222222:environment-account-
connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "environmentAccountId": "222222222222",
        "environmentName": "simple-env-connected",
        "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "lastModifiedAt": "2021-04-28T23:15:33.486000+00:00",
        "managementAccountId": "111111111111",
        "requestedAt": "2021-04-28T23:13:50.847000+00:00",
        "roleArn": "arn:aws:iam::222222222222:role/service-role/env-account-proton-
service-role",
        "status": "CONNECTED"
    }
}
```

다음 명령을 실행하여 환경 계정 연결을 거부합니다.

```
$ aws proton reject-environment-account-connection \
    --id "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
```

응답:

```
{
    "environmentAccountConnection": {
        "arn": "arn:aws:proton:us-east-1:222222222222:environment-account-
connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "status": "REJECTED",
        "environmentAccountId": "222222222222",
        "environmentName": "simple-env-reject",
        "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "lastModifiedAt": "2021-04-28T23:13:50.847000+00:00",
        "managementAccountId": "111111111111",
        "requestedAt": "2021-04-28T23:13:50.847000+00:00",
        "roleArn": "arn:aws:iam::222222222222:role/service-role/env-account-proton-
service-role"
    }
}
```

환경 계정의 연결을 봅니다. 환경 계정 연결을 가져오기 하거나 나열하기 할 수 있습니다.

다음 명령 가져오기를 실행합니다.

```
$ aws proton get-environment-account-connection \
    --id "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
```

응답:

```
{
    "environmentAccountConnection": {
        "arn": "arn:aws:proton:region-id:222222222222:environment-account-
connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "environmentAccountId": "222222222222",
        "environmentName": "simple-env-connected",
        "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "lastModifiedAt": "2021-04-28T23:15:33.486000+00:00",
        "managementAccountId": "111111111111",
        "requestedAt": "2021-04-28T23:13:50.847000+00:00",
        "roleArn": "arn:aws:iam::222222222222:role/service-role/env-account-proton-
service-role",
        "status": "CONNECTED"
    }
}
```

환경 계정에서 환경 계정 연결을 삭제합니다.

> **Note**
>
> 이 환경 계정 연결을 삭제하면 환경 계정 및 명명된 환경에 대한 새 환경 연결이 수락될 때까지 환경 계정의 환경 인프라 리소스를 관리할 수 AWS Proton 없습니다. 환경 연결 없이 남아 있는 프로비저닝된 리소스를 정리하는 것은 사용자의 책임입니다.

다음 명령을 실행합니다.

```
$ aws proton delete-environment-account-connection \
    --id "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
```

응답:

```
{
    "environmentAccountConnection": {
```

```
        "arn": "arn:aws:proton:us-east-1:222222222222:environment-account-
connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "environmentAccountId": "222222222222",
        "environmentName": "simple-env-connected",
        "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "lastModifiedAt": "2021-04-28T23:13:50.847000+00:00",
        "managementAccountId": "111111111111",
        "requestedAt": "2021-04-28T23:13:50.847000+00:00",
        "roleArn": "arn:aws:iam::222222222222:role/service-role/env-account-proton-
service-role",
        "status": "CONNECTED"
    }
}
```

고객 관리형 환경

고객 관리형 환경을 사용하면 AWS Proton 환경으로 이미 배포한 VPC와 같은 기존 인프라를 사용할 수 있습니다. 고객 관리형 환경을 사용하는 동안 외부에서 자체 공유 리소스를 프로비저닝할 수 있습니다 AWS Proton. 그러나 AWS Proton 가 AWS Proton 서비스 배포 시 관련 프로비저닝 출력을 입력으로 사용하도록 허용할 수 있습니다. 출력이 변경될 수 있는 경우 AWS Proton 는 업데이트를 수락할 수 있습니다. 그러나 프로비저닝은 외부에서 관리되므로는 환경을 직접 변경할 수 AWS Proton 없습니다 AWS Proton.

환경이 생성된 후에는가 Amazon ECS 클러스터 이름 또는 Amazon VPC ID와 같이 환경을 AWS Proton 만든 경우 AWS Proton 생성된 것과 동일한 출력을에 제공해야 합니다. IDs

이 기능을 사용하면 AWS Proton 서비스 템플릿에서이 환경으로 AWS Proton 서비스 리소스를 배포하고 업데이트할 수 있습니다. 그러나 환경 자체는의 템플릿 업데이트를 통해 수정되지 않습니다 AWS Proton. 환경에 대한 업데이트를 실행하고 해당 출력을 업데이트할 책임은 사용자에게 있습니다 AWS Proton.

단일 계정에 AWS Proton 관리형 환경과 고객 관리형 환경이 혼합된 여러 환경을 보유할 수 있습니다. 두 번째 계정을 연결하고 기본 계정의 AWS Proton 템플릿을 사용하여 연결된 두 번째 계정의 환경 및 서비스에 대한 배포 및 업데이트를 실행할 수도 있습니다.

고객 관리 환경을 사용하는 방법

관리자가 가장 먼저 해야 할 일은 가져온 고객 관리 환경 템플릿을 등록하는 것입니다. 템플릿 번들에 매니페스트나 인프라 파일을 제공하지 마십시오. 스키마만 제공하세요.

아래 스키마는 열린 API 형식을 사용하여 출력 목록을 요약하고 CloudFormation 템플릿에서 출력을 복제합니다.

> **Important**
>
> 출력에는 문자열 입력만 허용됩니다.

다음 예제는 해당 Fargate CloudFormation 템플릿에 대한 템플릿의 출력 섹션 코드 조각입니다.

```
Outputs:
  ClusterName:
    Description: The name of the ECS cluster
    Value: !Ref 'ECSCluster'
  ECSTaskExecutionRole:
    Description: The ARN of the ECS role
    Value: !GetAtt 'ECSTaskExecutionRole.Arn'
  VpcId:
    Description: The ID of the VPC that this stack is deployed in
    Value: !Ref 'VPC'
[...]
```

AWS Proton 가져온 해당 환경의 스키마는 다음과 유사합니다. 스키마에 기본값을 제공하지 마십시오.

```
schema:
  format:
    openapi: "3.0.0"
  environment_input_type: "EnvironmentOutput"
  types:
    EnvironmentOutput:
      type: object
      description: "Outputs of the environment"
      properties:
        ClusterName:
          type: string
          description: "The name of the ECS cluster"
        ECSTaskExecutionRole:
          type: string
          description: "The ARN of the ECS role"
        VpcId:
          type: string
          description: "The ID of the VPC that this stack is deployed in"
```

```
[...]
```

템플릿을 등록할 때이 템플릿을 가져오고 번들에 대한 Amazon S3 버킷 위치를 제공함을 나타냅니다.는 템플릿을 초안에 넣기 전에 스키마에 CloudFormation 템플릿 파라미터만 포함`environment_input_type`되고 템플릿 파라미터는 포함되지 않음을 AWS Proton 확인합니다.

가져온 환경을 만들려면 다음을 제공합니다.

- 배포 시 사용할 IAM 역할.
- 필수 출력 값이 포함된 사양.

콘솔 또는 일반 환경 배포와 유사한 프로세스를 AWS CLI 사용하여 둘 다 제공할 수 있습니다.

CodeBuild 프로비저닝 역할 생성

CloudFormation 및 Terraform과 같은 코드형 인프라(IaaC) 도구에는 다양한 유형의 AWS 리소스에 대한 권한이 필요합니다. 예를 들어, IaaC 템플릿이 S3 버킷을 선언하는 경우 S3 버킷을 생성, 읽기, 업데이트 및 삭제할 수 있는 권한이 필요합니다. 역할을 필요한 최소 권한으로 제한하는 것이 보안 모범 사례로 간주됩니다. AWS 리소스의 범위를 고려할 때 IaaC 템플릿에 대한 최소 권한 정책을 생성하는 것은 어렵습니다. 특히 이러한 템플릿으로 관리되는 리소스가 나중에 변경될 수 있는 경우 더욱 그렇습니다. 예를 들어 관리 중인 템플릿에 대한 최신 편집에서 RDS 데이터베이스 리소스를 AWS Proton추가합니다.

올바른 권한을 구성하면 IaC를 원활하게 배포할 수 있습니다. AWS Proton CodeBuild 프로비저닝은 고객 계정에 있는 CodeBuild 프로젝트에서 고객이 제공한 임의의 CLI 명령을 실행합니다. 일반적으로 이러한 명령은 AWS CDK와 같은 코드형 인프라 (IaaC) 도구를 사용하여 인프라를 생성하고 삭제합니다. 템플릿이 CodeBuild 프로비저닝을 사용하는 AWS 리소스가 배포되면 AWS 는에서 관리하는 CodeBuild 프로젝트에서 빌드를 시작합니다 AWS. CodeBuild에 역할이 전달되며, CodeBuild는 명령을 실행하는 것으로 가정합니다. CodeBuild 프로비저닝 역할이라고 하는 이 역할은 고객이 제공하며 인프라를 프로비저닝하는 데 필요한 권한을 포함합니다. CodeBuild에서만 수임할 수 있으며 심지어는 수임할 AWS Proton 수 없습니다.

역할 만들기

CodeBuild 프로비저닝 역할은 IAM 콘솔 또는 AWS CLI에서 생성할 수 있습니다. AWS CLI에서 생성하려면:

```
aws iam create-role --role-name AWSProtonCodeBuildProvisioning --assume-role-policy-
document '{"Version": "2012-10-17",       "Statement":[{"Effect":"Allow","Principal":
{"Service":"codebuild.amazonaws.com"},"Action":"sts:AssumeRole"}]}'
aws iam attach-role-policy --role-name AWSProtonCodeBuildProvisioning --policy-arn
 arn:aws:iam::aws:policy/AWSProtonCodeBuildProvisioningBasicAccess
```

또한 CodeBuild 서비스에서 빌드를 실행하는 데 필요한 최소한의 권한이 포함된 AWSProtonCodeBuildProvisioningBasicAccess를 첨부합니다.

콘솔을 사용하려는 경우 역할을 생성할 때 다음 사항을 확인합니다.

1. 신뢰할 수 있는 엔터티에서 AWS 서비스를 선택한 다음 CodeBuild를 선택합니다.
2. 권한 추가 단계에서 첨부하려는 AWSProtonCodeBuildProvisioningBasicAccess 및 기타 정책을 선택합니다.

관리자 액세스

AdministratorAccess 정책을 CodeBuild 프로비저닝 역할에 연결하면 권한 부족으로 인해 IaaC 템플릿이 실패하지 않도록 보장됩니다. 또한 환경 템플릿 또는 서비스 템플릿을 생성할 수 있는 사람은 누구나 관리자가 아니더라도 관리자 수준 작업을 수행할 수 있습니다. CodeBuild 프로비저닝 역할과 AdministatorAccess 함께를 사용하는 것은 권장 AWS Proton 하지 않습니다. CodeBuild 프로비저닝 역할과 함께 AdministratorAccess을 사용하기로 결정했다면 샌드박스 환경에서 사용합니다.

IAM 콘솔에서 또는 다음 명령을 실행하여 AdministratorAccess이 있는 역할을 만들 수 있습니다.

```
aws iam create-role --role-name AWSProtonCodeBuildProvisioning --assume-role-policy-
document '{"Version": "2012-10-17",       "Statement":[{"Effect":"Allow","Principal":
{"Service":"codebuild.amazonaws.com"},"Action":"sts:AssumeRole"}]}'
aws iam attach-role-policy --role-name AWSProtonCodeBuildProvisioning --policy-arn
 arn:aws:iam::aws:policy/AdministratorAccess
```

최소 범위 지정 역할 생성

최소 권한으로 역할을 생성하려는 경우 다음과 같은 여러 가지 방법이 있습니다.

- 관리자 권한으로 배포한 다음 역할의 범위를 좁히세요. IAM 액세스 분석기를 사용하는 것이 좋습니다.
- 관리형 정책을 사용하여 사용하려는 서비스에 대한 액세스 권한을 부여하세요.

AWS CDK

와 AWS CDK 함께를 사용하고 각 환경 계정/리전`cdk bootstrap`에서 AWS Proton를 실행한 경우에 대한 역할이 이미 있습니다`cdk deploy`. 이 경우 다음 정책을 CodeBuild 프로비저닝 역할에 더하세요.

```
{
    "Action": "sts:AssumeRole",
    "Resource": [
        "arn:aws:iam::account-id:role/cdk-*-deploy-role-*",
        "arn:aws:iam::account-id:role/cdk-*-file-publishing-role-*"
    ],
    "Effect": "Allow"
}
```

사용자 지정 VPC

사용자 지정 VPC에서 CodeBuild를 실행하기로 결정한 경우 CodeBuild 역할에 다음과 같은 권한이 필요합니다.

```
{
    "Effect": "Allow",
    "Action": [
        "ec2:CreateNetworkInterface"
    ],
    "Resource": [
        "arn:aws:ec2:region:account-id:network-interface/*",
        "arn:aws:ec2:region:account-id:subnet/*",
        "arn:aws:ec2:region:account-id:security-group/*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "ec2:DeleteNetworkInterface"
    ],
    "Resource": [
        "arn:aws:ec2:region:account-id:*/*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
```

```
        "ec2:DescribeDhcpOptions",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeVpcs"
    ],
    "Resource": "*"
},
{
    "Effect": "Allow",
    "Action": [
        "ec2:CreateNetworkInterfacePermission"
    ],
    "Resource": "arn:aws:ec2:region:account-id:network-interface/*",
    "Condition": {
        "StringEquals": {
            "ec2:AuthorizedService": "codebuild.amazonaws.com"
        }
    }
}
```

AmazonEC2FullAccess 관리형 정책을 사용할 수도 있지만 여기에는 필요하지 않을 수 있는 권한이 포함되어 있습니다. CLI를 사용하여 관리되는 정책을 첨부하려면,

```
aws iam create-role --role-name AWSProtonCodeBuildProvisioning --assume-role-policy-
document '{"Version": "2012-10-17",       "Statement":[{"Effect":"Allow","Principal":
{"Service":"codebuild.amazonaws.com"},"Action":"sts:AssumeRole"}]}'
aws iam attach-role-policy --role-name AWSProtonCodeBuildProvisioning --policy-arn
 arn:aws:iam::aws:policy/AdministratorAccess
```

AWS Proton 서비스

AWS Proton 서비스는 일반적으로 여러 서비스 인스턴스와 파이프라인을 포함하는 서비스 템플릿의 인스턴스화입니다. AWS Proton 서비스 인스턴스는 특정 환경의 서비스 템플릿 인스턴스화입니다. 서비스 템플릿은 AWS Proton 서비스의 인프라 및 선택적 서비스 파이프라인에 대한 완전한 정의입니다.

서비스 인스턴스를 배포한 후 CI/CD 파이프라인을 표시하는 소스 코드 푸시 또는 서비스 템플릿의 새 버전으로 서비스를 업데이트하여 인스턴스를 업데이트할 수 있습니다.는 서비스 템플릿의 새 버전을 사용할 수 있게 되면 새 버전으로 서비스를 업데이트할 수 있도록 AWS Proton 프롬프트합니다. 서비스가 업데이트되면가 서비스 및 서비스 인스턴스를 AWS Proton 다시 배포합니다.

이 장에서는 생성, 보기, 업데이트 및 삭제 작업을 사용하여 서비스를 관리하는 방법을 보여줍니다. 자세한 내용은 AWS Proton 서비스 API 참조를 참조하세요.

주제

- 서비스 생성
- 서비스 데이터 보기
- 서비스 편집
- 서비스 삭제
- 서비스 인스턴스 데이터 보기
- 서비스 인스턴스를 업데이트
- 서비스 파이프라인 업데이트

서비스 생성

를 사용하여 애플리케이션을 배포하려면 개발자 AWS Proton는 서비스를 생성하고 다음 입력을 제공합니다.

1. 플랫폼 팀이 게시한 AWS Proton 서비스 템플릿의 이름입니다.
2. 서비스의 이름.
3. 배포하려는 서비스 인스턴스 수입니다.
4. 사용할 환경 선택.
5. 서비스 파이프라인이 포함된 서비스 템플릿을 사용하는 경우 코드 리포지토리에 대한 연결(선택 사항).

서비스에는 무엇이 있나요?

AWS Proton 서비스를 생성할 때 두 가지 유형의 서비스 템플릿 중에서 선택할 수 있습니다.

- 서비스 파이프라인이 포함된 서비스 템플릿(기본값).
- 서비스 파이프라인을 포함하지 않는 서비스 템플릿.

서비스를 생성할 때 서비스 인스턴스 하나 이상 만들어야 합니다.

서비스 인스턴스와 선택적 파이프라인은 서비스와 연결되어 있습니다. 서비스 생성 및 삭제 작업의 컨텍스트 내에서만 파이프라인을 생성하거나 삭제할 수 있습니다. 서비스에 인스턴스 추가 및 제거 방법에 대해 알아보려면 서비스 편집을 참조하세요.

> **Note**
>
> 환경은 환경에서 사용하는 것과 동일한 프로비저닝 방법을 사용하여 환경에서 또는 AWS자체 관리형 프로비저닝. AWS Proton provisions 서비스를 사용하도록 구성됩니다. 개발자가 서비스 인스턴스를 만들거나 업데이트해도 차이는 없으며 두 경우 모두 경험도 동일합니다. 프로비저닝 방법에 대한 자세한 내용은 the section called “프로비저닝 방법”을 참조하세요.

서비스 템플릿

메이저 버전과 마이너 버전의 서비스 템플릿을 모두 사용할 수 있습니다. 콘솔을 사용할 때는 서비스 템플릿의 최신 Recommended 메이저 버전과 마이너 버전을 선택합니다. 를 사용하고 서비스 템플릿의 메이저 버전만 AWS CLI 지정하는 경우 해당 최신 Recommended 마이너 버전을 암시적으로 지정합니다.

다음은 메이저 템플릿 버전과 마이너 템플릿 버전 간의 차이점과 용도에 대한 설명입니다.

- 템플릿의 새 버전은 플랫폼 팀 구성원의 승인을 받는 즉시 Recommended이 됩니다. 즉, 해당 버전을 사용하여 새 서비스가 생성되고 기존 서비스를 새 버전으로 업데이트하라는 메시지가 표시됩니다.
- AWS Proton를 통해 플랫폼 팀은 서비스 인스턴스를 서비스 템플릿의 새 마이너 버전으로 자동으로 업데이트할 수 있습니다. 마이너 버전은 이전 버전과 호환되어야 합니다.
- 메이저 버전에서는 업데이트 프로세스의 일부로 새 입력을 제공해야 하므로 서비스를 해당 서비스 템플릿의 메이저 버전으로 업데이트해야 합니다. 메이저 버전은 이전 버전과 호환되지 않습니다.

서비스 생성

다음 절차에서는 AWS Proton 콘솔 또는를 사용하여 서비스 파이프라인이 있거나 없는 서비스를 AWS CLI 생성하는 방법을 보여줍니다.

AWS Management Console

다음 콘솔 단계에 표시된 대로 서비스를 생성합니다.

1. AWS Proton 콘솔에서 서비스를 선택합니다.
2. 서비스 생성을 선택합니다.
3. 서비스 템플릿 선택 페이지에서 템플릿을 선택하고 구성을 선택합니다.

   활성화된 파이프라인을 사용하지 않으려면 해당 서비스의 파이프라인 제외로 표시된 템플릿을 선택합니다.
4. 서비스 구성 페이지의 서비스 설정 단원에서 서비스 이름을 입력합니다.
5. (선택 사항) 서비스 설명을 입력합니다.
6. 서비스 리포지토리 설정 단원에서,

   a. CodeStar 연결의 경우 목록에서 연결을 선택합니다.

   b. 리포지토리 ID의 경우 목록에서 소스 코드 리포지토리의 이름을 선택합니다.

   c. 브랜치 이름의 경우 목록에서 소스 코드 저장소 브랜치의 이름을 선택합니다.
7. (선택 사항) 태그 단원에서 새 태그 추가 를 선택하고 키와 값을 입력하여 고객 관리형 태그를 생성합니다.
8. 다음을 선택합니다.
9. 사용자 지정 설정 구성 페이지, 서비스 인스턴스 섹션, 새 인스턴스 단원에서. `required` 파라미터 값을 입력해야 합니다. `optional` 파라미터 값을 입력하거나 지정된 경우 기본값을 사용할 수 있습니다.
10. 파이프라인 입력 단원에서 `required` 파라미터 값을 입력해야 합니다. `optional` 파라미터 값을 입력하거나 지정된 경우 기본값을 사용할 수 있습니다.
11. 다음을 선택하고 입력 내용을 검토합니다.
12. 생성(Create)을 선택합니다.

    서비스 세부 정보 및 상태와 서비스의 AWS 관리형 태그 및 고객 관리형 태그를 확인합니다.
13. 탐색 창에서 서비스를 선택합니다.

새 페이지에 서비스 목록과 함께 상태 및 기타 서비스 세부 정보가 표시됩니다.

AWS CLI

를 사용하는 경우 소스 코드 디렉터리에 `.aws-proton/service.yaml`있는 YAML 형식의 spec 파일에서 서비스 입력을 AWS CLI지정합니다.

CLI `get-service-template-minor-version` 명령을 사용하여 스펙 파일에 값을 제공하는 스키마 필수 및 선택적 파라미터를 볼 수 있습니다.

`pipelineProvisioning: "CUSTOMER_MANAGED"`가 있는 서비스 템플릿을 사용하려면 사양에 `pipeline:` 섹션을 포함하지 말고 `create-service` 명령에 `-repository-connection-arn`, `-repository-id` 및 `-branch-name` 파라미터를 포함하지 마십시오.

다음 CLI 단계와 같이 서비스 파이프라인을 사용하여 서비스를 생성합니다.

1. 다음 CLI 예제 명령에 표시된 대로 파이프라인의 서비스 역할을 설정합니다.

   명령:

   ```
   $ aws proton update-account-settings \
           --pipeline-service-role-arn "arn:aws:iam::123456789012:role/AWS
    ProtonServiceRole"
   ```

2. 다음 목록은 서비스 템플릿 스키마를 기반으로 하는 서비스 파이프라인 및 인스턴스 입력을 포함하는 예제 사양을 보여줍니다.

   Spec:

   ```
   proton: ServiceSpec

   pipeline:
     my_sample_pipeline_required_input: "hello"
     my_sample_pipeline_optional_input: "bye"

   instances:
     - name: "acme-network-dev"
       environment: "ENV_NAME"
       spec:
         my_sample_service_instance_required_input: "hi"
         my_sample_service_instance_optional_input: "ho"
   ```

다음 CLI 예제 명령 및 응답에 표시된 대로 파이프라인을 사용하여 서비스를 만듭니다.

명령:

```
$ aws proton create-service \
        --name "MySimpleService" \
        --branch-name "mainline" \
        --template-major-version "1" \
        --template-name "fargate-service" \
        --repository-connection-arn "arn:aws:codestar-connections:region-
id:123456789012:connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111" \
        --repository-id "myorg/myapp" \
        --spec "file://spec.yaml"
```

응답:

```
{
    "service": {
        "arn": "arn:aws:proton:region-id:123456789012:service/MySimpleService",
        "createdAt": "2020-11-18T19:50:27.460000+00:00",
        "lastModifiedAt": "2020-11-18T19:50:27.460000+00:00",
        "name": "MySimpleService",
        "repositoryConnectionArn": "arn:aws:codestar-connections:region-
id:123456789012:connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "repositoryId": "myorg/myapp",
        "status": "CREATE_IN_PROGRESS",
        "templateName": "fargate-service"
    }
}
```

다음 CLI 예제 명령 및 응답에 표시된 대로 서비스 파이프라인을 사용하지 않는 서비스를 만듭니다.

다음은 서비스 파이프라인 입력을 포함하지 않는 예제 사양을 보여줍니다.

Spec:

```
proton: ServiceSpec

instances:
  - name: "acme-network-dev"
    environment: "ENV_NAME"
```

```
    spec:
      my_sample_service_instance_required_input: "hi"
      my_sample_service_instance_optional_input: "ho"
```

프로비저닝된 서비스 파이프라인 없이 서비스를 생성하려면 다음 CLI 예제 명령 및 응답에 표시된 대로 **spec.yaml**에 대한 경로를 제공하고 리포지토리 파라미터는 포함하지 마십시오.

명령:

```
$ aws proton create-service \
        --name "MySimpleServiceNoPipeline" \
        --template-major-version "1" \
        --template-name "fargate-service" \
        --spec "file://spec-no-pipeline.yaml"
```

응답:

```
{
    "service": {
        "arn": "arn:aws:proton:region-id:123456789012:service/
MySimpleServiceNoPipeline",
        "createdAt": "2020-11-18T19:50:27.460000+00:00",
        "lastModifiedAt": "2020-11-18T19:50:27.460000+00:00",
        "name": "MySimpleServiceNoPipeline",
        "status": "CREATE_IN_PROGRESS",
        "templateName": "fargate-service-no-pipeline"
    }
}
```

서비스 데이터 보기

AWS Proton 콘솔 또는를 사용하여 서비스 세부 정보 데이터를 보고 나열할 수 있습니다 AWS CLI.

AWS Management Console

다음 단계에 표시된 대로 AWS Proton 콘솔을 사용하여 서비스 세부 정보를 나열하고 확인합니다.

1. 서비스 목록을 보려면 탐색 창에서 서비스를 선택합니다.
2. 세부 데이터를 보려면 서비스 이름을 선택합니다.

   서비스 세부 데이터 보기

AWS CLI

다음 CLI 예제 명령 및 응답에 표시된 대로 서비스 파이프라인을 통해 서비스의 세부 정보를 볼 수 있습니다.

명령:

```
$ aws proton get-service \
    --name "simple-svc"
```

응답:

```
{
    "service": {
        "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc",
        "branchName": "mainline",
        "createdAt": "2020-11-28T22:40:50.512000+00:00",
        "lastModifiedAt": "2020-11-28T22:44:51.207000+00:00",
        "name": "simple-svc",
        "pipeline": {
            "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc/
pipeline/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
            "createdAt": "2020-11-28T22:40:50.512000+00:00",
            "deploymentStatus": "SUCCEEDED",
            "lastDeploymentAttemptedAt": "2020-11-28T22:40:50.512000+00:00",
            "lastDeploymentSucceededAt": "2020-11-28T22:40:50.512000+00:00",
            "spec": "proton: ServiceSpec\npipeline:\n
 my_sample_pipeline_required_input: hello\n  my_sample_pipeline_optional_input:
 bye\ninstances:\n- name: instance-svc-simple\n  environment: my-simple-
env\n  spec:\n    my_sample_service_instance_required_input: hi\n
 my_sample_service_instance_optional_input: ho\n",
            "templateMajorVersion": "1",
            "templateMinorVersion": "1",
            "templateName": "svc-simple"
        },
        "repositoryConnectionArn": "arn:aws:codestar-connections:region-
id:123456789012:connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
        "repositoryId": "myorg/myapp",
        "spec": "proton: ServiceSpec\npipeline:\n
 my_sample_pipeline_required_input: hello\n  my_sample_pipeline_optional_input:
 bye\ninstances:\n- name: instance-svc-simple\n  environment: my-simple-
env\n  spec:\n    my_sample_service_instance_required_input: hi\n
 my_sample_service_instance_optional_input: ho\n",
```

```
        "status": "ACTIVE",
        "templateName": "svc-simple"
    }
}
```

다음 CLI 예제 명령 및 응답에 표시된 대로 서비스 파이프라인을 사용하지 않고 서비스의 세부 정보를 볼 수 있습니다.

명령:

```
$ aws proton get-service \
    --name "simple-svc-no-pipeline"
```

응답:

```
{
    "service": {
        "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc-without-
pipeline",
        "createdAt": "2020-11-28T22:40:50.512000+00:00",
        "lastModifiedAt": "2020-11-28T22:44:51.207000+00:00",
        "name": "simple-svc-without-pipeline",
        "spec": "proton: ServiceSpec\ninstances:\n- name: instance-svc-simple\n
 environment: my-simple-env\n  spec:\n    my_sample_service_instance_required_input:
 hi\n    my_sample_service_instance_optional_input: ho\n",
        "status": "ACTIVE",
        "templateName": "svc-simple-no-pipeline"
    }
}
```

서비스 편집

AWS Proton 서비스를 다음과 같이 편집할 수 있습니다.

- 서비스 설명을 편집합니다.
- 서비스 인스턴스를 추가 및 제거하여 서비스를 편집합니다.

서비스 설명 편집

콘솔 또는를 사용하여 서비스 설명을 AWS CLI 편집할 수 있습니다.

AWS Management Console

다음 단계에 설명된 대로 콘솔을 사용하여 서비스를 편집합니다.

서비스 목록에서.

1. AWS Proton 콘솔에서 서비스를 선택합니다.
2. 서비스 목록에서 업데이트하려는 서비스 왼쪽에 있는 라디오 버튼을 선택합니다.
3. 편집을 선택합니다.
4. 서비스 구성 페이지에서 양식을 작성하고 다음을 선택합니다.
5. 사용자 지정 설정 구성 페이지에서 다음을 선택합니다.
6. 편집 내용을 검토하고 변경 사항 저장을 선택합니다.

서비스 세부 정보 페이지에서.

1. AWS Proton 콘솔에서 서비스를 선택합니다.
2. 서비스 목록에서 수정하려는 서비스의 이름을 선택합니다.
3. 서비스 세부 정보 페이지에서 편집을 선택합니다.
4. 서비스 구성 페이지에서 양식을 작성하고 다음을 선택합니다.
5. 사용자 지정 설정 페이지에서 양식을 작성하고 다음을 선택합니다.
6. 편집 내용을 검토하고 변경 사항 저장을 선택합니다.

AWS CLI

다음 CLI 예제 명령 및 응답에 표시된 대로 설명을 편집합니다.

명령:

```
$ aws proton update-service \
    --name "MySimpleService" \
    --description "Edit by updating description"
```

응답:

```
{
```

```
    "service": {
        "arn": "arn:aws:proton:region-id:123456789012:service/MySimpleService",
        "branchName": "main",
        "createdAt": "2021-03-12T22:39:42.318000+00:00",
        "description": "Edit by updating description",
        "lastModifiedAt": "2021-03-12T22:44:21.975000+00:00",
        "name": "MySimpleService",
        "repositoryConnectionArn": "arn:aws:codestar-connections:region-
id:123456789012:connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "repositoryId": "my-repository/myorg-myapp",
        "status": "ACTIVE",
        "templateName": "fargate-service"
    }
}
```

서비스를 편집하여 서비스 인스턴스 추가 또는 제거

AWS Proton 서비스의 경우 편집된 사양을 제출하여 서비스 인스턴스를 추가하거나 삭제할 수 있습니다. 요청이 성공하려면 다음 조건을 충족해야 합니다.

- 편집 요청을 제출할 때 서비스 및 파이프라인이 이미 편집 또는 삭제되어 있지 않아야 합니다.
- 편집한 명세서에는 서비스 파이프라인을 수정하는 편집 내용이나 삭제할 수 없는 기존 서비스 인스턴스에 대한 편집 내용이 포함되지 않습니다.
- 사양을 편집해도 연결된 구성 요소가 있는 기존 서비스 인스턴스는 제거되지 않습니다. 이러한 서비스 인스턴스를 삭제하려면 먼저 구성 요소를 업데이트하여 서비스 인스턴스에서 분리해야 합니다. 구성 요소에 대한 자세한 내용은 구성 요소를 참조하세요.

삭제 실패 인스턴스는 DELETE_FAILED 상태의 서비스 인스턴스입니다. 서비스 편집을 요청하면는 편집 프로세스의 일부로 삭제에 실패한 인스턴스를 제거하려고 AWS Proton 시도합니다. 서비스 인스턴스 중 하나라도 삭제에 실패한 경우 콘솔 또는 AWS CLI에 표시되지 않더라도 해당 인스턴스와 연결된 리소스가 남아 있을 수 있습니다. 삭제에 실패한 인스턴스 인프라 리소스를 확인하고 정리하여가 자동으로 제거할 AWS Proton 수 있도록 합니다.

서비스의 서비스 인스턴스 할당량은 AWS Proton 할당량을 참조하세요. 또한 서비스를 생성한 후에는 서비스에 대한 최소 1개의 서비스 인스턴스를 유지 관리해야 합니다. 업데이트 프로세스 중에는 추가하거나 제거할 기존 서비스 인스턴스와 인스턴스 수를 AWS Proton 계산합니다. 삭제에 실패한 인스턴스는 이 수에 포함되며, spec 편집 시 해당 인스턴스를 고려해야 합니다.

콘솔 또는 AWS CLI 를 사용하여 서비스 인스턴스 추가 또는 제거

AWS Management Console

콘솔을 사용하여 서비스 인스턴스를 추가하거나 제거하려면 서비스를 편집합니다.

AWS Proton 콘솔에서

1. 탐색 창에서 서비스를 선택합니다.
2. 편집할 서비스를 선택합니다.
3. 편집을 선택합니다.
4. (선택 사항) 서비스 구성 페이지에서 서비스 이름 또는 설명을 편집한 후 다음을 선택합니다.
5. 사용자 지정 설정 구성 페이지에서 삭제를 선택하여 서비스 인스턴스를 삭제하고 새 인스턴스 추가를 선택하여 서비스 인스턴스를 추가하고 양식을 작성합니다.
6. 다음을 선택합니다.
7. 업데이트를 검토하고 변경 사항 저장을 선택합니다.
8. 서비스 인스턴스 삭제를 확인하라는 모달이 표시됩니다. 지침을 따르고 예, 삭제를 선택합니다.
9. 서비스 세부 정보 페이지에서 서비스의 상태 세부 정보를 확인합니다.

AWS CLI

다음 AWS CLI 예제 명령 및 응답과 **spec** 같이가 편집된 서비스 인스턴스를 추가하고 삭제합니다.

CLI를 사용할 때는 spec이 삭제할 서비스 인스턴스를 제외하고 추가할 서비스 인스턴스와 삭제 대상으로 표시하지 않은 기존 서비스 인스턴스를 모두 포함해야 합니다.

다음 목록은 편집 전 예제 spec와 사양별로 배포된 서비스 인스턴스 목록을 보여줍니다. 이 사양은 이전 예제에서 서비스 설명을 편집하는 데 사용되었습니다.

Spec:

```
proton: ServiceSpec

pipeline:
  my_sample_pipeline_optional_input: "abc"
  my_sample_pipeline_required_input: "123"

instances:
```

```
  - name: "my-instance"
    environment: "simple-env"
    spec:
      my_sample_service_instance_optional_input: "def"
      my_sample_service_instance_required_input: "456"
  - name: "my-other-instance"
    environment: "simple-env"
    spec:
      my_sample_service_instance_required_input: "789"
```

다음 예제 CLI `list-service-instances` 명령 및 응답은 서비스 인스턴스를 추가 또는 삭제하기 전의 활성 인스턴스를 보여줍니다.

명령:

```
$ aws proton list-service-instances \
    --service-name "MySimpleService"
```

응답:

```
{
    "serviceInstances": [
        {
            "arn": "arn:aws:proton:region-id:123456789012:service/MySimpleService/
service-instance/my-other-instance",
            "createdAt": "2021-03-12T22:39:42.318000+00:00",
            "deploymentStatus": "SUCCEEDED",
            "environmentName": "simple-env",
            "lastDeploymentAttemptedAt": "2021-03-12T22:39:43.109000+00:00",
            "lastDeploymentSucceededAt": "2021-03-12T22:39:43.109000+00:00",
            "name": "my-other-instance",
            "serviceName": "example-svc",
            "templateMajorVersion": "1",
            "templateMinorVersion": "0",
            "templateName": "fargate-service"
        },
        {
            "arn": "arn:aws:proton:region-id:123456789012:service/MySimpleService/
service-instance/my-instance",
            "createdAt": "2021-03-12T22:39:42.318000+00:00",
            "deploymentStatus": "SUCCEEDED",
            "environmentName": "simple-env",
            "lastDeploymentAttemptedAt": "2021-03-12T22:39:43.160000+00:00",
```

```
            "lastDeploymentSucceededAt": "2021-03-12T22:39:43.160000+00:00",
            "name": "my-instance",
            "serviceName": "example-svc",
            "serviceTemplateArn": "arn:aws:proton:region-id:123456789012:service-
template/fargate-service",
            "templateMajorVersion": "1",
            "templateMinorVersion": "0",
            "templateName": "fargate-service"
        }
    ]
}
```

다음 목록은 인스턴스를 삭제 및 추가하는 데 사용된 편집 예제 spec를 보여줍니다. 이름이 지정된 기존 인스턴스 my-instance가 제거되고 이름이 지정된 새 인스턴스 yet-another-instance가 추가됩니다.

Spec:

```
proton: ServiceSpec

pipeline:
  my_sample_pipeline_optional_input: "abc"
  my_sample_pipeline_required_input: "123"

instances:
  - name: "my-other-instance"
    environment: "simple-env"
    spec:
      my_sample_service_instance_required_input: "789"
  - name: "yet-another-instance"
    environment: "simple-env"
    spec:
      my_sample_service_instance_required_input: "789"
```

값이 spec 에 있는 경우 "${Proton::CURRENT_VAL}"을 원본 spec에서 보존할 파라미터 값을 지정하는 데 사용할 수 있습니다. 서비스 데이터 보기에 설명된 대로 서비스의 원본 spec을 보는 데 get-service을 사용합니다.

다음 목록은 "${Proton::CURRENT_VAL}"을 사용하여 기존 서비스 인스턴스가 유지되도록 spec이 파라미터 값 변경 사항을 포함하지 않도록 하는 방법을 보여줍니다.

Spec:

```
proton: ServiceSpec

pipeline:
  my_sample_pipeline_optional_input: "${Proton::CURRENT_VAL}"
  my_sample_pipeline_required_input: "${Proton::CURRENT_VAL}"

instances:
  - name: "my-other-instance"
    environment: "simple-env"
    spec:
      my_sample_service_instance_required_input: "${Proton::CURRENT_VAL}"
  - name: "yet-another-instance"
    environment: "simple-env"
    spec:
      my_sample_service_instance_required_input: "789"
```

다음 목록은 서비스를 편집하기 위한 CLI 명령 및 응답을 보여줍니다.

명령:

```
$ aws proton update-service
    --name "MySimpleService" \
    --description "Edit by adding and deleting a service instance" \
    --spec "file://spec.yaml"
```

응답:

```
{
    "service": {
        "arn": "arn:aws:proton:region-id:123456789012:service/MySimpleService",
        "branchName": "main",
        "createdAt": "2021-03-12T22:39:42.318000+00:00",
        "description": "Edit by adding and deleting a service instance",
        "lastModifiedAt": "2021-03-12T22:55:48.169000+00:00",
        "name": "MySimpleService",
        "repositoryConnectionArn": "arn:aws:codestar-connections:region-
id:123456789012:connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "repositoryId": "my-repository/myorg-myapp",
        "status": "UPDATE_IN_PROGRESS",
        "templateName": "fargate-service"
    }
}
```

다음 `list-service-instances` 명령 및 응답은 명명된 기존 인스턴스 `my-instance`가 제거되고 이름이 지정된 새 인스턴스 `yet-another-instance`가 추가되었음을 확인합니다.

명령:

```
$ aws proton list-service-instances \
    --service-name "MySimpleService"
```

응답:

```
{
    "serviceInstances": [
        {
            "arn": "arn:aws:proton:region-id:123456789012:service/MySimpleService/
service-instance/yet-another-instance",
            "createdAt": "2021-03-12T22:39:42.318000+00:00",
            "deploymentStatus": "SUCCEEDED",
            "environmentName": "simple-env",
            "lastDeploymentAttemptedAt": "2021-03-12T22:56:01.565000+00:00",
            "lastDeploymentSucceededAt": "2021-03-12T22:56:01.565000+00:00",
            "name": "yet-another-instance",
            "serviceName": "MySimpleService",
            "templateMajorVersion": "1",
            "templateMinorVersion": "0",
            "templateName": "fargate-service"
        },
        {
            "arn": "arn:aws:proton:region-id:123456789012:service/MySimpleService/
service-instance/my-other-instance",
            "createdAt": "2021-03-12T22:39:42.318000+00:00",
            "deploymentStatus": "SUCCEEDED",
            "environmentName": "simple-env",
            "lastDeploymentAttemptedAt": "2021-03-12T22:39:43.109000+00:00",
            "lastDeploymentSucceededAt": "2021-03-12T22:39:43.109000+00:00",
            "name": "my-other-instance",
            "serviceName": "MySimpleService",
            "templateMajorVersion": "1",
            "templateMinorVersion": "0",
            "templateName": "fargate-service"
        }
    ]
}
```

서비스 인스턴스 추가 또는 제거 시 발생하는 상황

서비스 인스턴스를 삭제하고 추가하기 위해 서비스 편집을 제출하면가 다음 작업을 AWS Proton 수행합니다.

- 서비스를 UPDATE_IN_PROGRESS로 설정합니다.
- 서비스에 파이프라인이 있는 경우 상태를 IN_PROGRESS로 설정하고 파이프라인 작업을 차단합니다.
- 삭제할 모든 서비스 인스턴스를 DELETE_IN_PROGRESS로 설정합니다.
- 서비스 작업을 차단합니다.
- 삭제하도록 표시한 서비스 인스턴스 상의 작업을 차단합니다.
- 새 서비스 인스턴스를 생성합니다.
- 삭제 대상으로 등록한 인스턴스를 삭제합니다.
- 삭제에 실패한 인스턴스를 제거하려고 시도합니다.
- 추가 및 삭제가 완료되면 서비스 파이프라인(있는 경우)을 다시 프로비저닝하고, 서비스를 ACTIVE로 설정하고 서비스 및 파이프라인 작업을 사용 설정합니다.

AWS Proton 는 다음과 같이 실패 모드를 해결하려고 시도합니다.

- 하나 이상의 서비스 인스턴스를 생성하지 못한 경우는 새로 생성된 모든 서비스 인스턴스의 프로비저닝을 해제 AWS Proton 하고 spec를 이전 상태로 되돌립니다. 서비스 인스턴스는 삭제되지 않으며 파이프라인을 어떤 식으로든 수정하지 않습니다.
- 하나 이상의 서비스 인스턴스를 삭제하지 못한 경우는 삭제된 인스턴스 없이 파이프라인을 AWS Proton 다시 프로비저닝합니다. spec은 추가된 인스턴스를 포함하고 삭제 대상으로 표시된 인스턴스는 제외하도록 업데이트되었습니다.
- 파이프라인 프로비저닝에 실패할 경우 롤백은 시도되지 않으며 서비스와 파이프라인 모두 실패한 업데이트 상태를 반영합니다.

태깅 및 서비스 편집

서비스 편집의 일부로 서비스 인스턴스를 추가하면 AWS 관리형 태그가에 전파되고 새 인스턴스 및 프로비저닝된 리소스에 대해 자동으로 생성됩니다. 새 태그를 만들면 해당 태그는 새 인스턴스에만 적용됩니다. 기존 서비스 고객 관리형 태그도 새 인스턴스에 전파됩니다. 자세한 내용은 AWS Proton 리소스 및 태그 지정 단원을 참조하세요.

서비스 삭제

AWS Proton 콘솔 또는를 사용하여 인스턴스 및 파이프라인이 있는 AWS Proton 서비스를 삭제할 수 있습니다 AWS CLI.

구성 요소가 연결된 서비스 인스턴스가 있는 서비스는 삭제할 수 없습니다. 이러한 서비스를 삭제하려면 먼저 연결된 모든 컴포넌트를 업데이트하여 해당 서비스 인스턴스에서 분리해야 합니다. 구성 요소에 대한 자세한 내용은 구성 요소를 참조하세요.

AWS Management Console

다음 단계에 설명된 대로 콘솔을 사용하여 서비스를 삭제합니다.

서비스 세부 정보 페이지에서.

1. AWS Proton 콘솔에서 서비스를 선택합니다.
2. 서비스 목록에서 삭제하려는 서비스의 이름을 선택합니다.
3. 서비스 세부 정보 페이지에서 작업을 선택한 다음 삭제를 선택합니다.
4. 삭제 작업을 확인하라는 모달이 표시됩니다.
5. 지침을 따르고 예, 삭제를 선택합니다.

AWS CLI

다음 CLI 예제 명령 및 응답에 표시된 대로 설명을 삭제합니다.

명령:

```
$ aws proton delete-service \
    --name "simple-svc"
```

응답:

```
{
    "service": {
        "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc",
        "branchName": "mainline",
        "createdAt": "2020-11-28T22:40:50.512000+00:00",
        "description": "Edit by updating description",
        "lastModifiedAt": "2020-11-29T00:30:39.248000+00:00",
        "name": "simple-svc",
```

```
        "repositoryConnectionArn": "arn:aws:codestar-connections:region-
id:123456789012:connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "repositoryId": "myorg/myapp",
        "status": "DELETE_IN_PROGRESS",
        "templateName": "fargate-service"
    }
}
```

서비스 인스턴스 데이터 보기

AWS Proton 서비스 인스턴스 세부 정보 데이터를 보는 방법을 알아봅니다. 콘솔 또는 AWS CLI를 사용할 수 있습니다.

서비스 인스턴스는 서비스에 속합니다. 서비스 편집, 생성 및 삭제 작업의 컨텍스트 내에서만 인스턴스를 생성하거나 삭제할 수 있습니다. 서비스에 인스턴스 추가 및 제거 방법에 대해 알아보려면 서비스 편집을 참조하세요.

AWS Management Console

다음 단계에 표시된 대로 AWS Proton 콘솔을 사용하여 인스턴스 세부 정보를 나열하고 확인합니다.

1. 인스턴스 목록을 보려면 탐색 창에서 서비스 인스턴스를 선택합니다.
2. 세부 데이터를 보려면 서비스 인스턴스 이름을 선택합니다.

   인스턴스 세부 데이터를 봅니다.

AWS CLI

다음 예제 명령 및 응답에 표시된 대로 서비스 인스턴스를 나열 및 조회합니다.

명령:

```
$ aws proton list-service-instances
```

응답:

```
{
    "serviceInstances": [
```

```
        {
            "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc/
service-instance/instance-one",
            "createdAt": "2020-11-28T22:40:50.512000+00:00",
            "deploymentStatus": "SUCCEEDED",
            "environmentArn": "arn:aws:proton:region-id:123456789012:environment/
simple-env",
            "lastDeploymentAttemptedAt": "2020-11-28T22:40:50.512000+00:00",
            "lastDeploymentSucceededAt": "2020-11-28T22:40:50.512000+00:00",
            "name": "instance-one",
            "serviceName": "simple-svc",
            "templateMajorVersion": "1",
            "templateMinorVersion": "0",
            "templateName": "fargate-service"
        }
    ]
}
```

명령:

```
$ aws proton get-service-instance \
    --name "instance-one" \
    --service-name "simple-svc"
```

응답:

```
{
    "serviceInstance": {
        "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc/service-
instance/instance-one",
        "createdAt": "2020-11-28T22:40:50.512000+00:00",
        "deploymentStatus": "SUCCEEDED",
        "environmentName": "simple-env",
        "lastDeploymentAttemptedAt": "2020-11-28T22:40:50.512000+00:00",
        "lastDeploymentSucceededAt": "2020-11-28T22:40:50.512000+00:00",
        "name": "instance-one",
        "serviceName": "simple-svc",
        "spec": "proton: ServiceSpec\npipeline:\n
 my_sample_pipeline_optional_input: hello world\n
 my_sample_pipeline_required_input: pipeline up\ninstances:\n- name: instance-one\n
 environment: my-simple-env\n  spec:\n    my_sample_service_instance_optional_input:
 Ola\n    my_sample_service_instance_required_input: Ciao\n",
        "templateMajorVersion": "1",
```

```
        "templateMinorVersion": "0",
        "templateName": "svc-simple"
    }
}
```

서비스 인스턴스를 업데이트

AWS Proton 서비스 인스턴스를 업데이트하고 업데이트를 취소하는 방법을 알아봅니다.

서비스 인스턴스는 서비스에 속합니다. 서비스 편집, 생성 및 삭제 작업의 컨텍스트 내에서만 인스턴스를 생성하거나 삭제할 수 있습니다. 서비스에 인스턴스 추가 및 제거 방법에 대해 알아보려면 서비스 편집을 참조하세요.

다음 목록에 설명된 대로 서비스 인스턴스를 업데이트하는 데는 네 가지 모드가 있습니다. 를 사용할 때 AWS CLI`deployment-type` 필드는 모드를 정의합니다. 콘솔을 사용할 때 이러한 모드는 서비스 인스턴스 세부 정보 페이지의 작업에서 드롭다운되는 편집 및 최신 마이너 버전 및 최신 메이저 버전으로 업데이트 작업에 매핑됩니다.

`NONE`

이 모드에서는 배포가 이루어지지 않습니다. 요청된 메타데이터 파라미터만 업데이트됩니다.

`CURRENT_VERSION`

이 모드에서는 서비스 인스턴스가 배포되고 사용자가 제공한 새 사양으로 업데이트됩니다. 요청된 파라미터만 업데이트됩니다. 이 `deployment-type`을 사용할 때 마이너 버전 또는 메이저 버전 파라미터를 포함하지 마십시오.

`MINOR_VERSION`

이 모드에서는 서비스 인스턴스가 기본적으로 현재 사용 중인 메이저 버전의 게시된, 권장되는 최신 마이너 버전으로 배포 및 업데이트됩니다. 현재 사용 중인 메이저 버전과 다른 마이너 버전을 지정할 수도 있습니다.

`MAJOR_VERSION`

이 모드에서는 기본적으로 현재 템플릿의 게시된 권장(최신) 메이저 버전 및 마이너 버전으로 서비스 인스턴스가 배포 및 업데이트됩니다. 사용 중인 메이저 버전보다 상위의 다른 메이저 버전과 마이너 버전을 지정할 수도 있습니다(선택 사항).

deploymentStatus가 배포 취소를 시도하는 경우 서비스 인스턴스 업데이트 배포 취소IN_PROGRESS AWS Proton 를 시도할 수 있습니다. 취소 성공은 보장되지 않습니다.

업데이트 배포를 취소하면는 다음 단계에 나열된 대로 배포 취소를 AWS Proton 시도합니다.

- 배포 상태를 CANCELLING로 설정합니다.
- IN_PROGRESS인 경우 진행 중인 배포를 중지하고 배포에 의해 생성된 모든 새 리소스를 삭제합니다.
- 배포 상태를 CANCELLED로 설정합니다.
- 리소스 상태를 배포가 시작되기 전의 상태로 되돌립니다.

서비스 인스턴스 배포를 취소하는 방법에 대한 자세한 내용은 AWS Proton API 참조에서 CancelServiceInstanceDeployment를 참조하세요.

콘솔 또는를 사용하여 업데이트를 AWS CLI 수행하거나 업데이트 배포를 취소합니다.

AWS Management Console

다음 단계에 따라 콘솔을 사용하여 서비스 인스턴스를 업데이트합니다.

1. AWS Proton 콘솔로 돌아가 탐색 창에서 서비스 인스턴스를 선택합니다.
2. 서비스 인스턴스 목록에서 업데이트하려는 서비스 인스턴스의 이름을 선택합니다.
3. 작업을 선택한 다음 업데이트 옵션 중 하나를 선택하고, 사양이나 작업을 업데이트하려면 편집을 선택한 다음 최신 마이너 버전으로 업데이트 또는 최신 메이저 버전으로 업데이트를 선택합니다.
4. 양식을 작성하고 검토 페이지가 표시될 때까지 다음을 선택합니다.
5. 편집 내용을 검토하고 업데이트를 선택합니다.

AWS CLI

CLI 예제 명령 및 응답에 표시된 대로 서비스 인스턴스를 새 마이너 버전으로 업데이트합니다.

서비스 인스턴스를 수정된 spec로 업데이트할 때 "${Proton::CURRENT_VAL}"를 사용하여 원본 spec에서 보존할 파라미터 값을 표시하는 데 사용할 수 있습니다(값이 spec에 존재하는 경우). 서비스 데이터 보기에 설명된 대로 get-service을 사용하여 서비스 인스턴스의 원본 spec를 봅니다.

다음은 spec에서 "${Proton::CURRENT_VAL}"를 사용하는 방법을 나타낸 예시입니다.

Spec:

```
proton: ServiceSpec

pipeline:
  my_sample_pipeline_optional_input: "${Proton::CURRENT_VAL}"
  my_sample_pipeline_required_input: "${Proton::CURRENT_VAL}"

instances:
  - name: "my-instance"
    environment: "simple-env"
    spec:
      my_sample_service_instance_optional_input: "${Proton::CURRENT_VAL}"
      my_sample_service_instance_required_input: "${Proton::CURRENT_VAL}"
  - name: "my-other-instance"
    environment: "simple-env"
    spec:
      my_sample_service_instance_required_input: "789"
```

명령: 업데이트

```
$ aws proton update-service-instance \
    --name "instance-one" \
    --service-name "simple-svc" \
    --spec "file://service-spec.yaml" \
    --template-major-version "1" \
    --template-minor-version "1" \
    --deployment-type "MINOR_VERSION"
```

응답:

```
{
    "serviceInstance": {
        "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc/service-
instance/instance-one",
        "createdAt": "2021-04-02T21:29:59.962000+00:00",
        "deploymentStatus": "IN_PROGRESS",
        "environmentName": "arn:aws:proton:region-id:123456789012:environment/
simple-env",
        "lastDeploymentAttemptedAt": "2021-04-02T21:38:00.823000+00:00",
        "lastDeploymentSucceededAt": "2021-04-02T21:29:59.962000+00:00",
        "name": "instance-one",
```

```
        "serviceName": "simple-svc",
        "templateMajorVersion": "1",
        "templateMinorVersion": "0",
        "templateName": "svc-simple"
    }
}
```

명령: 상태 가져오기 및 확인

```
$ aws proton get-service-instance \
    --name "instance-one" \
    --service-name "simple-svc"
```

응답:

```
{
    "serviceInstance": {
        "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc/service-
instance/instance-one",
        "createdAt": "2021-04-02T21:29:59.962000+00:00",
        "deploymentStatus": "SUCCEEDED",
        "environmentName": "simple-env",
        "lastDeploymentAttemptedAt": "2021-04-02T21:38:00.823000+00:00",
        "lastDeploymentSucceededAt": "2021-04-02T21:38:00.823000+00:00",
        "name": "instance-one",
        "serviceName": "simple-svc",
        "spec": "proton: ServiceSpec\n\npipeline:\n
 my_sample_pipeline_optional_input: \"abc\"\n  my_sample_pipeline_required_input:
 \"123\"\n\ninstances:\n  - name: \"instance-one\"\n    environment: \"simple-
env\"\n    spec:\n      my_sample_service_instance_optional_input: \"def\"\n
      my_sample_service_instance_required_input: \"456\"\n  - name: \"my-
other-instance\"\n    environment: \"kls-simple-env\"\n    spec:\n
 my_sample_service_instance_required_input: \"789\"\n",
        "templateMajorVersion": "1",
        "templateMinorVersion": "1",
        "templateName": "svc-simple"
    }
}
```

AWS Management Console

다음 단계에 표시된 대로 콘솔을 사용하여 서비스 인스턴스 배포를 취소합니다.

1. AWS Proton 콘솔로 돌아가 탐색 창에서 서비스 인스턴스를 선택합니다.
2. 서비스 인스턴스 목록에서 취소하려는 배포 업데이트가 있는 서비스 인스턴스의 이름을 선택합니다.
3. 업데이트 배포 상태가 진행 중이면 서비스 인스턴스 세부 정보 페이지에서 작업을 선택한 다음 배포 취소를 선택합니다.
4. 취소 확인을 요청하는 모달이 표시됩니다. 배포 취소를 선택합니다.
5. 업데이트 배포 상태가 취소 중으로 설정된 다음 취소가 완료되면 취소됨으로 설정됩니다.

AWS CLI

다음 CLI 예제 명령 및 응답에 표시된 대로 새로운 마이너 버전 2로 IN_PROGRESS 서비스 인스턴스 배포 업데이트를 취소합니다.

이 예제에 사용된 템플릿에는 업데이트 배포가 성공하기 전에 취소가 시작되도록 대기 조건이 포함되어 있습니다.

명령: 취소

```
$ aws proton cancel-service-instance-deployment \
    --service-instance-name "instance-one" \
    --service-name "simple-svc"
```

응답:

```
{
    "serviceInstance": {
        "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc/service-
instance/instance-one",
        "createdAt": "2021-04-02T21:29:59.962000+00:00",
        "deploymentStatus": "CANCELLING",
        "environmentName": "simple-env",
        "lastDeploymentAttemptedAt": "2021-04-02T21:45:15.406000+00:00",
        "lastDeploymentSucceededAt": "2021-04-02T21:38:00.823000+00:00",
        "name": "instance-one",
        "serviceName": "simple-svc",
        "spec": "proton: ServiceSpec\npipeline:\n
 my_sample_pipeline_optional_input: abc\n  my_sample_pipeline_required_input:
 '123'\ninstances:\n- name: my-instance\n  environment: MySimpleEnv
\n  spec:\n    my_sample_service_instance_optional_input: def\n
```

```
 my_sample_service_instance_required_input: '456'\n- name: my-other-instance\n
 environment: MySimpleEnv\n  spec:\n    my_sample_service_instance_required_input:
 '789'\n",
        "templateMajorVersion": "1",
        "templateMinorVersion": "1",
        "templateName": "svc-simple"
    }
}
```

명령: 상태 가져오기 및 확인

```
$ aws proton get-service-instance \
    --name "instance-one" \
    --service-name "simple-svc"
```

응답:

```
{
    "serviceInstance": {
        "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc/service-
instance/instance-one",
        "createdAt": "2021-04-02T21:29:59.962000+00:00",
        "deploymentStatus": "CANCELLED",
        "deploymentStatusMessage": "User initiated cancellation.",
        "environmentName": "simple-env",
        "lastDeploymentAttemptedAt": "2021-04-02T21:45:15.406000+00:00",
        "lastDeploymentSucceededAt": "2021-04-02T21:38:00.823000+00:00",
        "name": "instance-one",
        "serviceName": "simple-svc",
        "spec": "proton: ServiceSpec\n\npipeline:\n
 my_sample_pipeline_optional_input: \"abc\"\n  my_sample_pipeline_required_input:
 \"123\"\n\ninstances:\n  - name: \"instance-one\"\n    environment: \"simple-
env\"\n    spec:\n      my_sample_service_instance_optional_input: \"def\"\n
      my_sample_service_instance_required_input: \"456\"\n  - name: \"my-
other-instance\"\n    environment: \"kls-simple-env\"\n    spec:\n
 my_sample_service_instance_required_input: \"789\"\n",
        "templateMajorVersion": "1",
        "templateMinorVersion": "1",
        "templateName": "svc-simple"
    }
}
```

서비스 파이프라인 업데이트

AWS Proton 서비스 파이프라인을 업데이트하고 업데이트를 취소하는 방법을 알아봅니다.

서비스 파이프라인은 서비스에 속합니다. 서비스 생성 및 삭제 작업의 컨텍스트 내에서만 파이프라인을 생성하거나 삭제할 수 있습니다.

다음 목록에 설명된 대로 서비스 파이프라인을 업데이트하는 데는 네 가지 모드가 있습니다. 를 사용할 때 AWS CLI`deployment-type` 필드는 모드를 정의합니다. 콘솔을 사용할 때 이러한 모드는 파이프라인 편집 및 권장 버전으로 업데이트에 매핑됩니다.

`NONE`

이 모드에서는 배포가 이루어지지 않습니다. 요청된 메타데이터 파라미터만 업데이트됩니다.

`CURRENT_VERSION`

이 모드에서는 서비스 파이프라인이 배포되고 사용자가 제공한 새 사양으로 업데이트됩니다. 요청된 파라미터만 업데이트됩니다. 이 `deployment-type`을 사용할 때 마이너 버전 또는 메이저 버전 파라미터를 포함하지 마십시오.

`MINOR_VERSION`

이 모드에서는 서비스 파이프라인이 기본적으로 현재 사용 중인 메이저 버전의 게시된, 권장되는 최신 마이너 버전으로 배포 및 업데이트됩니다. 현재 사용 중인 메이저 버전과 다른 마이너 버전을 지정할 수도 있습니다.

`MAJOR_VERSION`

이 모드에서는 서비스 파이프라인이 기본적으로 현재 템플릿의 게시된 권장되는 최신 메이저 버전 및 마이너 버전으로 배포 및 업데이트됩니다. 사용 중인 메이저 버전보다 상위의 다른 메이저 버전과 마이너 버전을 지정할 수도 있습니다(선택 사항).

`deploymentStatus`가 배포 취소를 시도하는 경우 서비스 파이프라인 업데이트 배포 취소`IN_PROGRESS` AWS Proton 를 시도할 수 있습니다. 취소 성공은 보장되지 않습니다.

업데이트 배포를 취소하면는 다음 단계에 나열된 대로 배포 취소를 AWS Proton 시도합니다.

- 배포 상태를 `CANCELLING`로 설정합니다.

- IN_PROGRESS인 경우 진행 중인 배포를 중지하고 배포에 의해 생성된 모든 새 리소스를 삭제합니다.
- 배포 상태를 CANCELLED로 설정합니다.
- 리소스 상태를 배포가 시작되기 전의 상태로 되돌립니다.

서비스 파이프라인 배포 취소에 대한 자세한 내용은 AWS Proton API 참조의 CancelServicePipelineDeployment를 참조하세요.

콘솔 또는를 사용하여 업데이트를 AWS CLI 수행하거나 업데이트 배포를 취소합니다.

AWS Management Console

다음 단계에 설명된 대로 콘솔을 사용하여 서비스 파이프라인을 편집합니다.

1. AWS Proton 콘솔에서 서비스를 선택합니다.
2. 서비스 목록에서 파이프라인을 업데이트할 서비스의 이름을 선택합니다.
3. 서비스 세부정보 페이지에는 개요 및 파이프라인이라는 두 개의 탭이 있습니다. 파이프라인을 선택합니다.
4. 사양을 업데이트하려면 파이프라인 편집을 선택하고 각 양식을 작성하고 최종 양식을 완료할 때까지 다음을 선택한 다음 파이프라인 업데이트를 선택합니다.

   새 버전으로 업데이트하고 파이프라인 템플릿에서 새 버전을 사용할 수 있음을 나타내는 정보 아이콘이 있는 경우 새 템플릿 버전의 이름을 선택합니다.

   a. 권장 버전으로 업데이트를 선택합니다.
   b. 각 양식을 작성하고 최종 양식을 작성할 때까지 다음 을 선택하고 업데이트 를 선택합니다.

AWS CLI

다음 CLI 예제 명령 및 응답에 표시된 대로 서비스 파이프라인을 새 마이너 버전으로 업데이트합니다.

서비스 파이프라인을 수정된 spec로 업데이트할 때 "${Proton::CURRENT_VAL}"를 사용하여 원본 spec에서 보존할 파라미터 값을 표시하는 데 사용할 수 있습니다(값이 spec에 존재하는 경우). 서비스 데이터 보기에 설명된 대로 서비스 파이프라인의 원본 spec을 보는 데 get-service을 사용합니다.

다음은 spec에서 "${Proton::CURRENT_VAL}"를 사용하는 방법을 나타낸 예시입니다.

Spec:

```
proton: ServiceSpec

pipeline:
  my_sample_pipeline_optional_input: "${Proton::CURRENT_VAL}"
  my_sample_pipeline_required_input: "${Proton::CURRENT_VAL}"

instances:
  - name: "my-instance"
    environment: "simple-env"
    spec:
      my_sample_service_instance_optional_input: "${Proton::CURRENT_VAL}"
      my_sample_service_instance_required_input: "${Proton::CURRENT_VAL}"
  - name: "my-other-instance"
    environment: "simple-env"
    spec:
      my_sample_service_instance_required_input: "789"
```

명령: 업데이트

```
$ aws proton update-service-pipeline \
    --service-name "simple-svc" \
    --spec "file://service-spec.yaml" \
    --template-major-version "1" \
    --template-minor-version "1" \
    --deployment-type "MINOR_VERSION"
```

응답:

```
{
    "pipeline": {
        "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc/pipeline/
a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "createdAt": "2021-04-02T21:29:59.962000+00:00",
        "deploymentStatus": "IN_PROGRESS",
        "lastDeploymentAttemptedAt": "2021-04-02T21:39:28.991000+00:00",
        "lastDeploymentSucceededAt": "2021-04-02T21:29:59.962000+00:00",
        "spec": "proton: ServiceSpec\n\npipeline:\n
 my_sample_pipeline_optional_input: \"abc\"\n  my_sample_pipeline_required_input:
```

```
 \"123\"\n\ninstances:\n  - name: \"my-instance\"\n    environment: \"MySimpleEnv
\"\n    spec:\n      my_sample_service_instance_optional_input: \"def
\"\n      my_sample_service_instance_required_input: \"456\"\n  - name:
 \"my-other-instance\"\n    environment: \"MySimpleEnv\"\n    spec:\n
 my_sample_service_instance_required_input: \"789\"\n",
        "templateMajorVersion": "1",
        "templateMinorVersion": "0",
        "templateName": "svc-simple"
    }
}
```

명령: 상태 가져오기 및 확인

```
$ aws proton get-service \
    --name "simple-svc"
```

응답:

```
{
    "service": {
        "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc",
        "branchName": "main",
        "createdAt": "2021-04-02T21:29:59.962000+00:00",
        "lastModifiedAt": "2021-04-02T21:30:54.364000+00:00",
        "name": "simple-svc",
        "pipeline": {
            "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc/
pipeline",
            "createdAt": "2021-04-02T21:29:59.962000+00:00",
            "deploymentStatus": "SUCCEEDED",
            "lastDeploymentAttemptedAt": "2021-04-02T21:39:28.991000+00:00",
            "lastDeploymentSucceededAt": "2021-04-02T21:39:28.991000+00:00",
            "spec": "proton: ServiceSpec\n\npipeline:\n
 my_sample_pipeline_optional_input: \"abc\"\n  my_sample_pipeline_required_input:
 \"123\"\n\ninstances:\n  - name: \"instance-one\"\n    environment: \"simple-
env\"\n    spec:\n      my_sample_service_instance_optional_input: \"def
\"\n      my_sample_service_instance_required_input: \"456\"\n  - name:
 \"my-other-instance\"\n    environment: \"simple-env\"\n    spec:\n
 my_sample_service_instance_required_input: \"789\"\n",
            "templateMajorVersion": "1",
            "templateMinorVersion": "1",
            "templateName": "svc-simple"
        },
```

```
        "repositoryConnectionArn": "arn:aws:codestar-connections:region-
id:123456789012:connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "repositoryId": "repo-name/myorg-myapp",
        "spec": "proton: ServiceSpec\n\npipeline:\n
 my_sample_pipeline_optional_input: \"abc\"\n  my_sample_pipeline_required_input:
 \"123\"\n\ninstances:\n  - name: \"instance-one\"\n    environment: \"simple-
env\"\n    spec:\n      my_sample_service_instance_optional_input: \"def
\"\n      my_sample_service_instance_required_input: \"456\"\n  - name:
 \"my-other-instance\"\n    environment: \"simple-env\"\n    spec:\n
 my_sample_service_instance_required_input: \"789\"\n",
        "status": "ACTIVE",
        "templateName": "svc-simple"
    }
}
```

AWS Management Console

다음 단계에 표시된 대로 콘솔을 사용하여 서비스 파이프라인 배포를 취소합니다.

1. AWS Proton 콘솔을 열고, 탐색 창에서 서비스를 선택합니다.
2. 서비스 목록에서 취소하려는 배포 업데이트가 있는 파이프라인의 이름을 선택합니다.
3. 서비스 세부 정보 페이지에서 파이프라인 탭을 선택합니다.
4. 업데이트 배포 상태가 진행 중이면 서비스 파이프라인 세부 정보 페이지에서 배포 취소를 선택합니다.
5. 취소 확인을 요청하는 모달이 표시됩니다. 배포 취소를 선택합니다.
6. 업데이트 배포 상태가 취소 중으로 설정된 다음 취소가 완료되면 취소됨으로 설정됩니다.

AWS CLI

다음 CLI 예제 명령 및 응답에 표시된 대로 마이너 버전 2로 IN_PROGRESS 서비스 파이프라인 배포 업데이트를 취소합니다.

이 예제에 사용된 템플릿에는 업데이트 배포가 성공하기 전에 취소가 시작되도록 대기 조건이 포함되어 있습니다.

명령: 취소

```
$ aws proton cancel-service-pipeline-deployment \
    --service-name "simple-svc"
```

응답:

```
{
    "pipeline": {
        "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc/pipeline",
        "createdAt": "2021-04-02T21:29:59.962000+00:00",
        "deploymentStatus": "CANCELLING",
        "lastDeploymentAttemptedAt": "2021-04-02T22:02:45.095000+00:00",
        "lastDeploymentSucceededAt": "2021-04-02T21:39:28.991000+00:00",
        "templateMajorVersion": "1",
        "templateMinorVersion": "1",
        "templateName": "svc-simple"
    }
}
```

명령: 상태 가져오기 및 확인

```
$ aws proton get-service \
    --name "simple-svc"
```

응답:

```
{
    "service": {
        "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc",
        "branchName": "main",
        "createdAt": "2021-04-02T21:29:59.962000+00:00",
        "lastModifiedAt": "2021-04-02T21:30:54.364000+00:00",
        "name": "simple-svc",
        "pipeline": {
            "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc/
pipeline",
            "createdAt": "2021-04-02T21:29:59.962000+00:00",
            "deploymentStatus": "CANCELLED",
            "deploymentStatusMessage": "User initiated cancellation.",
            "lastDeploymentAttemptedAt": "2021-04-02T22:02:45.095000+00:00",
            "lastDeploymentSucceededAt": "2021-04-02T21:39:28.991000+00:00",
            "spec": "proton: ServiceSpec\n\npipeline:\n
 my_sample_pipeline_optional_input: \"abc\"\n  my_sample_pipeline_required_input:
 \"123\"\n\ninstances:\n  - name: \"instance-one\"\n    environment: \"simple-
env\"\n    spec:\n      my_sample_service_instance_optional_input: \"def
\"\n      my_sample_service_instance_required_input: \"456\"\n  - name:
```

```
 \"my-other-instance\"\n    environment: \"simple-env\"\n    spec:\n
 my_sample_service_instance_required_input: \"789\"\n",
            "templateMajorVersion": "1",
            "templateMinorVersion": "1",
            "templateName": "svc-simple"
        },
        "repositoryConnectionArn": "arn:aws:codestar-connections:region-
id:123456789012:connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "repositoryId": "repo-name/myorg-myapp",
        "spec": "proton: ServiceSpec\n\npipeline:\n
 my_sample_pipeline_optional_input: \"abc\"\n  my_sample_pipeline_required_input:
 \"123\"\n\ninstances:\n  - name: \"instance-one\"\n    environment: \"simple-
env\"\n    spec:\n      my_sample_service_instance_optional_input: \"def
\"\n      my_sample_service_instance_required_input: \"456\"\n  - name:
 \"my-other-instance\"\n    environment: \"simple-env\"\n    spec:\n
 my_sample_service_instance_required_input: \"789\"\n",
        "status": "ACTIVE",
        "templateName": "svc-simple"
    }
}
```

AWS Proton 구성 요소

구성 요소는 AWS Proton 리소스의 한 유형입니다. 서비스 템플릿에 유연성을 더합니다. 구성 요소는 플랫폼 팀에 핵심 인프라 패턴을 확장하고 개발자가 애플리케이션 인프라의 여러 측면을 관리할 수 있도록 지원하는 보호 장치를 정의하는 메커니즘을 제공합니다.

에서 AWS Proton 관리자는 개발 팀 및 애플리케이션에서 사용되는 표준 인프라를 정의합니다. 하지만 개발팀은 특정 사용 사례에 맞게 Amazon Simple Queue Service(Amazon SQS) 대기열이나 Amazon DynamoDB 테이블과 같은 추가 리소스를 포함해야 할 수도 있습니다. 이러한 애플리케이션별 리소스는 특히 초기 애플리케이션 개발 과정에서 자주 변경될 수 있습니다. 관리자 작성 템플릿에서 이러한 빈번한 변경 사항을 유지 관리하려면 관리 및 확장이 어려울 수 있습니다. 관리자는 실제 관리자 추가 값 없이 더 많은 템플릿을 유지해야 합니다. 애플리케이션 개발자가 애플리케이션에 대한 템플릿을 작성하도록 하는 대안은 AWS Fargate 작업과 같은 주요 아키텍처 구성 요소를 표준화하는 관리자의 기능을 제거하므로 이상적이지 않습니다. 여기에서 구성 요소가 들어옵니다.

개발자는 구성 요소를 사용하여 관리자가 환경 및 서비스 템플릿에서 정의한 것 이상으로 추가 리소스를 애플리케이션에 추가할 수 있습니다. 그런 다음 개발자는 구성 요소를 서비스 인스턴스에 연결합니다.는 환경 및 서비스 인스턴스에 대한 리소스를 프로비저닝하는 것처럼 구성 요소에 의해 정의된 인프라 리소스를 AWS Proton 프로비저닝합니다.

구성 요소는 완전히 통합된 환경을 위해 서비스 인스턴스 입력을 읽고 서비스 인스턴스에 출력을 제공할 수 있습니다. 예를 들어 구성 요소가 서비스 인스턴스에서 사용할 Amazon Simple Storage Service(S3) 버킷을 추가하는 경우, 구성 요소 템플릿은 환경 및 서비스 인스턴스 이름을 고려하여 버킷의 이름을 지정할 수 있습니다. 가 서비스 템플릿을 AWS Proton 렌더링하여 서비스 인스턴스를 프로비저닝할 때 서비스 인스턴스는 버킷을 참조하여 사용할 수 있습니다.

AWS Proton 현재가 지원하는 구성 요소는 직접 정의된 구성 요소입니다. 구성 요소의 인프라를 정의하는 코드형 인프라(IaC) 파일을 AWS Proton API 또는 콘솔에 직접 전달합니다. 이는 템플릿 번들에서 IaC를 정의하고 번들을 템플릿 리소스로 등록한 다음 템플릿 리소스를 사용하여 환경이나 서비스를 생성하는 환경이나 서비스와는 다릅니다.

> **Note**
>
> 직접 정의된 구성 요소를 통해 개발자는 추가 인프라를 정의하고 프로비저닝할 수 있습니다.는 동일한 AWS Identity and Access Management (IAM) 역할을 사용하여 동일한 환경에서 실행되는 직접 정의된 모든 구성 요소를 AWS Proton 프로비저닝합니다.

관리자는 다음 두 가지 방법으로 개발자가 구성 요소로 수행할 수 있는 작업을 제어할 수 있습니다.

- 지원되는 구성 요소 소스 - 관리자는 서비스 템플릿 버전의 속성을 기반으로 구성 요소를 AWS Proton 서비스 인스턴스에 연결하도록 허용할 수 있습니다. 기본적으로 개발자는 서비스 인스턴스에 구성 요소를 연결할 수 없습니다.

  이 속성에 대한 자세한 내용은 AWS Proton API 참조의 CreateServiceTemplateVersion API 작업의 supportedComponentSources 매개변수를 참조합니다.

  > **Note**
  >
  > 템플릿 동기화를 사용하는 경우는 리포지토리의 서비스 템플릿 번들에 변경 사항을 커밋할 때 서비스 템플릿 버전을 암시적으로 AWS Proton 생성합니다. 이 경우 서비스 템플릿 버전 생성 시 지원되는 구성 요소 소스를 지정하는 대신 각 서비스 템플릿 메이저 버전과 관련된 파일에 이 속성을 지정합니다. 자세한 내용은 the section called “서비스 템플릿 동기화” 단원을 참조하십시오.

- 구성 요소 역할 - 관리자는 환경에 구성 요소 역할을 할당할 수 있습니다. 관리자는 환경에서 직접 정의된 구성 요소에 의해 정의된 인프라를 프로비저닝할 때이 역할을 수 AWS Proton 임합니다. 따라서 구성 요소 역할은 개발자가 환경에서 직접 정의된 구성 요소를 사용하여 추가할 수 있는 인프라의 범위를 좁힙니다. 구성 요소 역할이 없는 경우 개발자는 환경에서 직접 정의된 구성 요소를 만들 수 없습니다.

  구성 요소 역할 할당에 대한 자세한 내용은 AWS Proton API 참조의 CreateEnvironment API 작업의 componentRolearn 매개변수를 참조합니다.

  > **Note**
  >
  > 구성 요소 역할은 자체 관리형 프로비저닝 환경에서 사용되지 않습니다.

주제

- 구성 요소는 다른 AWS Proton 리소스와 어떻게 비교되나요?
- AWS Proton 콘솔의 구성 요소
- AWS Proton API 및의 구성 요소 AWS CLI
- 구성 요소 관련 자주 묻는 질문
- 구성 요소 상태

- 코드 파일로서의 구성 요소 인프라
- 구성 요소 CloudFormation 예제

구성 요소는 다른 AWS Proton 리소스와 어떻게 비교되나요?

여러 가지 면에서 구성 요소는 다른 AWS Proton 리소스와 유사합니다. 인프라는 CloudFormation YAML 또는 Terraform HCL 형식으로 작성된 IaC 템플릿 파일에 정의됩니다.는 AWS관리형 프로비저닝 또는 자체 관리형 프로비저닝을 사용하여 구성 요소 인프라를 프로비저닝할 AWS Proton 수 있습니다. ???

그러나 구성 요소는 다음과 같은 몇 가지 면에서 다른 AWS Proton 리소스와 다릅니다.

- 분리된 상태 — 구성 요소는 서비스 인스턴스에 연결되고 인프라를 확장하도록 설계되었지만 서비스 인스턴스에 연결되지 않는 분리된 상태일 수도 있습니다. 구성 요소 상태에 대한 자세한 내용은 the section called “구성 요소 상태”을 참조합니다.
- 스키마 없음 - 구성 요소에는 템플릿 번들처럼 연결된 스키마가 없습니다. 구성 요소 입력은 서비스에 의해 정의됩니다. 구성 요소는 서비스 인스턴스에 연결될 때 입력을 사용할 수 있습니다.
- 고객 관리형 구성 요소 없음 -는 AWS Proton 항상 구성 요소 인프라를 프로비저닝합니다. 자체 리소스를 가져오는 버전의 구성 요소는 없습니다. 사용자 관리 환경에 대한 자세한 내용은 the section called “생성”을 참조합니다.
- 템플릿 리소스 없음 - 직접 정의된 구성 요소에는 환경 및 서비스 템플릿과 유사한 관련 템플릿 리소스가 없습니다. IaC 템플릿 파일을 구성 요소에 직접 제공합니다. 마찬가지로 구성 요소 인프라를 프로비저닝하기 위한 템플릿 언어 및 렌더링 엔진을 정의하는 매니페스트를 직접 제공합니다. 템플릿 번들 작성과 비슷한 방식으로 템플릿 파일과 매니페스트를 작성합니다. 그러나 직접 정의된 구성 요소를 사용하면 IaC 파일을 특정 위치에 번들로 저장할 필요가 없으며 IaC 파일 AWS Proton 에서 템플릿 리소스를 생성하지 않아도 됩니다.
- CodeBuild 기반 프로비저닝 없음 CodeBuild 기반 프로비저닝이라고 하는 자체 사용자 지정 프로비저닝 스크립트를 사용하여 직접 정의된 구성 요소를 프로비저닝할 수 없습니다. 자세한 내용은 the section called “CodeBuild 프로비저닝” 단원을 참조하십시오.

AWS Proton 콘솔의 구성 요소

AWS Proton 콘솔을 사용하여 AWS Proton 구성 요소를 생성, 업데이트, 확인 및 사용합니다.

다음 콘솔 페이지는 구성 요소와 관련이 있습니다. 최상위 콘솔 페이지로 바로 연결되는 링크가 포함되어 있습니다.

- 구성 요소 - AWS 계정의 구성 요소 목록을 봅니다. 새 구성 요소를 만들고 기존 구성 요소를 업데이트 또는 삭제할 수 있습니다. 목록에서 구성 요소 이름을 선택하면 해당 세부 정보 페이지를 볼 수 있습니다.

  환경 세부 정보 및 서비스 인스턴스 세부 정보 페이지에도 유사한 목록이 있습니다. 이 목록에는 보고 있는 리소스와 관련된 구성 요소만 표시됩니다. 이러한 목록 중 하나에서 구성 요소를 생성하면는 구성 요소 생성 페이지에서 연결된 환경을 AWS Proton 미리 선택합니다.
- 구성 요소 세부 정보 - 구성 요소 세부 정보 페이지를 보려면 구성 요소 목록에서 구성 요소 이름을 선택합니다.

  세부 정보 페이지에서 구성 요소 세부 정보 및 상태를 보고 구성 요소를 업데이트하거나 삭제합니다. 출력(예: 프로비저닝된 리소스 ARNs), 프로비저닝된 CloudFormation 스택 및 할당된 태그 목록을 보고 관리합니다.
- 구성 요소 생성 - 구성 요소를 생성합니다. 구성 요소 이름과 설명을 입력하고, 관련 리소스를 선택하고, 구성 요소 소스 IaC 파일을 지정하고, 태그를 할당합니다.
- 구성 요소 업데이트 - 구성 요소를 업데이트하려면 구성 요소 목록에서 구성 요소를 선택한 다음 작업 메뉴에서 구성 요소 업데이트를 선택합니다. 또는 구성 요소 세부 정보 페이지에서 업데이트를 선택합니다.

  대부분의 구성 요소 세부 정보를 업데이트할 수 있습니다. 코드에서 직접 업데이트할 수 없습니다. 또한 업데이트 성공 후 구성 요소를 재배포할지 여부를 선택할 수 있습니다.
- 환경 구성 - 환경을 만들거나 업데이트할 때 구성 요소 역할을 지정할 수 있습니다. 이 역할은 환경에서 직접 정의된 구성 요소를 실행하는 기능을 제어하고 구성 요소를 프로비저닝할 수 있는 권한을 제공합니다.
- 새 서비스 템플릿 버전 생성 - 서비스 템플릿 버전을 생성할 때 템플릿 버전에 지원되는 구성 요소 소스를 지정할 수 있습니다. 이를 통해 이 템플릿 버전을 기반으로 서비스의 서비스 인스턴스에 구성 요소를 연결하는 기능이 제어됩니다.

AWS Proton API 및의 구성 요소 AWS CLI

AWS Proton API 또는를 사용하여 AWS Proton 구성 요소를 AWS CLI 생성, 업데이트, 확인 및 사용합니다.

다음 API 작업은 AWS Proton 구성 요소 리소스를 직접 관리합니다.

- CreateComponent — AWS Proton 컴포넌트를 생성합니다.
- DeleteComponent — AWS Proton 구성 요소를 삭제합니다.

- GetComponent - 구성 요소에 대한 세부 데이터를 가져옵니다.
- ListComponentOutputs — 구성 요소 코드형 인프라(IaC) 출력 목록을 가져옵니다.
- ListComponentProvisionedResources - 구성 요소에 대해 프로비저닝된 리소스를 세부 정보와 함께 나열합니다.
- ListComponents - 구성 요소를 요약 데이터와 함께 나열합니다. 환경, 서비스 또는 단일 서비스 인스턴스별로 결과 목록을 필터링할 수 있습니다.

다른 AWS Proton 리소스의 다음 API 작업에는 구성 요소와 관련된 몇 가지 기능이 있습니다.

- CreateEnvironment, UpdateEnvironment -이 환경에서 직접 정의된 구성 요소를 프로비저닝할 때 AWS Proton 가 사용하는 IAM 서비스 역할의 Amazon 리소스 이름(ARN)을 지정하는 `componentRoleArn` 데 사용합니다. 이는 직접 정의된 구성 요소가 프로비저닝할 수 있는 인프라의 범위를 결정합니다.
- CreateServiceTemplateVersion - 지원되는 구성 요소 소스를 지정하는 데 `supportedComponentSources`를 사용합니다. 지원되는 소스가 있는 구성 요소를 이 서비스 템플릿 버전을 기반으로 서비스 인스턴스에 연결할 수 있습니다.

구성 요소 관련 자주 묻는 질문

구성 요소의 수명 주기는 어떻게 되나요?

구성 요소는 연결된 상태이거나 분리된 상태일 수 있습니다. 서비스 인스턴스에 연결하여 대부분의 시간 동안 인프라를 개선하도록 설계되었습니다. 분리된 구성 요소는 과도기 상태이므로 제어되고 안전한 방식으로 구성 요소를 삭제하거나 다른 서비스 인스턴스에 연결할 수 있습니다. 자세한 내용은 the section called “구성 요소 상태”을 참조합니다.

연결된 구성 요소를 삭제할 수 없는 이유는 무엇입니까?

해결 방법: 연결된 구성 요소를 삭제하려면 구성 요소를 업데이트하여 서비스 인스턴스에서 분리하고 서비스 인스턴스 안정성을 확인한 다음 구성 요소를 삭제하십시오.

이것이 왜 필요한가요? 연결된 구성 요소는 애플리케이션이 런타임 기능을 수행하는 데 필요한 추가 인프라를 제공합니다. 서비스 인스턴스는 구성 요소 출력을 사용하여 이 인프라의 리소스를 탐지하고 사용할 수 있습니다. 구성 요소를 삭제하여 해당 인프라 리소스를 제거하면 연결된 서비스 인스턴스가 중단될 수 있습니다.

추가된 안전 조치로 AWS Proton 에서는 구성 요소를 삭제하기 전에 구성 요소를 업데이트하고 서비스 인스턴스에서 분리해야 합니다. 그런 다음 서비스 인스턴스의 유효성을 검사하여 제대로 배포되고 작

동하는지 확인할 수 있습니다. 문제가 감지되면 구성 요소를 서비스 인스턴스에 빠르게 다시 연결한 다음 문제를 해결할 수 있습니다. 서비스 인스턴스에 구성 요소에 대한 종속성이 없다고 확신하면 구성 요소를 안전하게 삭제할 수 있습니다.

구성 요소에 연결된 서비스 인스턴스를 직접 변경할 수 없는 이유는 무엇입니까?

해결 방법: 첨부 파일을 변경하려면 구성 요소를 업데이트하여 서비스 인스턴스에서 분리하고 구성 요소 및 서비스 인스턴스 안정성을 확인한 다음 구성 요소를 새 서비스 인스턴스에 연결합니다.

이것이 왜 필요한가요? 구성 요소는 서비스 인스턴스에 연결되도록 설계되었습니다. 구성 요소는 인프라 리소스 이름 지정 및 구성에 서비스 인스턴스 입력을 사용할 수 있습니다. 연결된 서비스 인스턴스를 변경하면 구성 요소가 중단될 수 있습니다 (이전 FAQ, 연결된 구성 요소를 삭제할 수 없는 이유는 무엇입니까? 에서 설명한 것처럼 서비스 인스턴스가 중단될 수 있음). 예를 들어 구성 요소의 IaC 템플릿에 정의된 리소스의 이름이 바뀌거나 심지어 교체될 수도 있습니다.

추가된 안전 조치로 AWS Proton 에서는 구성 요소를 업데이트하고 서비스 인스턴스에서 분리해야 다른 서비스 인스턴스에 연결할 수 있습니다. 그런 다음 구성 요소를 새 서비스 인스턴스에 연결하기 전에 구성 요소와 서비스 인스턴스 모두의 안정성을 검증할 수 있습니다.

구성 요소 상태

AWS Proton 구성 요소는 기본적으로 두 가지 상태일 수 있습니다.

- 연결됨 — 구성 요소가 서비스 인스턴스에 연결되어 있습니다. 서비스 인스턴스의 런타임 기능을 지원하는 인프라를 정의합니다. 구성 요소는 개발자가 정의한 인프라를 사용하여 환경 및 서비스 템플릿에 정의된 인프라를 확장합니다.

  일반적인 구성 요소는 수명 주기의 대부분의 유용한 부분에서 연결된 상태입니다.
- 분리됨 - 구성 요소는 AWS Proton 환경과 연결되며 환경의 서비스 인스턴스에 연결되지 않습니다.

  이는 단일 서비스 인스턴스 이상으로 구성 요소의 수명을 연장하기 위한 전환 상태입니다.

다음 표는 여러 구성 요소 상태를 최상위 수준으로 비교한 것입니다.

	연결됨	분리됨
주의 주요 목적	서비스 인스턴스의 인프라를 확장합니다.	서비스 인스턴스 연결 간에 구성 요소의 인프라를 유지 관리합니다.

	연결됨	분리됨
다음과 연결	서비스 인스턴스 및 환경	환경
주요 특정 속성	• 서비스 이름 • 서비스 인스턴스 이름 • 사양	• 환경 이름
삭제될 수 있음	✗ 아니요	✓ 예
다른 서비스 인스턴스로 업데이트 가능	✗ 아니요	✓ 예
입력을 읽을 수 있음	✓ 예	✗ 아니요

구성 요소의 주요 목적은 서비스 인스턴스에 연결하고 추가 리소스로 인프라를 확장하는 것입니다. 연결된 구성 요소는 사양에 따라 서비스 인스턴스의 입력을 읽을 수 있습니다. 구성 요소를 직접 삭제하거나 다른 서비스 인스턴스에 연결할 수 없습니다. 해당 서비스 인스턴스나 관련 서비스 및 환경도 삭제할 수 없습니다. 이러한 작업을 수행하려면 먼저 구성 요소를 업데이트하여 서비스 인스턴스에서 분리합니다.

단일 서비스 인스턴스의 수명 주기 이후에도 구성 요소의 인프라를 유지 관리하려면 서비스 및 서비스 인스턴스 이름을 제거하여 구성 요소를 업데이트하고 서비스 인스턴스에서 분리합니다. 이 분리 상태는 과도기 상태입니다. 구성 요소에는 입력이 없습니다. 인프라는 프로비저닝된 상태로 유지되며 업데이트할 수 있습니다. 구성 요소가 연결되었을 때 연결된 리소스 (서비스 인스턴스, 서비스) 를 삭제할 수 있습니다. 구성 요소를 삭제하거나 서비스 인스턴스에 다시 연결되도록 업데이트할 수 있습니다.

코드 파일로서의 구성 요소 인프라

구성 요소 코드형 인프라(IaC) 파일은 다른 AWS Proton 리소스와 유사합니다. 여기에서 구성 요소별 몇 가지 세부 정보를 알아보세요. 에 대한 IaC 파일 작성에 대한 자세한 내용은 섹션을 AWS Proton참조하세요템플릿 작성 및 번들.

구성 요소와 함께 매개변수 사용

AWS Proton 파라미터 네임스페이스에는 서비스 IaC 파일이 연결된 구성 요소의 이름과 출력을 가져오기 위해 참조할 수 있는 일부 파라미터가 포함되어 있습니다. 네임스페이스에는 구성 요소 IaC 파일이 참조하여 구성 요소가 연결된 환경, 서비스 및 서비스 인스턴스의 입력, 출력 및 리소스 값을 가져올 수 있는 매개변수도 포함되어 있습니다.

구성 요소에는 자체 입력이 없으며 연결된 서비스 인스턴스에서 입력을 가져옵니다. 구성 요소는 환경 출력을 읽을 수도 있습니다.

구성 요소 및 관련 서비스 IaC 파일의 매개변수 사용에 대한 자세한 내용은 the section called “구성 요소 CloudFormation IaC 파라미터”을 참조합니다. AWS Proton 파라미터에 대한 일반 정보와 파라미터 네임스페이스의 전체 참조는 섹션을 참조하세요the section called “파라미터”.

견고한 IaC 파일 작성

관리자는 서비스 템플릿 버전을 생성할 때 템플릿 버전에서 생성된 서비스 인스턴스에 연결된 구성 요소를 허용할지 여부를 결정할 수 있습니다. AWS Proton API 참조에서 CreateServiceTemplateVersion API 작업의 upportedComponentSources 매개변수를 참조합니다. 그러나 향후 서비스 인스턴스의 경우 인스턴스를 생성하고 구성 요소를 추가할지 여부를 결정하며 구성 요소 (직접 정의된 구성 요소의 경우) 작성자 (구성 요소 IaC) 는 일반적으로 서비스 템플릿을 사용하는 개발자와 다릅니다. 따라서 구성 요소가 서비스 인스턴스에 연결될 것이라고 보장할 수 없습니다. 또한 특정 구성 요소 출력 이름의 존재 여부 또는 이러한 출력 값의 유효성 및 안전성을 보장할 수 없습니다.

AWS Proton 및 Jinja 구문은 다음과 같은 방법으로 이러한 문제를 해결하고 장애 없이 렌더링되는 강력한 서비스 템플릿을 작성하는 데 도움이 됩니다.

- AWS Proton 파라미터 필터 - 구성 요소 출력 속성을 참조할 때 파라미터 값을 검증, 필터링 및 형식 지정하는 한정자인 파라미터 필터를 사용할 수 있습니다. 자세한 정보와 지침은 the section called “CloudFormation 파라미터 필터”을 참조합니다.
- 단일 속성 기본값 - 구성 요소의 단일 리소스 또는 출력 속성을 참조하는 경우 `default` 필터를 사용하여 기본값이 있든 없든 서비스 템플릿 렌더링이 실패하지 않도록 할 수 있습니다. 구성 요소나 참조하는 특정 출력 매개변수가 없는 경우 기본값 (또는 기본값을 지정하지 않은 경우 빈 문자열) 이 대신 렌더링되고 렌더링이 성공합니다. 자세한 내용은 the section called “기본값 제공”을 참조합니다.

  예시:

  - `{{ service_instance.components.default.name | default("") }}`
  - `{{ service_instance.components.default.outputs.my-output | default("17") }}`

> **Note**
>
> 직접 정의된 구성 요소를 지정하는 네임스페이스 `.default` 부분을 참조된 속성이 없을 때 기본값을 제공하는 `default` 필터와 혼동하지 마십시오.

- 전체 개체 참조 – 전체 구성 요소 또는 구성 요소의 출력 컬렉션을 참조하는 경우 AWS Proton 는 빈 개체, `{}`를 반환하므로 서비스 템플릿 렌더링이 실패하지 않습니다. 코드에서 직접 SDK의 보안 인증 정보를 제공할 필요가 없습니다. 빈 개체를 사용할 수 있는 컨텍스트에서 참조하거나 `{{ if .. }}` 조건을 사용하여 빈 개체를 테스트해야 합니다.

  예시:

  - `{{ service_instance.components.default }}`
  - `{{ service_instance.components.default.outputs }}`

구성 요소 CloudFormation 예제

다음은 AWS Proton 직접 정의된 구성 요소의 전체 예제와 AWS Proton 서비스에서 구성 요소를 사용하는 방법입니다. 이 구성 요소는 Amazon Simple Storage Service(S3) 버킷 및 관련 액세스 정책을 프로비저닝합니다. 서비스 인스턴스는 이 버킷을 참조하여 사용할 수 있습니다. 버킷 이름은 환경, 서비스, 서비스 인스턴스 및 구성 요소의 이름을 기반으로 합니다. 즉, 버킷은 특정 서비스 인스턴스를 확장하는 구성 요소 템플릿의 특정 인스턴스와 결합됩니다. 개발자는 이 구성 요소 템플릿을 기반으로 여러 구성 요소를 생성하여 다양한 서비스 인스턴스 및 기능 요구 사항에 맞게 Amazon S3 버킷을 프로비저닝할 수 있습니다.

이 예제에서는 다양한 필수 코드형 CloudFormation 인프라(IaC) 파일을 작성하고 필수 AWS Identity and Access Management (IAM) 역할을 생성하는 방법을 다룹니다. 이 예제에서는 소유자 역할별로 단계를 그룹화합니다.

관리자 단계

개발자가 서비스와 함께 구성 요소를 사용할 수 있도록 하려면

1. 환경에서 실행되는 직접 정의된 구성 요소가 프로비저닝할 수 있는 리소스의 범위를 좁히는 AWS Identity and Access Management (IAM) 역할을 생성합니다.는 나중에이 역할을 AWS Proton 수임하여 환경에서 직접 정의된 구성 요소를 프로비저닝합니다.

   이 예시에서는 다음 정책을 사용합니다.

Example직접 정의된 구성 요소 역할

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudformation:CancelUpdateStack",
        "cloudformation:CreateChangeSet",
        "cloudformation:DeleteChangeSet",
        "cloudformation:DescribeStacks",
        "cloudformation:ContinueUpdateRollback",
        "cloudformation:DetectStackResourceDrift",
        "cloudformation:DescribeStackResourceDrifts",
        "cloudformation:DescribeStackEvents",
        "cloudformation:CreateStack",
        "cloudformation:DeleteStack",
        "cloudformation:UpdateStack",
        "cloudformation:DescribeChangeSet",
        "cloudformation:ExecuteChangeSet",
        "cloudformation:ListChangeSets",
        "cloudformation:ListStackResources"
      ],
      "Resource": "arn:aws:cloudformation:*:123456789012:stack/AWSProton-*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:CreateBucket",
        "s3:DeleteBucket",
        "s3:GetBucket*",
        "iam:CreatePolicy",
        "iam:DeletePolicy",
        "iam:GetPolicy",
        "iam:ListPolicyVersions",
        "iam:DeletePolicyVersion"
      ],
      "Resource": "*",
      "Condition": {
```

```
        "ForAnyValue:StringEquals": {
          "aws:CalledVia": "cloudformation.amazonaws.com"
        }
      }
    }
  ]
}
```

2. 환경을 만들거나 업데이트할 때 이전 단계에서 만든 역할을 입력합니다. AWS Proton 콘솔의 환경 구성 페이지에서 구성 요소 역할을 지정합니다. AWS Proton API 또는를 사용하는 경우 CreateEnvironment 또는 UpdateEnvironment API 작업`componentRoleArn`의를 AWS CLI지정합니다.
3. 서비스 인스턴스에 연결된 직접 정의된 구성 요소를 참조하는 서비스 템플릿을 생성합니다.

   이 예제는 구성 요소가 서비스 인스턴스에 연결되어 있지 않아도 손상되지 않는 강력한 서비스 템플릿을 작성하는 방법을 보여줍니다.

   Example컴포넌트를 사용한 서비스 클라우드포메이션 IaC 파일

```
# service/instance_infrastructure/cloudformation.yaml

Resources:
  TaskDefinition:
    Type: AWS::ECS::TaskDefinition
    Properties:
      TaskRoleArn: !Ref TaskRole
      ContainerDefinitions:
        - Name: '{{service_instance.name}}'
          # ...
          {% if service_instance.components.default.outputs | length > 0 %}
          Environment:
            {{ service_instance.components.default.outputs |
                proton_cfn_ecs_task_definition_formatted_env_vars }}
          {% endif %}

  # ...

  TaskRole:
    Type: AWS::IAM::Role
    Properties:
      # ...
      ManagedPolicyArns:
```

```
        - !Ref BaseTaskRoleManagedPolicy
        {{ service_instance.components.default.outputs
            | proton_cfn_iam_policy_arns }}

  # Basic permissions for the task
  BaseTaskRoleManagedPolicy:
    Type: AWS::IAM::ManagedPolicy
    Properties:
      # ...
```

4. 직접 정의된 구성 요소를 지원되는 것으로 선언하는 새 서비스 템플릿 마이너 버전을 생성합니다.

   - Amazon S3의 템플릿 번들 - AWS Proton 콘솔에서 서비스 템플릿 버전을 생성할 때 지원되는 구성 요소 소스에서 직접 정의를 선택합니다. AWS Proton API 또는를 사용하는 경우 CreateServiceTemplateVersion 또는 UpdateServiceTemplateVersion API 작업의 `supportedComponentSources` 파라미터`DIRECTLY_DEFINED`에를 AWS CLI지정합니다.
   - 템플릿 동기화 - 메이저 버전 디렉터리의 `.template-registration.yaml` 파일에 `supported_component_sources:` 항목으로 `DIRECTLY_DEFINED`를 지정한 서비스 템플릿 번들 리포지토리에 변경 내용을 커밋합니다. 이 파일에 대한 자세한 내용은 the section called "서비스 템플릿 동기화"을 참조합니다.

5. 서비스 템플릿 마이너 버전을 가져올 수 있는 권한을 부여합니다. 자세한 내용은 the section called "게시"을 참조합니다.
6. 이 서비스 템플릿을 사용하는 개발자의 IAM 역할에서 `proton:CreateComponent`를 허용해야 합니다.

개발자 단계

서비스 인스턴스와 함께 직접 정의된 구성 요소를 사용하려면

1. 관리자가 구성 요소 지원을 받아 만든 서비스 템플릿 버전을 사용하는 서비스를 생성합니다. 또는 기존 서비스 인스턴스 중 하나를 업데이트하여 최신 템플릿 버전을 사용할 수도 있습니다.
2. Amazon S3 버킷 및 관련 액세스 정책을 프로비저닝하고 이러한 리소스를 출력으로 노출하는 구성 요소 IaC 템플릿 파일을 작성하십시오.

   Example컴포넌트 CloudFormation IAC 파일

```
# cloudformation.yaml

# A component that defines an S3 bucket and a policy for accessing the bucket.
```

```
Resources:
  S3Bucket:
    Type: AWS::S3::Bucket
    Properties:
      BucketName: '{{environment.name}}-{{service.name}}-{{service_instance.name}}-
{{component.name}}'
  S3BucketAccessPolicy:
    Type: AWS::IAM::ManagedPolicy
    Properties:
      PolicyDocument:
        Version: "2012-10-17"
        Statement:
          - Effect: Allow
            Action:
              - 's3:Get*'
              - 's3:List*'
              - 's3:PutObject'
            Resource: !GetAtt S3Bucket.Arn
Outputs:
  BucketName:
    Description: "Bucket to access"
    Value: !GetAtt S3Bucket.Arn
  BucketAccessPolicyArn:
    Value: !Ref S3BucketAccessPolicy
```

3. AWS Proton API 또는를 사용하는 경우 구성 요소에 대한 매니페스트 파일을 AWS CLI작성합니다.

   Example직접 정의된 구성 요소 매니페스트

   ```
   infrastructure:
     templates:
       - file: "cloudformation.yaml"
         rendering_engine: jinja
         template_language: cloudformation
   ```

4. 직접 정의된 구성 요소를 생성합니다. 관리자가 구성 요소를 프로비저닝하기 위해 정의한 구성 요소 역할을 AWS Proton 가정합니다.

   AWS Proton 콘솔의 구성 요소 페이지에서 구성 요소 생성을 선택합니다. 구성 요소 설정의 경우 구성 요소 이름과 선택적 구성 요소 설명을 입력합니다. 구성 요소 연결의 경우 서비스 인스턴스에 구성 요소 연결을 선택합니다. 환경, 서비스, 서비스 인스턴스를 선택합니다. 구성 요소 소스에서 원하는 CloudFormation항목을 선택한 다음 구성 요소 IaC 파일을 선택합니다.

> **Note**
>
> 코드에서 직접 SDK의 매니페스트를 제공할 필요가 없습니다.

AWS Proton API 또는를 사용하는 경우 CreateComponent API 작업을 AWS CLI사용합니다. 구성 요소를 `name` 설정하고 선택 항목 `description`을 설정합니다. `environmentName`, `serviceName` 및 `serviceInstanceName`를 설정합니다. `templateSource` 및 `manifest`를 생성한 파일의 경로로 설정합니다.

> **Note**
>
> 서비스 및 서비스 인스턴스 이름을 지정할 때 환경 이름을 지정하는 것은 선택 사항입니다. 이 두 가지의 조합은 AWS 계정에서 고유하며 서비스 인스턴스에서 환경을 확인할 AWS Proton 수 있습니다.

5. 서비스 인스턴스를 업데이트하여 재배포합니다.는 렌더링된 서비스 인스턴스 템플릿에서 구성 요소의 출력을 AWS Proton 사용하여 애플리케이션이 구성 요소가 프로비저닝한 Amazon S3 버킷을 사용할 수 있도록 합니다.

에서 git 리포지토리 사용 AWS Proton

AWS Proton 는 다양한 용도로 git 리포지토리를 사용합니다. 다음 목록은 AWS Proton 리소스와 연결된 리포지토리 유형을 분류합니다. 리포지토리에 반복적으로 연결하여 콘텐츠를 푸시하거나 리포지토리에서 콘텐츠를 가져오는 AWS Proton 기능의 경우 AWS 계정 AWS Proton 에서에 리포지토리 링크를 등록해야 합니다. 리포지토리 링크는 리포지토리에 연결할 때 사용할 AWS Proton 수 있는 속성 세트입니다.는 AWS Proton 현재 GitHub, GitHub Enterprise 및 BitBucket을 지원합니다.

개발자 리포지토리

코드 리포지토리 — 개발자가 애플리케이션 코드를 저장하는 데 사용하는 리포지토리. 코드 배포에 사용됩니다. AWS Proton 는이 리포지토리와 직접 상호 작용하지 않습니다. 개발자는 파이프라인이 포함된 서비스를 프로비저닝할 때 애플리케이션 코드를 읽을 수 있는 리포지토리 이름과 브랜치를 제공합니다. AWS Proton 이 정보를 해당 파이프라인이 제공하는 파이프라인에 전달합니다.

자세한 내용은 the section called “생성”을 참조하세요.

관리자 리포지토리

템플릿 리포지토리 - 관리자가 AWS Proton 템플릿 번들을 저장하는 리포지토리입니다. 템플릿 동기화에 사용됨. 관리자가에서 템플릿을 생성하면 템플릿 리포지토리를 가리키고 새 템플릿을 템플릿과 동기화된 상태로 AWS Proton 유지할 AWS Proton수 있습니다. 관리자가 리포지토리의 템플릿 번들을 업데이트하면 AWS Proton 가 자동으로 새 템플릿 버전을 생성합니다. 동기화에 사용하기 AWS Proton 전에 템플릿 리포지토리를에 연결합니다.

자세한 내용은 the section called “템플릿 동기화 구성” 단원을 참조하십시오.

> **Note**
>
> Amazon Simple Storage Service(Amazon S3)에 템플릿을 계속 업로드하고 AWS Proton 템플릿 관리 APIs하여 새 템플릿 또는 템플릿 버전을 생성하는 경우 템플릿 리포지토리가 필요하지 않습니다.

자체 관리형 프로비저닝 리포지토리

인프라 리포지토리 - 렌더링된 인프라 템플릿을 호스팅하는 리포지토리. 리소스 인프라의 자체 관리형 프로비저닝에 사용됨. 관리자가 자체 관리형 프로비저닝을 위한 환경을 생성하면 리포지토리를 제공합니다.는이 리포지토리에 풀 요청(PRs)을 AWS Proton 제출하여 환경에 대한 인프라와 환

경에 배포된 모든 서비스 인스턴스를 생성합니다. 자체 관리 인프라 프로비저닝에 사용하기 전에 인프라 리포지토리를 AWS Proton 에 연결합니다.

파이프라인 리포지토리 - 파이프라인을 생성하는 데 사용되는 리포지토리. 파이프라인의 자체 관리형 프로비저닝에 사용됨. 추가 리포지토리를 사용하여 파이프라인을 프로비저닝하면 AWS Proton 가 파이프라인 구성을 개별 환경 또는 서비스와 독립적으로 저장할 수 있습니다. 모든 자체 관리형 프로비저닝 서비스를 위한 단일 파이프라인 리포지토리만 제공하면 됩니다. 자체 관리형 파이프라인 프로비저닝에 사용하기 AWS Proton 전에 파이프라인 리포지토리를에 연결합니다.

자세한 내용은 the section called “AWS관리형 프로비저닝” 단원을 참조하십시오.

주제

리포지토리로 연결되는 링크를 생성합니다.

콘솔 또는 CLI를 사용하여 리포지토리에 대한 링크를 만들 수 있습니다. 리포지토리 링크를 생성하면 가 서비스 연결 역할을 AWS Proton 생성합니다.

AWS Management Console

다음 콘솔 단계에 표시된 대로 리포지토리에 대한 링크를 생성합니다.

1. AWS Proton 콘솔에서 리포지토리를 선택합니다.
2. 리포지토리 생성을 선택합니다.
3. 새 리포지토리 연결 페이지의 리포지토리 세부 정보 단원에서,
   a. 리포지토리 제공자를 선택합니다.
   b. 기존 연결 중 하나를 선택합니다. 연결이 없는 경우 새 CodeStar 연결 추가를 선택하여 연결을 생성한 다음 AWS Proton 콘솔로 돌아가서 연결 목록을 새로 고치고 새 연결을 선택합니다.
   c. 연결된 소스 코드 리포지토리에서 선택합니다.
4. [선택 사항] 태그 단원에서 새 태그 추가를 한 번 이상 선택하고 키 및 값 쌍을 입력합니다.
5. 리포지토리 생성을 선택합니다.

6. 연결된 리포지토리의 세부 데이터를 조회합니다.

AWS CLI

리포지토리로 연결되는 링크를 생성 및 등록합니다.

다음 명령을 실행합니다.

```
$ aws proton create-repository \
    --name myrepos/environments \
    --connection-arn "arn:aws:codestar-connections:region-
id:123456789012:connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111" \
    --provider "GITHUB" \
    --encryption-key "arn:aws:kms:region-id:123456789012:key/bPxRfiCYEXAMPLEKEY" \
    --tags key=mytag1,value=value1 key=mytag2,value=value2
```

마지막 두 파라미터 --encryption-key, --tags는 선택사항입니다.

응답:

```
{
    "repository": {
        "arn": "arn:aws:proton:region-id:123456789012:repository/github:myrepos/
environments",
        "connectionArn": "arn:aws:codestar-connections:region-
id:123456789012:connection/2ad03b28-a7c4-EXAMPLE11111",
        "encryptionKey": "arn:aws:kms:region-id:123456789012:key/
bPxRfiCYEXAMPLEKEY",
        "name": "myrepos/environments",
        "provider": "GITHUB"
    }
}
```

리포지토리 링크를 생성한 후 다음 예제 명령과 같이 AWS 및 고객 관리형 태그 목록을 볼 수 있습니다. AWS Proton 는 AWS 관리형 태그를 자동으로 생성합니다. AWS CLI를 사용하여 고객 관리 태그를 수정하고 생성할 수도 있습니다. 자세한 내용은 AWS Proton 리소스 및 태그 지정 단원을 참조하십시오.

명령:

```
$ aws proton list-tags-for-resource \
```

```
    --resource-arn "arn:aws:proton:region-id:123456789012:repository/github:myrepos/
environments"
```

연결된 리포지토리 데이터 보기

콘솔 또는 AWS CLI를 사용하여 연결된 리포지토리 세부 정보를 삭제할 수 있습니다. git 리포지토리를 동기화하는 데 사용되는 리포지토리 링크의 경우 AWS Proton를 사용하여 리포지토리 동기화 정의 및 상태를 검색할 수 있습니다 AWS CLI.

AWS Management Console

AWS Proton 콘솔을 사용하여 연결된 리포지토리의 세부 정보를 나열하고 볼 수 있습니다.

1. 연결된 리포지토리를 나열하려면 탐색 창에서 리포지토리를 선택합니다.
2. 세부 데이터를 보려면 리포지토리 이름을 선택합니다.

AWS CLI

연결된 리포지토리를 나열합니다.

다음 명령을 실행합니다.

```
$ aws proton list-repositories
```

응답:

```
{
    "repositories": [
        {
            "arn": "arn:aws:proton:region-id:123456789012:repository/github:myrepos/
templates",
            "name": "myrepos/templates",
            "provider": "GITHUB"
        },
        {
            "arn": "arn:aws:proton:region-id:123456789012:repository/github:myrepos/
environments",
            "name": "myrepos/environments",
            "provider": "GITHUB"
        }
```

```
    ]
}
```

연결된 리포지토리의 세부 정보를 봅니다.

다음 명령을 실행합니다.

```
$ aws proton get-repository \
    --name myrepos/templates \
    --provider "GITHUB"
```

응답:

```
{
    "repository": {
        "arn": "arn:aws:proton:region-id:123456789012:repository/github:myrepos/
templates",
        "name": "myrepos/templates",
        "provider": "GITHUB"
    }
}
```

동기화된 리포지토리를 나열합니다.

다음 예제는 템플릿 동기화를 위해 구성한 리포지토리를 나열합니다.

다음 명령을 실행합니다.

```
$ aws proton list-repository-sync-definitions \
    --branch "main" \
    --repository-name myrepos/templates \
    --repository-provider "GITHUB" \
    --sync-type "TEMPLATE_SYNC"
```

리포지토리 동기화 상태를 봅니다.

다음 예는 템플릿 동기화 저장소의 동기화 상태를 검색하는 예제입니다.

다음 명령을 실행합니다.

```
$ aws proton get-repository-sync-status \
```

```
    --branch "main" \
    --repository-name myrepos/templates \
    --repository-provider "GITHUB" \
    --sync-type "TEMPLATE_SYNC"
```

응답:

```
{
    "latestSync": {
        "events": [
            {
                "event": "Clone started",
                "time": "2021-11-21T00:26:35.883000+00:00",
                "type": "CLONE_STARTED"
            },
            {
                "event": "Updated configuration",
                "time": "2021-11-21T00:26:41.894000+00:00",
                "type": "CONFIG_UPDATED"
            },
            {
                "event": "Starting syncs for commit 62c03ff86eEXAMPLE1111111",
                "externalId": "62c03ff86eEXAMPLE1111111",
                "time": "2021-11-21T00:26:44.861000+00:00",
                "type": "STARTING_SYNC"
            }
        ],
        "startedAt": "2021-11-21T00:26:29.728000+00:00",
        "status": "SUCCEEDED"
    }
}
```

리포지토리 링크 삭제

콘솔 또는 AWS CLI를 사용하여 리포지토리 링크를 삭제할 수 있습니다.

> **Note**
>
> 리포지토리 링크를 삭제하면 계정에 AWS Proton 있는 등록된 링크만 제거됩니다 AWS . 리포지토리에서 어떤 정보도 삭제하지 않습니다.

AWS Management Console

콘솔을 사용하여 리포지토리 링크를 삭제합니다.

리포지토리 세부 정보 페이지에서.

1. AWS Proton 콘솔에서 리포지토리를 선택합니다.
2. 리포지토리 목록에서 삭제하려는 리포지토리 왼쪽에 있는 라디오 버튼을 선택합니다.
3. 삭제를 선택합니다.
4. 삭제 작업을 확인하라는 모달이 표시됩니다.
5. 지침을 따르고 예, 삭제를 선택합니다.

AWS CLI

리포지토리 링크를 삭제합니다.

다음 명령을 실행합니다.

```
$ aws proton delete-repository \
    --name myrepos/templates \
    --provider"GITHUB"
```

응답:

```
{
    "repository": {
        "arn": "arn:aws:proton:region-id:123456789012:repository/github:myrepos/
templates",
        "name": "myrepos/templates",
        "provider": "GITHUB"
    }
}
```

모니터링 AWS Proton

모니터링은 AWS Proton 및 AWS 솔루션의 안정성, 가용성 및 성능을 유지하는 데 중요한 부분입니다. 다음 섹션에서는 사용할 수 있는 모니터링 도구에 대해 설명합니다 AWS Proton.

EventBridge AWS Proton 로 자동화

Amazon EventBridge에서 AWS Proton 이벤트를 모니터링할 수 있습니다. EventBridge는 자체 애플리케이션, software-as-a-service(SaaS) 애플리케이션 및에서 실시간 데이터 스트림을 제공합니다 AWS 서비스. AWS 리소스 상태 변경에 응답하도록 이벤트를 구성할 수 있습니다. EventBridge는이 데이터를 AWS Lambda 및 Amazon Simple Notification Service와 같은 대상 서비스로 라우팅합니다. 이러한 이벤트는 CloudWatch Events에 나타나는 이벤트와 동일합니다. CloudWatch Events는 AWS 리소스의 변경 사항을 설명하는 시스템 이벤트의 스트림을 거의 실시간으로 제공합니다. 자세한 내용은 EventBridge User Guide(EventBridge 사용 설명서)의 What Is EventBridge?(EventBridge란?)를 참조하세요.

EventBridge를 사용하여 AWS Proton 프로비저닝 워크플로의 상태 변경에 대한 알림을 받습니다.

이벤트 유형

이벤트는 이벤트 패턴 및 대상을 포함하는 규칙으로 구성됩니다. 이벤트 패턴 및 대상 개체를 선택하여 규칙을 구성합니다.

이벤트 패턴

각 규칙은 모니터링할 이벤트의 소스 및 유형, 이벤트 대상이 포함된 이벤트 패턴으로 표현됩니다. 이벤트를 모니터링하려면 모니터링할 서비스를 이벤트 소스로 사용하여 규칙을 만듭니다. 예를 들어 AWS Proton 을 이벤트 소스로 사용하여 배포 상태가 변경될 때 규칙을 트리거하는 이벤트 패턴으로 규칙을 만들 수 있습니다.

대상

새 규칙은 선택한 서비스를 이벤트 대상으로 수신합니다. 알림을 보내거나, 상태 정보를 캡처하거나, 교정 작업을 수행하거나, 이벤트를 시작하거나, 기타 작업을 수행하도록 대상 서비스를 설정할 수 있습니다.

이벤트 객체에는 ID, 계정, 세부 정보 유형 AWS 리전, 소스, 버전, 리소스, 시간(선택 사항)의 표준 필드가 포함됩니다. 세부 정보 필드는 이벤트의 사용자 지정 필드를 포함하는 중첩된 개체입니다.

AWS Proton 이벤트는 최대한으로 내보내집니다. 최선의 작업 전달이란 서비스가 모든 이벤트를 EventBridge에 전송하려고 시도하지만 드물게 이벤트가 전달되지 않을 수 있음을 의미합니다.

이벤트를 내보낼 수 있는 각 AWS Proton 리소스에 대해 다음 표에는 세부 정보 유형 값, 세부 정보 필드 및 (사용 가능한 경우) `status` 및 `previousStatus` 세부 정보 필드의 값 목록에 대한 참조가 나열되어 있습니다. 리소스가 삭제된 경우 `status` 세부 정보 필드 값은 `DELETED`입니다.

Resource	세부 정보 유형 값	세부 정보 필드

`EnvironmentTemplate`	AWS Proton 환경 템플릿 상태 변경	`name`
		`status`
		`previousStatus`

`EnvironmentTemplateVersion`	AWS Proton 환경 템플릿 버전 상태 변경	`name`
		`majorVersion`
		`minorVersion`
		`status`
		`previousStatus`
		상태 값

`ServiceTemplate`	AWS Proton 서비스 템플릿 상태 변경	`name`
		`status`
		`previousStatus`

`ServiceTemplateVersion`	AWS Proton 서비스 템플릿 버전 상태 변경	`name`

Resource	세부 정보 유형 값	세부 정보 필드
		majorVersion minorVersion status previousStatus 상태 값
Environment	AWS Proton 환경 상태 변경	name status previousStatus
Service	AWS Proton 서비스 상태 변경	name status previousStatus 상태 값
ServiceInstance	AWS Proton 서비스 인스턴스 상태 변경	name serviceName status previousStatus

Resource	세부 정보 유형 값	세부 정보 필드

`ServicePipeline`	AWS Proton 서비스 파이프라인 상태 변경	`serviceName`
		`status`
		`previousStatus`

`EnvironmentAccountConnection`	AWS Proton 환경 계정 연결 상태 변경	`id`
		`status`
		`previousStatus`
		상태 값

`Component`	AWS Proton 구성 요소 상태 변경	`name`
		`status`
		`previousStatus`

AWS Proton 이벤트 예제

다음 예제에서는가 EventBridge로 이벤트를 전송할 AWS Proton 수 있는 방법을 보여줍니다.

서비스 템플릿

```
{
    "source": "aws.proton",
    "detail-type": ["AWS Proton Service Template Status Change"],
    "time": "2021-03-22T23:21:40.734Z",
    "resources": ["arn:aws:proton:region_id:123456789012:service-template/sample-
service-template-name"],
    "detail": {
        "name": "sample-service-template-name",
        "status": "PUBLISHED",
        "previousStatus": "DRAFT"
```

```
    }
}
```

서비스 템플릿 버전

```
{
    "source": "aws.proton",
    "detail-type": ["AWS Proton Service Template Version Status Change"],
    "time": "2021-03-22T23:21:40.734Z",
    "resources": ["arn:aws:proton:region_id:123456789012:service-template/sample-
service-template-name:1.0"],
    "detail": {
        "name": "sample-service-template-name",
        "majorVersion": "1",
        "minorVersion": "0",
        "status": "REGISTRATION_FAILED",
        "previousStatus": "REGISTRATION_IN_PROGRESS"
    }
}
```

환경

```
{
    "source": "aws.proton",
    "detail-type": ["AWS Proton Environment Status Change"],
    "time": "2021-03-22T23:21:40.734Z",
    "resources": ["arn:aws:proton:region_id:123456789012:environment/sample-
environment"],
    "detail": {
        "name": "sample-environment",
        "status": "DELETE_FAILED",
        "previousStatus": "DELETE_IN_PROGRESS"
    }
}
```

EventBridgeTutorial: AWS Proton 서비스 상태 변경에 대한 Amazon Simple Notification Service 알림 전송

이 자습서에서는 AWS Proton 서비스의 상태 변경을 캡처하는 AWS Proton 사전 구성된 이벤트 규칙을 사용합니다. EventBridge는 SNS 주제로 상태 변경 사항을 보냅니다. 주제를 구독하면 Amazon SNS가 AWS Proton 서비스에 대한 상태 변경 이메일을 보냅니다.

사전 조건

`Active` 상태의 기존 AWS Proton 서비스가 있습니다. 이 자습서에서는 이 서비스에 서비스 인스턴스를 추가한 다음 인스턴스를 삭제할 수 있습니다.

AWS Proton 서비스를 생성해야 하는 경우 섹션을 참조하세요시작하기. 자세한 내용은 AWS Proton 할당량 및 the section called “편집” 섹션을 참조하세요.

1단계: Amazon SNS 주제 생성 및 구독

2단계에서 생성한 이벤트 규칙의 이벤트 대상이 될 SNS 주제를 만듭니다.

SNS 주제 생성

1. SNS 콘솔을 열고 로그인합니다.
2. 탐색 창에서 주제를 선택한 다음, 주제 생성을 선택합니다.
3. 주제 생성 페이지에서,
   a. 유형에서 표준을 선택합니다.
   b. 이름에 **`tutorial-service-status-change`**를 입력하고 주제 생성을 선택합니다.
4. 자습서-서비스-상태-변경 세부 정보 페이지에서 구독 생성을 선택합니다.
5. 구독 생성 페이지에서,
   a. 프로토콜(Protocol)에서 이메일(Email)을 선택합니다.
   b. 엔드포인트(Endpoint)에 현재 액세스 권한이 있는 이메일 주소를 입력하고 구독 생성(Create subscription)을 선택합니다.
6. 이메일 계정을 확인하고 구독 확인 이메일 메시지를 기다립니다. 메시지를 수신하면 구독 확인을 선택합니다.

2단계: 이벤트 규칙 등록

AWS Proton 서비스의 상태 변경을 캡처하는 이벤트 규칙을 등록합니다. 자세한 내용은 사전 조건 단원을 참조하십시오.

이벤트 규칙을 생성합니다.

1. EventBridge 콘솔을 엽니다.

2. 탐색 창에서 이벤트와 규칙을 선택합니다.
3. 규칙 페이지의 규칙 단원에서 규칙 생성을 선택합니다.
4. 규칙 생성 페이지에서,

   a. 이름 및 설명 섹션의 이름에 **tutorial-rule**을 입력합니다.

   b. 패턴 정의 단원에서 이벤트 패턴을 선택합니다.

      i. Event matching pattern(이벤트 일치 패턴)에서 Pre-defined pattern by service(서비스별 사전 정의됨)를 선택합니다.

      ii. 서비스 공급자(Service provider)에 AWS를 선택합니다.

      iii. 서비스 이름(Service name)에서 AWS Proton을 선택합니다.

      iv. 이벤트 유형에서 AWS Proton 서비스 상태 변경을 선택합니다.

      이벤트 패턴은 텍스트 편집기에 표시됩니다.

      v. AWS Proton 콘솔을 엽니다.

      vi. 탐색 창에서 서비스를 선택합니다.

      vii. 서비스 페이지에서 AWS Proton 서비스 이름을 선택합니다.

      viii. 서비스 세부 정보 페이지에서 서비스 리소스 이름(ARN)을 복사합니다.

      ix. EventBridge 콘솔과 자습서 규칙으로 돌아가서 텍스트 편집기에서 편집을 선택합니다.

      x. 텍스트 편집기에서 `"resources":`에 8단계에서 복사한 서비스 ARN을 입력합니다.

```
{
    "source": ["aws.proton"],
    "detail-type": ["AWS Proton Service Status Change"],
    "resources": ["arn:aws:proton:region-id:123456789012:service/your-service"]
}
```

      xi. 이벤트 패턴을 저장합니다.

   c. 대상 선택 단원에서,

      i. 대상에서 SNS 주제를 선택합니다.

      ii. 주제에서는 자습서-서비스-상태-변경을 선택합니다.

   d. 생성(Create)을 선택합니다.

3단계: 이벤트 규칙 테스트

AWS Proton 서비스에 인스턴스를 추가하여 이벤트 규칙이 작동하는지 확인합니다.

1. AWS Proton 콘솔로 전환합니다.
2. 탐색 창에서 서비스를 선택합니다.
3. 서비스 페이지에서 서비스 이름을 선택합니다.
4. 서비스 세부 정보 페이지에서 편집을 선택합니다.
5. 서비스 구성 페이지에서 다음을 선택합니다.
6. 사용자 지정 설정 구성 페이지, 서비스 인스턴스 단원에서, 새 인스턴스 추가를 선택합니다.
7. 새 인스턴스의 양식을 작성합니다.
   a. 인스턴스 이름을 입력합니다.
   b. 기존 인스턴스에 대해 선택한 것과 동일한 호환 환경을 선택합니다.
   c. 필수 입력 값을 입력합니다.
   d. 다음을 선택합니다.
8. 의견 내용을 검토하고 업데이트를 선택합니다.
9. 서비스 상태가 이면 이메일을 `Active`확인하여 상태 업데이트를 제공하는 AWS 알림을 받았는지 확인합니다.

```
{
    "version": "0",
    "id": "af76c382-2b3c-7a0a-cf01-936dff228276",
    "detail-type": "AWS Proton Service Status Change",
    "source": "aws.proton",
    "account": "123456789012",
    "time": "2021-06-29T20:40:16Z",
    "region": "region-id",
    "resources": ["arn:aws:proton:region-id:123456789012:service/your-service"],
    "detail": {
        "previousStatus": "ACTIVE",
        "status": "UPDATE_IN_PROGRESS",
        "name": "your-service"
    }
}
```

```
{
```

```
    "version": "0",
    "id": "87131e29-ad95-bda2-cd30-0ce825dfb0cd",
    "detail-type": "AWS Proton Service Status Change",
    "source": "aws.proton",
    "account": "123456789012",
    "time": "2021-06-29T20:42:27Z",
    "region": "region-id",
    "resources": ["arn:aws:proton:region-id:123456789012:service/your-service"],
    "detail": {
        "previousStatus": "UPDATE_IN_PROGRESS",
        "status": "ACTIVE",
        "name": "your-service"
    }
}
```

4단계: 정리

SNS 주제 및 구독을 삭제하고 EventBridge 규칙을 삭제하세요.

SNS 주제 삭제 및 구독

1. SNS 콘솔로 이동합니다.
2. 탐색 창에서 구독을 선택합니다.
3. 구독 페이지에서 이름이 지정된 `tutorial-service-status-change` 주제에 대한 구독을 선택한 다음 삭제를 선택합니다.
4. 탐색 창에서 주제를 선택합니다.
5. 주제 페이지에서 주제 이름이 지정된 `tutorial-service-status-change`를 선택한 다음 삭제를 선택합니다.
6. 삭제를 확인하라는 메시지가 표시됩니다. 지침을 따르고 삭제를 선택합니다.

EventBridge 규칙을 삭제합니다.

1. EventBridge 콘솔로 이동합니다.
2. 탐색 창에서 이벤트와 규칙을 선택합니다.
3. 규칙 페이지에서 주제 이름이 지정된 `tutorial-rule`를 선택한 다음 삭제를 선택합니다.
4. 삭제를 확인하라는 메시지가 표시됩니다. 삭제를 선택합니다.

추가된 서비스 인스턴스를 삭제합니다.

1. AWS Proton 콘솔로 이동합니다.
2. 탐색 창에서 서비스를 선택합니다.
3. 서비스 페이지에서 서비스 이름을 선택합니다.
4. 서비스 세부 정보 페이지에서 편집을 선택하고 다음을 선택합니다.
5. 사용자 지정 설정 구성 페이지의 서비스 인스턴스 단원에서 이 자습서의 일부로 만든 서비스 인스턴스의 삭제를 선택한 후 다음을 선택합니다.
6. 의견 내용을 검토하고 업데이트를 선택합니다.
7. 삭제를 확인하라는 메시지가 표시됩니다. 지침을 따르고 예, 삭제를 선택합니다.

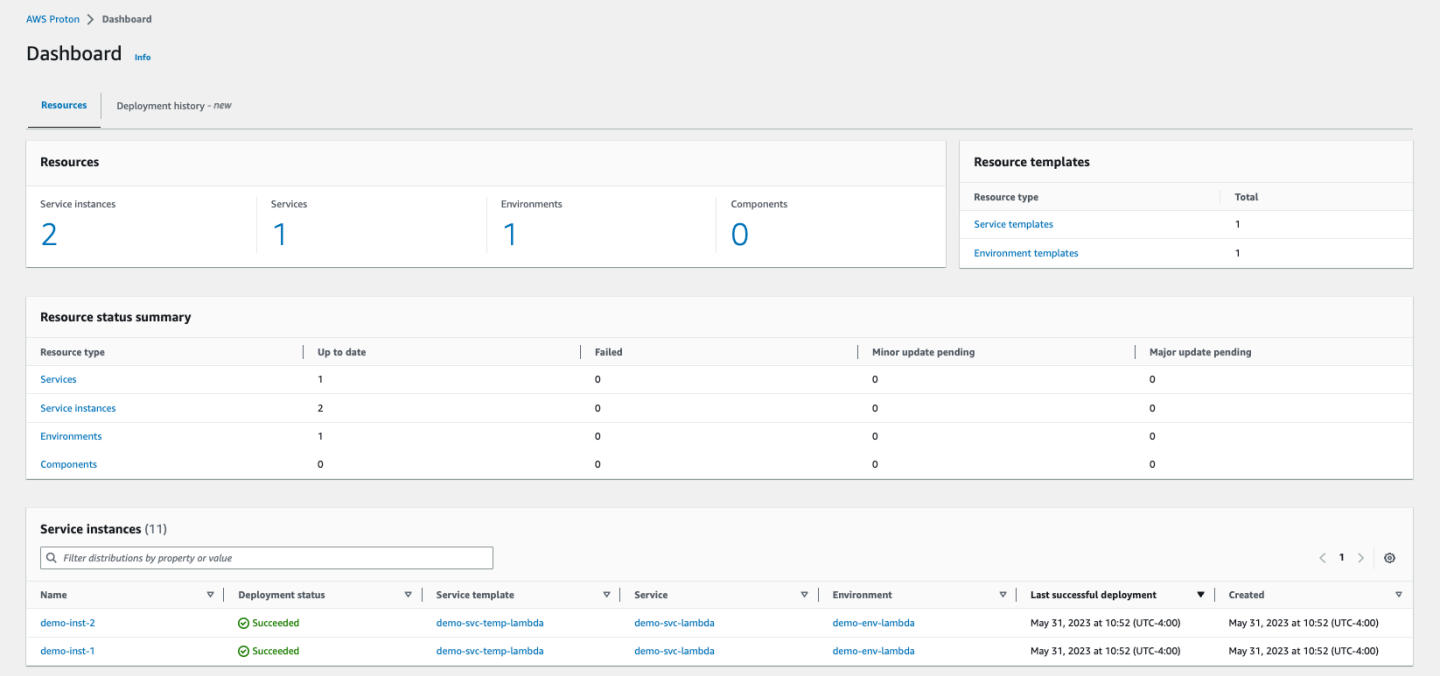
AWS Proton 대시보드를 사용하여 인프라를 최신 상태로 유지

AWS Proton 대시보드는 AWS 계정의 AWS Proton 리소스에 대한 요약을 제공하며, 특히 업데이트된 배포된 리소스가 어떻게 되는지에 중점을 둡니다. 배포된 리소스는 관련 템플릿의 권장 버전을 사용하면 최신 상태를 유지합니다. 배포된 리소스가 만료되면 메이저 또는 마이너 템플릿 버전 업데이트가 필요할 수 있습니다.

AWS Proton 콘솔에서 대시보드 보기

AWS Proton 대시보드를 보려면 AWS Proton 콘솔을 연 다음 탐색 창에서 대시보드를 선택합니다.

리소스

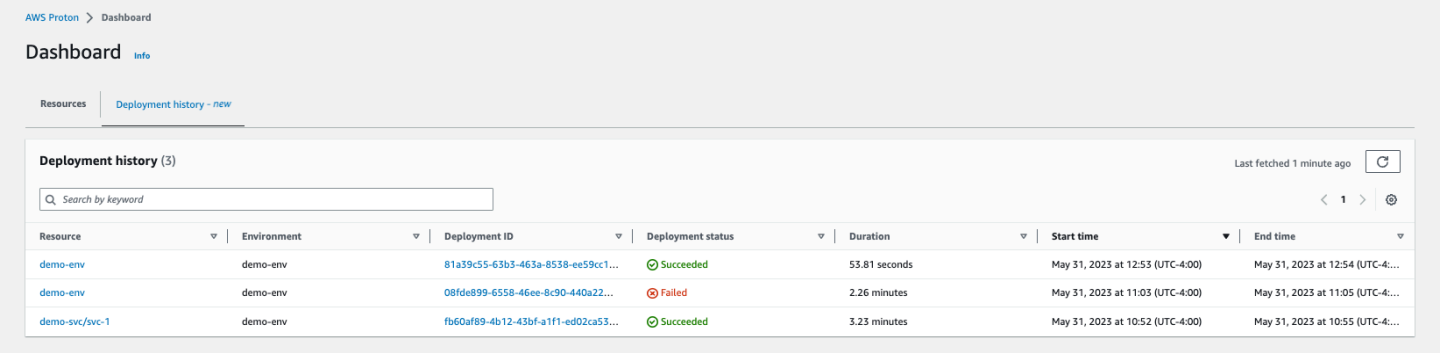


대시보드의 첫 번째 탭에는 계정의 모든 리소스 수가 표시됩니다. 리소스 탭에는 서비스 인스턴스, 서비스, 환경, 구성 요소의 수와 리소스 템플릿이 표시됩니다. 또한 배포된 각 리소스 유형의 리소스 수를 해당 유형의 리소스 상태별로 분류합니다. 서비스 인스턴스 테이블에는 배포 상태, 연결된 AWS Proton 리소스, 사용 가능한 업데이트, 일부 타임스탬프 등 각 서비스 인스턴스의 세부 정보가 표시됩니다.

모든 테이블 속성을 기준으로 서비스 인스턴스 목록을 필터링할 수 있습니다. 예를 들어 특정 기간 내에 배포된 서비스 인스턴스 또는 메이저 또는 마이너 버전 권장 사항에 비해 최신 상태가 아닌 서비스 인스턴스를 필터링하여 확인할 수 있습니다.

서비스 인스턴스 이름을 선택하면 적절한 버전 업데이트를 수행할 수 있는 서비스 인스턴스 세부 정보 페이지로 이동합니다. 다른 AWS Proton 리소스 이름을 선택하여 세부 정보 페이지로 이동하거나 리소스 유형을 선택하여 해당 리소스 목록으로 이동합니다.

배포 기록



배포 기록 탭에서는 배포에 대한 세부 정보를 볼 수 있습니다. 배포 기록 테이블에서 배포 상태는 물론 환경 및 배포 ID를 추적할 수 있습니다. 리소스 이름 또는 배포 ID를 선택하여 배포 상태 메시지 및 리소스 출력과 같은 더 자세한 정보를 볼 수 있습니다. 또한 이 테이블을 사용하면 모든 테이블 속성을 기준으로 필터링할 수 있습니다.

의 보안 AWS Proton

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 서비스 에서 실행되는 인프라를 보호할 책임이 있습니다 AWS 클라우드. AWS 또한는 안전하게 사용할 수 있는 서비스를 제공합니다. 타사 감사자는 AWS 규정 준수 프로그램 일환으로 보안의 효과를 정기적으로 테스트하고 확인합니다. 에 적용되는 규정 준수 프로그램에 대한 자세한 내용은 규정 준수 AWS 서비스 프로그램 제공 범위규정 준수 AWS Proton참조하세요.
- 클라우드의 보안 - 사용자의 책임은 AWS 서비스 사용하는에 따라 결정됩니다. 또한 귀하는 귀사의 데이터 민감도, 귀사의 요구 사항, 관련 법률 및 규정을 비롯한 기타 요소에 대해서도 책임이 있습니다.

이 설명서는를 사용할 때 공동 책임 모델을 적용하는 방법을 이해하는 데 도움이 됩니다 AWS Proton. 다음 주제에서는 보안 및 규정 준수 목표를 충족하도록 AWS Proton 를 구성하는 방법을 보여줍니다. 또한 리소스를 모니터링하고 보호하는 AWS Proton 데 도움이 AWS 서비스 되는 다른를 사용하는 방법을 알아봅니다.

주제

- 에 대한 자격 증명 및 액세스 관리 AWS Proton
- 의 구성 및 취약성 분석 AWS Proton
- 의 데이터 보호 AWS Proton
- 의 인프라 보안 AWS Proton
- 에서 로깅 및 모니터링 AWS Proton
- 의 복원력 AWS Proton
- 에 대한 보안 모범 사례 AWS Proton
- 교차 서비스 혼동된 대리인 방지
- CodeBuild 프로비저닝 사용자 지정 VPC 지원

에 대한 자격 증명 및 액세스 관리 AWS Proton

AWS Identity and Access Management (IAM)는 관리자가 AWS 리소스에 대한 액세스를 안전하게 제어하는 데 도움이 AWS 서비스 되는 입니다. IAM 관리자는 누가 AWS Proton 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

AWS Identity and Access Management (IAM)를 사용하는 방법은 역할에 따라 다릅니다.

- 서비스 사용자 - 기능에 액세스할 수 없는 경우 관리자에게 권한 요청(참조AWS Proton 자격 증명 및 액세스 문제 해결)
- 서비스 관리자 - 사용자 액세스 결정 및 권한 요청 제출(AWS Proton 에서 IAM을 사용하는 방법 참조)
- IAM 관리자 - 액세스를 관리하기 위한 정책 작성(에 대한 자격 증명 기반 정책 예제 AWS Proton 참조)

ID를 통한 인증

인증은 자격 증명 자격 증명을 AWS 사용하여에 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 정책을 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 Proton 에서 IAM을 사용하는 방법

IAM을 사용하여에 대한 액세스를 관리하기 전에 사용할 수 있는 IAM 기능을 AWS Proton알아봅니다 AWS Proton.

에서 사용할 수 있는 IAM 기능 AWS Proton

IAM 특성	AWS Proton 지원
자격 증명 기반 정책	예
리소스 기반 정책	아니요
정책 작업	예
정책 리소스	예
정책 조건 키	예

IAM 특성	AWS Proton 지원
ACL	아니요
ABAC(정책의 태그)	예
임시 보안 인증	예
엔터티 권한	예
서비스 역할	예
서비스 연결 역할	예

AWS Proton 및 기타에서 대부분의 IAM 기능을 AWS 서비스 사용하는 방법을 개괄적으로 알아보려면 IAM 사용 설명서의 AWS 서비스 IAM으로 작업하는 섹션을 참조하세요.

에 대한 자격 증명 기반 정책 AWS Proton

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 있는지를 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.

에 대한 자격 증명 기반 정책 예제 AWS Proton

자격 AWS Proton 증명 기반 정책의 예를 보려면 섹션을 참조하세요에 대한 자격 증명 기반 정책 예제 AWS Proton.

내의 리소스 기반 정책 AWS Proton

리소스 기반 정책 지원: 아니요

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 관리자

는 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 또는가 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.

에 대한 정책 작업 AWS Proton

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.

AWS Proton 작업 목록을 보려면 서비스 승인 참조의에서 정의한 작업을 AWS Proton 참조하세요.

의 정책 작업은 작업 앞에 다음 접두사를 AWS Proton 사용합니다.

```
proton
```

단일 문에서 여러 작업을 지정하려면 쉼표로 구분합니다.

```
"Action": [
      "proton:action1",
      "proton:action2"
         ]
```

와일드카드(*)를 사용하여 여러 작업을 지정할 수 있습니다. 예를 들어, `List`라는 단어로 시작하는 모든 작업을 지정하려면 다음 작업을 포함합니다.

```
"Action": "proton:List*"
```

자격 AWS Proton 증명 기반 정책의 예를 보려면 섹션을 참조하세요에 대한 자격 증명 기반 정책 예제 AWS Proton.

에 대한 정책 리소스 AWS Proton

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": "*"
```

AWS Proton 리소스 유형 및 해당 ARNs 목록을 보려면 서비스 승인 참조의에서 정의한 리소스를 AWS Proton 참조하세요. 각 리소스의 ARN을 지정할 수 있는 작업을 알아보려면 AWS Proton에서 정의한 작업을 참조하세요.

자격 AWS Proton 증명 기반 정책의 예를 보려면 섹션을 참조하세요에 대한 자격 증명 기반 정책 예제 AWS Proton.

에 대한 정책 조건 키 AWS Proton

서비스별 정책 조건 키 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.

`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.

AWS Proton 조건 키 목록을 보려면 서비스 승인 참조의에 대한 조건 키를 AWS Proton 참조하세요. 조건 키를 사용할 수 있는 작업과 리소스를 알아보려면 에서 정의한 작업을 AWS Proton 참조하세요.

리소스에 대한 액세스를 제한하는 조건 키 기반 정책의 예제는 에 대한 조건 키 기반 정책 예제 AWS Proton에서 확인할 수 있습니다.

의 액세스 제어 목록(ACLs) AWS Proton

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.

ACL(액세스 제어 목록)은 리소스에 연결할 수 있는 피부여자 목록입니다. 이 목록은 연결된 리소스에 액세스할 수 있는 권한을 계정에 부여합니다.

를 사용한 ABAC(속성 기반 액세스 제어) AWS Proton

ABAC 지원(정책의 태그): 예

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.

AWS Proton 리소스 태그 지정에 대한 자세한 내용은 섹션을 참조하세요AWS Proton 리소스 및 태그 지정.

에서 임시 자격 증명 사용 AWS Proton

임시 자격 증명 지원: 예

임시 자격 증명은 AWS 리소스에 대한 단기 액세스를 제공하며 페더레이션 또는 전환 역할을 사용할 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 작업하는 AWS 서비스 섹션을 참조하세요.

에 대한 교차 서비스 보안 주체 권한 AWS Proton

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.

에 대한 서비스 역할 AWS Proton

서비스 역할 지원: 예

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.

자세한 내용은 AWS Proton IAM 서비스 역할 정책 예제 단원을 참조하십시오.

> **Warning**
>
> 서비스 역할에 대한 권한을 변경하면 AWS Proton 기능이 중단될 수 있습니다. 에서 관련 지침을 AWS Proton 제공하는 경우에만 서비스 역할을 편집합니다.

에 대한 서비스 연결 역할 AWS Proton

서비스 연결 역할 지원: 예

서비스 연결 역할은에 연결된 서비스 역할의 한 유형입니다 AWS 서비스. 서비스는 사용자를 대신하여 작업을 수행하기 위해 역할을 수임할 수 있습니다. 서비스 연결 역할은에 표시 AWS 계정 되며 서비스가 소유합니다. IAM 관리자는 서비스 연결 역할의 권한을 볼 수 있지만 편집은 할 수 없습니다.

자세한 내용은 에 대한 서비스 연결 역할 사용 AWS Proton 단원을 참조하십시오.

에 대한 정책 예제 AWS Proton

다음 섹션에서 AWS Proton IAM 정책 예제를 찾습니다.

주제

- 에 대한 자격 증명 기반 정책 예제 AWS Proton
- AWS Proton IAM 서비스 역할 정책 예제
- 에 대한 조건 키 기반 정책 예제 AWS Proton

에 대한 자격 증명 기반 정책 예제 AWS Proton

기본적으로 사용자 및 역할에는 AWS Proton 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 AWS Proton에서 정의한 작업 및 리소스 유형에 대한 자세한 내용은 서비스 권한 부여 참조의 에 대한 작업, 리소스 및 조건 키를 AWS Proton 참조하세요.

주제

- 정책 모범 사례
- 에 대한 자격 증명 기반 정책 예제 링크 AWS Proton

정책 모범 사례

자격 증명 기반 정책에 따라 계정에서 사용자가 AWS Proton 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.

에 대한 자격 증명 기반 정책 예제 링크 AWS Proton

에 대한 자격 증명 기반 정책 예제 예제 링크 AWS Proton

- AWS 에 대한 관리형 정책 AWS Proton
- AWS Proton IAM 서비스 역할 정책 예제
- 에 대한 조건 키 기반 정책 예제 AWS Proton

AWS Proton IAM 서비스 역할 정책 예제

관리자는 환경 및 서비스 템플릿에 정의된 대로가 AWS Proton 생성하는 리소스를 소유하고 관리합니다. 가 대신 리소스를 AWS Proton 생성하도록 허용하는 IAM 서비스 역할을 계정에 연결합니다. 관리자는가 애플리케이션을 AWS Proton 환경에서 AWS Proton 서비스로 AWS Proton 배포할 때 개발자가 나중에 소유하고 관리하는 리소스에 대한 IAM 역할과 AWS Key Management Service 키를 제공합니다. AWS KMS 및 데이터 암호화에 대한 자세한 내용은 섹션을 참조하세요의 데이터 보호 AWS Proton.

서비스 역할은가 사용자를 대신하여 리소스를 AWS Proton 호출하도록 허용하는 Amazon Web Services(IAM) 역할입니다. 서비스 역할을 지정한 경우 AWS Proton 에서는 역할의 자격 증명을 사용합니다. 서비스 역할을 사용하여가 수행할 AWS Proton 수 있는 작업을 명시적으로 지정합니다.

IAM 서비스를 사용하여 서비스 역할과 해당 권한 정책을 생성합니다. 서비스 역할 생성에 대한 자세한 내용은 IAM 사용 설명서의 AWS 서비스에 권한을 위임할 역할 생성을 참조하세요.

AWS Proton 를 사용하여 프로비저닝하기 위한 서비스 역할 CloudFormation

플랫폼 팀의 구성원은 관리자가 AWS Proton 서비스 역할을 생성하고 환경을 환경의 CloudFormation 서비스 역할(CreateEnvironment API 작업의 `protonServiceRoleArn` 파라미터)로 생성할 AWS Proton 때에 제공할 수 있습니다. 이 역할을 사용하면 환경 또는 환경에서 실행되는 서비스 인스턴스

가 AWS관리형 프로비저닝을 사용하고 인프라를 프로비저닝할 때가 사용자를 대신하여 다른 서비스에 AWS Proton API를 호출 AWS CloudFormation 할 수 있습니다.

AWS Proton 서비스 역할에는 다음 IAM 역할 및 신뢰 정책을 사용하는 것이 좋습니다. AWS Proton 콘솔을 사용하여 환경을 생성하고 새 역할을 생성하도록 선택하면이 정책이 자동으로 생성하는 서비스 역할에 AWS Proton 추가됩니다. 이 정책에 대한 권한을 축소할 때는 `Access Denied` 오류가 AWS Proton 실패한다는 점에 유의하세요.

> **Important**
>
> 다음 예제에 표시된 정책은 계정에 템플릿을 등록할 수 있는 모든 사람에게 관리자 권한을 부여한다는 점에 유의합니다. AWS Proton 템플릿에서 정의할 리소스를 알 수 없으므로 이러한 정책에는 광범위한 권한이 있습니다. 환경에 배포될 특정 리소스로 권한 범위를 좁히는 것이 좋습니다.

AWS Proton 에 대한 서비스 역할 정책 예제 CloudFormation

를 AWS 계정 ID*123456789012*로 바꿉니다.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudformation:CancelUpdateStack",
        "cloudformation:ContinueUpdateRollback",
        "cloudformation:CreateChangeSet",
        "cloudformation:CreateStack",
        "cloudformation:DeleteChangeSet",
        "cloudformation:DeleteStack",
        "cloudformation:DescribeChangeSet",
        "cloudformation:DescribeStackDriftDetectionStatus",
        "cloudformation:DescribeStackEvents",
        "cloudformation:DescribeStackResourceDrifts",
        "cloudformation:DescribeStacks",
        "cloudformation:DetectStackResourceDrift",
        "cloudformation:ExecuteChangeSet",
```

```
        "cloudformation:ListChangeSets",
        "cloudformation:ListStackResources",
        "cloudformation:UpdateStack"
      ],
      "Resource": "arn:aws:cloudformation:*:123456789012:stack/AWSProton-*"
    },
    {
      "Effect": "Allow",
      "NotAction": [
        "organizations:*",
        "account:*"
      ],
      "Resource": "*",
      "Condition": {
        "ForAnyValue:StringEquals": {
          "aws:CalledVia": [
            "cloudformation.amazonaws.com"
          ]
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "organizations:DescribeOrganization",
        "account:ListRegions"
      ],
      "Resource": "*",
      "Condition": {
        "ForAnyValue:StringEquals": {
          "aws:CalledVia": [
            "cloudformation.amazonaws.com"
          ]
        }
      }
    }
  ]
}
```

AWS Proton 서비스 신뢰 정책

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
    "Sid": "ServiceTrustRelationshipWithConfusedDeputyPrevention",
    "Effect": "Allow",
    "Principal": {
      "Service": "proton.amazonaws.com"
    },
    "Action": "sts:AssumeRole",
    "Condition": {
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      },
      "ArnLike": {
        "aws:SourceArn": "arn:aws:proton:*:123456789012:environment/*"
      }
    }
  }
}
```

범위가 지정된 다운 AWS관리형 프로비저닝 서비스 역할 정책

다음은 S3 리소스를 프로비저닝하기 위한 AWS Proton 서비스만 필요한 경우 사용할 수 있는 범위가 축소된 AWS Proton 서비스 역할 정책의 예입니다.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudformation:CancelUpdateStack",
        "cloudformation:ContinueUpdateRollback",
        "cloudformation:CreateChangeSet",
        "cloudformation:CreateStack",
```

```
        "cloudformation:DeleteChangeSet",
        "cloudformation:DeleteStack",
        "cloudformation:DescribeChangeSet",
        "cloudformation:DescribeStackDriftDetectionStatus",
        "cloudformation:DescribeStackEvents",
        "cloudformation:DescribeStackResourceDrifts",
        "cloudformation:DescribeStacks",
        "cloudformation:DetectStackResourceDrift",
        "cloudformation:ExecuteChangeSet",
        "cloudformation:ListChangeSets",
        "cloudformation:ListStackResources",
        "cloudformation:UpdateStack"
      ],
      "Resource": "arn:aws:cloudformation:*:123456789012:stack/AWSProton-*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:*"
      ],
      "Resource": "*",
      "Condition": {
        "ForAnyValue:StringEquals": {
          "aws:CalledVia": [
            "cloudformation.amazonaws.com"
          ]
        }
      }
    }
  ]
}
```

AWS Proton CodeBuild 프로비저닝을 위한 서비스 역할

플랫폼 팀의 구성원은 관리자가 AWS Proton 서비스 역할을 생성하고 환경을 환경의 CodeBuild 서비스 역할(CreateEnvironment API 작업의 `codebuildRoleArn` 파라미터)로 생성할 AWS Proton 때에 제공할 수 있습니다. 이 역할을 사용하면 환경 또는 해당 환경에서 실행되는 서비스 인스턴스가 CodeBuild 프로비저닝을 사용하여 인프라를 프로비저닝할 때가 사용자를 대신하여 다른 서비스에 AWS Proton API를 호출할 수 있습니다.

AWS Proton 콘솔을 사용하여 환경을 생성하고 새 역할을 생성하도록 선택하면는 관리자 권한이 있는 정책을 자동으로 생성하는 서비스 역할에 AWS Proton 추가합니다. 자체 역할을 생성하고 권한을 축소할 때 `Access Denied` 오류 시가 AWS Proton 실패한다는 점에 유의하세요.

> **Important**
>
> 생성한 역할에 AWS Proton 연결하는 정책은 계정에 템플릿을 등록할 수 있는 모든 사용자에게 관리자 권한을 부여합니다. AWS Proton 템플릿에서 정의할 리소스를 알 수 없으므로 이러한 정책에는 광범위한 권한이 있습니다. 환경에 배포될 특정 리소스로 권한 범위를 좁히는 것이 좋습니다.

AWS Proton CodeBuild에 대한 서비스 역할 정책 예제

다음 예제는 CodeBuild가 AWS Cloud Development Kit (AWS CDK)를 사용하여 리소스를 프로비저닝할 수 있는 권한을 제공합니다.

를 AWS 계정 ID*123456789012*로 바꿉니다.

JSON

```
{
 "Version":"2012-10-17",
 "Statement": [
   {
     "Action": [
       "logs:CreateLogStream",
       "logs:CreateLogGroup",
       "logs:PutLogEvents"
     ],
     "Resource": [
       "arn:aws:logs:us-east-1:123456789012:log-group:/aws/codebuild/AWSProton-
Shell-*",
       "arn:aws:logs:us-east-1:123456789012:log-group:/aws/codebuild/AWSProton-
Shell-*:*"
     ],
     "Effect": "Allow"
   },
   {
     "Action": "proton:NotifyResourceDeploymentStatusChange",
     "Resource": "arn:aws:proton:us-east-1:123456789012:*",
```

```
      "Effect": "Allow"
    },
    {
      "Action": "sts:AssumeRole",
      "Resource": [
        "arn:aws:iam::123456789012:role/cdk-*-deploy-role-*",
        "arn:aws:iam::123456789012:role/cdk-*-file-publishing-role-*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

AWS Proton CodeBuild 신뢰 정책

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
    "Sid": "CodeBuildTrustRelationshipWithConfusedDeputyPrevention",
    "Effect": "Allow",
    "Principal": {
      "Service": "codebuild.amazonaws.com"
    },
    "Action": "sts:AssumeRole",
    "Condition": {
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      },
      "ArnLike": {
        "aws:SourceArn": "arn:aws:proton:*:123456789012:environment/*"
      }
    }
  }
}
```

AWS Proton 파이프라인 서비스 역할

서비스 파이프라인을 프로비저닝하려면 다른 서비스에 대한 API 호출 권한이 AWS Proton 필요합니다. 필수 서비스 역할은 환경을 만들 때 제공하는 서비스 역할과 비슷합니다. 그러나 파이프라인을 생성하기 위한 역할은 AWS 계정의 모든 서비스 간에 공유되며 콘솔에서 또는 UpdateAccountSettings API 작업을 통해 이러한 역할을 계정 설정으로 제공합니다.

AWS Proton 콘솔을 사용하여 계정 설정을 업데이트하고 CloudFormation 또는 CodeBuild 서비스 역할에 대한 새 역할을 생성하도록 선택하면가 생성하는 서비스 역할에 AWS Proton 추가하는 정책은 이전 섹션 AWS-관리형 프로비저닝 역할 및에 설명된 정책과 동일합니다CodeBuild 프로비저닝 역할. 이 정책에 대한 권한을 축소할 때는 `Access Denied` 오류가 AWS Proton 실패한다는 점에 유의하세요.

> **Important**
>
> 이전 섹션의 예시 정책은 계정에 템플릿을 등록할 수 있는 모든 사람에게 관리자 권한을 부여한다는 점에 유의합니다. AWS Proton 템플릿에서 정의할 리소스를 알 수 없으므로 이러한 정책에는 광범위한 권한이 있습니다. 파이프라인에 배포될 특정 리소스로 권한 범위를 좁히는 것이 좋습니다.

AWS Proton 구성 요소 역할

플랫폼 팀의 구성원은 관리자가 AWS Proton 서비스 역할을 생성하고 환경을 환경의 CloudFormation 구성 요소 역할(CreateEnvironment API 작업의 `componentRoleArn` 파라미터)로 생성할 AWS Proton 때에 제공할 수 있습니다. 이 역할은 직접 정의된 구성 요소가 프로비저닝할 수 있는 인프라의 범위를 좁힙니다. 구성 요소에 대한 자세한 내용은 구성 요소를 참조하세요.

다음 예제 정책은 Simple Storage Service(S3) 버킷 및 관련 액세스 정책을 프로비저닝하는 직접 정의된 구성 요소를 생성할 수 있도록 지원합니다.

AWS Proton 구성 요소 역할 정책 예제

를 AWS 계정 ID*123456789012*로 바꿉니다.

JSON

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

```
      "Effect": "Allow",
      "Action": [
        "cloudformation:CancelUpdateStack",
        "cloudformation:CreateChangeSet",
        "cloudformation:DeleteChangeSet",
        "cloudformation:DescribeStacks",
        "cloudformation:ContinueUpdateRollback",
        "cloudformation:DetectStackResourceDrift",
        "cloudformation:DescribeStackResourceDrifts",
        "cloudformation:DescribeStackEvents",
        "cloudformation:CreateStack",
        "cloudformation:DeleteStack",
        "cloudformation:UpdateStack",
        "cloudformation:DescribeChangeSet",
        "cloudformation:ExecuteChangeSet",
        "cloudformation:ListChangeSets",
        "cloudformation:ListStackResources"
      ],
      "Resource": "arn:aws:cloudformation:*:123456789012:stack/AWSProton-*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:CreateBucket",
        "s3:DeleteBucket",
        "s3:GetBucket*",
        "iam:CreatePolicy",
        "iam:DeletePolicy",
        "iam:GetPolicy",
        "iam:ListPolicyVersions",
        "iam:DeletePolicyVersion"
      ],
      "Resource": "*",
      "Condition": {
        "ForAnyValue:StringEquals": {
          "aws:CalledVia": "cloudformation.amazonaws.com"
        }
      }
    }
  ]
}
```

AWS Proton 구성 요소 신뢰 정책

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
    "Sid": "ServiceTrustRelationshipWithConfusedDeputyPrevention",
    "Effect": "Allow",
    "Principal": {
      "Service": "proton.amazonaws.com"
    },
    "Action": "sts:AssumeRole",
    "Condition": {
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      },
      "ArnLike": {
        "aws:SourceArn": "arn:aws:proton:*:123456789012:environment/*"
      }
    }
  }
}
```

에 대한 조건 키 기반 정책 예제 AWS Proton

다음 예제 IAM 정책은 `Condition` 블록에 지정된 템플릿과 일치하는 AWS Proton 작업에 대한 액세스를 거부합니다. 이러한 조건 키는 작업, 리소스 및 조건 키 AWS Proton에 나열된 작업에서만 지원됩니다. `DeleteEnvironmentTemplate` 등의 다른 작업에 대한 권한을 관리하려면 리소스 수준 액세스 제어를 사용해야 합니다.

특정 AWS Proton 템플릿에 대한 템플릿 작업을 거부하는 정책의 예:

JSON

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

```
            "Action": ["proton:*"],
            "Resource": "*",
            "Condition": {
                "StringEqualsIfExists": {
                    "proton:EnvironmentTemplate":
 ["arn:aws:proton:region_id:123456789012:environment-template/my-environment-
template"]
                }
            }
        },
        {
            "Effect": "Deny",
            "Action": ["proton:*"],
            "Resource": "*",
            "Condition": {
                "StringEqualsIfExists": {
                    "proton:ServiceTemplate":
 ["arn:aws:proton:region_id:123456789012:service-template/my-service-template"]
                }
            }
        }
    ]
}
```

다음 예제 정책에서 첫 번째 리소스 수준 문은 `Resource` 블록에 나열된 서비스 AWS Proton 템플릿과 `ListServiceTemplates`일치하는 이외의 템플릿 작업에 대한 액세스를 거부합니다. 두 번째 문은 `Condition` 블록에 나열된 템플릿과 일치하는 AWS Proton 작업에 대한 액세스를 거부합니다.

특정 템플릿과 일치하는 AWS Proton 작업을 거부하는 정책의 예:

JSON

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

```
            "Resource": "arn:aws:proton:us-east-1:123456789012:service-template/
my-service-template"
        },
        {
            "Effect": "Deny",
            "Action": [
                "proton:*"
            ],
            "Resource": "*",
            "Condition": {
                "StringEqualsIfExists": {
                    "proton:ServiceTemplate": [
                        "arn:aws:proton:us-east-1:123456789012:service-template/
my-service-template"
                    ]
                }
            }
        }
    ]
}
```

최종 정책 예제에서는 Condition 블록에 나열된 특정 서비스 템플릿과 일치하는 개발자 AWS Proton 작업을 허용합니다.

특정 템플릿과 일치하는 AWS Proton 개발자 작업을 허용하는 정책 예제:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "proton:ListServiceTemplates",
                "proton:ListServiceTemplateVersions",
                "proton:ListServices",
                "proton:ListServiceInstances",
                "proton:ListEnvironments",
                "proton:GetServiceTemplate",
                "proton:GetServiceTemplateVersion",
                "proton:GetService",
```

```
                "proton:GetServiceInstance",
                "proton:GetEnvironment",
                "proton:CreateService",
                "proton:UpdateService",
                "proton:UpdateServiceInstance",
                "proton:UpdateServicePipeline",
                "proton:DeleteService",
                "codestar-connections:ListConnections"
            ],
            "Resource": "*",
            "Condition": {
                "StringEqualsIfExists": {
                    "proton:ServiceTemplate":
 "arn:aws:proton:region_id:123456789012:service-template/my-service-template"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "codestar-connections:PassConnection"
            ],
            "Resource": "arn:aws:codestar-connections:*:*:connection/*",
            "Condition": {
                "StringEquals": {
                    "codestar-connections:PassedToService":
 "proton.amazonaws.com"
                }
            }
        }

    ]
}
```

AWS 에 대한 관리형 정책 AWS Proton

사용자, 그룹 및 역할에 권한을 추가하려면 직접 정책을 작성하는 것보다 AWS 관리형 정책을 사용하는 것이 더 쉽습니다. 팀에 필요한 권한만 제공하는 IAM 고객 관리형 정책을 생성하기 위해서는 시간과 전문 지식이 필요합니다. 빠르게 시작하려면 AWS 관리형 정책을 사용할 수 있습니다. 이 정책은 일반적인 사용 사례를 다루며 사용자의 AWS 계정에서 사용할 수 있습니다. AWS 관리형 정책에 대한 자세한 내용은 IAM 사용 설명서의 AWS 관리형 정책을 참조하세요.

AWS 서비스 AWS 관리형 정책을 유지 관리하고 업데이트합니다. AWS 관리형 정책에서는 권한을 변경할 수 없습니다. 서비스에서 때때로 추가 권한을 AWS 관리형 정책에 추가하여 새로운 기능을 지원합니다. 이 유형의 업데이트는 정책이 연결된 모든 ID(사용자, 그룹 및 역할)에 적용됩니다. 서비스는 새로운 기능이 시작되거나 새 작업을 사용할 수 있을 때 AWS 관리형 정책에 업데이트됩니다. 서비스는 AWS 관리형 정책에서 권한을 제거하지 않으므로 정책 업데이트로 인해 기존 권한이 손상되지 않습니다.

또한는 여러 서비스에 걸쳐 있는 직무에 대한 관리형 정책을 AWS 지원합니다. 예를 들어 ReadOnlyAccess AWS 관리형 정책은 모든 AWS 서비스 및 리소스에 대한 읽기 전용 액세스를 제공합니다. 서비스에서 새 기능을 시작하면 AWS 가 새 작업 및 리소스에 대한 읽기 전용 권한을 추가합니다. 직무 정책의 목록과 설명은 IAM 사용 설명서의 직무에 관한AWS 관리형 정책을 참조하세요.

AWS Proton 는 리소스 및 API 작업에 대한 다양한 수준의 제어를 허용하는 사용자, 그룹 또는 역할에 연결할 수 있는 관리형 IAM 정책 및 신뢰 관계를 제공합니다. 이러한 정책은 직접 적용할 수도 있고, 사용자 고유의 정책을 생성하기 위한 시작 지점으로 정책을 사용할 수도 있습니다.

다음 신뢰 관계는 각 AWS Proton 관리형 정책에 사용됩니다.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
    "Sid": "ExampleTrustRelationshipWithProtonConfusedDeputyPrevention",
    "Effect": "Allow",
    "Principal": {
      "Service": "proton.amazonaws.com"
    },
    "Action": "sts:AssumeRole",
    "Condition": {
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      },
      "ArnLike": {
        "aws:SourceArn": "arn:aws:proton:*:123456789012:environment/*"
      }
    }
  }
}
```

AWS 관리형 정책: AWSProtonFullAccess

를 IAM 엔터AWSProtonFullAccess티에 연결할 수 있습니다. AWS Proton 는가 사용자를 대신하여 작업을 AWS Proton 수행할 수 있도록 허용하는 서비스 역할에도이 정책을 연결합니다.

이 정책은 AWS Proton 작업에 대한 전체 액세스 권한과가 AWS Proton 의존하는 다른 AWS 서비스 작업에 대한 제한된 액세스를 허용하는 관리 권한을 부여합니다.

이 정책에는 다음 키 액션 네임스페이스가 포함되어 있습니다.

- proton – 관리자에게 AWS Proton API에 대한 전체 액세스 권한을 허용합니다.
- iam – 관리자가 역할을 AWS Proton에 전달할 수 있습니다. 이는가 관리자를 대신하여 다른 서비스에 API 호출을 AWS Proton 할 수 있도록 하기 위해 필요합니다.
- kms – 관리자가 고객 관리 키에 권한 부여를 추가할 수 있습니다.
- codeconnections - 관리자가 코드 연결을 나열하고 전달할 수 있도록 허용합니다 AWS Proton.

자세한 내용은 AWSProtonFullAccess를 참조하세요.

AWS 관리형 정책: AWSProtonDeveloperAccess

를 IAM 엔터AWSProtonDeveloperAccess티에 연결할 수 있습니다. AWS Proton 는가 사용자를 대신하여 작업을 AWS Proton 수행하도록 허용하는 서비스 역할에도이 정책을 연결합니다.

이 정책은 AWS Proton 작업 및가 AWS Proton 의존하는 다른 AWS 작업에 대한 제한된 액세스를 허용하는 권한을 부여합니다. 이러한 권한의 범위는 AWS Proton 서비스를 생성하고 배포하는 개발자의 역할을 지원하도록 설계되었습니다.

이 정책은 AWS Proton 템플릿 및 환경 생성, 삭제 및 업데이트 APIs에 대한 액세스를 제공하지 않습니다. 개발자에게 이 정책이 제공하는 것보다 훨씬 더 제한된 권한이 필요한 경우 범위를 좁혀 최소 권한을 부여하는 사용자 지정 정책을 만드는 것이 좋습니다.

이 정책에는 다음 키 액션 네임스페이스가 포함되어 있습니다.

- proton – 기여자가 제한된 AWS Proton API 세트에 액세스할 수 있도록 허용합니다.
- codeconnections - 기고자가 코드 연결을 나열하고 전달할 수 있도록 허용합니다 AWS Proton.

자세한 내용은 AWSProtonDeveloperAccess를 참조하세요.

AWS 관리형 정책: AWSProtonReadOnlyAccess

를 IAM 엔터AWSProtonReadOnlyAccess티에 연결할 수 있습니다. AWS Proton 는가 사용자를 대신하여 작업을 AWS Proton 수행하도록 허용하는 서비스 역할에도이 정책을 연결합니다.

이 정책은 AWS Proton 작업에 대한 읽기 전용 액세스와가 AWS Proton 의존하는 다른 AWS 서비스 작업에 대한 제한된 읽기 전용 액세스를 허용하는 권한을 부여합니다.

이 정책에는 다음 키 액션 네임스페이스가 포함되어 있습니다.

- proton – 기여자가 AWS Proton API에 대한 읽기 전용 액세스를 허용합니다.

자세한 내용은 AWSProtonReadOnlyAccess를 참조하세요.

AWS 관리형 정책: AWSProtonSyncServiceRolePolicy

AWS Proton 는이 템플릿 동기화를 AWS Proton 수행하도록 허용하는 AWSServiceRoleForProtonSync 서비스 연결 역할에이 정책을 연결합니다.

이 정책은 AWS Proton 작업 및가 AWS Proton 의존하는 다른 AWS 서비스 작업에 대한 제한된 액세스를 허용하는 권한을 부여합니다.

이 정책에는 다음 키 액션 네임스페이스가 포함되어 있습니다.

- proton - APIs에 AWS Proton 대한 제한된 AWS Proton 액세스 동기화를 허용합니다.
- codeconnections - CodeConnections APIs AWS Proton 제한된 동기화 액세스를 허용합니다.

자세한 내용은 AWSProtonSyncServiceRolePolicy를 참조하세요.

AWS 관리형 정책: AWSProtonCodeBuildProvisioningBasicAccess

권한 CodeBuild는 AWS Proton CodeBuild 프로비저닝에 대한 빌드를 실행해야 합니다. AWSProtonCodeBuildProvisioningBasicAccess을 CodeBuild 프로비저닝 역할에 연결할 수 있습니다.

이 정책은 AWS Proton CodeBuild 프로비저닝이 작동할 수 있는 최소 권한을 부여합니다. CodeBuild가 빌드 로그를 생성할 수 있는 권한을 부여합니다. 또한 Proton이 코드형 인프라(IaC) 출력을 AWS Proton 사용자에게 제공할 수 있는 권한을 부여합니다. IaC 도구가 인프라를 관리하는 데 필요한 권한은 제공하지 않습니다.

이 정책에는 다음 키 액션 네임스페이스가 포함되어 있습니다.

- `logs` - CodeBuild에서 빌드 로그를 생성할 수 있도록 합니다. 이 권한이 없으면 CodeBuild를 시작할 수 없습니다.
- `proton` - CodeBuild 프로비저닝 명령이 지정된 AWS Proton 리소스`aws proton notify-resource-deployment-status-change`에 대한 IaaC 출력을 업데이트하기 위해를 호출하도록 허용합니다.

자세한 내용은 AWSProtonCodeBuildProvisioningBasicAccess를 참조하세요.

AWS 관리형 정책: AWSProtonCodeBuildProvisioningServiceRolePolicy

AWS Proton 는이 CodeBuild 기반 프로비저닝을 수행하도록 허용하는 AWSServiceRoleForProtonCodeBuildProvisioning 서비스 연결 역할에이 정책을 연결합니다. AWS Proton CodeBuild

이 정책은가 AWS Proton 의존하는 AWS 서비스 작업에 대한 제한된 액세스를 허용하는 권한을 부여합니다.

이 정책에는 다음 키 액션 네임스페이스가 포함되어 있습니다.

- `cloudformation` - CloudFormation APIs에 대한 제한된 액세스를 AWS Proton CodeBuild 기반 프로비저닝에 허용합니다.
- `codebuild` - AWS Proton CodeBuild API에 대한 제한된 액세스를CodeBuild 기반 프로비저닝에 허용합니다. APIs
- `iam` – 관리자가 역할을 AWS Proton에 전달할 수 있습니다. 이는가 관리자를 대신하여 다른 서비스에 API 호출을 AWS Proton 할 수 있도록 하기 위해 필요합니다.
- `servicequotas` - AWS Proton 가 CodeBuild 동시 빌드 제한을 확인할 수 있도록 허용하여 적절한 빌드 대기열을 보장합니다.

자세한 내용은 AWSProtonCodeBuildProvisioningServiceRolePolicy를 참조하세요.

AWS 관리형 정책: AWSProtonServiceGitSyncServiceRolePolicy

AWS Proton 는이 서비스 동기화를 AWS Proton 수행하도록 허용하는 AWSServiceRoleForProtonServiceSync 서비스 연결 역할에이 정책을 연결합니다.

이 정책은 AWS Proton 작업 및가 AWS Proton 의존하는 다른 AWS 서비스 작업에 대한 제한된 액세스를 허용하는 권한을 부여합니다.

이 정책에는 다음 키 액션 네임스페이스가 포함되어 있습니다.

- `proton` - AWS Proton APIs에 대한 제한된 액세스 AWS Proton 동기화를 허용합니다.

자세한 내용은 AWSProtonServiceGitSyncServiceRolePolicy를 참조하세요.

AWS Proton AWS 관리형 정책에 대한 업데이트

이 서비스가 이러한 변경 사항을 추적하기 시작한 AWS Proton 이후부터의 AWS 관리형 정책 업데이트에 대한 세부 정보를 봅니다. 이 페이지의 변경 사항에 대한 자동 알림을 받으려면 AWS Proton 문서 기록 페이지에서 RSS 피드를 구독하세요.

변경	설명	Date
AWSProtonCodeBuildProvisioningServiceRolePolicy - 기존 정책에 대한 업데이트	가 CodeBuild 기반 프로비저닝을 AWS Proton 수행하도록 허용하는 서비스 연결 역할에 대한 관리형 정책은 이제 CloudFormation `TagResource` 및 `UntagResource` API 작업을 호출할 수 있는 권한을 부여합니다. 이러한 권한은 리소스에 대한 태그 지정 작업을 수행하는 데 필요합니다.	2024년 6월 15일
AWSProtonFullAccess - 기존 정책 업데이트	Git 리포지토리와 Git 동기화를 사용하는 서비스 연결 역할에 대한 관리형 정책이 두 서비스 접두사가 있는 리소스에 대해 업데이트되었습니다. 자세한 내용은 AWS CodeConnections 및 관리형 정책에 대한 서비스 연결 역할 사용을 참조하세요. https://docs.aws.amazon.com/dtconsole/latest/userguide/security-iam-awsmanpol.html	2024년 4월 25일

변경	설명	Date
AWSProtonDeveloperAccess - 기존 정책 업데이트	Git 리포지토리와 Git 동기화를 사용하는 서비스 연결 역할에 대한 관리형 정책이 두 서비스 접두사가 있는 리소스에 대해 업데이트되었습니다. 자세한 내용은 AWS CodeConnections 및 관리형 정책에 대한 서비스 연결 역할 사용을 참조하세요. https://docs.aws.amazon.com/dtconsole/latest/userguide/security-iam-awsmanpol.html	2024년 4월 25일
AWSProtonSyncServiceRolePolicy - 기존 정책 업데이트	Git 리포지토리와 Git 동기화를 사용하는 서비스 연결 역할에 대한 관리형 정책이 두 서비스 접두사가 있는 리소스에 대해 업데이트되었습니다. 자세한 내용은 AWS CodeConnections 및 관리형 정책에 대한 서비스 연결 역할 사용을 참조하세요. https://docs.aws.amazon.com/dtconsole/latest/userguide/security-iam-awsmanpol.html	2024년 4월 25일
AWSProtonCodeBuildProvisioningServiceRolePolicy - 기존 정책 업데이트	AWS Proton 는 CodeBuild 프로비저닝을 사용하기 위해 계정에 필요한 CodeBuild 동시 빌드 제한이 있는지 확인하는 권한을 추가하도록이 정책을 업데이트했습니다.	2023년 5월 12일

변경	설명	Date
AWSProtonServiceGitSyncServiceRolePolicy - 새 정책	AWS Proton 는가 서비스 동기화를 수행할 수 있도록 허용하는 새 정책을 추가 AWS Proton 했습니다. 해당 정책은 AWSServiceRoleForProtonServiceSync 서비스 연결에서 사용됩니다.	2023년 3월 31일
AWSProtonDeveloperAccess - 기존 정책 업데이트	AWS Proton 는 템플릿 요약, 배포된 템플릿 리소스 및 오래된 리소스를 볼 수 있는 새 `GetResourcesSummary` 작업을 추가했습니다.	2022년 11월 18일
AWSProtonReadOnlyAccess - 기존 정책 업데이트	AWS Proton 는 템플릿 요약, 배포된 템플릿 리소스 및 오래된 리소스를 볼 수 있는 새 `GetResourcesSummary` 작업을 추가했습니다.	2022년 11월 18일
AWSProtonCodeBuildProvisioningBasicAccess - 새 정책	AWS Proton 는 CodeBuild 에 빌드 for AWS Proton CodeBuild 프로비저닝을 실행하는 데 필요한 권한을 부여하는 새 정책을 추가했습니다.	2022년 11월 16일
AWSProtonSyncServiceRolePolicy - 새 정책	AWS Proton 는 AWS Proton 가 CodeBuild 기반 프로비저닝과 관련된 작업을 수행할 수 있도록 허용하는 새 정책을 추가했습니다. 해당 정책은 AWSServiceRoleForProtonCodeBuildProvisioning 서비스 연결에서 사용됩니다.	2022년 9월 2일

변경	설명	Date
AWSProtonFullAccess - 기존 정책 업데이트	AWS Proton 는 새 AWS Proton API 작업에 대한 액세스를 제공하고 일부 AWS Proton 콘솔 작업에 대한 권한 문제를 해결하기 위해이 정책을 업데이트했습니다.	2022년 3월 30일
AWSProtonDeveloperAccess - 기존 정책 업데이트	AWS Proton 이 정책을 업데이트하여 새 AWS Proton API 작업에 대한 액세스를 제공하고 일부 AWS Proton 콘솔 작업에 대한 권한 문제를 수정합니다.	2022년 3월 30일
AWSProtonReadOnlyAccess - 기존 정책 업데이트	AWS Proton 이 정책을 업데이트하여 새 AWS Proton API 작업에 대한 액세스를 제공하고 일부 AWS Proton 콘솔 작업에 대한 권한 문제를 수정합니다.	2022년 3월 30일
AWSProtonSyncServiceRolePolicy - 새 정책	AWS Proton 는가 템플릿 동기화와 관련된 작업을 수행할 수 있도록 허용하는 새 정책을 추가 AWS Proton 했습니다. 해당 정책은 AWSServiceRoleForProtonSync 서비스 연결에서 사용됩니다.	2021년 11월 23일
AWSProtonFullAccess - 새 정책	AWS Proton 는 AWS Proton API 작업 및 AWS Proton 콘솔에 대한 관리 역할 액세스를 제공하는 새 정책을 추가했습니다.	2021년 6월 9일

변경	설명	Date
AWSProtonDeveloperAccess - 새 정책	AWS Proton 는 AWS Proton API 작업 및 AWS Proton 콘솔에 대한 개발자 역할 액세스를 제공하는 새 정책을 추가했습니다.	2021년 6월 9일
AWSProtonReadOnlyAccess - 새 정책	AWS Proton 는 AWS Proton API 작업 및 AWS Proton 콘솔에 대한 읽기 전용 액세스를 제공하는 새 정책을 추가했습니다.	2021년 6월 9일
AWS Proton 가 변경 내용 추적을 시작했습니다.	AWS Proton 가 AWS 관리형 정책에 대한 변경 내용 추적을 시작했습니다.	2021년 6월 9일

에 대한 서비스 연결 역할 사용 AWS Proton

AWS Proton 는 AWS Identity and Access Management (IAM) 서비스 연결 역할을 사용합니다. 서비스 연결 역할은 직접 연결된 고유한 유형의 IAM 역할입니다 AWS Proton. 서비스 연결 역할은에서 사전 정의 AWS Proton 하며 서비스가 사용자를 대신하여 다른 AWS 서비스를 호출하는 데 필요한 모든 권한을 포함합니다.

주제

- AWS Proton 동기화에 역할 사용
- CodeBuild 기반 프로비저닝을 위한 역할 사용

AWS Proton 동기화에 역할 사용

AWS Proton 는 AWS Identity and Access Management (IAM) 서비스 연결 역할을 사용합니다. 서비스 연결 역할은 직접 연결된 고유한 유형의 IAM 역할입니다 AWS Proton. 서비스 연결 역할은에서 사전 정의 AWS Proton 하며 서비스가 사용자를 대신하여 다른 AWS 서비스를 호출하는 데 필요한 모든 권한을 포함합니다.

필요한 권한을 수동으로 추가할 필요가 없으므로 서비스 연결 역할을 더 AWS Proton 쉽게 설정할 수 있습니다.는 서비스 연결 역할의 권한을 AWS Proton 정의하며, 달리 정의되지 않은 한 만 해당 역할을 수임할 AWS Proton 수 있습니다. 정의된 권한에는 신뢰 정책과 권한 정책이 포함되며 이 권한 정책은 다른 IAM 엔터티에 연결할 수 없습니다.

먼저 관련 리소스를 삭제한 후에만 서비스 연결 역할을 삭제할 수 있습니다. 이렇게 하면 AWS Proton 리소스에 대한 액세스 권한을 실수로 제거할 수 없기 때문에 리소스가 보호됩니다.

서비스 연결 역할을 지원하는 다른 서비스에 대한 자세한 내용은 AWS IAM으로 작업하는 서비스를 참조하고 서비스 연결 역할 열에서 예인 서비스를 찾습니다. 해당 서비스에 대한 서비스 연결 역할 설명서를 보려면 예 링크를 선택합니다.

에 대한 서비스 연결 역할 권한 AWS Proton

AWS Proton 는 AWSServiceRoleForProtonSync 및 AWSServiceRoleForProtonServiceSync라는 두 가지 서비스 연결 역할을 사용합니다.

AWSServiceRoleForProtonSync 서비스 연결 역할은 역할을 수임하기 위해 다음 서비스를 신뢰합니다.

- `sync.proton.amazonaws.com`

라는 역할 권한 정책은가 지정된 리소스에서 다음 작업을 완료 AWS Proton 하도록 `AWSProtonSyncServiceRolePolicy` 허용합니다.

- 작업: AWS Proton 템플릿 및 템플릿 버전 생성, 관리 및 읽기
- 작업: CodeConnections에서 연결 사용

이 정책에 대한 자세한 내용은 AWS 관리형 정책: AWSProtonSyncServiceRolePolicy 섹션을 참조하세요.

AWSServiceRoleForProtonServiceSync 서비스 연결 역할은 역할을 수임하기 위해 다음 서비스를 신뢰합니다.

- `service-sync.proton.amazonaws.com`

라는 역할 권한 정책은가 지정된 리소스에서 다음 작업을 완료 AWS Proton 하도록 `AWSProtonServiceGitSyncServiceRolePolicy` 허용합니다.

- 작업: AWS Proton 서비스 및 서비스 인스턴스 생성, 관리 및 읽기

이 정책에 대한 자세한 내용은 AWS 관리형 정책: AWSProtonServiceGitSyncServiceRolePolicy 섹션을 참조하세요.

IAM 엔터티(사용자, 그룹, 역할 등)가 서비스 연결 역할을 생성하고 편집하거나 삭제할 수 있도록 권한을 구성할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 권한을 참조하세요.

에 대한 서비스 연결 역할 생성 AWS Proton

서비스 연결 역할은 수동으로 생성할 필요가 없습니다. AWS Management Console AWS CLI, 또는 AWS API AWS Proton AWS Proton 에서 동기화할 리포지토리 또는 서비스를 구성하면가 서비스 연결 역할을 생성합니다.

이 서비스 연결 역할을 삭제했다가 다시 생성해야 하는 경우 동일한 프로세스를 사용하여 계정에서 역할을 다시 생성할 수 있습니다. 동기화할 리포지토리 또는 서비스를 구성하면 AWS Proton가 서비스 연결 역할을 다시 AWS Proton 생성합니다.

AWSServiceRoleForProtonSync 서비스 연결 역할을 다시 생성하려면 동기화를 위한 리포지토리를 구성하고, AWSServiceRoleForProtonServiceSync를 다시 생성하려면 동기화를 위한 서비스를 구성해야 합니다.

에 대한 서비스 연결 역할 편집 AWS Proton

AWS Proton 에서는 AWSServiceRoleForProtonSync 서비스 연결 역할을 편집할 수 없습니다. 서비스 연결 역할을 생성한 후에는 다양한 엔터티가 역할을 참조할 수 있기 때문에 역할 이름을 변경할 수 없습니다. 하지만 IAM을 사용하여 역할의 설명을 편집할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 편집을 참조하세요.

에 대한 서비스 연결 역할 삭제 AWS Proton

AWSServiceRoleForProtonSync 역할을 수동으로 삭제할 필요가 없습니다. AWS CLI, 또는 AWS API 에서 리포지토리 동기화를 위해 AWS Proton 연결된 모든 리포지토리 AWS Management Console를 삭제하면 AWS Proton 가 리소스를 정리하고 서비스 연결 역할을 삭제합니다.

AWS Proton 서비스 연결 역할에 지원되는 리전

AWS Proton 는 서비스를 사용할 수 AWS 리전 있는 모든에서 서비스 연결 역할 사용을 지원합니다. 자세한 설명은 AWS 일반 참조의 AWS Proton 엔드포인트 및 할당량을 참조하세요.

CodeBuild 기반 프로비저닝을 위한 역할 사용

AWS Proton 는 AWS Identity and Access Management (IAM) 서비스 연결 역할을 사용합니다. 서비스 연결 역할은 직접 연결된 고유한 유형의 IAM 역할입니다 AWS Proton. 서비스 연결 역할은에서 사전 정의 AWS Proton 하며 서비스가 사용자를 대신하여 다른 AWS 서비스를 호출하는 데 필요한 모든 권한을 포함합니다.

필요한 권한을 수동으로 추가할 필요가 없으므로 서비스 연결 역할을 더 AWS Proton 쉽게 설정할 수 있습니다.는 서비스 연결 역할의 권한을 AWS Proton 정의하며, 달리 정의되지 않은 한 만 해당 역할을 수임할 AWS Proton 수 있습니다. 정의된 권한에는 신뢰 정책과 권한 정책이 포함되며 이 권한 정책은 다른 IAM 엔터티에 연결할 수 없습니다.

먼저 관련 리소스를 삭제한 후에만 서비스 연결 역할을 삭제할 수 있습니다. 이렇게 하면 AWS Proton 리소스에 대한 액세스 권한을 실수로 제거할 수 없기 때문에 리소스가 보호됩니다.

서비스 연결 역할을 지원하는 다른 서비스에 대한 자세한 내용은 AWS IAM으로 작업하는 서비스를 참조하고 서비스 연결 역할 열에서 예인 서비스를 찾습니다. 해당 서비스에 대한 서비스 연결 역할 설명서를 보려면 예 링크를 선택합니다.

에 대한 서비스 연결 역할 권한 AWS Proton

AWS Proton 는 AWSServiceRoleForProtonCodeBuildProvisioning - AWS Proton CodeBuild 프로비저닝을 위한 서비스 연결 역할이라는 서비스 연결 역할을 사용합니다.

AWSServiceRoleForProtonCodeBuildProvisioning 서비스 연결 역할은 역할을 수임하기 위해 다음 서비스를 신뢰합니다.

- `codebuild.proton.amazonaws.com`

라는 역할 권한 정책은가 지정된 리소스에서 다음 작업을 완료 AWS Proton 하도록 `AWSProtonCodeBuildProvisioningServiceRolePolicy` 허용합니다.

- 작업: CloudFormation 스택과 변환의 생성, 관리 및 읽기
- 작업: CodeBuild 프로젝트 및 빌드를 생성, 관리 및 읽기

이 정책에 대한 자세한 내용은 AWS 관리형 정책: AWSProtonCodeBuildProvisioningServiceRolePolicy 섹션을 참조하세요.

IAM 엔터티(사용자, 그룹, 역할 등)가 서비스 연결 역할을 생성하고 편집하거나 삭제할 수 있도록 권한을 구성할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 권한을 참조하세요.

에 대한 서비스 연결 역할 생성 AWS Proton

서비스 연결 역할은 수동으로 생성할 필요가 없습니다. 의 AWS Management Console AWS CLI, 또는 AWS API AWS Proton AWS Proton 에서 CodeBuild 기반 프로비저닝을 사용하는 환경을 생성하면가 서비스 연결 역할을 생성합니다.

이 서비스 연결 역할을 삭제했다가 다시 생성해야 하는 경우 동일한 프로세스를 사용하여 계정에서 역할을 다시 생성할 수 있습니다. 에서 CodeBuild 기반 프로비저닝을 사용하는 환경을 생성하면가 서비스 연결 역할을 다시 AWS Proton AWS Proton 생성합니다.

에 대한 서비스 연결 역할 편집 AWS Proton

AWS Proton 에서는 AWSServiceRoleForProtonCodeBuildProvisioning 서비스 연결 역할을 편집할 수 없습니다. 서비스 연결 역할을 생성한 후에는 다양한 개체가 역할을 참조할 수 있기 때문에 역할 이름을 변경할 수 없습니다. 하지만 IAM을 사용하여 역할의 설명을 편집할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 편집을 참조하세요.

에 대한 서비스 연결 역할 삭제 AWS Proton

서비스 연결 역할이 필요한 기능 또는 서비스가 더 이상 필요 없는 경우에는 해당 역할을 삭제하는 것이 좋습니다. 따라서 적극적으로 모니터링하거나 유지하지 않는 미사용 엔터티가 없도록 합니다. 그러나 수동으로 삭제 AWS Proton 하려면 먼저에서 CodeBuild 기반 프로비저닝을 사용하는 모든 환경 및 서비스(인스턴스 및 파이프라인)를 삭제해야 합니다.

수동으로 서비스 연결 역할 삭제

IAM 콘솔 AWS CLI, 또는 AWS API를 사용하여 AWSServiceRoleForProtonCodeBuildProvisioning 서비스 연결 역할을 삭제합니다. 자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 삭제를 참조하십시오.

AWS Proton 서비스 연결 역할에 지원되는 리전

AWS Proton 는 서비스를 사용할 수 AWS 리전 있는 모든에서 서비스 연결 역할 사용을 지원합니다. 자세한 내용은AWS 일반 참조에서 AWS Proton 엔드포인트 및 할당량을 참조하세요.

AWS Proton 자격 증명 및 액세스 문제 해결

다음 정보를 사용하여 및 IAM으로 작업할 때 발생할 수 있는 일반적인 문제를 진단 AWS Proton 하고 수정할 수 있습니다.

주제

- 에서 작업을 수행할 권한이 없음 AWS Proton
- iam:PassRole을 수행하도록 인증되지 않음
- 내 외부의 사람이 내 AWS Proton 리소스에 액세스 AWS 계정 하도록 허용하고 싶습니다.

에서 작업을 수행할 권한이 없음 AWS Proton

에서 작업을 수행할 권한이 없다는 AWS Management Console 메시지가 표시되면 관리자에게 문의하여 도움을 받아야 합니다. 관리자는 로그인 보안 인증 정보를 제공한 사람입니다.

다음 예제 오류는 `mateojackson` IAM 사용자가 콘솔을 사용하여 가상 *`my-example-widget`* 리소스에 대한 세부 정보를 보려고 하지만 가상 `proton:`*`GetWidget`* 권한이 없을 때 발생합니다.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 proton:GetWidget on resource: my-example-widget
```

이 경우, Mateo는 *`my-example-widget`* 작업을 사용하여 `proton:`*`GetWidget`* 리소스에 액세스하도록 허용하는 정책을 업데이트하라고 관리자에게 요청합니다.

iam:PassRole을 수행하도록 인증되지 않음

`iam:PassRole` 작업을 수행할 수 있는 권한이 없다는 오류가 수신되면 AWS Proton에 역할을 전달할 수 있도록 정책을 업데이트해야 합니다.

일부 AWS 서비스 에서는 새 서비스 역할 또는 서비스 연결 역할을 생성하는 대신 기존 역할을 해당 서비스에 전달할 수 있습니다. 이렇게 하려면 역할을 서비스에 전달할 권한이 있어야 합니다.

다음 예 오류는 `marymajor`라는 IAM 사용자가 콘솔을 사용하여 AWS Proton에서 작업을 수행하려고 하는 경우에 발생합니다. 하지만 작업을 수행하려면 서비스 역할이 부여한 권한이 서비스에 있어야 합니다. Mary는 서비스에 역할을 전달할 권한이 없습니다.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

이 경우, Mary가 `iam:PassRole`작업을 수행할 수 있도록 Mary의 정책을 업데이트해야 합니다.

도움이 필요한 경우 AWS 관리자에게 문의하세요. 관리자는 로그인 자격 증명을 제공한 사람입니다.

내 외부의 사람이 내 AWS Proton 리소스에 액세스 AWS 계정 하도록 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.

- 에서 이러한 기능을 AWS Proton 지원하는지 여부를 알아보려면 섹션을 참조하세요AWS Proton 에서 IAM을 사용하는 방법.
- 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.
- 타사에 리소스에 대한 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 타사가 AWS 계정 소유한에 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.

의 구성 및 취약성 분석 AWS Proton

AWS Proton 는 고객이 제공한 코드에 대한 패치 또는 업데이트를 제공하지 않습니다. 고객은에서 실행 중인 서비스 및 애플리케이션의 소스 코드 AWS Proton 와 서비스 및 환경 템플릿 번들에 제공된 코드를 포함하여 자체 코드에 패치를 업데이트하고 적용할 책임이 있습니다.

고객은 환경 및 서비스에서 인프라 리소스를 업데이트하고 패치를 적용할 책임이 있습니다. AWS Proton 는 리소스를 자동으로 업데이트하거나 패치를 적용하지 않습니다. 고객은 아키텍처의 리소스에 대한 설명서를 참조하여 각 아키텍처의 패치 정책을 이해해야 합니다.

고객이 요청한 환경 및 서비스 업데이트를 서비스 및 환경 템플릿의 마이너 버전에 제공하는 것 외에 AWS Proton 는 고객이 서비스 및 환경 템플릿과 템플릿 번들에서 정의하는 리소스에 대한 패치 또는 업데이트를 제공하지 않습니다.

자세한 내용은 다음 리소스를 참조하세요.

- 공동 책임 모델

- Web Services: 보안 프로세스의 개요

의 데이터 보호 AWS Proton

AWS Proton 는 AWS 공동 책임 모델데이터 보호에 대한 규정 및 지침을 포함하는 준수합니다. AWS 는 모든를 실행하는 글로벌 인프라를 보호할 책임이 있습니다 AWS 서비스.는이 인프라에서 호스팅되는 데이터에 대한 제어를 AWS 유지합니다. 고객 콘텐츠 및 개인 데이터 처리를 위한 보안 구성 제어 포함. AWS 고객 및 APN 파트너, 데이터 컨트롤러 또는 데이터 프로세서로 작동 는에 저장한 모든 개인 데이터에 대해 책임을 집니다. AWS 클라우드

데이터 보호를 위해 자격 AWS 계정 증명을 보호하고 AWS Identity and Access Management (IAM)를 사용하여 개별 사용자 계정을 설정하여 각 사용자에게 직무를 수행하는 데 필요한 권한만 부여하는 것이 좋습니다. 또한 다음과 같은 방법으로 데이터를 보호하는 것이 좋습니다.

- 각 계정에 다중 인증(MFA)을 사용하세요.
- SSL/TLS를 사용하여 AWS 리소스와 통신합니다. TLS 1.2 이상을 권장합니다.
- 를 사용하여 API 및 사용자 활동 로깅을 설정합니다 AWS CloudTrail.
- 내의 모든 기본 보안 제어와 함께 AWS 암호화 솔루션을 사용합니다 AWS 서비스.

이름 필드와 같은 자유 형식 필드에 고객 계정 번호와 같은 중요 식별 정보를 절대 입력하지 마십시오. 여기에는 AWS Proton 또는 기타에서 콘솔, API AWS CLI또는 AWS SDKs를 AWS 서비스 사용하여 작업하는 경우가 포함됩니다. 리소스 식별자 또는 AWS 리소스 관리와 관련된 유사한 항목의 자유 형식 텍스트 필드에 입력하는 모든 데이터는 진단 로그에 포함하기 위해 선택될 수 있습니다. 외부 서버에 URL을 제공할 때 해당 서버에 대한 요청을 검증하기 위해 자격 증명 정보를 URL에 포함시키지 마십시오.

데이터 보호에 대한 자세한 내용은 AWS 보안 블로그의 AWS 공동 책임 모델 및 GDPR 블로그 게시물을 참조하세요.

서버 측 저장 데이터 암호화

템플릿 번들을 저장하는 S3 버킷에 저장된 템플릿 번들의 민감한 데이터를 암호화하도록 선택한 경우 등록된 AWS Proton 템플릿에 연결할 수 있도록가 템플릿 번들을 AWS Proton 검색할 수 있도록 SSE-S3 또는 SSE-KMS 키를 사용해야 합니다.

전송 중 암호화

모든 서비스 대 서비스 통신은 SSL/TLS를 사용하여 전송 중에 암호화됩니다.

AWS Proton 암호화 키 관리

내에서 AWS Proton모든 고객 데이터는 기본적으로 AWS Proton 소유 키를 사용하여 암호화됩니다. 고객 소유 및 관리형 AWS KMS 키를 제공하는 경우 모든 고객 데이터는 다음 단락에 설명된 대로 고객이 제공한 키를 사용하여 암호화됩니다.

AWS Proton 템플릿을 생성할 때 키를 지정하고 자격 증명을 AWS Proton 사용하여가 키를 사용하도록 허용하는 권한 부여 AWS Proton 를 생성합니다.

권한 부여를 수동으로 사용 중지하거나 지정된 키를 비활성화 또는 삭제하면 AWS Proton 이 지정된 키로 암호화된 데이터를 읽을 수 없어 `ValidationException` 문제가 발생합니다.

AWS Proton 암호화 컨텍스트

AWS Proton 는 암호화 컨텍스트 헤더를 지원합니다. 암호화 컨텍스트는 데이터에 대한 추가 컨텍스트 정보를 포함할 수 있는 선택적 키-값 페어 집합입니다. 암호화 컨텍스트에 대한 일반 내용은 AWS Key Management Service 개발자 안내서의 AWS Key Management Service 개념 - 암호화 컨텍스트를 참조하세요.

암호화 컨텍스트는 보안되지 않은 임의의 데이터를 포함하는 키-값 페어 세트입니다. 데이터 암호화 요청에 암호화 컨텍스트를 포함하는 경우, AWS KMS 는 암호화된 데이터에 암호화 컨텍스트를 암호 방식으로 바인딩합니다. 따라서 동일한 암호화 컨텍스트로 전달해야 이 데이터를 해독할 수 있습니다.

고객은 암호화 컨텍스트를 사용하여 감사 기록 및 로그에서 고객 관리 키의 사용을 식별할 수 있습니다. AWS CloudTrail 및 CloudWatch Logs 같은 로그에서 일반 텍스트에 나타나기도 합니다.

AWS Proton 는 고객 지정 또는 외부 지정 암호화 컨텍스트를 사용하지 않습니다.

AWS Proton 는 다음과 같은 암호화 컨텍스트를 추가합니다.

```
{
  "aws:proton:template": "<proton-template-arn>",
  "aws:proton:resource": "<proton-resource-arn>"
}
```

첫 번째 암호화 컨텍스트는 리소스가 연결된 AWS Proton 템플릿을 식별하고 고객 관리형 키 권한 및 권한 부여에 대한 제약 조건 역할을 합니다.

두 번째 암호화 컨텍스트는 암호화된 AWS Proton 리소스를 식별합니다.

다음 예제에서는 AWS Proton 암호화 컨텍스트 사용을 보여줍니다.

서비스 인스턴스를 만드는 개발자.

```
{
  "aws:proton:template": "arn:aws:proton:region_id:123456789012:service-template/my-
template",
  "aws:proton:resource": "arn:aws:proton:region_id:123456789012:service/my-service/
service-instance/my-service-instance"
}
```

템플릿을 만드는 관리자.

```
{
  "aws:proton:template": "arn:aws:proton:region_id:123456789012:service-template/my-
template",
  "aws:proton:resource": "arn:aws:proton:region_id:123456789012:service-template/my-
template"
}
```

의 인프라 보안 AWS Proton

관리형 서비스인는 AWS 글로벌 네트워크 보안으로 보호 AWS Proton 됩니다. AWS 보안 서비스 및가 인프라를 AWS 보호하는 방법에 대한 자세한 내용은 AWS 클라우드 보안을 참조하세요. 인프라 보안 모범 사례를 사용하여 환경을 설계하려면 보안 원칙 AWS Well-Architected Framework의 인프라 보호를 참조하세요 AWS .

AWS 에서 게시한 API 호출을 사용하여 네트워크를 AWS Proton 통해에 액세스합니다. 클라이언트는 다음을 지원해야 합니다.

- Transport Layer Security(TLS). TLS 1.2는 필수이며 TLS 1.3을 권장합니다.
- DHE(Ephemeral Diffie-Hellman) 또는 ECDHE(Elliptic Curve Ephemeral Diffie-Hellman)와 같은 완전 전송 보안(PFS)이 포함된 암호 제품군. Java 7 이상의 최신 시스템은 대부분 이러한 모드를 지원합니다.

네트워크 격리를 개선하기 위해 다음 섹션에 설명된 AWS PrivateLink 대로를 사용할 수 있습니다.

AWS Proton 및 인터페이스 VPC 엔드포인트(AWS PrivateLink)

인터페이스 VPC 엔드포인트를 생성 AWS Proton 하여 VPC와 간에 프라이빗 연결을 설정할 수 있습니다. 인터페이스 엔드포인트는 인터넷 게이트웨이AWS PrivateLink, NAT 디바이스, VPN 연결 또는

AWS Direct Connect 연결 없이 비공개로 AWS Proton APIs에 액세스할 수 있는 기술로 구동됩니다. VPC의 인스턴스는 AWS Proton APIs. VPC와 간의 트래픽 AWS Proton 은 Amazon 네트워크를 벗어나지 않습니다.

각 인터페이스 엔드포인트는 서브넷에서 하나 이상의 Elastic Network Interfaces로 표현됩니다.

자세한 내용은 Amazon VPC 사용 설명서의 인터페이스 VPC 엔드포인트(AWS PrivateLink)를 참조하세요.

AWS Proton VPC 엔드포인트에 대한 고려 사항

에 대한 인터페이스 VPC 엔드포인트를 설정하기 전에 Amazon VPC 사용 설명서의 인터페이스 엔드포인트 속성 및 제한 사항을 검토해야 AWS Proton합니다.

AWS Proton 는 VPC에서 모든 API 작업을 호출할 수 있도록 지원합니다.

VPC 엔드포인트 정책은에 대해 지원됩니다 AWS Proton. 기본적으로 엔드포인트를 통해에 대한 전체 액세스 AWS Proton 가 허용됩니다. 자세한 내용은 Amazon VPC 사용 설명서의 VPC 엔드포인트를 통해 서비스에 대한 액세스 제어를 참조하세요.

에 대한 인터페이스 VPC 엔드포인트 생성 AWS Proton

Amazon VPC 콘솔 또는 ()를 사용하여 AWS Proton 서비스에 대한 VPC 엔드포인트를 생성할 수 있습니다 AWS Command Line Interface AWS CLI. 자세한 내용은 Amazon VPC 사용 설명서의 인터페이스 엔드포인트 생성을 참조하세요.

다음 서비스 이름을 AWS Proton 사용하여 용 VPC 엔드포인트를 생성합니다.

- com.amazonaws.*region*.proton

엔드포인트에 대해 프라이빗 DNS를 활성화하는 경우 리전의 기본 DNS 이름, 예를 들어를 AWS Proton 사용하여에 API 요청을 할 수 있습니다`proton.`*region*`.amazonaws.com`.

자세한 내용은 Amazon VPC 사용 설명서의 인터페이스 엔드포인트를 통해 서비스 액세스를 참조하세요.

에 대한 VPC 엔드포인트 정책 생성 AWS Proton

AWS Proton에 대한 액세스를 제어하는 VPC 엔드포인트에 엔드포인트 정책을 연결할 수 있습니다. 이 정책은 다음 정보를 지정합니다.

- 작업을 수행할 수 있는 위탁자.
- 수행할 수 있는 작업.
- 작업을 수행할 수 있는 리소스.

자세한 내용은 Amazon VPC 사용 설명서의 VPC 엔드포인트를 통해 서비스에 대한 액세스 제어를 참조하세요.

예: AWS Proton 작업에 대한 VPC 엔드포인트 정책

다음은에 대한 엔드포인트 정책의 예입니다 AWS Proton. 엔드포인트에 연결되면이 정책은 모든 리소스의 모든 보안 주체에 대해 나열된 AWS Proton 작업에 대한 액세스 권한을 부여합니다.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Principal": "*",
      "Action": [
        "proton:ListServiceTemplates",
        "proton:ListServiceTemplateMajorVersions",
        "proton:ListServiceTemplateMinorVersions",
        "proton:ListServices",
        "proton:ListServiceInstances",
        "proton:ListEnvironments",
        "proton:GetServiceTemplate",
        "proton:GetServiceTemplateMajorVersion",
        "proton:GetServiceTemplateMinorVersion",
        "proton:GetService",
        "proton:GetServiceInstance",
        "proton:GetEnvironment",
        "proton:CreateService",
        "proton:UpdateService",
        "proton:UpdateServiceInstance",
        "proton:UpdateServicePipeline",
        "proton:DeleteService"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
```

```
  ]
}
```

에서 로깅 및 모니터링 AWS Proton

모니터링은 AWS Proton 및 다른 AWS 솔루션의 안정성, 가용성 및 성능을 유지하는 데 중요한 부분입니다.는에서 실행되는 인스턴스를 모니터링하고 AWS Proton, 이상이 있을 때 이를 보고하고, 필요한 경우 자동 조치를 취할 수 있는 다음과 같은 모니터링 도구를 AWS 제공합니다.

현재 AWS Proton 자체는 Amazon CloudWatch Logs 또는와 통합되지 않습니다 AWS Trusted Advisor. 관리자는 CloudWatch를 구성하고 사용하여 서비스 및 환경 템플릿에 정의된 AWS 서비스 다른를 모니터링할 수 있습니다. AWS Proton 는와 통합됩니다 AWS CloudTrail.

- Amazon CloudWatch는 AWS 리소스와 AWS 에서 실행하는 애플리케이션을 실시간으로 모니터링합니다. 지표를 수집 및 추적하고, 사용자 지정 대시보드를 생성할 수 있으며, 지정된 지표가 지정한 임곗값에 도달하면 사용자에게 알리거나 조치를 취하도록 경보를 설정할 수 있습니다. 예를 들어 CloudWatch에서 Amazon EC2 인스턴스의 CPU 사용량 또는 기타 지표를 추적하고 필요할 때 자동으로 새 인스턴스를 시작할 수 있습니다. 자세한 내용은 CloudWatch 사용 설명서를 참조하세요.
- Amazon CloudWatch Logs로 Amazon EC2 인스턴스, CloudTrail, 기타 소스의 로그 파일을 모니터링, 저장 및 액세스할 수 있습니다. CloudWatch Logs는 로그 파일의 정보를 모니터링하고 특정 임곗값에 도달하면 사용자에게 알릴 수 있습니다. 또한 매우 내구력 있는 스토리지에 로그 데이터를 저장할 수 있습니다. 자세한 내용은 Amazon CloudWatch Logs 사용 설명서를 참조하세요.
- AWS CloudTrail는에 의해 또는를 대신하여 수행된 API 호출 및 관련 이벤트를 캡처 AWS 계정 하고 사용자가 지정한 Amazon S3 버킷으로 로그 파일을 전송합니다. 호출한 사용자 및 계정 AWS, 호출이 수행된 소스 IP 주소, 호출이 발생한 시기를 식별할 수 있습니다. 자세한 내용은 AWS CloudTrail 사용 설명서를 참조하십시오.
- EventBridge: 애플리케이션을 다양한 소스의 데이터와 쉽게 연결할 수 있는 서버리스 이벤트 버스 서비스입니다. EventBridge는 자체 애플리케이션, Software-as-a-Service(SaaS) 애플리케이션의 실시간 데이터 스트림을 제공하고 해당 데이터를 Lambda AWS 서비스 와 같은 대상으로 라우팅합니다. 이를 통해 서비스에서 발생하는 이벤트를 모니터링하고 이벤트 기반 아키텍처를 구축할 수 있습니다. 자세한 내용은 EventBridge AWS Proton 로 자동화 및 EventBridge 사용 설명서를 참조하세요.

의 복원력 AWS Proton

AWS 글로벌 인프라는 AWS 리전 및 가용 영역을 중심으로 구축됩니다.는 물리적으로 분리되고 격리된 여러 가용 영역을 AWS 리전제공하며,이 가용 영역은 지연 시간이 짧고 처리량이 높으며 중복성이

높은 네트워킹과 연결됩니다. 가용 영역을 사용하면 중단 없이 영역 간에 자동으로 장애 극복 조치가 이루어지는 애플리케이션 및 데이터베이스를 설계하고 운영할 수 있습니다. 가용 영역은 기존의 단일 또는 다중 데이터 센터 인프라보다 가용성, 내결함성, 확장성이 뛰어납니다.

AWS 리전및 가용 영역에 대한 자세한 내용은 AWS 글로벌 인프라를 참조하세요.

는 AWS 글로벌 인프라 외에도 데이터 복원력 및 백업 요구 사항을 지원하는 기능을 AWS Proton 제공합니다.

AWS Proton 백업

AWS Proton 는 모든 고객 데이터의 백업을 유지합니다. 전체 중단의 경우이 백업을 사용하여 이전의 유효한 상태에서 AWS Proton 및 고객 데이터를 복원할 수 있습니다.

에 대한 보안 모범 사례 AWS Proton

AWS Proton 는 자체 보안 정책을 개발하고 구현할 때 고려해야 할 보안 기능을 제공합니다. 다음 모범 사례는 일반적인 지침이며 완벽한 보안 솔루션을 나타내지는 않습니다. 이러한 모범 사례는 환경에 적절하지 않거나 충분하지 않을 수 있으므로 참고용으로만 사용해 주세요.

주제

- IAM을 사용하여 액세스 제어
- 템플릿 및 템플릿 번들에 자격 증명을 포함하지 마십시오.
- 암호화를 사용하여 민감한 데이터 보호
- API 호출을 보고 기록 AWS CloudTrail 하는 데 사용

IAM을 사용하여 액세스 제어

IAM은 사용자와 사용자의 권한을 관리하는 데 사용할 수 AWS 서비스 있는 입니다 AWS. 에서 IAM AWS Proton 을 사용하여 템플릿, 환경 또는 서비스 관리와 같이 관리자와 개발자가 수행할 수 있는 AWS Proton 작업을 지정할 수 있습니다. IAM 서비스 역할을 사용하여가 사용자를 대신하여 다른 서비스를 AWS Proton 호출하도록 허용할 수 있습니다.

AWS Proton 및 IAM 역할에 대한 자세한 내용은 섹션을 참조하세요에 대한 자격 증명 및 액세스 관리 AWS Proton.

최소 권한 액세스 구현. 자세한 내용은 AWS Identity and Access Management 사용 설명서에서 IAM의 정책 및 권한을 참조하세요.

템플릿 및 템플릿 번들에 자격 증명을 포함하지 마십시오.

CloudFormation 템플릿 및 템플릿 번들에 민감한 정보를 포함하는 대신 스택 템플릿에 동적 참조를 사용하는 것이 좋습니다.

동적 참조는 AWS Systems Manager Parameter Store 또는와 같은 다른 서비스에 저장되고 관리되는 외부 값을 참조할 수 있는 간단하고 강력한 방법을 제공합니다 AWS Secrets Manager. 동적 참조 사용 시 CloudFormation는 스택 및 변경 세트 작업 중에 필요한 경우 지정된 참조의 값을 검색하고 적절한 리소스에 값을 전달합니다. 하지만 CloudFormation은 실제 참조 값을 저장하지 않습니다. 자세한 내용은 CloudFormation 사용 설명서의 동적 참조를 사용하여 템플릿 값 지정을 참조하세요.

AWS Secrets Manager는 데이터베이스와 다른 서비스의 자격 증명을 안전하게 암호화, 저장 및 검색하는 데 효과적입니다. AWS Systems Manager 파라미터 스토어는 구성 데이터 관리를 위한 안전한 계층적 스토리지를 제공합니다.

템플릿 파라미터에 대한 자세한 내용은 CloudFormation 사용 설명서의 https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/parameters-section-structure.html를 참조하세요.

암호화를 사용하여 민감한 데이터 보호

내에서 AWS Proton모든 고객 데이터는 기본적으로 AWS Proton 소유 키를 사용하여 암호화됩니다.

플랫폼 팀원은에 고객 관리형 키를 제공하여 민감한 데이터를 AWS Proton 암호화하고 보호할 수 있습니다. S3 버킷에 민감한 저장 데이터를 암호화합니다. 자세한 내용은 의 데이터 보호 AWS Proton 단원을 참조하십시오.

API 호출을 보고 기록 AWS CloudTrail 하는 데 사용

AWS CloudTrail 는에서 API를 호출하는 모든 사용자를 추적합니다 AWS 계정. API 호출은 누구나 AWS Proton API, 콘솔 또는 AWS Proton AWS CLI 명령을 사용할 때마다 기록됩니다 AWS Proton . 로깅을 활성화하고 로그를 저장할 S3 버킷을 지정합니다. 이렇게 하면 필요한 경우 계정에서 누가 어떤 AWS Proton 호출을 했는지 감사할 수 있습니다. 자세한 내용은 에서 로깅 및 모니터링 AWS Proton 단원을 참조하십시오.

교차 서비스 혼동된 대리인 방지

혼동된 대리자 문제는 작업을 수행할 권한이 없는 엔터티가 권한이 더 많은 엔터티에게 작업을 수행하도록 강요할 수 있는 보안 문제입니다. 에서 AWS교차 서비스 가장은 혼동된 대리자 문제를 초래할 수

있습니다. 교차 서비스 가장은 한 서비스(직접 호출하는 서비스)가 다른 서비스(직접 호출되는 서비스)를 직접 호출할 때 발생할 수 있습니다. 직접 호출하는 서비스는 다른 고객의 리소스에 대해 액세스 권한이 없는 방식으로 작동하게 권한을 사용하도록 조작될 수 있습니다. 이를 방지하기 위해는 계정의 리소스에 대한 액세스 권한이 부여된 서비스 보안 주체를 사용하여 모든 서비스에 대한 데이터를 보호하는 데 도움이 되는 도구를 AWS 제공합니다.

리소스 정책에서 aws:SourceArn 및 aws:SourceAccount 전역 조건 컨텍스트 키를 사용하여 리소스에 다른 서비스를 AWS Proton 제공하는 권한을 제한하는 것이 좋습니다. 만약 aws:SourceArn 값에 Amazon S3 버킷 ARN과 같은 계정 ID가 포함되어 있지 않은 경우, 권한을 제한하려면 두 전역 조건 컨텍스트 키를 모두 사용해야 합니다. 두 전역 조건 컨텍스트 키와 계정을 포함한 aws:SourceArn 값을 모두 사용하는 경우, aws:SourceAccount 값 및 aws:SourceArn 값의 계정은 동일한 정책 명령문에서 사용할 경우 반드시 동일한 계정 ID를 사용해야 합니다. 하나의 리소스만 교차 서비스 액세스와 연결되도록 허용하려는 경우 aws:SourceArn을 사용하세요. 해당 계정의 모든 리소스가 교차 서비스 사용과 연결되도록 허용하려는 경우 aws:SourceAccount를 사용하세요.

의 값은가 AWS Proton 저장하는 리소스여야 aws:SourceArn 합니다.

혼동된 대리인 문제로부터 보호하는 가장 효과적인 방법은 리소스의 전체 ARN이 포함된 aws:SourceArn글로벌 조건 컨텍스트 키를 사용하는 것입니다. 리소스의 전체 ARN을 모를 경우 또는 여러 리소스를 지정하는 경우, ARN의 알 수 없는 부분에 대해 와일드카드(*)를 포함한 aws:SourceArn전역 조건 컨텍스트 키를 사용합니다. 예제: arn:aws::proton:*:*123456789012*:environment/*.

다음 예제에서는의 aws:SourceArn 및 aws:SourceAccount 전역 조건 컨텍스트 키를 사용하여 혼동된 대리자 문제를 AWS Proton 방지하는 방법을 보여줍니다.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Sid": "ExampleProtonConfusedDeputyPreventionPolicy",
        "Effect": "Allow",
        "Principal": {"Service": "proton.amazonaws.com"},
        "Action": "sts:AssumeRole",
        "Condition": {
            "StringEquals": {
                "aws:SourceAccount": "123456789012"
            },
            "ArnLike": {
```

```
                "aws:SourceArn": "arn:aws:proton:*:123456789012:environment/*"
            }
        }
    }
}
```

CodeBuild 프로비저닝 사용자 지정 VPC 지원

AWS Proton CodeBuild 프로비저닝은 AWS Proton 환경 계정에 있는 CodeBuild 프로젝트에서 고객이 제공한 임의의 CLI 명령을 실행합니다. 이러한 명령은 일반적으로 CDK와 같은 코드형 인프라(IaC) 도구를 사용하여 리소스를 관리합니다. VPC에 리소스가 있는 경우 CodeBuild에서 리소스에 액세스하지 못할 수 있습니다. 이를 위해 CodeBuild는 특정 VPC 내에서 실행할 수 있는 기능을 지원합니다. 몇 가지 사용 사례는 다음과 같습니다.

- Python용 PyPI, Java용 Maven 및 Node.js용 npm과 같은 자체 호스팅된 내부 아티팩트 리포지토리의 종속성을 검색합니다.
- CodeBuild는 파이프라인을 등록하기 위해 특정 VPC의 Jenkins 서버에 액세스해야 합니다.
- VPC 엔드포인트를 통해서만 액세스할 수 있도록 구성된 S3 버킷의 개체에 액세스합니다.
- 빌드에서 사설 서브넷에 격리된 RDS 데이터베이스의 데이터에 대해 통합 테스트를 실행합니다.

자세한 내용은 CodeBuild 및 VPC 설명서를 참조하세요.

CodeBuild 프로비저닝을 사용자 지정 VPC에서 실행하려면에서 간단한 솔루션을 AWS Proton 제공합니다. 먼저 VPC ID, 서브넷, 보안 그룹을 환경 템플릿에 추가해야 합니다. 그런 다음 환경 사양에 해당 값을 입력합니다. 이렇게 하면 지정된 VPC를 대상으로 하는 CodeBuild 프로젝트가 생성됩니다.

환경 템플릿 업데이트

스키마

VPC ID, 서브넷 및 보안 그룹은 환경 사양에 존재할 수 있도록 템플릿 스키마에 추가해야 합니다.

예시 schema.yaml:

```
schema:
  format:
    openapi: "3.0.0"
```

```
  environment_input_type: "EnvironmentInputType"
  types:
    EnvironmentInputType:
      type: object
      properties:
        codebuild_vpc_id:
          type: string
        codebuild_subnets:
          type: array
          items:
            type: string
        codebuild_security_groups:
          type: array
          items:
            type: string
```

이렇게 하면 매니페스트에서 사용할 세 가지 새 속성이 추가됩니다.

- `codebuild_vpc_id`
- `codebuild_subnets`
- `codebuild_security_groups`

매니페스트

CodeBuild에서 VPC 설정을 구성하려면 템플릿 매니페스트에서 `project_properties`라는 선택적 속성을 사용할 수 있습니다. CodeBuild 프로젝트를 생성하는 CloudFormation 스택에의 콘텐츠가 `project_properties` 추가됩니다. 이렇게 하면 Amazon VPC CloudFormation 속성뿐만 아니라 빌드 제한 시간과 같이 지원되는 CodeBuild CloudFormation 속성도 추가할 수 있습니다. `proton-inputs.json`에 제공된 것과 동일한 데이터를 `project_properties`의 값에도 사용할 수 있습니다.

이 섹션을 `manifest.yaml`에 추가합니다.

```
project_properties:
          VpcConfig:
            VpcId: "{{ environment.inputs.codebuild_vpc_id }}"
            Subnets: "{{ environment.inputs.codebuild_subnets }}"
            SecurityGroupIds: "{{ environment.inputs.codebuild_security_groups }}"
```

결과 `manifest.yaml`는 다음과 같을 수 있습니다.

```
infrastructure:
  templates:
    - rendering_engine: codebuild
      settings:
        image: aws/codebuild/amazonlinux2-x86_64-standard:4.0
        runtimes:
          nodejs: 16
        provision:
          - npm install
          - npm run build
          - npm run cdk bootstrap
          - npm run cdk deploy -- --require-approval never
        deprovision:
          - npm install
          - npm run build
          - npm run cdk destroy -- --force
        project_properties:
          VpcConfig:
            VpcId: "{{ environment.inputs.codebuild_vpc_id }}"
            Subnets: "{{ environment.inputs.codebuild_subnets }}"
            SecurityGroupIds: "{{ environment.inputs.codebuild_security_groups }}"
```

환경 생성

CodeBuild 프로비저닝 VPC 지원 템플릿으로 환경을 생성할 때는 VPC ID, 서브넷, 보안 그룹을 제공해야 합니다.

해당 지역의 모든 VPC ID 목록을 가져오려면 다음 명령을 실행합니다.

```
aws ec2 describe-vpcs
```

모든 서브넷 ID 목록을 얻으려면 다음을 실행합니다.

```
aws ec2 describe-subnets --filters "Name=vpc-id,Values=vpc-id"
```

Important

프라이빗 서브넷만 포함하세요. 퍼블릭 서브넷을 제공하면 CodeBuild가 실패합니다. 퍼블릭 서브넷에는 인터넷 게이트웨이로 연결되는 기본 경로가 있지만 프라이빗 서브넷은 그렇지 않습니다.

다음 명령을 실행하여 보안 그룹 ID를 가져옵니다. 이러한 ID는 AWS Management Console을 통해서도 얻을 수 있습니다.

```
aws ec2 describe-security-groups --filters "Name=vpc-id,Values=vpc-id"
```

값은 다음과 비슷합니다.

```
vpc-id: vpc-045ch35y28dec3a05
subnets:
  - subnet-04029a82e6ae46968
  - subnet-0f500a9294fc5f26a
 security-groups:
  - sg-03bc4c4ce32d67e8d
```

CodeBuild 권한 보장

VPC를 지원하려면 엘라스틱 네트워크 인터페이스를 생성하는 기능과 같은 특정 권한이 필요합니다.

콘솔에서 환경을 생성하는 경우 환경 생성 마법사 중에 이러한 값을 입력하세요. 프로그래밍 방식으로 환경을 만들려는 경우 `spec.yaml`은 다음과 같이 표시됩니다.

```
proton: EnvironmentSpec

spec:
  codebuild_vpc_id: vpc-045ch35y28dec3a05
  codebuild_subnets:
    - subnet-04029a82e6ae46968
    - subnet-0f500a9294fc5f26a
  codebuild_security_groups:
    - sg-03bc4c4ce32d67e8d
```

AWS Proton 리소스 및 태그 지정

AWS Proton Amazon 리소스 이름(ARN)이 할당된 리소스에는 환경 템플릿과 해당 메이저 및 마이너 버전, 서비스 템플릿과 해당 메이저 및 마이너 버전, 환경, 서비스, 서비스 인스턴스, 구성 요소 및 리포지토리가 포함됩니다. 이러한 리소스에 태그를 지정하면 정리하고 식별하는 데 도움이 됩니다. 태그를 사용하여 용도, 소유자, 환경 또는 기타 기준으로 리소스를 분류할 수 있습니다. 자세한 내용은 태그 지정 전략을 참조하세요. AWS Proton 리소스를 추적하고 관리하기 위해 다음 섹션에 설명된 태깅 기능을 사용할 수 있습니다.

AWS 태그 지정

AWS 리소스에 태그 형태로 메타데이터를 할당할 수 있습니다. 각 태그는 사용자가 정의하는 키와 선택적 값으로 구성됩니다. 태그를 사용하면 리소스를 손쉽게 관리, 식별, 정리, 검색 및 필터링할 수 있습니다.

> **Important**
>
> 개인 식별 정보(PII)나 기타 기밀 정보 또는 민감한 정보를 태그에 추가하지 않습니다. 청구를 비롯한 여러 AWS 서비스에서 태그에 액세스할 수 있습니다. 태그는 프라이빗 또는 민감한 데이터에 사용하기 위한 것이 아닙니다.

각 태그는 두 부분으로 구성됩니다.

- 태그 키(예제: `CostCenter`, `Environment` 또는 `Project`). 태그 키는 대/소문자를 구별합니다.
- 태그 값(선택 사항)(예: `111122223333` 또는 `Production`). 태그 키처럼 태그 값은 대/소문자를 구별합니다.

태그에 적용되는 기본 이름 지정 및 사용 요구 사항은 다음과 같습니다.

- 각 리소스에는 최대 50개 사용자 생성 태그가 포함될 수 있습니다.

> **Note**
>
> `aws:` 접두사로 시작하는 시스템 생성 태그는 AWS 사용하도록 예약되어 있으며이 제한에 포함되지 않습니다. `aws:` 접두사로 시작하는 태그를 편집하거나 삭제할 수 없습니다.

- 각 리소스에 대해 각 태그 키는 고유해야 하며 각 태그 키는 하나의 값만 가질 수 있습니다.
- 태그 키는 UTF-8 형식의 최소 1자에서 최대 128자여야 합니다.
- 태그 값은 최소 1자, 최대 256자의 UTF-8 형식 유니코드 문자로 지정해야 합니다.
- 허용되는 문자는 UTF-8로 표현할 수 있는 문자, 숫자, 공백 및 * _ . : / = + - @ 문자입니다.

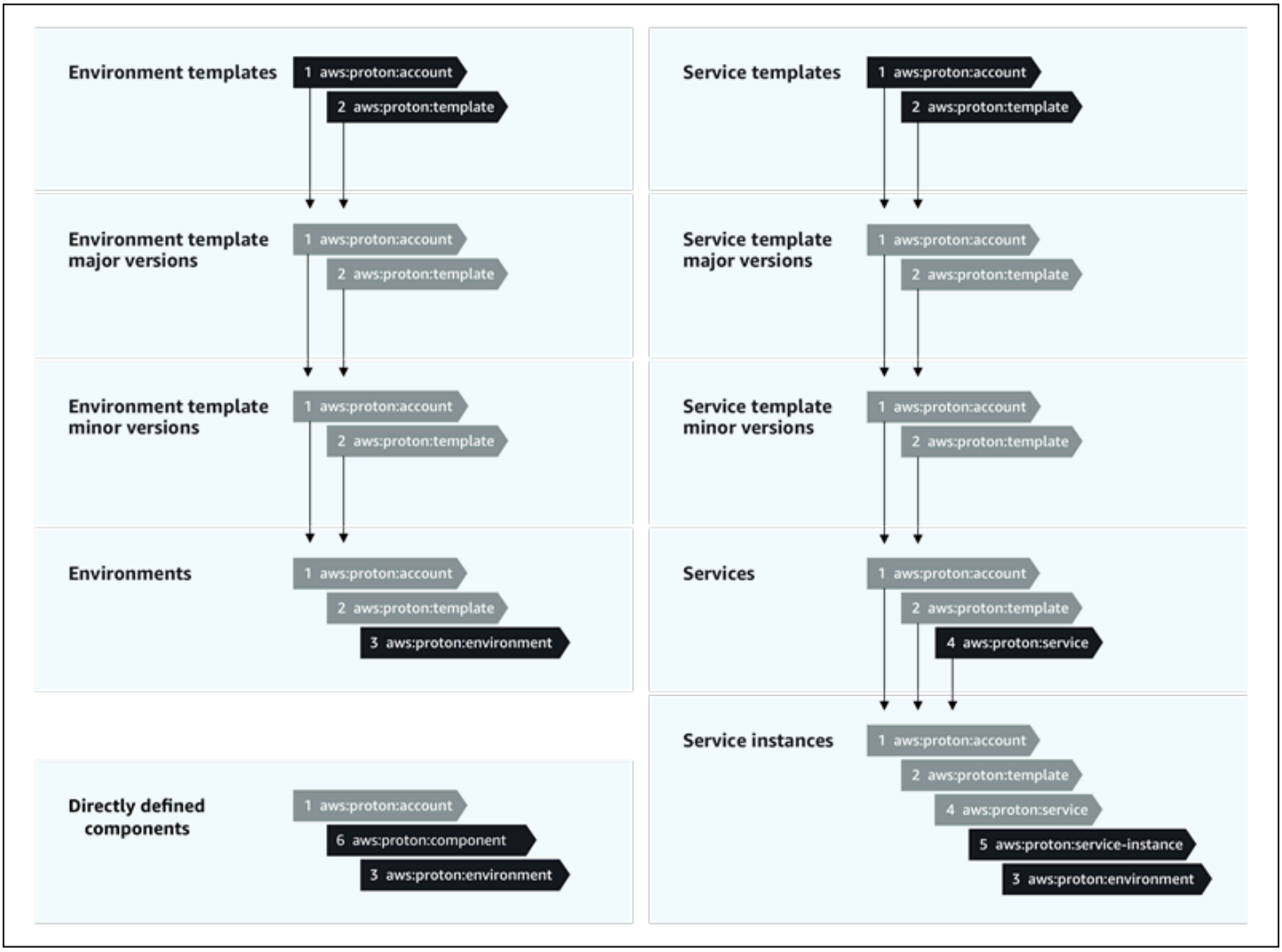
AWS Proton 태그 지정

를 사용하면 생성한 태그와가 AWS Proton 자동으로 생성하는 태그를 모두 사용할 AWS Proton수 있습니다.

AWS Proton AWS 관리형 태그

AWS Proton 리소스를 생성하면는 다음 다이어그램과 같이 새 리소스에 대한 AWS 관리형 태그를 AWS Proton 자동으로 생성합니다. AWS 관리형 태그는 나중에 새 AWS Proton 리소스를 기반으로 하는 다른 리소스로 전파됩니다. 예를 들어 환경 템플릿의 관리 태그는 해당 버전으로 전파되고 서비스의 관리 태그는 해당 서비스 인스턴스로 전파됩니다.

> **Note**
>
> AWS 관리형 태그는 환경 계정 연결에 대해 생성되지 않습니다. 자세한 내용은 the section called “계정 연결” 단원을 참조하십시오.



프로비저닝된 리소스에 태그 전파

서비스 및 환경 템플릿에 정의된 리소스와 같은 프로비저닝된 리소스가 AWS 태그 지정을 지원하는 경우 AWS 관리형 태그는 고객 관리형 태그로 프로비저닝된 리소스에 전파됩니다. 이러한 태그는 AWS 태그 지정을 지원하지 않는 프로비저닝된 리소스로 전파되지 않습니다.

AWS Proton 는 다음 표에 설명된 대로 AWS Proton 계정, 등록된 템플릿, 배포된 환경, 서비스 및 서비스 인스턴스별로 리소스에 태그를 적용합니다. AWS 관리형 태그를 사용하여 AWS Proton 리소스를 보고 관리할 수 있지만 수정할 수는 없습니다.

AWS 관리형 태그 키	전파 고객 관리형 키 전파	설명
`aws:proton:account`	`proton:account`	AWS Proton 리소스를 생성하고 배포하는 AWS 계정입니다.

AWS 관리형 태그 키	전파 고객 관리형 키 전파	설명
aws:proton:template	proton:template	선택한 템플릿의 ARN.
aws:proton:environment	proton:environment	선택한 환경의 ARN.
aws:proton:service	proton:service	선택한 서비스의 ARN.
aws:proton:service-instance	proton:service-instance	선택한 서비스 인스턴스의 ARN.
aws:proton:component	proton:component	선택한 구성 요소의 ARN.

다음은 AWS Proton 리소스에 대한 AWS 관리형 태그의 예입니다.

```
"aws:proton:template" = "arn:aws:proton:region-id:account-id:environment-template/env-template"
```

다음은 관리형 태그에서 전파된 프로비저닝된 리소스에 적용된 고객 AWS 관리형 태그의 예입니다.

```
"proton:environment:database" = "arn:aws:proton:region-id:account-id:rds/env-db"
```

AWS관리형 프로비저닝을 사용하면 전파된 태그를 프로비저닝된 리소스에 직접 AWS Proton 적용합니다.

자체 관리형 프로비저닝을 사용하면는 프로비저닝 풀 요청(PR)에서 제출하는 렌더링된 IaC 파일과 함께 전파된 태그를 사용할 수 있도록 AWS Proton 합니다. 태그는 문자열 맵 변수 proton_tags에 제공됩니다. 에 AWS Proton 태그를 포함하려면 Terraform 구성에서이 변수를 참조하는 것이 좋습니다default_tags. 이렇게 하면 프로비저닝된 모든 리소스에 AWS Proton 태그가 전파됩니다.

다음 예제는 환경 Terraform 템플릿에서의 태그 전파 방법을 보여줍니다.

proton_tags 변수 정의는 다음과 같습니다.

proton.environment.variables.tf:

```
variable "environment" {
```

```
  type = object({
    inputs = map(string)
    name = string
  })
}

variable "proton_tags" {
  type = map(string)
  default = null
}
```

이 변수에 태그 값을 할당하는 방법은 다음과 같습니다.

proton.auto.tfvars.json:

```
{
  "environment": {
    "name": "dev",
    "inputs": {
      "ssm_parameter_value": "MyNewParamValue"
    }
  }

  "proton_tags" : {
    "proton:account" : "123456789012",
    "proton:template" : "arn:aws:proton:us-east-1:123456789012:environment-template/
fargate-env",
    "proton:environment" : "arn:aws:proton:us-east-1:123456789012:environment/dev"
  }
}
```

프로비저닝된 리소스에 추가되도록 Terraform 구성에 AWS Proton 태그를 추가하는 방법은 다음과 같습니다.

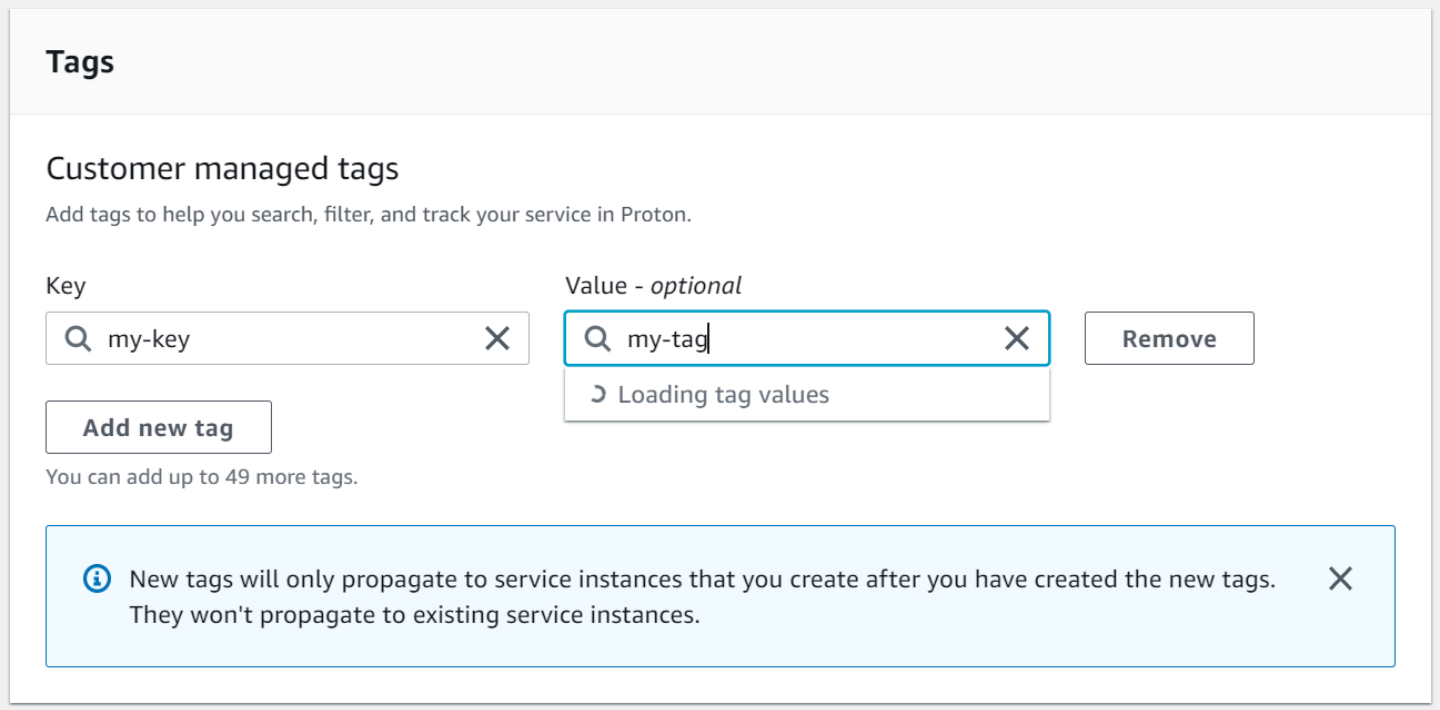
```
# Configure the AWS Provider
provider "aws" {
  region = var.aws_region
  default_tags {
    tags = var.proton_tags
  }
}
```

고객 관리형 태그

각 AWS Proton 리소스에는 최대 50개의 고객 관리형 태그 할당량이 있습니다. 고객 관리형 태그는 기존 AWS Proton 리소스 또는 프로비저닝된 리소스로 전파되지 않는다는 점을 제외하면 AWS 관리형 태그와 동일한 방식으로 하위 AWS Proton 리소스로 전파됩니다. 기존 하위 리소스가 AWS Proton 있는 리소스에 새 태그를 적용하고 기존 하위 리소스에 새 태그로 태그를 지정하려면 콘솔 또는를 사용하여 기존 각 하위 리소스에 수동으로 태그를 지정해야 합니다 AWS CLI.

콘솔과 CLI를 사용하여 태그 생성

콘솔을 사용하여 AWS Proton 리소스를 생성하면 다음 콘솔 스냅샷과 같이 생성 절차의 첫 번째 또는 두 번째 페이지에서 고객 관리형 태그를 생성할 수 있습니다. 새 태그 추가를 선택하고 키 및 값을 입력하고 진행합니다.



AWS Proton 콘솔을 사용하여 새 리소스를 생성한 후 세부 정보 페이지에서 AWS 관리형 및 고객 관리형 태그 목록을 볼 수 있습니다.

태그 생성 및 편집

1. AWS Proton 콘솔에서 태그 목록을 볼 수 있는 AWS Proton 리소스 세부 정보 페이지를 엽니다.
2. 태그 관리를 선택합니다.

3. 태그 관리 페이지에서 태그를 보고, 생성하고, 제거하고, 편집할 수 있습니다. 상단에 나열된 AWS 관리형 태그는 수정할 수 없습니다. 그러나 관리형 태그 뒤에 나열된 편집 필드를 사용하여 고객 AWS 관리형 태그를에 추가하고 수정할 수 있습니다.

   새 태그를 추가하려면 태그 생성을 선택합니다.
4. 새 태그의 키와 값을 입력합니다.
5. 태그를 편집하려면 선택한 키의 태그 값 필드에 값을 입력합니다.
6. 태그를 제거하려면 태그를 선택하고 삭제를 선택합니다.
7. 변경을 완료했으면 변경사항 저장을 선택합니다.

를 사용하여 태그 생성 AWS Proton AWS CLI

를 사용하여 태그를 확인, 생성, 제거 및 편집할 수 있습니다 AWS Proton AWS CLI.

다음 예와 같이 리소스에 태그 생성 또는 편집이 가능합니다.

```
$ aws proton tag-resource \
    --resource-arn "arn:aws:proton:region-id:account-id:service-template/webservice" \
    --tags '[{"key":"mykey1","value":"myval1"},{"key":"mykey2","value":"myval2"}]'
```

다음 예와 같이 리소스의 태그를 제거할 수 있습니다.

```
$ aws proton untag-resource \
    --resource-arn "arn:aws:proton:region-id:account-id:service-template/webservice" \
    --tag-keys '["mykey1","mykey2"]'
```

마지막 예와 같이 리소스의 태그를 나열할 수 있습니다.

```
$ aws proton list-tags-for-resource \
    --resource-arn "arn:aws:proton:region-id:account-id:service-template/webservice"
```

문제 해결 AWS Proton

의 문제를 해결하는 방법을 알아봅니다 AWS Proton.

주제

- CloudFormation 동적 파라미터를 참조하는 배포 오류

CloudFormation 동적 파라미터를 참조하는 배포 오류

CloudFormation 동적 변수를 참조하는 배포 오류가 표시되는 경우 해당 오류가 Jinja 이스케이프 변수인지 확인하세요. 이러한 오류는 Jinja가 동적 변수를 잘못 해석하여 발생할 수 있습니다. CloudFormation 동적 파라미터 구문은 AWS Proton 파라미터와 함께 사용하는 Jinja 구문과 매우 유사합니다.

CloudFormation 동적 변수 구문의 예:

`'{{resolve:secretsmanager:MySecret:SecretString:password:EXAMPLE1-90ab-cdef-fedc-ba987EXAMPLE}}'`.

예제 AWS Proton 파라미터 Jinja 구문:

`'{{ service_instance.environment.outputs.env-outputs }}'`.

이러한 오해 오류를 방지하기 위해 Jinja는 다음 예와 같이 CloudFormation 동적 파라미터를 이스케이프합니다.

이 예제는 CloudFormation 사용 설명서에서 가져온 것입니다. AWS Secrets Manager secret-name 및 json-key 세그먼트를 사용하여 보안 암호에 저장된 로그인 자격 증명을 검색할 수 있습니다.

```
MyRDSInstance:
  Type: AWS::RDS::DBInstance
  Properties:
    DBName: 'MyRDSInstance'
      AllocatedStorage: '20'
      DBInstanceClass: db.t2.micro
      Engine: mysql
      MasterUsername: '{{resolve:secretsmanager:MyRDSSecret:SecretString:username}}'
      MasterUserPassword:
 '{{resolve:secretsmanager:MyRDSSecret:SecretString:password}}'
```

CloudFormation 동적 파라미터를 이스케이프하려면 다음 두 가지 방법을 사용할 수 있습니다.

- `{% raw %} and {% endraw %}` 사이 블록을 묶습니다.

```
'{% raw %}'
MyRDSInstance:
  Type: AWS::RDS::DBInstance
  Properties:
    DBName: 'MyRDSInstance'
      AllocatedStorage: '20'
      DBInstanceClass: db.t2.micro
      Engine: mysql
      MasterUsername: '{{resolve:secretsmanager:MyRDSSecret:SecretString:username}}'
      MasterUserPassword:
 '{{resolve:secretsmanager:MyRDSSecret:SecretString:password}}'
'{% endraw %}'
```

- `"{{ }}"` 사이 파라미터를 묶습니다.

```
MyRDSInstance:
  Type: AWS::RDS::DBInstance
  Properties:
    DBName: 'MyRDSInstance'
      AllocatedStorage: '20'
      DBInstanceClass: db.t2.micro
      Engine: mysql
      MasterUsername:
 "{{ '{{resolve:secretsmanager:MyRDSSecret:SecretString:username}}' }}"
      MasterUserPassword:
 "{{ '{{resolve:secretsmanager:MyRDSSecret:SecretString:password}}' }}"
```

자세한 내용은 Jinja 이스케이프 참조하세요.

AWS Proton 할당량

다음 표에는 할당 AWS Proton 량이 나열되어 있습니다. 모든 값은 지원되는 AWS 리전별로 AWS 계정 당입니다.

리소스 할당량	기본 한도	조정 가능?
템플릿 번들의 최대 크기	10MB	X 아니요
템플릿 매니페스트 파일의 최대 크기	2MB	X 아니요
템플릿 스키마 파일의 최대 크기	2MB	X 아니요
각 템플릿 파일의 최대 크기	2MB	X 아니요
각 템플릿 이름의 최대 길이	100자	X 아니요
번들당 최대 CloudFormation 템플릿 파일 수	1	X 아니요
계정당 등록된 템플릿의 최대 수, 서비스 및 환경 템플릿 합산	1000	✓ 예
템플릿당 등록된 템플릿 버전의 최대 수	1000	✓ 예
CodeBuild 프로비저닝 번들당 최대 파일 수	500	X 아니요
계정당 최대 환경 수	1000	✓ 예
계정당 최대 서비스 수	1000	✓ 예
서비스당 최대 서비스 인스턴스 수	20	✓ 예
계정당 최대 구성 요소 수	1000	✓ 예
환경 계정당 최대 환경 계정 연결 수	1000	✓ 예

문서 이력

다음 표에서는 최신 릴리스 AWS Proton 및 고객 피드백과 관련된 설명서의 중요한 변경 사항을 설명합니다. 이 설명서에 대한 업데이트 알림을 받으려면 RSS 피드를 구독하면 됩니다.

- API 버전: 2020-07-20

변경 사항	설명	날짜
지원 종료 알림	2026년 10월 7일에 AWS 는에 대한 지원을 종료합니다 AWS Proton.	2025년 10월 7일
관리형 정책 업데이트	`AWSProtonCodeBuildProvisioningServiceRolePolicy` 정책이 업데이트되었습니다.	2024년 6월 15일
관리형 정책 업데이트	`AWSProtonDeveloperAccess` 정책이 업데이트되었습니다.	2024년 4월 25일
관리형 정책 업데이트	`AWSProtonFullAccess` 정책이 업데이트되었습니다.	2024년 4월 25일
관리형 정책 업데이트	`AWSProtonSyncServiceRolePolicy` 정책이 업데이트되었습니다.	2024년 4월 25일
관리형 정책 업데이트	`AWSProtonCodeBuildProvisioningServiceRolePolicy` 정책이 업데이트되었습니다.	2023년 5월 12일
서비스 동기화 구성	AWS Proton 는 서비스 동기화 구성에 대한 지원을 추가합니다.	2023년 3월 31일

CodeBuild	AWS Proton 는 CodeBuild 프로비저닝에 대한 지원을 추가합니다.	2022년 11월 16일
관리형 정책 업데이트	CodeBuild 프로비저닝을 위한 빌드를 실행하는 데 필요한 권한을 AWS Proton CodeBuild 에 부여하는 AWSProtonCodeBuildProvisioningBasicAccess 정책이 추가되었습니다.	2022년 11월 11일
Terraform 태그 전파	태깅 장에 Terraform 태그 전파를 추가했습니다.	2022년 9월 16일
API 마이그레이션 가이드	API 마이그레이션 가이드가 제거되었습니다.	2022년 8월 12일
AWS Proton 객체	AWS Proton 객체 및 다른 객체 AWS 및 타사 객체와의 관계에 대한 주제를 추가했습니다. AWS Proton 개체를 참조하세요.	2022년 7월 29일
링크된 리포지토리 분류	가이드 전반에 걸쳐 연결된 (등록된) 저장소의 목적과 사용법을 명확히 설명했습니다.	2022년 7월 18일
안내서 병합	두 개의 개별 관리자 및 사용자 안내서를 단일 안내서인 AWS Proton 사용자 안내서에 통합했습니다.	2022년 6월 30일

관리형 정책 업데이트	새 AWS Proton API 작업에 대한 액세스를 제공하고 일부 AWS Proton 콘솔 작업에 대한 권한 문제를 해결하도록 관리형 정책을 업데이트했습니다. 에 AWS 대한 관리형 정책을 AWS Proton 참조하세요.	2022년 6월 20일
CLI 시작하기	새 템플릿 라이브러리를 사용하는 새 자습서로 AWS CLI시작하기를 업데이트했습니다.	2022년 6월 14일
직접 정의된 구성 요소	구성 요소 장을 추가하고 안내서 전체에서 관련 내용을 수정했습니다.	2022년 6월 1일
AWS Proton 템플릿 라이브러리	AWS Proton 템플릿 라이브러리 주제가 추가되었습니다.	2022년 5월 6일
테라폼 정식 버전(GA)	풀 리퀘스트 프로비저닝의 이름을 자체 관리형 프로비저닝으로 변경했습니다. 프로비저닝 방법 주제를 추가했습니다.	2022년 3월 23일
리포지토리 태그 지정	데이터 스트림 리소스 태깅에 대한 지원이 추가되었습니다. 리포지토리로 연결되는 링크 생성을 참고합니다.	2022년 3월 23일
문서 업데이트	환경 계정 연결 태깅을 추가했습니다.	2021년 11월 26일

템플릿 동기화 및 Terraform 미리 보기	정식 버전을 위한 템플릿 동기화 특성이 포함된 자동 템플릿 버전 관리 및 미리 보기에서 Terraform을 통한 풀 리퀘스트 프로비저닝이 추가되었습니다. API 마이그레이션 가이드를 다시 읽어보세요.	2021년 11월 24일
설명서 업데이트	EventBridge 자습서, 워크플로 시작하기, AWS Proton 작동 방식 및 템플릿 번들 섹션 개선 사항이 추가되었습니다.	2021년 9월 17일
AWS Proton 콘솔 도움말 패널 릴리스	도움말 패널이 콘솔에 추가되었습니다. 콘솔 템플릿 버전 삭제 시 더 이상 하위 버전이 삭제되지 않습니다. API 마이그레이션 가이드가 제거되었습니다.	2021년 9월 8일
AWS Proton 정식 출시(GA) 릴리스	교차 계정 환경, EventBridge 모니터링, IAM 조건 키, 멱등성 지원이 추가되고 할당량이 증가했습니다.	2021년 6월 9일

서비스에 대한 서비스 인스턴스 추가 및 삭제 및 환경에 기존 외부 인프라 사용 AWS Proton	이 공개 평가판 릴리스에는 서비스에서 서비스 인스턴스를 추가 및 삭제하고, AWS Proton 환경에서 기존 외부 인프라를 사용하고, 환경, 서비스 인스턴스 및 파이프라인 배포를 취소할 수 있는 업데이트가 포함되어 있습니다. AWS Proton 이제는 PrivateLink를 지원합니다. 리소스가 마이너 버전을 사용하는 동안 실수로 마이너 버전이 삭제되는 것을 방지하기 위해 추가 삭제 확인 기능이 추가되었습니다.	2021년 4월 27일
를 사용하여 태그 지정 AWS Proton	공개 평가판 릴리스 2에는 AWS Proton 태그 지정과 서비스 파이프라인 없이 서비스를 시작할 수 있는 기능이 포함되어 있습니다.	2021년 3월 5일
최초 릴리스	이제 일부 지역에서 공개 미리 보기 버전을 사용할 수 있습니다.	2020년 12월 1일

AWS 용어집

최신 AWS 용어는 AWS 용어집 참조의 AWS 용어집을 참조하세요.