aws

개발자 가이드

AWS App Runner

AWS App Runner: 개발자 가이드

Table of Contents

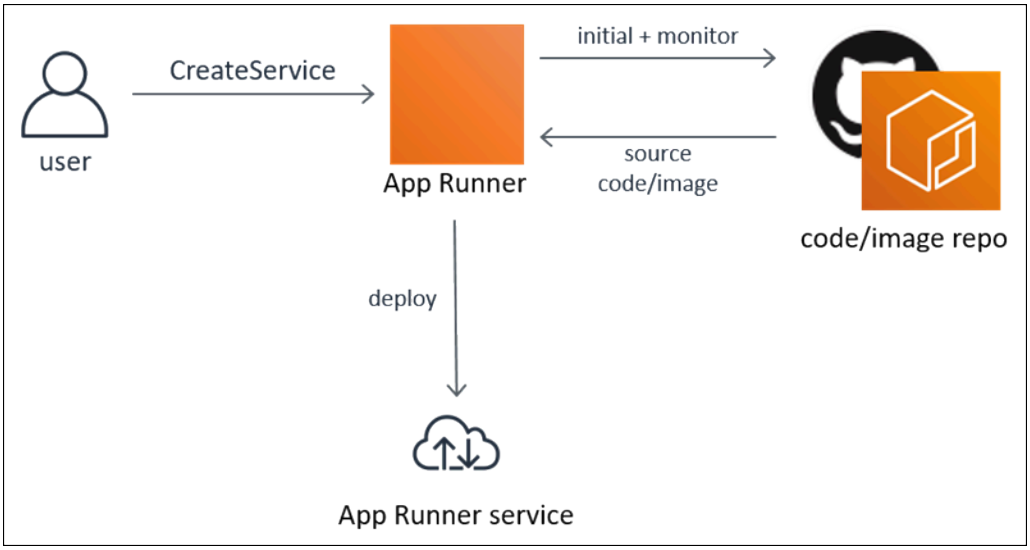
AWS App Runner 는 2026년 4월 30일부터 신규 고객에게 더 이상 공개되지 않습니다. App Runner를 사용하려면 해당 날짜 이전에 가입하세요. 기존 고객은 정상적으로 서비스를 계속 이용할 수 있습니다. 자세한 내용은 AWS App Runner 가용성 변경을 참조하세요.

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

AWS App Runner란 무엇인가요?

AWS App Runner 는 소스 코드 또는 컨테이너 이미지에서 AWS 클라우드의 확장 가능하고 안전한 웹 애플리케이션으로 직접 배포할 수 있는 빠르고 간단하며 비용 효율적인 방법을 제공하는 AWS 서비스입니다. 새로운 기술을 배우거나, 사용할 컴퓨팅 서비스를 결정하거나, AWS 리소스를 프로비저닝하고 구성하는 방법을 알 필요가 없습니다.

App Runner는 코드 또는 이미지 리포지토리에 직접 연결됩니다. 완전 관리형 운영, 고성능, 확장성 및 보안을 갖춘 자동 통합 및 전송 파이프라인을 제공합니다.



App Runner는 누구를 위한 것입니까?

개발자인 경우 App Runner를 사용하여 코드 또는 이미지 리포지토리의 새 버전을 배포하는 프로세스를 간소화할 수 있습니다.

운영 팀의 경우 App Runner는 커밋이 코드 리포지토리로 푸시되거나 새 컨테이너 이미지 버전이 이미지 리포지토리로 푸시될 때마다 자동 배포를 활성화합니다.

App Runner 액세스

다음 인터페이스 중 하나를 사용하여 App Runner 서비스 배포를 정의하고 구성할 수 있습니다.

- App Runner 콘솔 - App Runner 서비스를 관리하기 위한 웹 인터페이스를 제공합니다.
- App Runner API - App Runner 작업을 수행하기 위한 RESTful API를 제공합니다. 자세한 내용은 AWS App Runner API 참조를 참조하세요.

- AWS 명령줄 인터페이스(AWS CLI) - Amazon VPC를 비롯한 다양한 AWS 서비스에 대한 명령을 제공하며 Windows, macOS 및 Linux에서 지원됩니다. 자세한 내용은 AWS Command Line Interface 단원을 참조하십시오.
- AWS SDKs- 언어별 APIs 제공하고 서명 계산, 요청 재시도 처리, 오류 처리와 같은 많은 연결 세부 정보를 처리합니다. 자세한 정보는 AWS SDK를 참조하세요.

App Runner 요금

App Runner는 애플리케이션을 실행하는 비용 효율적인 방법을 제공합니다. App Runner 서비스가 사용하는 리소스에 대해서만 비용을 지불합니다. 요청 트래픽이 낮을 때 서비스가 더 적은 컴퓨팅 인스턴스로 스케일 다운됩니다. 프로비저닝된 인스턴스의 가장 낮은 수와 가장 높은 수, 인스턴스가 처리하는 가장 높은 로드 등 확장성 설정을 제어할 수 있습니다.

App Runner 자동 조정에 대한 자세한 내용은 섹션을 참조하세요the section called “Auto Scaling”.

요금 정보는 AWS App Runner 요금을 참조하세요.

다음에 있는 것

다음 주제에서 App Runner를 시작하는 방법을 알아봅니다.

- 설정 - App Runner를 사용하기 위한 사전 조건 단계를 완료합니다.
- 시작하기 - 첫 번째 애플리케이션을 App Runner에 배포합니다.

AWS App Runner 가용성 변경

신중한 고려 끝에 2026년 4월 30일부터 신규 고객을 AWS App Runner 응대하기로 결정했습니다. 기존 AWS App Runner 고객은 새로운 리소스 및 서비스 생성을 포함하여 서비스를 정상적으로 계속 사용할 수 있습니다. AWS 는 보안 및 가용성에 계속 투자 AWS App Runner하지만 새로운 기능을 도입할 계획은 없습니다.

마이그레이션할 때 Amazon Elastic Container Service(Amazon ECS) Express 모드를 탐색하는 것이 좋습니다 AWS App Runner. Amazon ECS Express 모드는 App Runner의 운영 단순성을 유지하면서 더 광범위한 Amazon ECS 기능 세트에 대한 액세스를 제공합니다. 단일 API 호출을 통해 컨테이너 이미지와 두 개의 IAM 역할을 제공하고 Amazon ECS는 Fargate의 ECS 서비스, Application Load Balancer, Auto Scaling 및 네트워킹을 포함하여 AWS 계정에 전체 애플리케이션 스택을 프로비저닝합니다. Application Load Balancer Amazon ECS Express 모드 사용에 대한 추가 요금은 없습니다. 애플리케이션을 실행하기 위해 생성된 기본 AWS 리소스에 대해서만 비용을 지불합니다.

이 가이드에서는 기존 App Runner 서비스를 ECS Express Mode로 마이그레이션하고 DNS 라우팅을 사용하여 트래픽을 점진적으로 이동하는 방법을 설명합니다.

마이그레이션 개요

이 가이드에서는 DNS 가중치 기반 라우팅과 함께 블루/그린 배포 접근 방식을 사용하여 App Runner에서 ECS Express 모드로 트래픽을 마이그레이션합니다. 두 서비스 모두 마이그레이션 중에 동시에 실행됩니다. Amazon Route 53(또는 DNS 공급자)를 사용하여 App Runner 서비스에서 ECS Express Mode 서비스로 트래픽을 점진적으로 전환합니다. 작은 비율부터 시작하여 시간이 지남에 따라 증가합니다. 이 접근 방식을 사용하면 가동 중지 시간을 최소화하고 문제가 발생할 경우 DNS 가중치를 조정하여 롤백할 수 있습니다.

일반적인 마이그레이션에는 다음 단계가 포함됩니다.

1. 기존 App Runner 서비스의 구성 검토
2. 동일한 컨테이너 이미지를 사용하여 ECS Express Mode 서비스 생성
3. 사용자 지정 도메인을 사용하는 경우 ECS Express Mode 서비스에 대해 동일한 사용자 지정 도메인 구성
4. DNS 라우팅을 사용하여 App Runner에서 ECS Express 모드로 트래픽 전환
5. 마이그레이션을 완료하고 더 이상 필요하지 않은 경우 App Runner 서비스를 삭제합니다.

사전 조건

시작하기 전에 다음이 있는지 확인합니다.

- Amazon ECS, Amazon AWS App Runner Route 53 및 Application Load Balancer 리소스를 생성하고 관리할 수 있는 적절한 AWS Identity and Access Management 권한이 있는 AWS 계정
- AWS CLI AWS 계정의 자격 증명으로 설치 및 구성됨
- ECS Express Mode에 배포하기 위해 Amazon Elastic Container Registry(또는 다른 컨테이너 레지스트리)에 저장된 컨테이너 이미지
- ECS Express Mode에 필요한 IAM 역할: `ecsTaskExecutionRole` Amazon ECS 작업 실행 및 `ecsInfrastructureRoleForExpressServices` ECS Express Mode 인프라 프로비저닝

마이그레이션 중에 기존 사용자 지정 도메인을 보존하려면 다음 사항도 필요합니다.

- Amazon Route 53 또는 타사 도메인 등록 대행자를 `app.example.com`사용하여와 같이 제어하는 등록된 도메인 이름
- 사용자 지정 도메인과 일치하는 AWS Certificate Manager (ACM)의 SSL/TLS 인증서입니다. 리소스를 배포하는 AWS 리전 동일한에서 퍼블릭 ACM 인증서를 요청합니다. App Runner와 Amazon ECS Express 모드 모두 사용자 지정 도메인을 사용하여 HTTPS 액세스를 활성화하려면 ACM 인증서가 필요합니다.

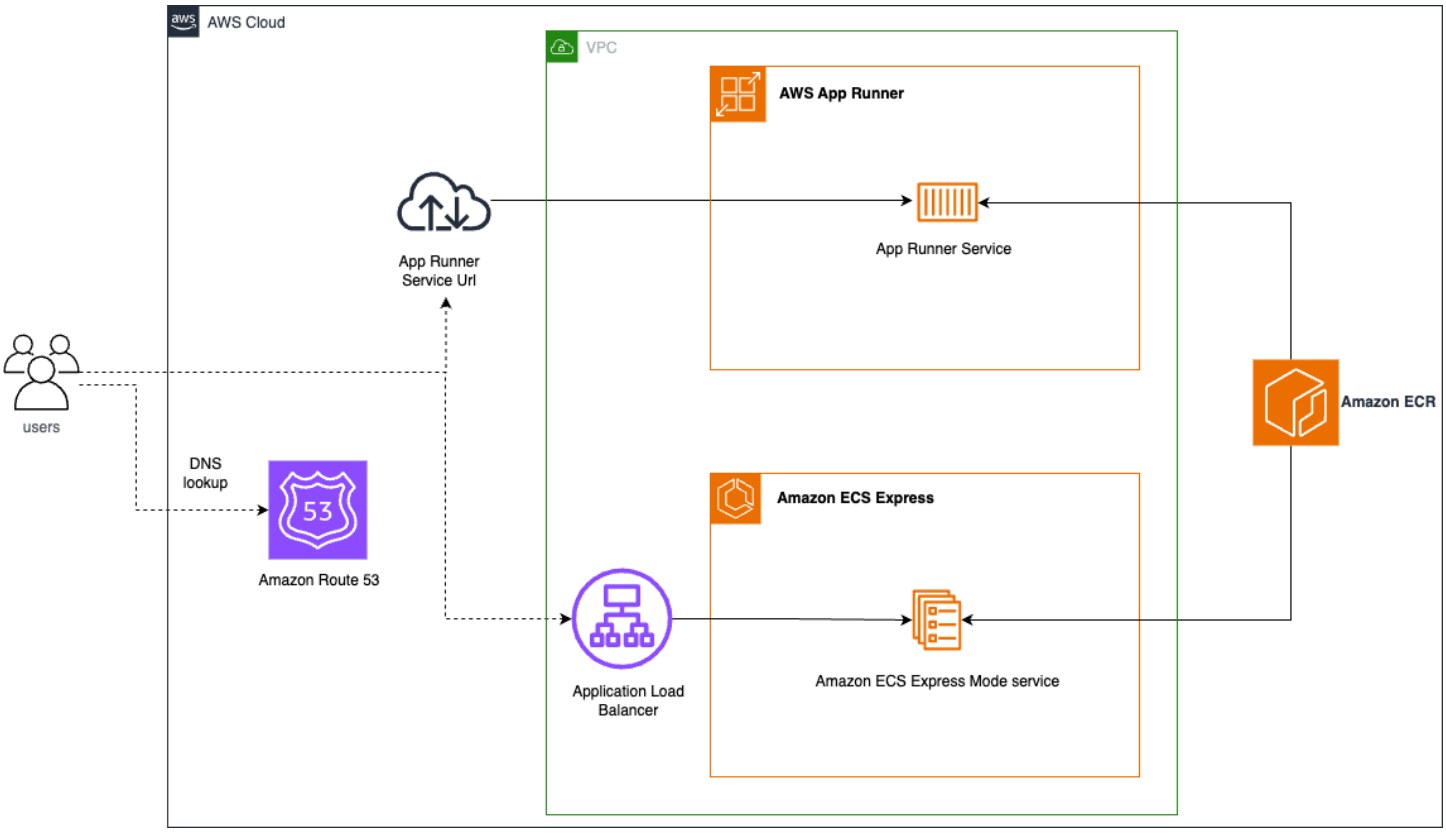
시작하기 전 준비 사항

- 컨테이너 이미지 요구 사항 - ECS Express Mode는 컨테이너 이미지를 배포합니다. App Runner 서비스가 소스 코드에서 배포된 경우 먼저 컨테이너 이미지를 생성하고 Amazon Elastic Container Registry와 같은 레지스트리로 푸시하는 빌드 단계를 추가합니다. 그런 다음 해당 이미지를 ECS Express Mode에 배포합니다. 소스 기반 배포 마이그레이션에 대한 소스 기반 배포 마이그레이션 자세한 내용은 섹션을 참조하세요.
- 도메인 동작 - App Runner 서비스가와 같은 사용자 지정 도메인을 이미 사용하는 `app.example.com`경우 마이그레이션 중에 동일한 호스트 이름을 재사용하고 DNS를 업데이트하여 App Runner와 ECS Express 모드 간에 트래픽을 점진적으로 이동할 수 있습니다.

  App Runner 서비스가 기본 App Runner 서비스 URL만 사용하는 경우 ECS Express Mode 서비스에는 다른 엔드포인트가 있습니다. 이 경우 점진적 트래픽 시프팅에 사용할 수 있는 공유 호스트 이름이 없습니다. ECS Express Mode 서비스를 생성하고 검증한 다음 새 엔드포인트를 사용하도록 클라이언트 또는 DNS를 업데이트해야 합니다.

마이그레이션 연습

다음 다이어그램은 Route 53을 사용하여 App Runner 서비스와 ECS Express Mode 서비스 간에 DNS 레코드를 이동하는 마이그레이션의 작동 방식을 보여줍니다.



1단계: 기존 App Runner 구성 검토

App Runner 콘솔에서 기존 서비스를 검토하고 이월하려는 값을 기록해 둡니다. 최소한 다음 사항에 유의하세요.

- 컨테이너 이미지
- 애플리케이션 포트
- 환경 변수
- 구성된 경우 사용자 지정 도메인 이름
- 구성된 경우 사용자 지정 도메인과 연결된 ACM 인증서

새 서비스로 전달하려는 다른 런타임 설정을 검토할 수도 있습니다.

사용자 지정 도메인 세부 정보는 섹션을 참조하세요the section called “사용자 지정 도메인 이름”.

2단계: ECS Express Mode 서비스 생성

App Runner 서비스에서 사용하는 것과 동일한 컨테이너 이미지를 사용하여 ECS Express Mode 서비스를 생성합니다. AWS Management Console 또는를 사용하여 서비스를 생성할 수 있습니다AWS CLI.

CLI 명령 예제:

```
aws ecs create-express-gateway-service \
    --execution-role-arn arn:aws:iam::123456789012:role/ecsTaskExecutionRole \
    --infrastructure-role-arn arn:aws:iam::123456789012:role/
ecsInfrastructureRoleForExpressServices \
    --primary-container '{
        "image": "123456789012.dkr.ecr.us-east-1.amazonaws.com/my-app:latest",
        "containerPort": 8080,
        "environment": [{
            "name": "ENV_VAR_NAME",
            "value": "value"
        }]
    }' \
    --service-name "my-application" \
    --health-check-path "/" \
    --scaling-target '{"minTaskCount":1,"maxTaskCount":4}' \
    --monitor-resources
```

이미지, 포트, 환경 변수 및 조정 값을 App Runner 서비스의 값으로 바꿉니다.

이 명령은 Fargate의 ECS 서비스, 대상 그룹 및 상태 확인이 포함된 Application Load Balancer, 자동 조정 정책, 보안 그룹 및 네트워킹 구성, 기본 URL을 포함하여 AWS 계정에 전체 애플리케이션 스택을 프로비저닝합니다.

프로비저닝에는 일반적으로 3~5분이 걸립니다. Amazon ECS 콘솔의 리소스 탭에서 진행 상황을 추적할 수 있습니다.

완료되면 콘솔에 표시된 기본 URL을 사용하여 ECS Express Mode 서비스를 테스트합니다. 트래픽 전환을 진행하기 전에 애플리케이션이 올바르게 작동하는지 확인합니다.

3단계: ECS Express 모드에 대한 사용자 지정 도메인 구성

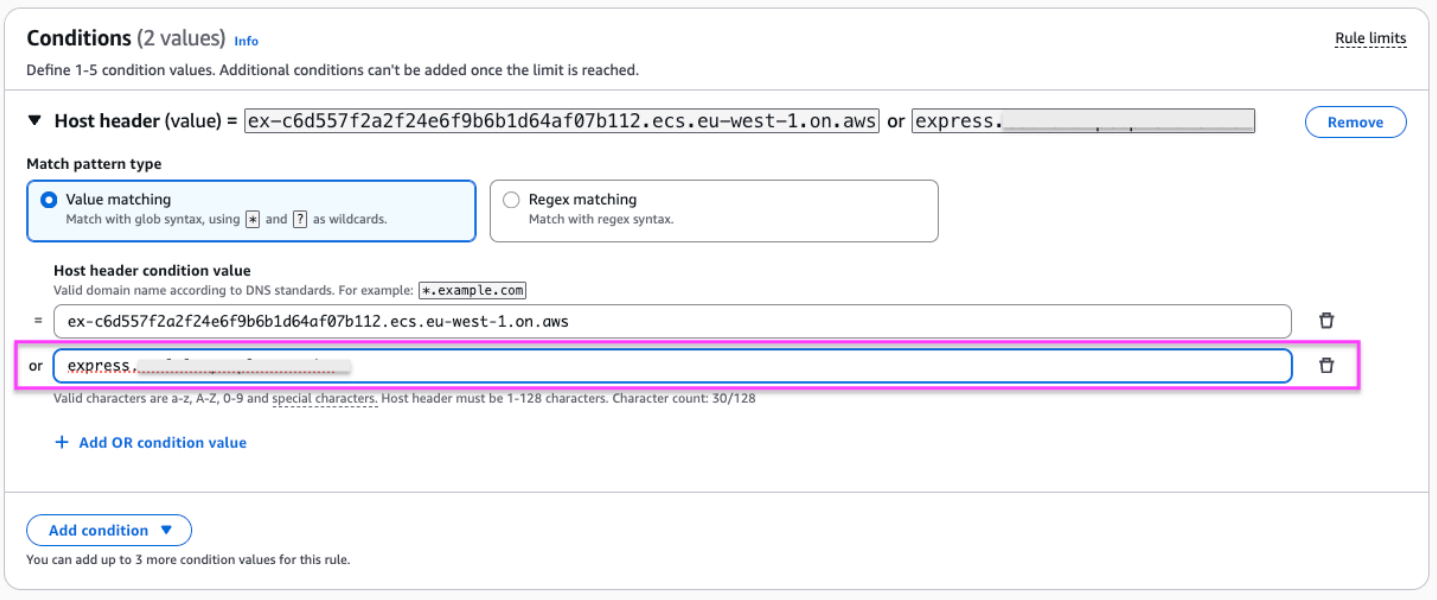
App Runner 서비스가 사용자 지정 도메인을 사용하는 경우 트래픽을 이동하기 전에 ECS Express Mode 서비스에 대해 동일한 사용자 지정 도메인을 구성합니다. 이 단계에서는 도메인에 대한 트

래픽을 수락하고 HTTPS에 ACM 인증서를 사용하도록 ECS Express Mode 서비스에 대해 생성된 Application Load Balancer를 구성합니다.

- Application Load Balancer 리스너 규칙에 사용자 지정 도메인을 호스트 헤더 조건으로 추가합니다. App Runner 서비스와 연결한 것과 동일한 도메인 이름을 사용합니다(예: `app.example.com`). 이렇게 하면 Application Load Balancer에 도메인에서 ECS Express Mode 대상 그룹으로 트래픽을 라우팅하도록 지시합니다.
- Application Load Balancer HTTPS 리스너에 SSL 인증서를 추가합니다. 1단계에서 기록한 ACM 인증서를 HTTPS 리스너에 추가합니다.

자세한 지침은 Amazon ECS 개발자 안내서의 서비스에 사용자 지정 도메인 추가를 참조하세요.

다음 이미지는 Application Load Balancer 리스너 규칙에서 호스트 헤더 조건을 구성하는 예를 보여줍니다.



4단계: Route 53 가중치 기반 라우팅을 사용하여 트래픽 이동

App Runner 서비스가 이미 사용자 지정 도메인을 사용하는 경우 Route 53 가중치 기반 라우팅을 사용하여 트래픽을 ECS Express Mode 서비스로 점진적으로 이동할 수 있습니다. 가중치 기반 라우팅을 사용하면 동일한 호스트 이름의 트래픽을 여러 엔드포인트로 라우팅할 수 있습니다. 각 엔드포인트는 자체 가중치가 있는 별도의 DNS 레코드로 정의되며 Route 53는 해당 가중치에 따라 요청을 분산합니다.

> **Note**
>
> 이 가이드에서는 Route 53를 예로 사용합니다. 다른 DNS 공급자를 사용하는 경우 공급자의 트래픽 관리 기능을 사용하여 동등한 DNS를 변경합니다.

기존 App Runner 레코드를 가중치 기반 레코드로 변환합니다.

1. Route 53 콘솔을 엽니다.
2. 호스팅 영역을 선택한 다음 도메인의 호스팅 영역을 선택합니다.
3. 현재 App Runner를 가리키는 호스트 이름의 기존 레코드(예: `app.example.com`)를 찾습니다.
4. 레코드를 편집하고 라우팅 정책을 가중치 적용으로 변경합니다.
5. 가중치를 로 설정합니다`100`(이렇게 하면 모든 초기 트래픽이 App Runner로 전달됩니다).
6. 레코드 ID에와 같은 설명 식별자를 입력합니다`app-runner-service`.
7. 변경 사항 저장을 선택합니다.

ECS Express 모드에 대한 가중치 기반 레코드 생성:

1. 동일한 호스팅 영역에 새 레코드를 생성합니다.
2. 동일한 레코드 이름을 사용합니다(예: `app.example.com`).
3. 동일한 레코드 유형을 사용합니다.
4. 라우팅 정책을 가중치 적용으로 설정합니다.
5. 트래픽 라우팅 대상에서 Application 및 Classic Load Balancer에 대한 별칭을 선택합니다.
6. 드롭다운에서 ECS Express Mode Application Load Balancer를 선택합니다.
7. 가중치를 로 설정합니다`0`(무게를 명시적으로 늘릴 때까지 ECS Express Mode로 트래픽 흐름 없음).
8. 레코드 ID에와 같은 설명 식별자를 입력합니다`ecs-express-service`.
9. 레코드 생성을 선택합니다.

트래픽을 점진적으로 전환합니다.

DNS 레코드가 구성되면 App Runner 가중치를 비례적으로 줄이면서 ECS Express Mode 가중치를 늘려 트래픽 이동을 시작합니다. 권장 접근 방식:

- ECS Express 모드를 10으로 설정/App Runner를 90으로 설정

- 서비스가 요청을 성공적으로 처리하는지 모니터링하고 검증합니다.
- 25/75로 증가
- 50/50으로 증가
- 75/25로 증가
- 100/0에 완료

각 단계에서 추가 트래픽을 이동하기 전에 애플리케이션을 테스트합니다. 언제든지 문제가 발생하면 가중치를 이전 값으로 다시 조정하여 롤백합니다.

> **Important**
>
> 검증 기간(예: 24~48시간) 동안 App Runner 서비스를 계속 실행하여 DNS 변경 사항이 전역적으로 전파되었는지 확인하고 필요한 경우 롤백 옵션을 제공합니다. 문제가 발생하면 Route 53 가중치를 App Runner로 빠르게 되돌릴 수 있습니다.

5단계: 마이그레이션 완료

ECS Express Mode 서비스가 프로덕션 트래픽을 올바르게 처리하고 검증 기간이 경과했는지 확인한 후 마이그레이션을 완료합니다.

- Route 53에서 App Runner를 가리키는 가중치 기반 레코드를 제거합니다(또는 가중치를 0으로 설정).
- App Runner 서비스에서 사용자 지정 도메인 연결을 제거합니다.

App Runner 서비스를 삭제합니다.

```
aws apprunner delete-service --service-arn your-app-runner-service-arn
```

또한 더 이상 필요하지 않은 리소스는 모두 제거하는 것이 좋습니다.

- App Runner에 대한 Route 53 가중치 기반 라우팅 레코드
- Amazon Elastic Container Registry의 미사용 컨테이너 이미지
- 더 이상 필요하지 않은 경우 App Runner용으로 특별히 생성된 IAM 역할

> **Note**
>
> 서비스가 프로덕션 환경에서 실행 중인 경우 ECS Express Mode 서비스, Application Load Balancer 또는 관련 리소스를 삭제하지 마십시오.

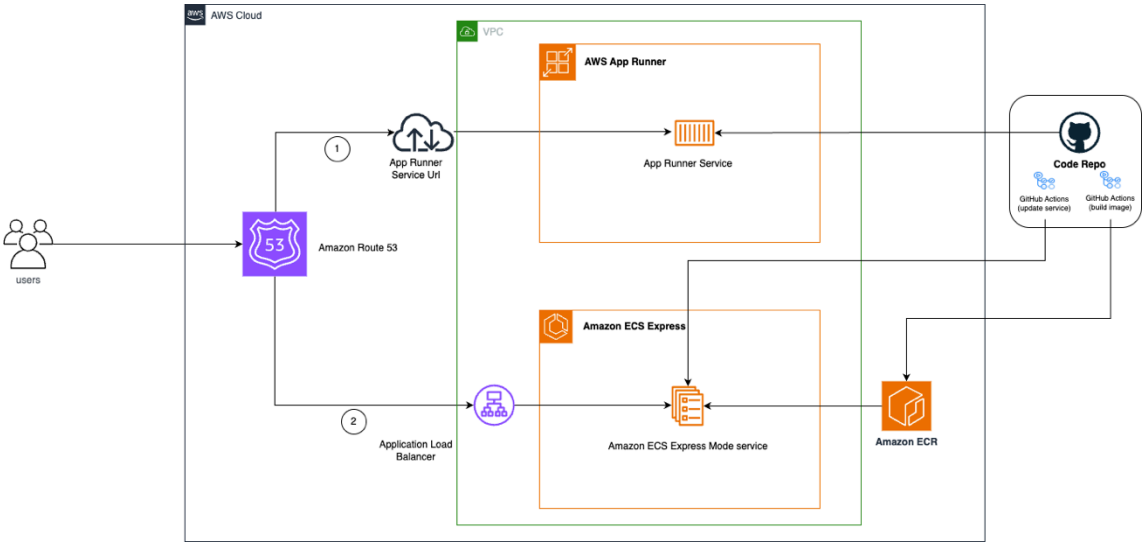
소스 기반 배포 마이그레이션

기존 App Runner 서비스가 컨테이너 이미지가 아닌 소스 코드에서 배포된 경우 ECS Express Mode에 배포하기 전에 컨테이너화 단계를 추가해야 합니다. App Runner와 달리 ECS Express 모드에는 컨테이너 이미지가 필요합니다. 그러나 Amazon ECS Deploy Express Service GitHub Action과 함께 GitHub Actions와 같은 CI/CD 도구를 사용하여 App Runner의 자동 배포 환경을 복제할 수 있습니다. GitHub

마이그레이션 워크플로에는 세 단계가 있습니다.

1. Dockerfile을 사용하여 컨테이너 이미지 빌드
2. Amazon Elastic Container Registry와 같은 컨테이너 레지스트리에 이미지 푸시
3. ECS Express 모드에 이미지 배포

다음 다이어그램은 GitHub 작업을 사용하여이 워크플로가 작동하는 방식을 보여줍니다.



애플리케이션 컨테이너화

애플리케이션에 아직 Dockerfile이 없는 경우 리포지토리 루트에 생성합니다. Dockerfile은 소스 코드를 컨테이너 이미지로 빌드하고 패키징하기 위한 청사진 역할을 합니다.

리포지토리 구조에는 다음이 포함되어야 합니다.

```
your-app/
### src/                    # Application source code
### Dockerfile              # Container build instructions
### package.json            # Dependencies and scripts
### .github/                # GitHub configuration
    ### workflows/          # GitHub Actions workflows
        ### deploy.yml      # ECS Express Mode deployment workflow
```

자동 배포를 위한 GitHub 작업 설정

코드 푸시 시 App Runner의 자동 배포를 복제하려면 다음을 사용하여 GitHub 작업을 구성합니다.

- OpenID Connect(OIDC) 공급자를 생성하여 GitHub Actions가 IAM 역할을 수임하도록 허용
- ECS Express 모드 및 Amazon Elastic Container Registry 권한을 사용하여 IAM 역할 생성 https://docs.aws.amazon.com/AmazonECR/latest/userguide/ECR_on_ECS.html
- ECS Express Mode에 필요한 두 가지 IAM 역할 생성
- ECS 리소스에 대한 GitHub 환경 변수 생성: `ECS_SERVICE`, `ECS_CLUSTER`, `AWS_ACCOUNT_ID`, `AWS_REGION`및 `ECR_REPOSITORY`

GitHub 작업 워크플로 예제

에서 워크플로 파일을 생성합니다`.github/workflows/deploy.yml`.

```
name: Build and Deploy to ECS

on:
  push:
    branches: [ main ]

env:
  AWS_REGION: ${{ vars.AWS_REGION }}
  AWS_ACCOUNT_ID: ${{ vars.AWS_ACCOUNT_ID }}
  ECR_REPOSITORY: ${{ vars.ECR_REPOSITORY }}
  ECS_SERVICE: ${{ vars.ECS_SERVICE }}
  ECS_CLUSTER: ${{ vars.ECS_CLUSTER }}

jobs:
  deploy:
```

```
    name: Deploy
    runs-on: ubuntu-latest
    environment: production
    permissions:
      id-token: write
      contents: read

    steps:
    - name: Checkout
      uses: actions/checkout@v6

    - name: Configure AWS credentials
      uses: aws-actions/configure-aws-credentials@v5
      with:
        aws-region: ${{ env.AWS_REGION }}
        role-to-assume: arn:aws:iam::${{ env.AWS_ACCOUNT_ID }}:role/github-actions-ecs-
role
        role-session-name: GitHubActionsECSDeployment

    - name: Login to Amazon ECR
      id: login-ecr
      uses: aws-actions/amazon-ecr-login@v2

    - name: Get short commit hash
      run: echo "IMAGE_TAG=${GITHUB_SHA:0:7}" >> $GITHUB_ENV

    - name: Build, tag, and push image to Amazon ECR
      id: build-image
      env:
        ECR_REGISTRY: ${{ steps.login-ecr.outputs.registry }}
      uses: docker/build-push-action@v6
      with:
        context: .
        push: true
        tags: ${{ env.ECR_REGISTRY }}/${{ env.ECR_REPOSITORY }}:latest,
${{ env.ECR_REGISTRY }}/${{ env.ECR_REPOSITORY }}:${{ env.IMAGE_TAG }}

    - name: Deploy to ECS Express Mode
      uses: aws-actions/amazon-ecs-deploy-express-service@v1
      env:
        ECR_REGISTRY: ${{ steps.login-ecr.outputs.registry }}
      with:
        service-name: ${{ env.ECS_SERVICE }}
        image: ${{ env.ECR_REGISTRY }}/${{ env.ECR_REPOSITORY }}:${{ env.IMAGE_TAG }}
```

```
        execution-role-arn: arn:aws:iam::${{ env.AWS_ACCOUNT_ID }}:role/
ecsTaskExecutionRole
        infrastructure-role-arn: arn:aws:iam::${{ env.AWS_ACCOUNT_ID }}:role/
ecsInfrastructureRoleForExpressServices
        cluster: ${{ env.ECS_CLUSTER }}
        container-port: 8080
        environment-variables: |
          [
            {"name": "ENV", "value": "Prod"}
          ]
        cpu: '1024'
        memory: '2048'
        health-check-path: /health
        min-task-count: 1
        max-task-count: 4
        auto-scaling-metric: AVERAGE_CPU
        auto-scaling-target-value: 70
```

코드 변경 사항을 기본 브랜치에 푸시하면 GitHub Actions는 새 컨테이너 이미지를 자동으로 빌드하고 Amazon Elastic Container Registry에 푸시한 다음 ECS Express Mode 서비스에 배포합니다. 이렇게 하면 App Runner를 사용한 자동 배포 경험이 복제됩니다.

ECS Express Mode 서비스가 실행되면 마이그레이션 연습의 3~5단계에 따라 사용자 지정 도메인을 구성하고, DNS 라우팅을 사용하여 트래픽을 이동하고, 마이그레이션을 완료합니다.

추가 리소스

- 를 사용하여 첫 번째 Express Mode 서비스 생성 AWS CLI
- 콘솔에서 첫 번째 Amazon ECS Express Mode 서비스 생성
- Express Mode 외부에서 리소스 업데이트
- the section called “사용자 지정 도메인 이름”
- Amazon Route 53 가중치 기반 라우팅

App Runner 설정

신규 AWS 고객인 경우 사용을 시작하기 전에이 페이지에 나열된 설정 사전 조건을 완료합니다 AWS App Runner.

이러한 설정 절차에서는 AWS Identity and Access Management (IAM) 서비스를 사용합니다. IAM에 대한 자세한 내용은 다음 참조 자료를 참조하세요.

- AWS Identity and Access Management (IAM)
- IAM 사용 설명서

에 가입 AWS 계정

이 없는 경우 다음 단계를 AWS 계정완료하여 생성합니다.

에 가입하려면 AWS 계정

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

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

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

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

관리자 액세스 권한이 있는 사용자 생성

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

보안 AWS 계정 루트 사용자

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

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

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

관리자 액세스 권한이 있는 사용자 생성

1. IAM Identity Center를 활성화합니다.

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

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

관리 액세스 권한이 있는 사용자로 로그인

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

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

추가 사용자에게 액세스 권한 할당

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

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

   지침은 AWS IAM Identity Center 사용 설명서의 그룹 추가를 참조하세요.

프로그래밍 방식 액세스 권한 부여

사용자는 AWS 외부에서와 상호 작용하려는 경우 프로그래밍 방식으로 액세스해야 합니다 AWS Management Console. 프로그래밍 방식 액세스를 부여하는 방법은에 액세스하는 사용자 유형에 따라 다릅니다 AWS.

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

프로그래밍 방식 액세스가 필요한 사용자	목적	방법
IAM	(권장) 콘솔 자격 증명을 임시 자격 증명으로 사용하여 AWS CLI, AWS SDKs 또는 AWS APIs.	사용하고자 하는 인터페이스에 대한 지침을 따릅니다. • 자세한 AWS CLI내용은 AWS Command Line Interface 사용 설명서의 AWS 로컬 개발을 위한 로그인을 참조하세요. • AWS SDKs 경우 SDK 및 도구 참조 안내서의 AWS 로컬 개발을 위한 로그인을 참조하세요. AWS SDKs
작업 인력 ID (IAM Identity Center에서 관리되는 사용자)	임시 자격 증명을 사용하여 AWS CLI, AWS SDKs 또는 AWS APIs.	사용하고자 하는 인터페이스에 대한 지침을 따릅니다. • 자세한 AWS CLI내용은 AWS Command Line Interface 사용 설명서의 AWS CLI 를 사용하도록 구성을 AWS IAM Identity Center 참조하세요. • AWS SDKs, 도구 및 AWS APIs 경우 SDK 및 도구 참조 안내서의 IAM Identity

프로그래밍 방식 액세스가 필요한 사용자	목적	방법
		Center 인증을 참조하세요. AWS SDKs
IAM	임시 자격 증명을 사용하여 AWS CLI, AWS SDKs 또는 AWS APIs.	IAM 사용 설명서의 AWS 리소스에서 임시 자격 증명 사용의 지침을 따릅니다.
IAM	(권장되지 않음) 장기 자격 증명을 사용하여 AWS CLI, AWS SDKs 또는 AWS APIs.	사용하고자 하는 인터페이스에 대한 지침을 따릅니다. • 자세한 AWS CLI내용은 AWS Command Line Interface 사용 설명서의 IAM 사용자 자격 증명을 사용하여 인증을 참조하세요. • AWS SDKs 및 도구의 경우 SDK 및 도구 참조 안내서의 장기 자격 증명을 사용하여 인증을 참조하세요. AWS SDKs • AWS APIs 경우 IAM 사용 설명서의 IAM 사용자의 액세스 키 관리를 참조하세요.

다음에 있는 것

사전 조건 단계를 완료했습니다. App Runner에 첫 번째 애플리케이션을 배포하려면 섹션을 참조하세요시작하기.

App Runner 시작하기

AWS App Runner 는 기존 컨테이너 이미지 또는 소스 코드를에서 실행 중인 웹 AWS 서비스로 직접 전환하는 빠르고 간단하며 비용 효율적인 방법을 제공하는 서비스입니다 AWS 클라우드.

이 자습서에서는 AWS App Runner 를 사용하여 App Runner 서비스에 애플리케이션을 배포하는 방법을 다룹니다. 소스 코드 및 배포, 서비스 빌드 및 서비스 런타임을 구성하는 방법을 안내합니다. 또한 코드 버전을 배포하고, 구성을 변경하고, 로그를 보는 방법도 보여줍니다. 마지막으로 자습서에서는 자습서의 절차에 따라 생성한 리소스를 정리하는 방법을 보여줍니다.

주제

사전 조건

자습서를 시작하기 전에 다음 작업을 수행해야 합니다.

1. 의 설정 단계를 완료합니다설정.
2. GitHub 리포지토리 또는 Bitbucket 리포지토리로 작업할지 여부를 결정합니다.
   - Bitbucket으로 작업하려면 Bitbucket 계정이 아직 없는 경우 먼저 Bitbucket 계정을 생성합니다. Bitbucket을 처음 사용하는 경우 Bitbucket 클라우드 설명서의 Bitbucket 시작하기를 참조하세요.
   - GitHub로 작업하려면 아직 없는 경우 GitHub 계정을 생성합니다. GitHub를 처음 사용하는 경우 GitHub 문서의 GitHub 시작하기를 참조하세요 GitHub.

     > **Note**
     >
     > 계정에서 여러 리포지토리 공급자에 대한 연결을 생성할 수 있습니다. 따라서 GitHub와 Bitbucket 리포지토리 모두에서 배포를 진행하려면이 절차를 반복할 수 있습니다. 다음에

를 통해 새 App Runner 서비스를 생성하고 다른 리포지토리 공급자에 대한 새 계정 연결을 생성합니다.

3. 리포지토리 공급자 계정에 리포지토리를 생성합니다. 이 자습서에서는 리포지토리 이름을 사용합니다`python-hello`. 다음 예제에 지정된 이름과 콘텐츠를 사용하여 리포지토리의 루트 디렉터리에 파일을 생성합니다.

python-hello 예제 리포지토리의 파일

Example requirements.txt

```
pyramid==2.0
```

Example server.py

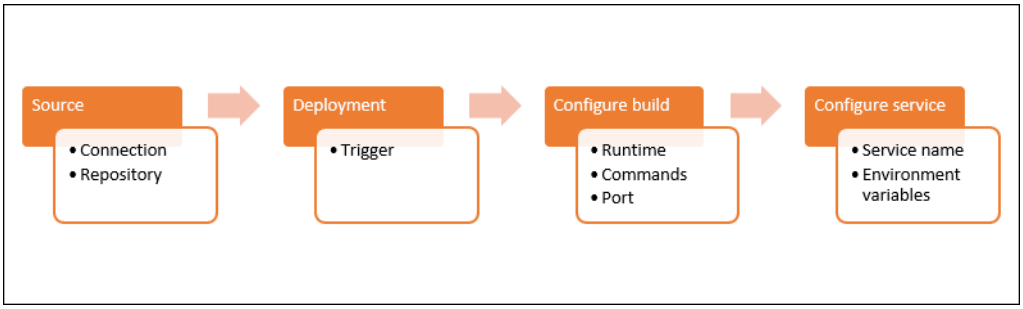
```
from wsgiref.simple_server import make_server
from pyramid.config import Configurator
from pyramid.response import Response
import os

def hello_world(request):
    name = os.environ.get('NAME')
    if name == None or len(name) == 0:
        name = "world"
    message = "Hello, " + name + "!\n"
    return Response(message)

if __name__ == '__main__':
    port = int(os.environ.get("PORT"))
    with Configurator() as config:
        config.add_route('hello', '/')
        config.add_view(hello_world, route_name='hello')
        app = config.make_wsgi_app()
    server = make_server('0.0.0.0', port, app)
    server.serve_forever()
```

1단계: App Runner 서비스 생성

이 단계에서는 GitHub 또는 Bitbucket에서의 일부로 생성한 예제 소스 코드 리포지토리를 기반으로 App Runner 서비스를 생성합니다the section called “사전 조건”. 이 예제에는 간단한 Python 웹 사이트가 포함되어 있습니다. 다음은 서비스를 생성하기 위해 수행하는 주요 단계입니다.

1. 소스 코드를 구성합니다.
2. 소스 배포를 구성합니다.
3. 애플리케이션 빌드를 구성합니다.
4. 서비스를 구성합니다.
5. 검토 및 확인합니다.

다음 다이어그램은 App Runner 서비스를 생성하는 단계를 간략하게 설명합니다.

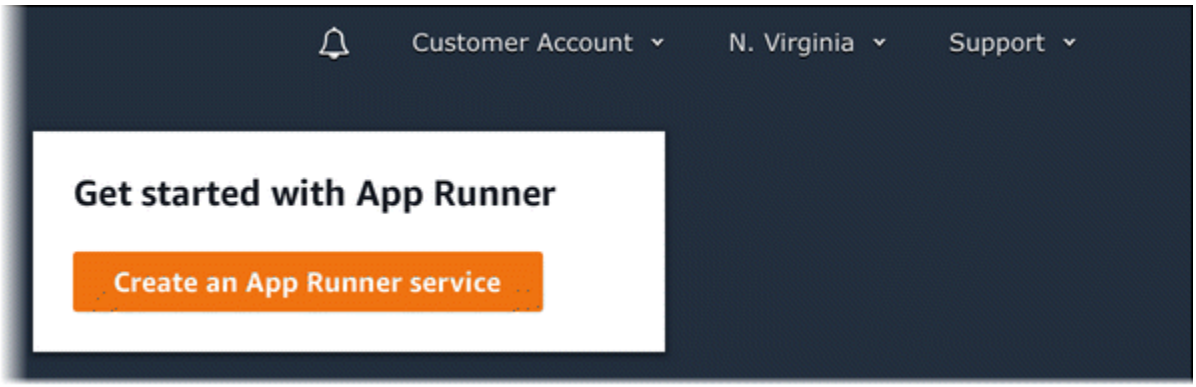


소스 코드 리포지토리를 기반으로 App Runner 서비스를 생성하려면

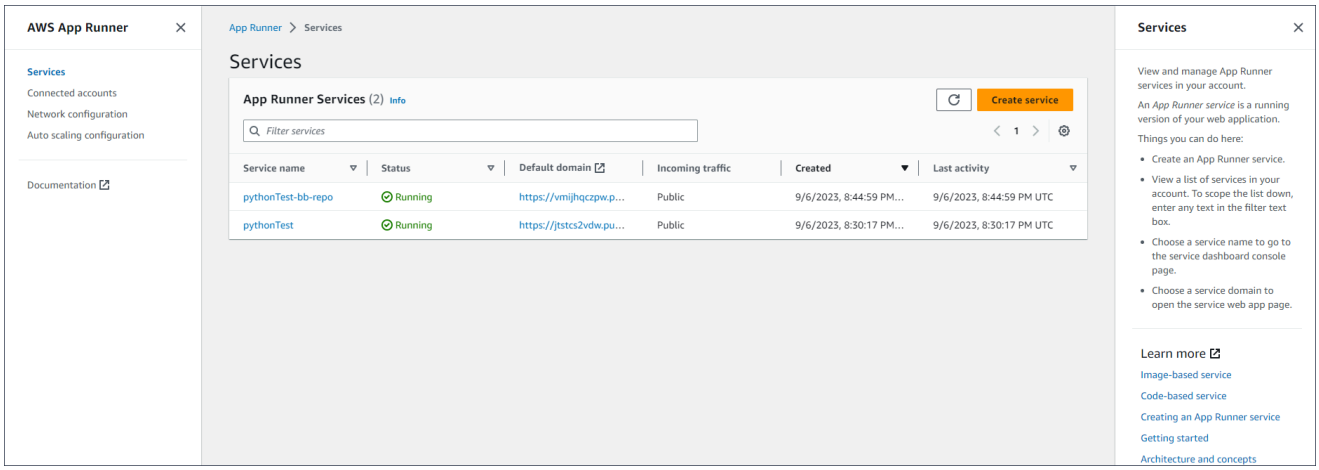
1. 소스 코드를 구성합니다.

   a. App Runner 콘솔을 열고 리전 목록에서를 선택합니다 AWS 리전.

   b. 에 아직 App Runner 서비스가 AWS 계정 없는 경우 콘솔 홈 페이지가 표시됩니다. App Runner 서비스 생성을 선택합니다.



에 기존 서비스가 AWS 계정 있는 경우 서비스 목록과 함께 서비스 페이지가 표시됩니다. 서비스 생성을 선택합니다.



c. 소스 및 배포 페이지의 소스 섹션의 리포지토리 유형에서 소스 코드 리포지토리를 선택합니다.

d. 공급자 유형을 선택합니다. GitHub 또는 Bitbucket을 선택합니다.

e. 그런 다음 새로 추가를 선택합니다. 메시지가 표시되면 GitHub 또는 Bitbucket 자격 증명을 제공합니다.

f. 이전에 선택한 공급자 유형에 따라 다음 단계 세트를 선택합니다.

> **Note**
>
> GitHub용 AWS 커넥터를 GitHub 계정에 설치하는 다음 단계는 일회성 단계입니다. 연결을 재사용하여이 계정의 리포지토리를 기반으로 여러 App Runner 서비스를 생성할 수 있습니다. 기존 연결이 있는 경우 연결을 선택하고 리포지토리 선택으로 건너뜁니다.
>
> Bitbucket 계정의 AWS 커넥터에도 동일하게 적용됩니다. GitHub와 Bitbucket을 App Runner 서비스의 소스 코드 리포지토리로 사용하는 경우 각 공급자에 대해 하나의 AWS 커넥터를 설치해야 합니다. 그런 다음 각 커넥터를 재사용하여 더 많은 App Runner 서비스를 생성할 수 있습니다.

- GitHub의 경우 다음 단계를 따릅니다.

  i. 다음 화면에서 연결 이름을 입력합니다.

  ii. App Runner와 함께 GitHub를 처음 사용하는 경우 다른 설치를 선택합니다.

iii. AWS GitHub용 커넥터 대화 상자에서 메시지가 표시되면 GitHub 계정 이름을 선택합니다.

iv. GitHub용 AWS 커넥터를 승인하라는 메시지가 표시되면 AWS 연결 승인을 선택합니다.

v. GitHub용 AWS 커넥터 설치 대화 상자에서 설치를 선택합니다.

계정 이름은 선택한 GitHub 계정/조직으로 표시됩니다. 이제 계정에서 리포지토리를 선택할 수 있습니다.

vi. 리포지토리에서 생성한 예제 리포지토리인를 선택합니다`python-hello`. 브랜치에서 리포지토리의 기본 브랜치 이름(예: 기본)을 선택합니다.

vii. 소스 디렉터리를 기본값으로 둡니다. 디렉터리는 기본적으로 리포지토리 루트로 설정됩니다. 이전 사전 조건 단계의 리포지토리 루트 디렉터리에 소스 코드를 저장했습니다.

- Bitbucket의 경우 다음 단계를 따릅니다.

i. 다음 화면에서 연결 이름을 입력합니다.

ii. App Runner와 함께 Bitbucket을 처음 사용하는 경우 다른 설치를 선택합니다.

iii. AWS CodeStar 액세스 요청 대화 상자에서 워크스페이스를 선택하고 Bitbucket 통합을 AWS CodeStar 위해에 액세스 권한을 부여할 수 있습니다. 워크스페이스를 선택한 다음 액세스 권한 부여를 선택합니다.

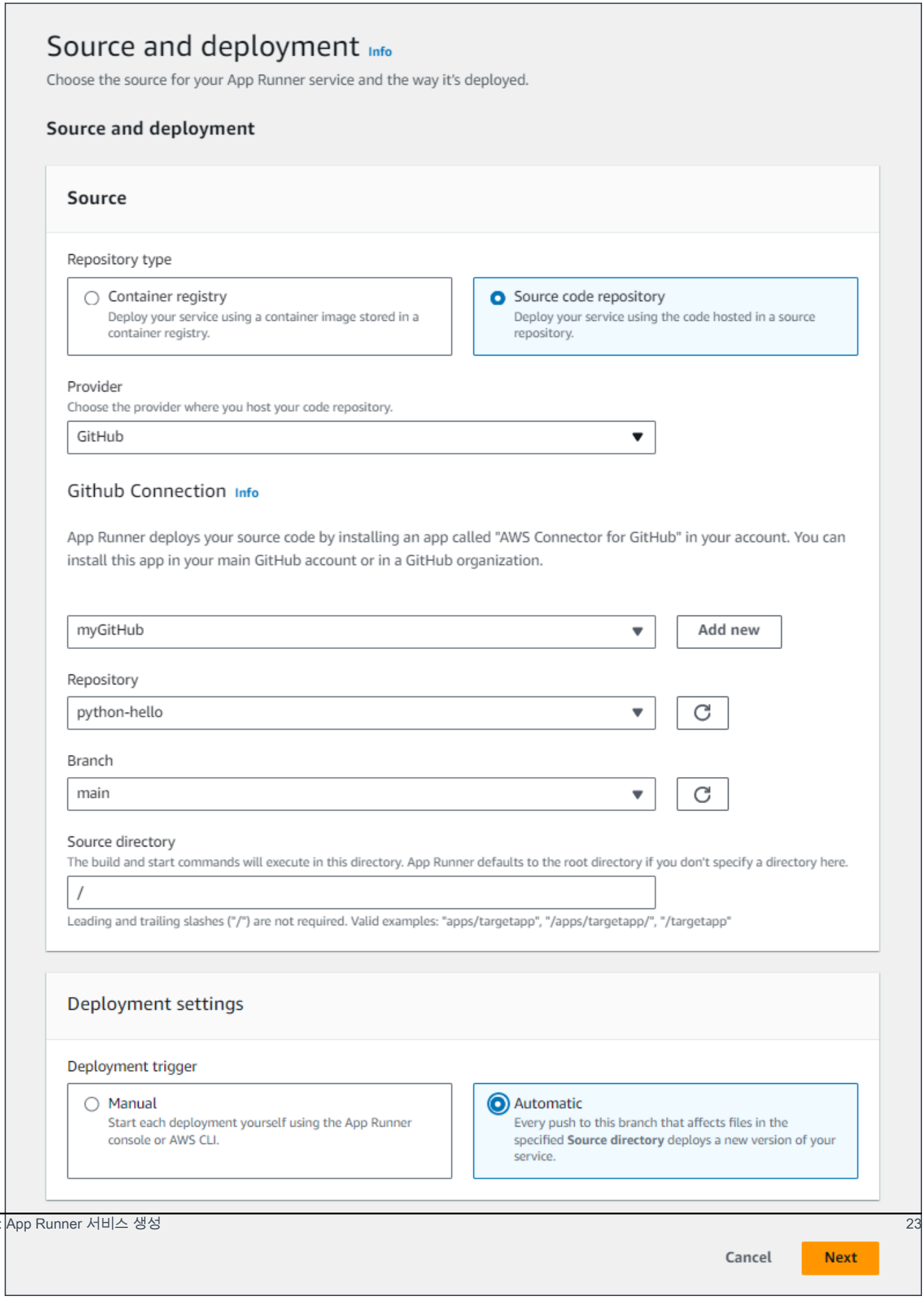
iv. 다음으로 AWS 콘솔로 리디렉션됩니다. Bitbucket 애플리케이션이 올바른 Bitbucket 워크스페이스로 설정되어 있는지 확인하고 다음을 선택합니다.

v. 리포지토리에서 생성한 예제 리포지토리인를 선택합니다`python-hello`. 브랜치에서 리포지토리의 기본 브랜치 이름(예: 기본)을 선택합니다.

vi. 소스 디렉터리를 기본값으로 둡니다. 디렉터리는 기본적으로 리포지토리 루트로 설정됩니다. 이전 사전 조건 단계의 리포지토리 루트 디렉터리에 소스 코드를 저장했습니다.

2. 배포 구성: 배포 설정 섹션에서 자동을 선택한 후 다음을 선택합니다.

> **Note**
>
> 자동 배포를 사용하면 리포지토리 소스 디렉터리에 대한 각 새 커밋이 새 버전의 서비스를 자동으로 배포합니다.


Source and deployment Info
Choose the source for your App Runner service and the way it's deployed.
Source and deployment
Source
Repository type
Container registry
Deploy your service using a container image stored in a container registry.
Source code repository
Deploy your service using the code hosted in a source repository.
Provider
Choose the provider where you host your code repository.
GitHub
Github Connection Info
App Runner deploys your source code by installing an app called "AWS Connector for GitHub" in your account. You can install this app in your main GitHub account or in a GitHub organization.
myGitHub
Add new
Repository
python-hello
Branch
main
Source directory
The build and start commands will execute in this directory. App Runner defaults to the root directory if you don't specify a directory here.
/
Leading and trailing slashes ("/") are not required. Valid examples: "apps/targetapp", "/apps/targetapp/", "/targetapp"
Deployment settings
Deployment trigger
Manual
Start each deployment yourself using the App Runner console or AWS CLI.
Automatic
Every push to this branch that affects files in the specified Source directory deploys a new version of your service.
Cancel
Next

3. 애플리케이션 빌드를 구성합니다.

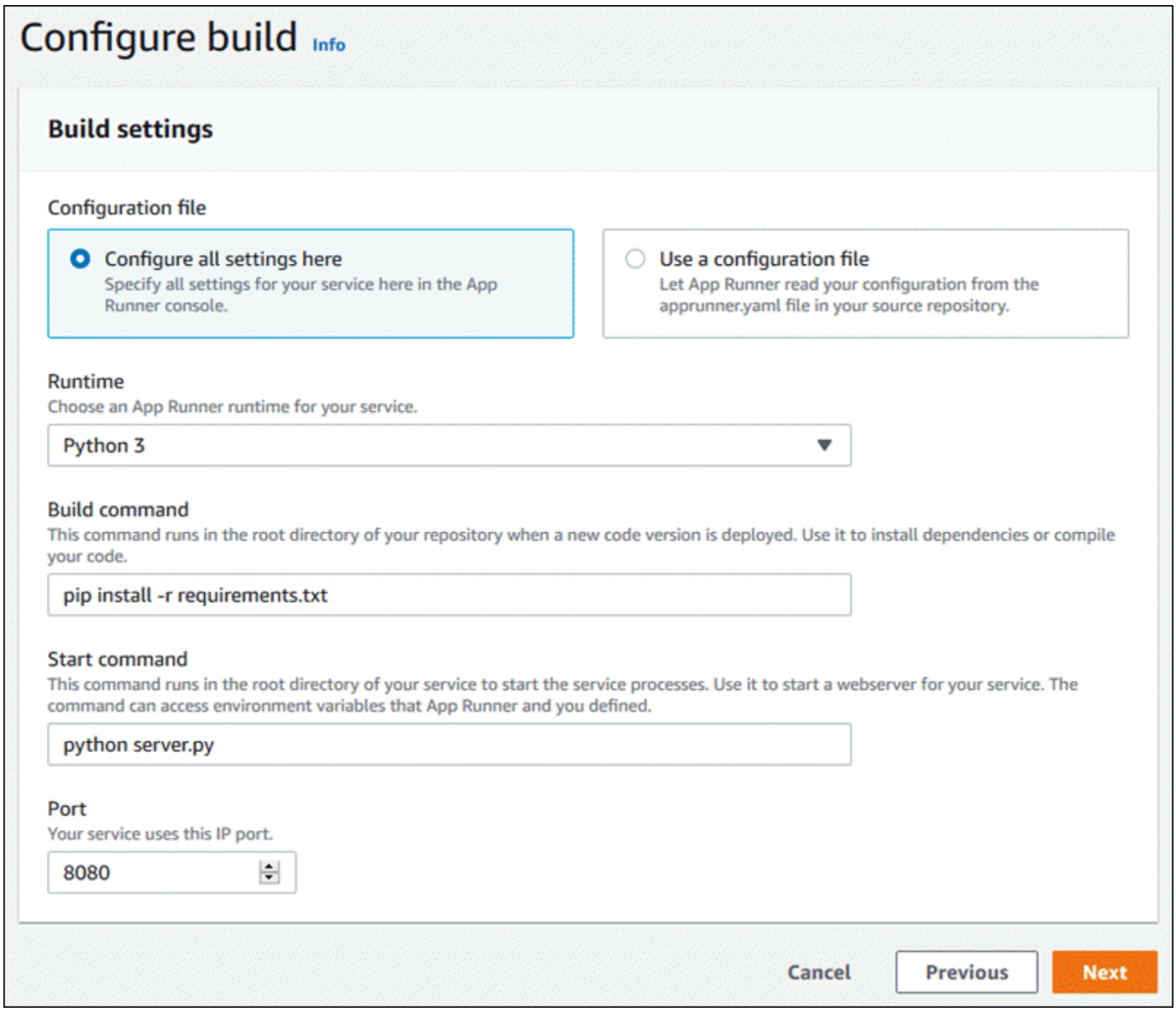
   a. 빌드 구성 페이지의 구성 파일에서 여기에서 모든 설정 구성을 선택합니다.

   b. 다음 빌드 설정을 제공합니다.

      - 런타임 - Python 3을 선택합니다.
      - 빌드 명령 -를 입력합니다**`pip install -r requirements.txt`**.
      - 시작 명령 -를 입력합니다**`python server.py`**.
      - 포트 -를 입력합니다**`8080`**.

   c. 다음을 선택합니다.

> **Note**
>
> Python 3 런타임은 기본 Python 3 이미지와 예제 Python 코드를 사용하여 Docker 이미지를 빌드합니다. 그런 다음이 이미지의 컨테이너 인스턴스를 실행하는 서비스를 시작합니다.



4. 서비스를 구성합니다.

   a. 서비스 구성 페이지의 서비스 설정 섹션에서 서비스 이름을 입력합니다.

   b. 환경 변수에서 환경 변수 추가를 선택합니다. 환경 변수에 다음 값을 입력합니다.

      - 소스 - 일반 텍스트 선택
      - 환경 변수 이름 - **NAME**
      - 환경 변수 값 - 모든 이름(예: 이름).

> **Note**
>
> 예제 애플리케이션은이 환경 변수에서 설정한 이름을 읽고 웹 페이지에 이름을 표시합니다.

c. 다음을 선택합니다.

Configure service Info

Service settings

Service name

python-test

Virtual CPU & memory

1 vCPU | 2 GB

Environment variables — *optional* Info

Add environment variables in plain text or reference them from Secrets Manager and SSM Parameter Store. Update IAM Policies using the IAM Policy template given below to securely reference secrets and configurations as environment variables.

No environment variables have been configured.

Add environment variable

You can add up to 50 more items.

▶ IAM policy templates

▶ Auto scaling Info

Configure automatic scaling behavior.

▶ Health check Info

Configure load balancer health checks.

▶ Security Info

Specify an Instance role and an AWS KMS encryption key

▶ Networking Info

Configure the way your service communicates with other applications, services, and resources.

▶ Observability

Configure observability tooling.

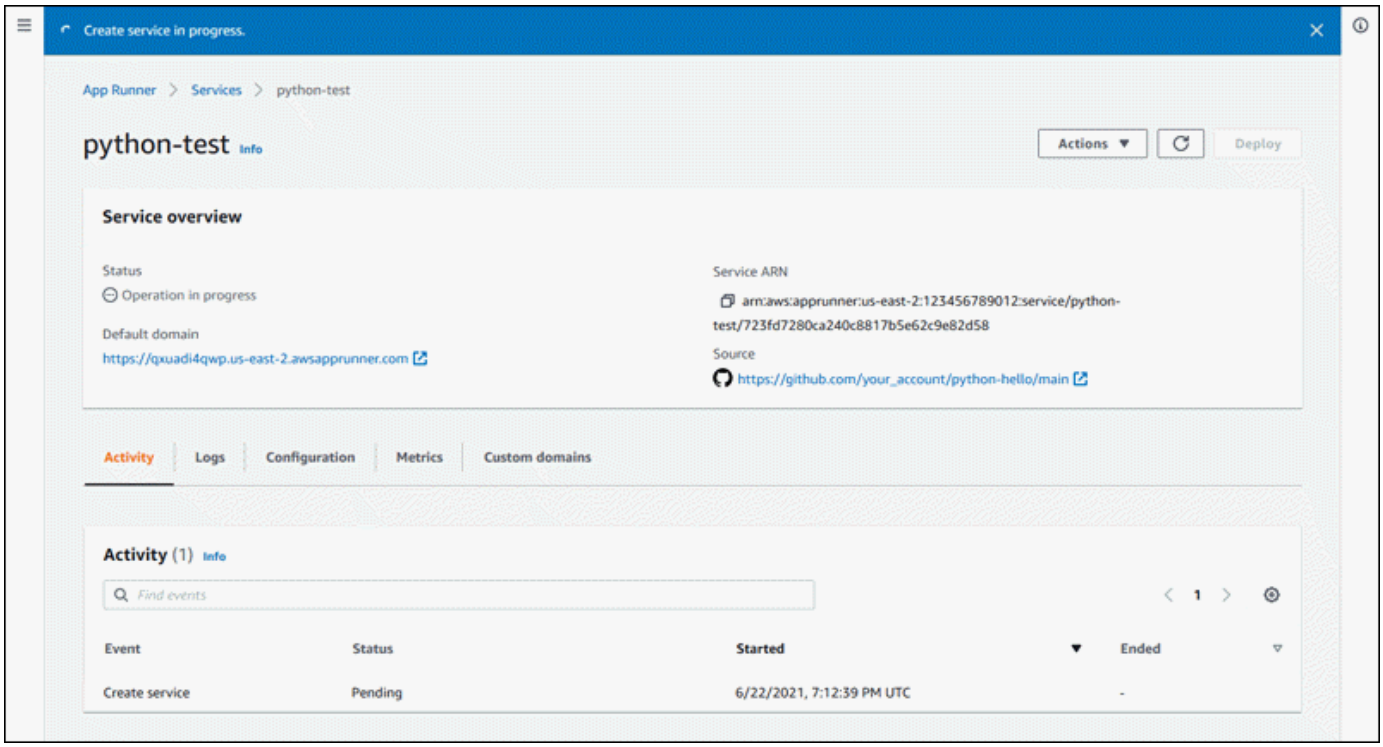
▼ Tags Info

Use tags to search and filter your resources, track your AWS costs, and control access permissions.

Tags — *optional*

5. 검토 및 생성 페이지에서 입력한 모든 세부 정보를 확인한 다음 생성 및 배포를 선택합니다.

   서비스가 성공적으로 생성되면 콘솔에 새 서비스의 서비스 개요와 함께 서비스 대시보드가 표시됩니다.



6. 서비스가 실행 중인지 확인합니다.

   a. 서비스 대시보드 페이지에서 서비스 상태가 실행 중일 때까지 기다립니다.

   b. 기본 도메인 값을 선택합니다. 기본 도메인 값은 서비스 웹 사이트의 URL입니다.

> **Note**
>
> App Runner 애플리케이션의 보안을 강화하기 위해 *.awsapprunner.com 도메인은 퍼블릭 접미사 목록(PSL)에 등록됩니다. 보안을 강화하려면 App Runner 애플리케이션의 기본 도메인 이름에 민감한 쿠키를 설정해야 하는 경우 `__Host-` 접두사가 있는 쿠키를 사용하는 것이 좋습니다. 이렇게 쿠키를 설정하면 교차 사이트 요청 위조 시도(CSRF)로부터 도메인을 보호하는 데 도움이 됩니다. 자세한 내용은 Mozilla 개발자 네트워크의 Set-Cookie 페이지를 참조하세요.

   웹 페이지에 Hello, ***your name***!이 표시됩니다.

2단계: 서비스 코드 변경

이 단계에서는 리포지토리 소스 디렉터리에서 코드를 변경합니다. App Runner CI/CD 기능은 변경 사항을 자동으로 빌드하고 서비스에 배포합니다.

서비스 코드를 변경하려면

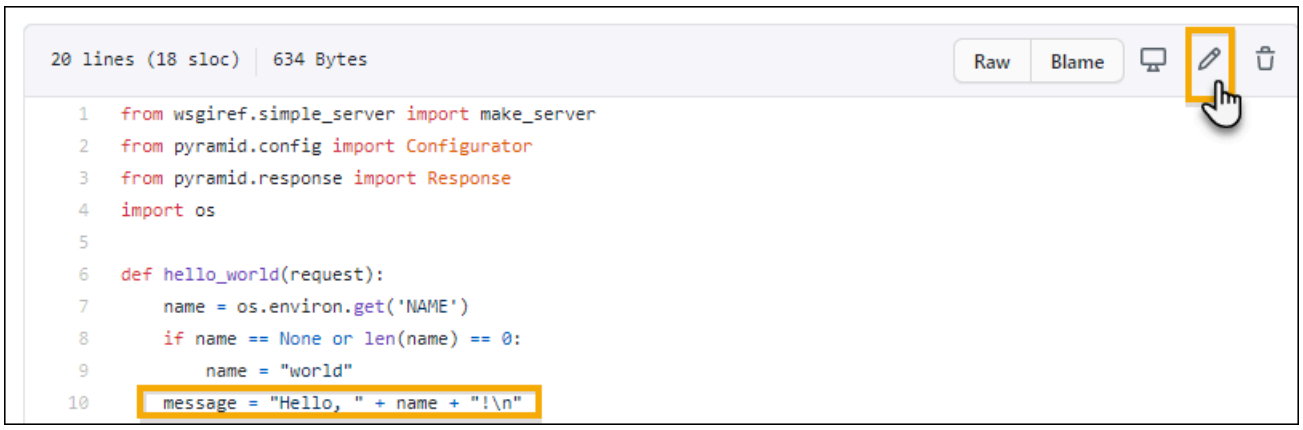
1. 예제 리포지토리로 이동합니다.
2. 라는 파일을 편집합니다`server.py`.
3. 변수에 할당된 표현식에서 텍스트를 `Hello`로 `message`변경합니다`Good morning`.
4. 변경 사항을 저장하고 리포지토리에 커밋합니다.
5. 다음 단계는 GitHub 리포지토리에서 서비스 코드를 변경하는 방법을 보여줍니다.

    a. 예제 GitHub 리포지토리로 이동합니다.

    b. 파일 이름을 선택하여 해당 파일로 `server.py` 이동합니다.

    c. 이 파일 편집(연필 아이콘)을 선택합니다.

    d. 변수에 할당된 표현식에서 텍스트를 `Hello`로 `message`변경합니다`Good morning`.



e. 변경 사항 커밋을 선택합니다.

6. 새 커밋이 App Runner 서비스에 배포되기 시작합니다. 서비스 대시보드 페이지에서 서비스 상태가 진행 중인 작업으로 변경됩니다.

   배포가 종료될 때까지 기다립니다. 서비스 대시보드 페이지에서 서비스 상태가 다시 실행 중으로 변경되어야 합니다.

7. 배포가 성공했는지 확인합니다. 서비스의 웹 페이지가 표시되는 브라우저 탭을 새로 고칩니다.

   이제 페이지에 수정된 메시지인 안녕하세요, ### ##!이 표시됩니다.

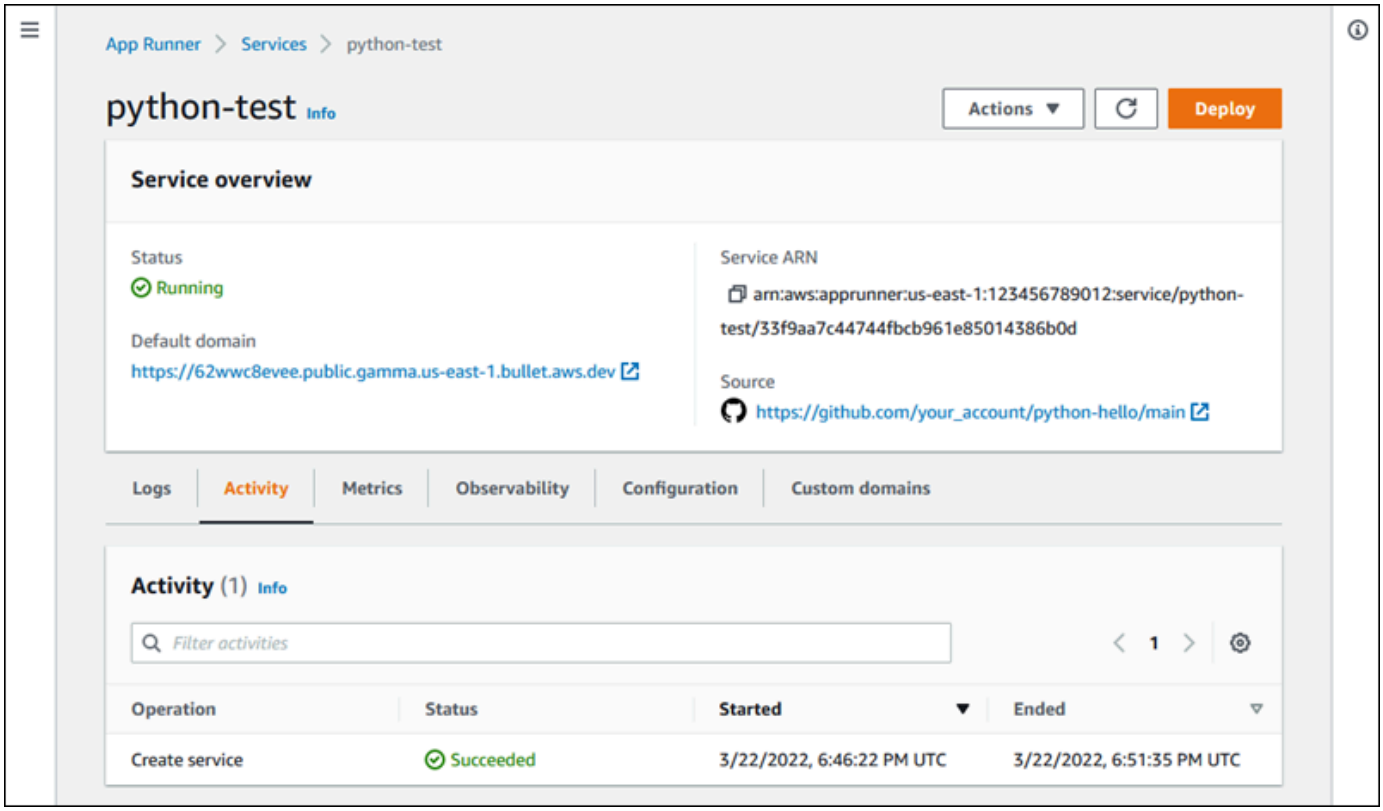
3단계: 구성 변경

이 단계에서는 서비스 구성 변경을 보여주기 위해 **NAME** 환경 변수 값을 변경합니다.

환경 변수 값을 변경하려면

1. App Runner 콘솔을 열고 리전 목록에서를 선택합니다 AWS 리전.
2. 탐색 창에서 서비스를 선택한 다음 App Runner 서비스를 선택합니다.

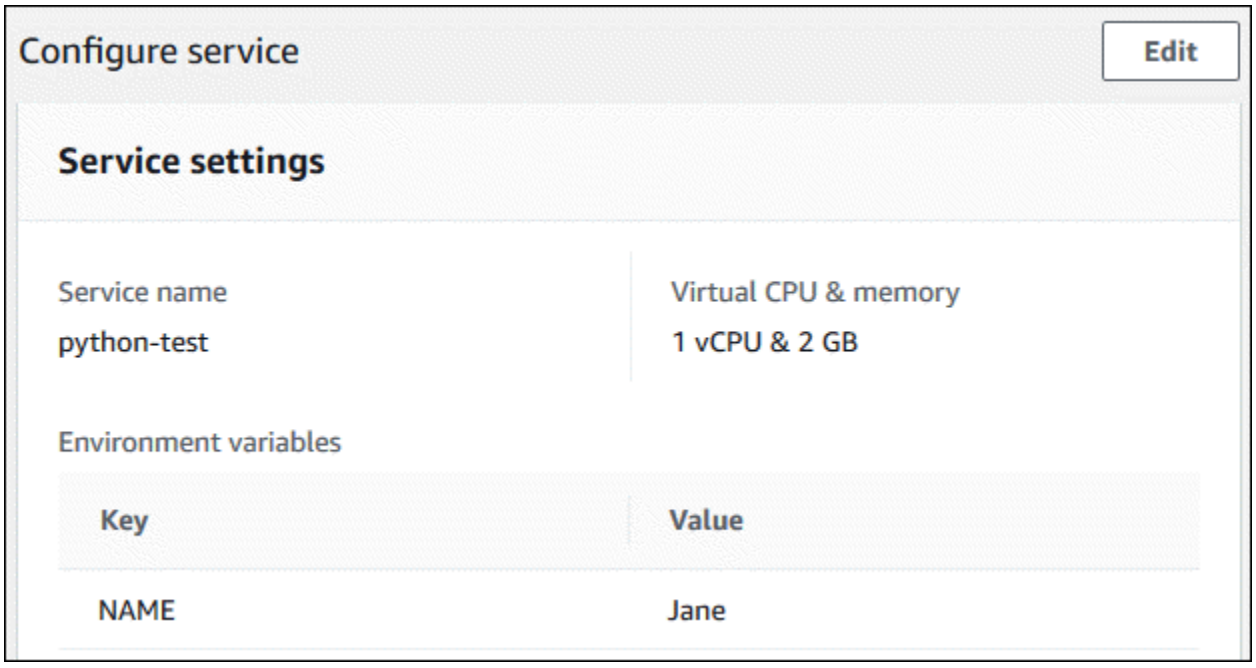
   콘솔에 서비스 개요와 함께 서비스 대시보드가 표시됩니다.



3. 서비스 대시보드 페이지에서 구성 탭을 선택합니다.

   콘솔에는 서비스 구성 설정이 여러 섹션에 표시됩니다.

4. 서비스 구성 섹션에서 편집을 선택합니다.



5. 키가 인 환경 변수의 경우 값을 다른 이름으로 **NAME**변경합니다.

6. [Apply changes]를 선택합니다.

   App Runner가 업데이트 프로세스를 시작합니다. 서비스 대시보드 페이지에서 서비스 상태가 진행 중인 작업으로 변경됩니다.

7. 업데이트가 종료될 때까지 기다립니다. 서비스 대시보드 페이지에서 서비스 상태가 다시 실행 중으로 변경되어야 합니다.

8. 업데이트가 성공했는지 확인합니다. 서비스의 웹 페이지가 표시되는 브라우저 탭을 새로 고칩니다.

   이제 페이지에 수정된 이름인 안녕하세요, **# ###** 표시됩니다!

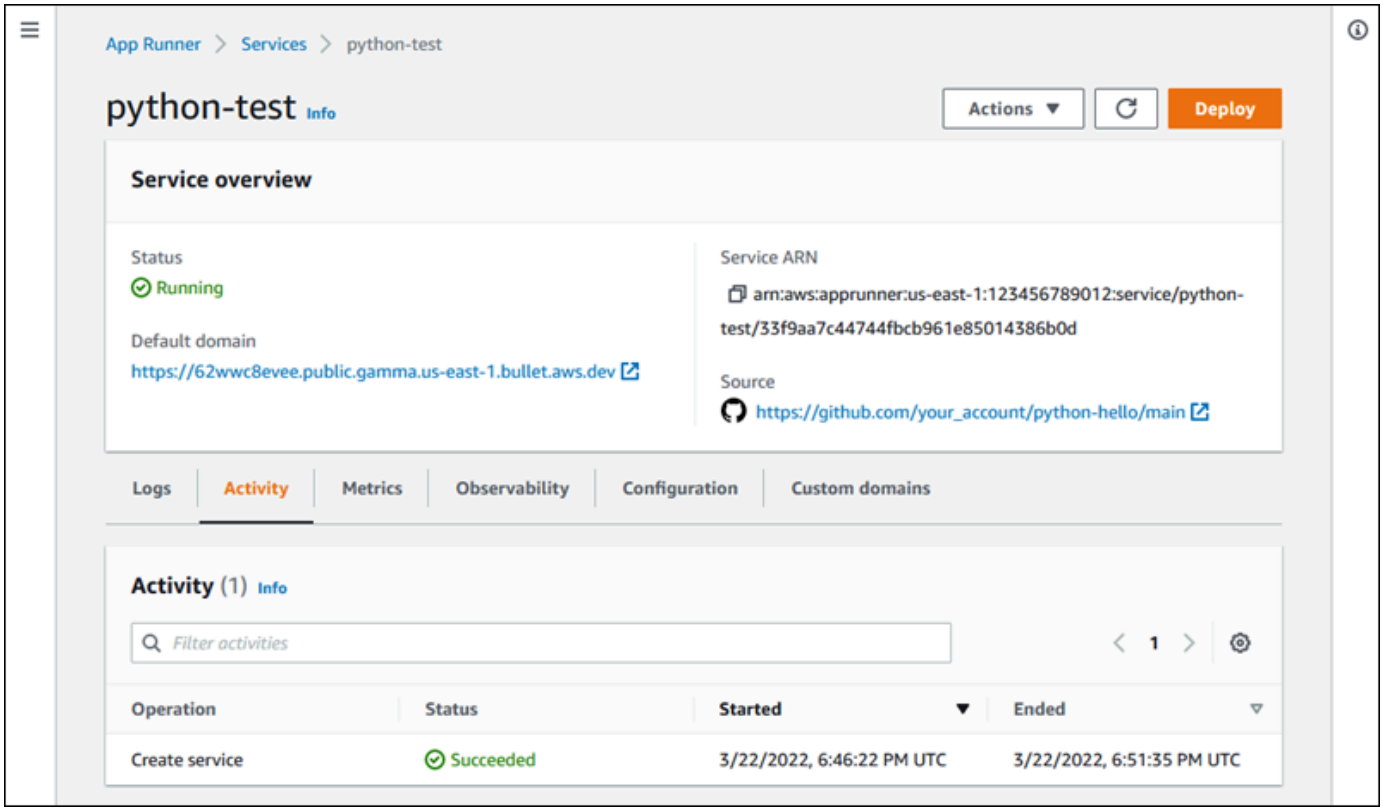
4단계: 서비스에 대한 로그 보기

이 단계에서는 App Runner 콘솔을 사용하여 App Runner 서비스에 대한 로그를 봅니다. App Runner는 로그를 Amazon CloudWatch Logs(CloudWatch Logs)로 스트리밍하고 서비스의 대시보드에 표시합니다. App Runner 로그에 대한 자세한 내용은 섹션을 참조하세요the section called “로그 (CloudWatch Logs)”.

서비스에 대한 로그를 보려면

1. App Runner 콘솔을 열고 리전 목록에서를 선택합니다 AWS 리전.
2. 탐색 창에서 서비스를 선택한 다음 App Runner 서비스를 선택합니다.

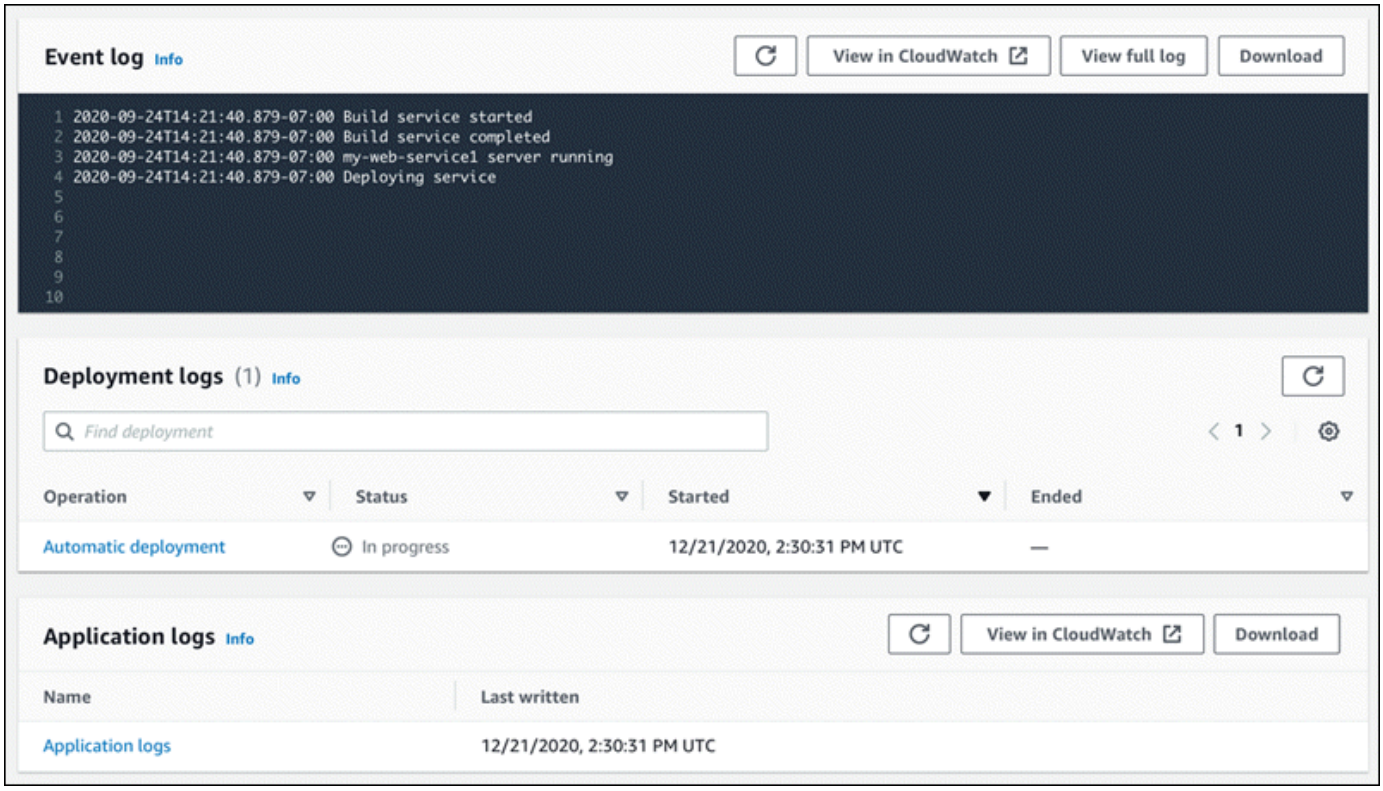
   콘솔에 서비스 개요와 함께 서비스 대시보드가 표시됩니다.



3. 서비스 대시보드 페이지에서 로그 탭을 선택합니다.

   콘솔에는 몇 가지 유형의 로그가 여러 섹션에 표시됩니다.

   - 이벤트 로그 - App Runner 서비스의 수명 주기 내 활동입니다. 콘솔에 최신 이벤트가 표시됩니다.
   - 배포 로그 - App Runner 서비스에 대한 소스 리포지토리 배포입니다. 콘솔에는 각 배포에 대해 별도의 로그 스트림이 표시됩니다.
   - 애플리케이션 로그 - App Runner 서비스에 배포된 웹 애플리케이션의 출력입니다. 콘솔은 실행 중인 모든 인스턴스의 출력을 단일 로그 스트림으로 결합합니다.



4. 특정 배포를 찾으려면 검색어를 입력하여 배포 로그 목록의 범위를 좁힙니다. 테이블에 나타나는 모든 값을 검색할 수 있습니다.
5. 로그의 콘텐츠를 보려면 전체 로그 보기(이벤트 로그) 또는 로그 스트림 이름(배포 및 애플리케이션 로그)을 선택합니다.
6. 다운로드를 선택하여 로그를 다운로드합니다. 배포 로그 스트림의 경우 먼저 로그 스트림을 선택합니다.
7. CloudWatch에서 보기를 선택하여 CloudWatch 콘솔을 열고 전체 기능을 사용하여 App Runner 서비스 로그를 탐색합니다. 배포 로그 스트림의 경우 먼저 로그 스트림을 선택합니다.

> **Note**
>
> CloudWatch 콘솔은 결합된 애플리케이션 로그 대신 특정 인스턴스의 애플리케이션 로그를 보려는 경우에 특히 유용합니다.

5단계: 정리

이제 App Runner 서비스를 생성하고, 로그를 보고, 몇 가지를 변경하는 방법을 배웠습니다. 이 단계에서는 서비스를 삭제하여 더 이상 필요하지 않은 리소스를 제거합니다.

서비스를 삭제하려면

1. 서비스 대시보드 페이지에서 작업을 선택한 다음 서비스 삭제를 선택합니다.
2. 확인 대화 상자에서 요청된 텍스트를 입력한 다음 삭제를 선택합니다.

   결과: 콘솔이 서비스 페이지로 이동합니다. 방금 삭제한 서비스는 삭제 중 상태로 표시됩니다. 잠시 후 목록에서 사라집니다.

이 자습서의 일부로 생성한 GitHub 및 Bitbucket 연결도 삭제하는 것이 좋습니다. 자세한 내용은 the section called “연결” 단원을 참조하십시오.

다음에 있는 것

이제 첫 번째 App Runner 서비스를 배포했으므로 다음 주제에서 자세히 알아보세요.

- 아키텍처 및 개념 - App Runner와 관련된 아키텍처, 주요 개념 및 AWS 리소스입니다.
- 이미지 기반 서비스 및 코드 기반 서비스 - App Runner가 배포할 수 있는 두 가지 유형의 애플리케이션 소스입니다.
- App Runner용 개발 - App Runner에 배포할 애플리케이션 코드를 개발하거나 마이그레이션할 때 알아야 할 사항입니다.
- App Runner 콘솔 - App Runner 콘솔을 사용하여 서비스를 관리하고 모니터링합니다.
- 서비스 관리 - App Runner 서비스의 수명 주기를 관리합니다.
- 관찰성 - 지표 모니터링, 로그 읽기, 이벤트 처리, 서비스 작업 호출 추적, HTTP 호출과 같은 애플리케이션 이벤트 추적을 통해 App Runner 서비스 작업에 대한 가시성을 확보합니다.
- App Runner 구성 파일 - App Runner 서비스의 빌드 및 런타임 동작에 대한 옵션을 지정하는 구성 기반 방법입니다.
- App Runner API - App Runner 애플리케이션 프로그래밍 인터페이스(API)를 사용하여 App Runner 리소스를 생성, 읽기, 업데이트 및 삭제합니다.
- 보안 - App Runner 및 기타 서비스를 사용하는 동안 AWS 및가 클라우드 보안을 보장하는 다양한 방법입니다.

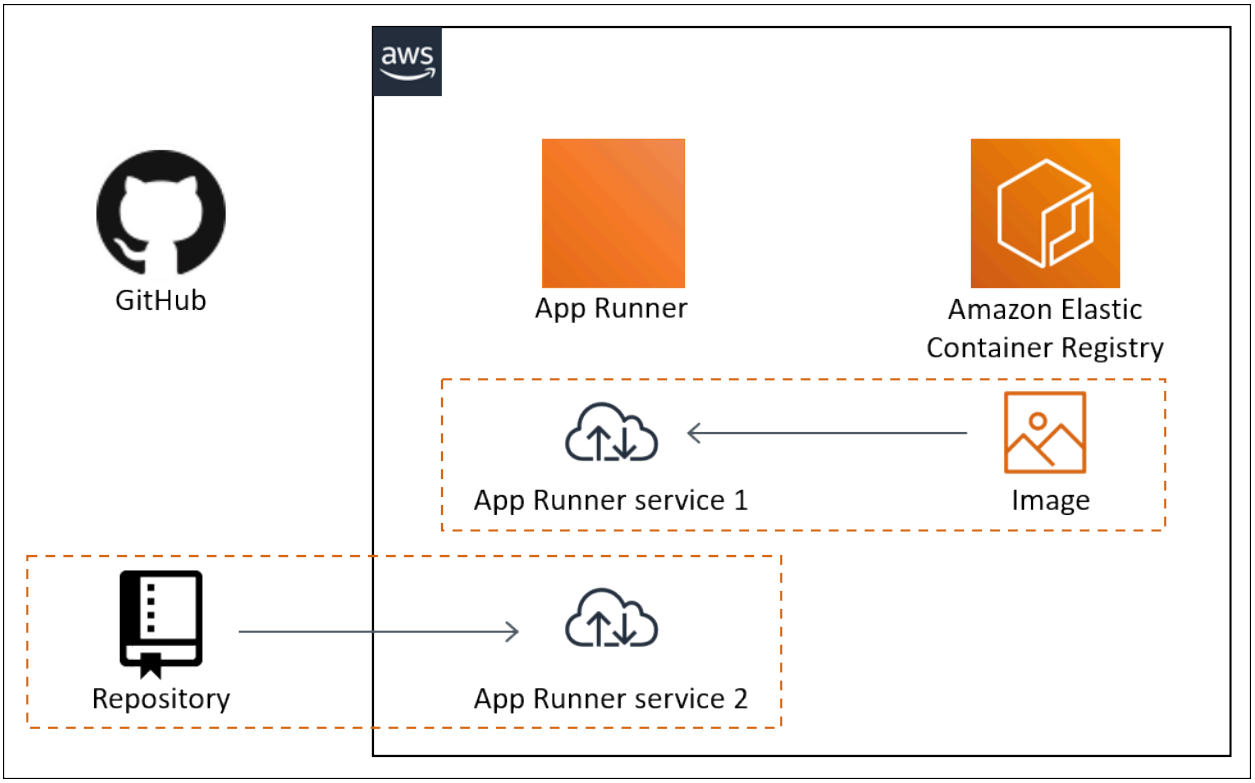
App Runner 아키텍처 및 개념

AWS App Runner 는 리포지토리에서 소스 코드 또는 소스 이미지를 가져와에서 실행 중인 웹 서비스를 생성하고 유지 관리합니다 AWS 클라우드. 일반적으로 서비스를 생성하려면 App Runner 작업 CreateService를 하나만 호출해야 합니다.

소스 이미지 리포지토리를 사용하면 App Runner가 웹 서비스를 실행하기 위해 배포할 수 있는 ready-to-use 가능한 컨테이너 이미지를 제공할 수 있습니다. 소스 코드 리포지토리를 사용하면 웹 서비스를 구축하고 실행하기 위한 코드와 지침을 제공하고 특정 런타임 환경을 대상으로 합니다. App Runner는 플랫폼 메이저 버전에 대해 각각 하나 이상의 관리형 런타임이 있는 여러 프로그래밍 플랫폼을 지원합니다.

현재 App Runner는 Bitbucket 또는 GitHub 리포지토리에서 소스 코드를 검색하거나의 Amazon Elastic Container Registry(Amazon ECR)에서 소스 이미지를 검색할 수 있습니다 AWS 계정.

다음 다이어그램은 App Runner 서비스 아키텍처의 개요를 보여줍니다. 다이어그램에는 두 가지 예제 서비스가 있습니다. 하나는 GitHub에서 소스 코드를 배포하고 다른 하나는 Amazon ECR에서 소스 이미지를 배포합니다. Bitbucket 리포지토리에도 동일한 흐름이 적용됩니다.



App Runner 개념

다음은 App Runner에서 실행 중인 웹 서비스와 관련된 주요 개념입니다.

- App Runner 서비스 - App Runner가 소스 코드 리포지토리 또는 컨테이너 이미지를 기반으로 애플리케이션을 배포하고 관리하는 데 사용하는 AWS 리소스입니다. App Runner 서비스는 애플리케이션의 실행 버전입니다. 서비스 생성에 대한 자세한 내용은 섹션을 참조하세요the section called “만들기”.
- 소스 유형 - App Runner 서비스를 배포하기 위해 제공하는 소스 리포지토리 유형: 소스 코드 또는 소스 이미지.
- 리포지토리 공급자 - 애플리케이션 소스(예: GitHub, Bitbucket 또는 Amazon ECR)가 포함된 리포지토리 서비스입니다.
- App Runner 연결 - App Runner가 리포지토리 공급자 계정(예: GitHub 계정 또는 조직)에 액세스할 수 있도록 하는 AWS 리소스입니다. 연결에 대한 자세한 내용은 the section called “연결” 섹션을 참조하세요.
- 런타임 - 소스 코드 리포지토리를 배포하기 위한 기본 이미지입니다. App Runner는 다양한 프로그래밍 플랫폼 및 버전을 위한 다양한 관리형 런타임을 제공합니다. 자세한 내용은 코드 기반 서비스 단원을 참조하십시오.
- 배포 - 소스 리포지토리(코드 또는 이미지)의 버전을 App Runner 서비스에 적용하는 작업입니다. 서비스에 대한 첫 번째 배포는 서비스 생성의 일부로 이루어집니다. 이후 배포는 다음 두 가지 방법 중 하나로 발생할 수 있습니다.
  - 자동 배포 - CI/CD 기능입니다. 리포지토리에 표시된 대로 애플리케이션의 각 버전을 자동으로 빌드(소스 코드용)하고 배포하도록 App Runner 서비스를 구성할 수 있습니다. 이는 소스 코드 리포지토리의 새 커밋이거나 소스 이미지 리포지토리의 새 이미지 버전일 수 있습니다.
  - 수동 배포 - 명시적으로 시작하는 App Runner 서비스에 대한 배포입니다.
- 사용자 지정 도메인 - App Runner 서비스와 연결하는 도메인입니다. 웹 애플리케이션의 사용자는이 도메인을 사용하여 기본 App Runner 하위 도메인 대신 웹 서비스에 액세스할 수 있습니다. 자세한 내용은 the section called “사용자 지정 도메인 이름” 단원을 참조하십시오.

> **Note**
>
> App Runner 애플리케이션의 보안을 강화하기 위해 *.awsapprunner.com 도메인은 퍼블릭 접미사 목록(PSL)에 등록됩니다. 보안을 강화하려면 App Runner 애플리케이션의 기본 도메인 이름에 민감한 쿠키를 설정해야 하는 경우 `__Host-` 접두사가 있는 쿠키를 사용하는 것이 좋습니다. 이렇게 쿠키를 설정하면 교차 사이트 요청 위조 시도(CSRF)로부터 도메인을

보호하는 데 도움이 됩니다. 자세한 내용은 Mozilla 개발자 네트워크의 Set-Cookie 페이지를 참조하세요.

- 유지 관리 - App Runner가 App Runner 서비스를 실행하는 인프라에서 가끔 수행하는 활동입니다. 유지 관리가 진행 중이면 서비스 상태가 일시적으로 몇 분 동안 `OPERATION_IN_PROGRESS` (콘솔에서 작업 진행 중)으로 변경됩니다. 이 시간 동안 서비스에 대한 작업(예: 배포, 구성 업데이트, 일시 중지/재개 또는 삭제)이 차단됩니다. 서비스 상태가 로 반환되면 몇 분 후에 작업을 다시 시도합니다`RUNNING`.

  > **Note**
  >
  > 작업이 실패하더라도 App Runner 서비스가 중단되었다는 의미는 아닙니다. 애플리케이션이 활성 상태이고 요청을 계속 처리합니다. 서비스에서 가동 중지가 발생할 가능성은 거의 없습니다.

  특히 App Runner는 서비스를 호스팅하는 기본 하드웨어에서 문제를 감지하면 서비스를 마이그레이션합니다. 서비스 가동 중지 시간을 방지하기 위해 App Runner는 서비스를 새 인스턴스 세트에 배포하고 트래픽을 인스턴스 세트로 이동합니다(블루-그린 배포). 가끔 요금이 일시적으로 약간 증가할 수 있습니다.

App Runner 지원 구성

App Runner 서비스를 구성할 때 서비스에 할당할 가상 CPU 및 메모리 구성을 지정합니다. 선택한 컴퓨팅 구성을 기준으로 비용을 지불합니다. 요금에 대한 자세한 내용은 AWS Resource Groups 요금을 참조하세요.

다음 표에서는 App Runner가 지원하는 vCPU 및 메모리 구성에 대한 정보를 제공합니다.

CPU	메모리
0.25 vCPU	0.5GB
0.25 vCPU	1GB
0.5 vCPU	1GB
1 vCPU	2GB

CPU	메모리
1 vCPU	3GB
1 vCPU	4GB
2 vCPU	4GB
2 vCPU	6GB
4 vCPU	8GB
4 vCPU	10GB
4 vCPU	12GB

App Runner 리소스

App Runner를 사용하는 경우에서 몇 가지 유형의 리소스를 생성하고 관리합니다 AWS 계정. 이러한 리소스는 코드에 액세스하고 서비스를 관리하는 데 사용됩니다.

다음 표에는 이러한 리소스에 대한 개요가 나와 있습니다.

리소스 이름	설명
Service	실행 중인 애플리케이션 버전을 나타냅니다. 이 가이드의 나머지 부분에서는 서비스 유형, 관리, 구성 및 모니터링에 대해 설명합니다. ARN: `arn:aws:apprunner:` *`region`*`:`*`account-id`* `:service/` *`service-name`* `[/`*`service-id`* `]`
Connection	App Runner 서비스에 타사 공급자에 저장된 프라이빗 리포지토리에 대한 액세스 권한을 제공합니다. 여러 서비스에서 공유하기 위한 별도의 리소스로 존재합니다. 연결에 대한 자세한 내용은 the section called “연결” 섹션을 참조하세요. ARN: `arn:aws:apprunner:` *`region`*`:`*`account-id`* `:connection/` *`connection-name`* `[/`*`connection-id`* `]`

리소스 이름	설명
AutoScalingConfiguration	App Runner 서비스에 애플리케이션의 자동 크기 조정을 제어하는 설정을 제공합니다. 여러 서비스에서 공유하기 위한 별도의 리소스로 존재합니다. 자동 스케일링에 대한 자세한 내용은 the section called “Auto Scaling” 섹션을 참조하세요. ARN: `arn:aws:apprunner:` *`region`*`:`*`account-id`* `:autoscalingconfiguration/` *`config-name`* `[/`*`config-revision`* `[/`*`config-id`* `]]`
ObservabilityConfiguration	App Runner 서비스에 대한 추가 애플리케이션 관찰성 기능을 구성합니다. 여러 서비스에서 공유하기 위한 별도의 리소스로 존재합니다. 관찰성 구성에 대한 자세한 내용은 섹션을 참조하세요the section called “관찰성 구성”. ARN: `arn:aws:apprunner:` *`region`*`:`*`account-id`* `:observabilityconfiguration/` *`config-name`* `[/`*`config-revision`* `[/`*`config-id`* `]]`
VpcConnector	App Runner 서비스에 대한 VPC 설정을 구성합니다. 여러 서비스에서 공유하기 위한 별도의 리소스로 존재합니다. VPC 기능에 대한 자세한 내용은 섹션을 참조하세요the section called “발신 트래픽”. ARN: `arn:aws:apprunner:` *`region`*`:`*`account-id`* `:vpcconnector/` *`connector-name`* `[/`*`connector-revision`* `[/`*`connector-id`* `]]`
VpcIngressConnection	수신 트래픽을 구성하는 데 사용되는 AWS App Runner 리소스입니다. VPC 인터페이스 엔드포인트와 App Runner 서비스 간의 연결을 설정하여 Amazon VPC 내에서만 App Runner 서비스에 액세스할 수 있도록 합니다. VPCIngressConnection의 기능에 대한 자세한 내용은 섹션을 참조하세요the section called “프라이빗 엔드포인트 활성화”. ARN: `arn:aws:apprunner:` *`region`*`:`*`account-id`* `:vpcingressconnection/` *`vpc-ingress-connection-name`* `[/`*`connector-id`* `]]`

App Runner 리소스 할당량

AWS 는 각각에서 AWS 리소스 사용에 대해 계정에 일부 할당량(한도라고도 함)을 부과합니다 AWS 리전. 다음 표에는 App Runner 리소스와 관련된 할당량이 나열되어 있습니다. 할당량은의 AWS App Runner 엔드포인트 및 할당량에도 나열됩니다AWS 일반 참조.

리소스 할당량		설명	기본값	조정 가능 여부
Services		계정에서 각각 생성할 수 있는 최대 서비스 수입니다 AWS 리전.	30	✓ 예
Connections		계정에서 각각 생성할 수 있는 최대 연결 수입니다 AWS 리전. 여러 서비스에서 하나의 VPC 커넥터를 사용할 수 있습니다.	10	✓ 예
Auto scaling configurations	이름	계정에서 각각 생성하는 오토 스케일링 구성에 포함할 수 있는 최대 고유 이름 수입니다 AWS 리전. 여러 서비스에서 하나의 Auto scaling 구성을 사용할 수 있습니다.	10	✓ 예
	이름당 개정	계정에서 각 고유 이름에 대해 생성할 수 있는 최대 Auto Scaling 구성 개정 수 AWS 리전 입니다. 여러 서비스에서 단일 Auto Scaling 구성 개정을 사용할 수 있습니다.	5	× 아니요
Observability configurations	이름	계정에서 각각 생성하는 관찰성 구성에 포함할 수 있는 최대 고유 이름 수입니다 AWS 리전. 여러 서비스에서 하나의 관찰성 구성을 사용할 수 있습니다.	10	✓ 예
	이름당 개정	AWS 리전 계정에서 각 고유 이름에 대해 생성할 수 있는 최대 관찰성 구성 개정 수입니다. 여러 서비스에서 단일 관찰성 구성 개정을 사용할 수 있습니다.	10	× 아니요

리소스 할당량	설명	기본값	조정 가능 여부
VPC connectors	계정에서 생성할 수 있는 각 VPC 커넥터의 최대 수입니다 AWS 리전. 여러 서비스에서 하나의 VPC 커넥터를 사용할 수 있습니다.	10	✓ 예
VPC Ingress Connection	계정에서 각각 생성할 수 있는 최대 VPC 수신 연결 수입니다 AWS 리전. 단일 VPC 수신 연결을 사용하여 여러 App Runner 서비스에 액세스할 수 있습니다.	1	× 아니요

대부분의 할당량은 조정 가능하며 할당량 증가를 요청할 수 있습니다. 자세한 내용은 Service Quotas 사용 설명서에서 할당량 증가 요청을 참조하세요.

소스 이미지를 기반으로 하는 App Runner 서비스

AWS App Runner 를 사용하여 기본적으로 서로 다른 두 가지 유형의 서비스 소스, 즉 소스 코드와 소스 이미지를 기반으로 서비스를 생성하고 관리할 수 있습니다. 소스 유형에 관계없이 App Runner는 서비스 시작, 실행, 조정 및 로드 밸런싱을 처리합니다. App Runner의 CI/CD 기능을 사용하여 소스 이미지 또는 코드의 변경 사항을 추적할 수 있습니다. App Runner는 변경 사항을 발견하면 자동으로 빌드(소스 코드용)하고 새 버전을 App Runner 서비스에 배포합니다.

이 장에서는 소스 이미지를 기반으로 하는 서비스에 대해 설명합니다. 소스 코드 기반 서비스에 대한 자세한 내용은 섹션을 참조하세요코드 기반 서비스.

소스 이미지는 이미지 리포지토리에 저장된 퍼블릭 또는 프라이빗 컨테이너 이미지입니다. App Runner는 이미지를 가리키며이 이미지를 기반으로 컨테이너를 실행하는 서비스를 시작합니다. 빌드 단계는 필요하지 않습니다. 대신 ready-to-deploy 수 있는 이미지를 제공합니다.

> **Note**
>
> 컨테이너 이미지를 제공할 때 사용자는 이러한 이미지를 정기적으로 업데이트하고 패치를 적용할 책임이 있습니다. App Runner가 인프라를 관리하는 동안 제공된 컨테이너 이미지의 보안 및 up-to-date 상태를 확인해야 합니다. 자세한 내용은 AWS App Runner 설명서를 참조하세요.

이미지 리포지토리 공급자

App Runner는 다음 이미지 저장소 제공업체를 지원합니다.

- Amazon Elastic Container Registry(Amazon ECR) -에 비공개인 이미지를 저장합니다 AWS 계정.
- Amazon Elastic Container Registry Public(Amazon ECR Public) - 공개적으로 읽을 수 있는 이미지를 저장합니다.

공급자 사용 사례

- AWS 계정의 Amazon ECR에 저장된 이미지 사용
- 다른 AWS 계정의 Amazon ECR에 저장된 이미지 사용
- Amazon ECR Public에 저장된 이미지 사용

AWS 계정의 Amazon ECR에 저장된 이미지 사용

Amazon ECR은 이미지를 리포지토리에 저장합니다. 프라이빗 및 퍼블릭 리포지토리가 있습니다. 프라이빗 리포지토리에서 App Runner 서비스에 이미지를 배포하려면 App Runner가 Amazon ECR에서 이미지를 읽을 수 있는 권한이 필요합니다. App Runner에 권한을 부여하려면 App Runner에 액세스 역할을 제공해야 합니다. 이는 필요한 Amazon ECR 작업 권한이 있는 AWS Identity and Access Management (IAM) 역할입니다. App Runner 콘솔을 사용하여 서비스를 생성할 때 계정에서 기존 역할을 선택할 수 있습니다. 또는 IAM 콘솔을 사용하여 새 사용자 지정 역할을 생성할 수 있습니다. 또는 App Runner 콘솔에서 관리형 정책에 따라 역할을 생성하도록 선택할 수 있습니다.

App Runner API 또는를 사용하는 경우 2단계 프로세스를 AWS CLI완료합니다. 먼저 IAM 콘솔을 사용하여 액세스 역할을 생성합니다. App Runner가 제공하는 관리형 정책을 사용하거나 사용자 지정 권한을 입력할 수 있습니다. 그런 다음 CreateService API 작업을 사용하여 서비스 생성 중에 액세스 역할을 제공합니다.

App Runner 서비스 생성에 대한 자세한 내용은 섹션을 참조하세요the section called “만들기”.

다른 AWS 계정의 Amazon ECR에 저장된 이미지 사용

App Runner 서비스를 생성할 때 서비스가 속한 계정이 아닌 다른 AWS 계정에 속하는 Amazon ECR 리포지토리에 저장된 이미지를 사용할 수 있습니다. 이전 섹션에서 동일 계정 이미지에 대해 나열된 것 외에도 교차 계정 이미지를 사용할 때 고려해야 할 몇 가지 추가 고려 사항이 있습니다.

- 교차 계정 리포지토리에는 정책이 연결되어 있어야 합니다. 리포지토리 정책은 액세스 역할에 리포지토리의 이미지를 읽을 수 있는 권한을 제공합니다. 이를 위해 다음 정책을 사용합니다. `Principal` 요소의 역할을 액세스 역할의 Amazon 리소스 이름(ARN)으로 바꿉니다.

  JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {"AWS": "arn:aws:iam::123456789012:role/MyApplicationRole"},
      "Action": [
        "ecr:BatchGetImage",
        "ecr:DescribeImages",
        "ecr:GetDownloadUrlForLayer"
        ],
        "Resource": "arn:aws:ecr:us-east-1:111122223333:repository/*"
```

```
    }
  ]
}
```

Amazon ECR 리포지토리에 리포지토리 정책을 연결하는 방법에 대한 자세한 내용은 Amazon Elastic Container Registry 사용 설명서의 리포지토리 정책 설명 설정을 참조하세요.

- App Runner는 서비스가 있는 계정과 다른 계정의 Amazon ECR 이미지에 대한 자동 배포를 지원하지 않습니다.

Amazon ECR Public에 저장된 이미지 사용

Amazon ECR Public은 공개적으로 읽을 수 있는 이미지를 저장합니다. 다음은 App Runner 서비스의 맥락에서 알아야 하는 Amazon ECR과 Amazon ECR Public의 주요 차이점입니다.

- Amazon ECR 퍼블릭 이미지는 공개적으로 읽을 수 있습니다. Amazon ECR 퍼블릭 이미지를 기반으로 서비스를 생성할 때 액세스 역할을 제공할 필요가 없습니다. 리포지토리에는 연결된 정책이 필요하지 않습니다.
- App Runner는 Amazon ECR 퍼블릭 이미지에 대한 자동(연속) 배포를 지원하지 않습니다.

Amazon ECR Public에서 직접 서비스 시작

Amazon ECR 퍼블릭 갤러리에서 호스팅되는 호환되는 웹 애플리케이션의 컨테이너 이미지를 App Runner에서 실행되는 웹 서비스로 직접 시작할 수 있습니다. 갤러리를 탐색할 때 갤러리 페이지에서 App Runner로 시작을 찾아 이미지를 찾습니다. 이 옵션이 있는 이미지는 App Runner와 호환됩니다. 갤러리에 대한 자세한 내용은 Amazon ECR Public 사용 설명서의 Amazon ECR Public Gallery 사용을 참조하세요.

갤러리 이미지를 App Runner 서비스로 시작하려면

1. 이미지의 갤러리 페이지에서 App Runner로 시작을 선택합니다.

   결과: App Runner 콘솔이 새 브라우저 탭에서 열립니다. 콘솔에 서비스 생성 마법사가 표시되고 필요한 새 서비스 세부 정보가 대부분 미리 채워집니다.
2. 콘솔에 표시된 AWS 리전이 아닌 다른 리전에서 서비스를 생성하려면 콘솔 헤더에 표시된 리전을 선택합니다. 그런 다음 다른 리전을 선택합니다.
3. 포트에 이미지 애플리케이션이 수신 대기하는 포트 번호를 입력합니다. 일반적으로 이미지의 갤러리 페이지에서 찾을 수 있습니다.
4. 선택적으로 다른 구성 세부 정보를 변경합니다.
5. 다음을 선택하고 설정을 검토한 다음 생성 및 배포를 선택합니다.

이미지 예제

App Runner 팀은 Amazon ECR 퍼블릭 갤러리에 hello-app-runner 예제 이미지를 유지합니다. 이 예제를 사용하여 이미지 기반 App Runner 서비스 생성을 시작할 수 있습니다. 자세한 내용은 hello-app-runner를 참조하세요.

소스 코드를 기반으로 하는 App Runner 서비스

AWS App Runner 를 사용하여 기본적으로 서로 다른 두 가지 유형의 서비스 소스, 즉 소스 코드와 소스 이미지를 기반으로 서비스를 생성하고 관리할 수 있습니다. 소스 유형에 관계없이 App Runner는 서비스 시작, 실행, 조정 및 로드 밸런싱을 처리합니다. App Runner의 CI/CD 기능을 사용하여 소스 이미지 또는 코드의 변경 사항을 추적할 수 있습니다. App Runner는 변경 사항을 발견하면 자동으로 빌드(소스 코드용)하고 새 버전을 App Runner 서비스에 배포합니다.

이 장에서는 소스 코드를 기반으로 하는 서비스에 대해 설명합니다. 소스 이미지를 기반으로 하는 서비스에 대한 자세한 내용은 섹션을 참조하세요이미지 기반 서비스.

소스 코드는 App Runner가 빌드하고 배포하는 애플리케이션 코드입니다. App Runner를 코드 리포지토리의 소스 디렉터리로 가리키고 프로그래밍 플랫폼 버전에 해당하는 적절한 런타임을 선택합니다. App Runner는 런타임의 기본 이미지와 애플리케이션 코드를 기반으로 이미지를 빌드합니다. 그런 다음이 이미지를 기반으로 컨테이너를 실행하는 서비스를 시작합니다.

App Runner는 편리한 플랫폼별 관리형 런타임을 제공합니다. 이러한 각 런타임은 소스 코드에서 컨테이너 이미지를 빌드하고 이미지에 언어 런타임 종속성을 추가합니다. Dockerfile과 같은 컨테이너 구성 및 빌드 지침을 제공할 필요가 없습니다.

이 장의 하위 주제에서는 App Runner가 지원하는 다양한 플랫폼, 즉 다양한 프로그래밍 환경 및 버전에 대한 관리형 런타임을 제공하는 관리형 플랫폼에 대해 설명합니다.

주제

소스 코드 리포지토리 공급자

App Runner는 소스 코드 리포지토리에서 소스 코드를 읽어 배포합니다. App Runner는 GitHub와 Bitbucket이라는 두 개의 소스 코드 리포지토리 공급자를 지원합니다.

소스 코드 리포지토리 공급자에서 배포

소스 코드 리포지토리에서 App Runner 서비스에 소스 코드를 배포하기 위해 App Runner는 소스 코드에 대한 연결을 설정합니다. App Runner 콘솔을 사용하여 서비스를 생성할 때 App Runner가 소스 코드를 배포할 수 있도록 연결 세부 정보와 소스 디렉터리를 제공합니다.

연결

서비스 생성 절차의 일부로 연결 세부 정보를 제공합니다. App Runner API 또는를 사용하는 경우 AWS CLI연결은 별도의 리소스입니다. 먼저 CreateConnection API 작업을 사용하여 연결을 생성합니다. 그런 다음 CreateService API 작업을 사용하여 서비스 생성 중에 연결의 ARN을 제공합니다.

소스 디렉터리

서비스를 생성할 때 소스 디렉터리도 제공합니다. 기본적으로 App Runner는 리포지토리의 루트 디렉터리를 소스 디렉터리로 사용합니다. 소스 디렉터리는 애플리케이션의 소스 코드 및 구성 파일을 저장하는 소스 코드 리포지토리의 위치입니다. 빌드 및 시작 명령은 소스 디렉터리에서도 실행됩니다. App Runner API 또는 AWS CLI 를 사용하여 서비스를 생성하거나 업데이트할 때 CreateService 및 UpdateService API 작업에 소스 디렉터리를 제공합니다. 자세한 내용은 이어지는 소스 디렉터리 단원을 참조하십시오.

App Runner 서비스 생성에 대한 자세한 내용은 섹션을 참조하세요the section called “만들기”. App Runner 연결에 대한 자세한 내용은 섹션을 참조하세요the section called “연결”.

소스 디렉터리

App Runner 서비스를 생성할 때 리포지토리 및 브랜치와 함께 소스 디렉터리를 제공할 수 있습니다. 소스 디렉터리 필드의 값을 애플리케이션의 소스 코드 및 구성 파일을 저장하는 리포지토리 디렉터리 경로로 설정합니다. App Runner는 사용자가 제공하는 소스 디렉터리 경로에서 빌드 및 시작 명령을 실행합니다.

루트 리포지토리 디렉터리에서 소스 디렉터리 경로 값을 절대값으로 입력합니다. 값을 지정하지 않으면 기본적으로 리포지토리 루트 디렉터리라고도 하는 리포지토리 최상위 디렉터리로 설정됩니다.

또한 최상위 리포지토리 디렉터리 외에 다른 소스 디렉터리 경로를 제공할 수도 있습니다. 이는 모노리포지토리 아키텍처를 지원합니다. 즉, 여러 애플리케이션의 소스 코드가 하나의 리포지토리에 저장됩니다. 단일 모노레포에서 여러 App Runner 서비스를 생성하고 지원하려면 각 서비스를 생성할 때 서로 다른 소스 디렉터리를 지정합니다.

> **Note**
>
> 여러 App Runner 서비스에 동일한 소스 디렉터리를 지정하는 경우 두 서비스 모두 개별적으로 배포되고 작동합니다.

`apprunner.yaml` 구성 파일을 사용하여 서비스 파라미터를 정의하도록 선택한 경우 리포지토리의 소스 디렉터리 폴더에 배치합니다.

배포 트리거 옵션이 자동으로 설정된 경우 소스 디렉터리에서 커밋한 변경 사항이 자동 배포를 트리거합니다. 소스 디렉터리 경로의 변경 사항만 자동 배포를 트리거합니다. 소스 디렉터리의 위치가 자동 배포 범위에 어떤 영향을 미치는지 이해하는 것이 중요합니다. 자세한 내용은의 자동 배포를 참조하세요배포 방법.

> **Note**
>
> App Runner 서비스가 PHP 관리형 런타임을 사용하고 기본 루트 리포지토리 이외의 소스 디렉터리를 지정하려는 경우 올바른 PHP 런타임 버전을 사용하는 것이 중요합니다. 자세한 내용은 PHP 플랫폼 사용 단원을 참조하십시오.

App Runner 관리형 플랫폼

App Runner 관리형 플랫폼은 다양한 프로그래밍 환경을 위한 관리형 런타임을 제공합니다. 각 관리형 런타임을 사용하면 프로그래밍 언어 또는 런타임 환경의 버전을 기반으로 컨테이너를 쉽게 빌드하고 실행할 수 있습니다. 관리형 런타임을 사용하는 경우 App Runner는 관리형 런타임 이미지로 시작합니다. 이 이미지는 Amazon Linux Docker 이미지를 기반으로 하며 언어 런타임 패키지와 일부 도구 및 인기 있는 종속성 패키지를 포함합니다. App Runner는이 관리형 런타임 이미지를 기본 이미지로 사용하고 애플리케이션 코드를 추가하여 도커 이미지를 빌드합니다. 그런 다음이 이미지를 배포하여 컨테이너에서 웹 서비스를 실행합니다.

App Runner 콘솔 또는 CreateService API 작업을 사용하여 서비스를 생성할 때 App Runner 서비스에 대한 런타임을 지정합니다. 런타임을 소스 코드의 일부로 지정할 수도 있습니다. 코드 리포지토리에 포

함하는 App Runner 구성 파일의 `runtime` 키워드를 사용합니다. 관리형 런타임의 이름 지정 규칙은 *<language-name><major-version>*입니다.

App Runner는 모든 배포 또는 서비스 업데이트 시 서비스의 런타임을 최신 버전으로 업데이트합니다. 애플리케이션에 특정 버전의 관리형 런타임이 필요한 경우 App Runner 구성 파일의 `runtime-version` 키워드를 사용하여 지정할 수 있습니다. 메이저 또는 마이너 버전을 포함하여 모든 수준의 버전으로를 잠글 수 있습니다. App Runner는 서비스의 런타임만 하위 수준으로 업데이트합니다.

관리형 런타임 버전에 대한 지원 종료

관리형 언어 런타임의 공식 공급자 또는 커뮤니티가 공식적으로 버전을 수명 종료(EOL)로 선언하면 App Runner는 버전 상태를 지원 종료로 선언하여 다음 작업을 수행합니다. 서비스가 지원 종료에 도달한 관리형 언어 런타임 버전에서 실행 중인 경우 다음 정책 및 권장 사항이 적용됩니다.

언어 런타임 버전에 대한 지원 종료:

- 기존 서비스는 지원 종료에 도달한 런타임을 사용하더라도 계속 실행되고 트래픽을 제공합니다. 그러나 더 이상 업데이트, 보안 패치 또는 기술 지원을 받지 않는 지원되지 않는 런타임에서 실행됩니다.
- 지원 종료 런타임을 사용하는 기존 서비스에 대한 업데이트는 여전히 허용되지만 서비스에 지원 종료 런타임을 계속 사용하는 것은 권장하지 않습니다.
- 지원 종료 날짜에 도달한 런타임을 사용하여 새 서비스를 생성할 수 없습니다.

지원 종료 상태의 언어 런타임 버전에 필요한 작업:

- 서비스가 소스 이미지를 기반으로 하는 경우 해당 서비스에 대한 추가 작업이 필요하지 않습니다.
- 서비스가 소스 코드를 기반으로 하는 경우 지원되는 런타임 버전을 사용하도록 서비스 구성을 업데이트합니다. 이렇게 하려면 App Runner 콘솔에서 지원되는 런타임 버전을 선택하거나, apprunner.yaml 구성 파일의 `runtime` 필드를 업데이트하거나, CreateService/UpdateService API 작업 또는 IaC 도구를 사용하여 `runtime` 파라미터를 설정합니다. 지원되는 런타임 목록은이 장의 특정 런타임에 대한 릴리스 정보 페이지를 참조하세요.
- 또는 App Runner의 컨테이너 이미지 소스 옵션으로 전환할 수 있습니다. 자세한 내용은 이미지 기반 서비스 섹션을 참조하세요.

> **Note**
>
> Node.js 12, 14 또는 16에서 Node.js 22로 또는 Python 3.7 또는 3.8에서 Python 3.11로 이동하는 경우 Node.js 22 및 Python 3.11은 더 빠르고 효율적인 빌드를 제공하는 수정된 App Runner 빌드 프로세스를 사용한다는 점에 유의하세요. 업그레이드하기 전에 호환성을 보장하려면 다음 섹션의 빌드 프로세스 지침을 검토하는 것이 좋습니다.

다음 표에는 지정된 지원 종료 날짜가 있는 App Runner 관리형 런타임 버전이 나열되어 있습니다.

런타임 버전	App Runner 지원 종료 날짜
Python 3.8 지원 런타임	2025년 12월 1일
Python 3.7 지원 런타임	2025년 12월 1일
Node.js 18 지원되는 런타임	2025년 12월 1일
Node.js 16 지원되는 런타임	2025년 12월 1일
Node.js 14 지원되는 런타임	2025년 12월 1일
Node.js 12 지원되는 런타임	2025년 12월 1일
.NET 6*	2025년 12월 1일
PHP 8.1*	2025년 12월 31일
Ruby 3.1*	2025년 12월 1일
Go 1*	2025년 12월 1일

* App Runner는 별표(*)로 표시된 런타임에 대한 새 언어 버전을 릴리스하지 않습니다. 이러한 런타임은 .NET, PHP, Ruby 및 Go입니다. 이러한 런타임에 대해 코드 기반 서비스가 구성된 경우 다음 작업 중 하나를 수행하는 것이 좋습니다.

- 해당하는 경우 서비스 구성을 지원되는 다른 관리형 런타임으로 전환합니다.
- 또는 원하는 런타임 버전으로 사용자 지정 컨테이너 이미지를 빌드하고 App Runner의 이미지 기반 서비스 옵션을 사용하여 배포합니다. Amazon ECR에서 이미지를 호스팅할 수 있습니다.

관리형 런타임 버전 및 App Runner 빌드

App Runner는 최신 메이저 버전 런타임에서 실행되는 애플리케이션을 위한 업데이트된 빌드 프로세스를 제공합니다. 이 수정된 빌드 프로세스는 더 빠르고 효율적입니다. 또한 애플리케이션을 실행하는 데 필요한 소스 코드, 빌드 아티팩트 및 런타임만 포함하는 더 작은 공간으로 최종 이미지를 생성합니다.

최신 빌드 프로세스를 수정된 App Runner 빌드라고 하고 원래 빌드 프로세스를 원래 App Runner 빌드라고 합니다. 이전 버전의 런타임 플랫폼에 대한 변경 사항을 방지하기 위해 App Runner는 일반적으로 새로 릴리스된 주요 릴리스인 특정 런타임 버전에만 수정된 빌드를 적용합니다.

매우 구체적인 사용 사례에 맞게 수정된 빌드를 이전 버전과 호환하고 애플리케이션 빌드를 구성할 수 있는 유연성을 높이기 위해 `apprunner.yaml` 구성 파일에 새 구성 요소를 도입했습니다. 이는 선택적 `pre-run` 파라미터입니다. 다음 섹션의 빌드에 대한 다른 유용한 정보와 함께이 파라미터를 사용해야 하는 경우를 설명합니다.

다음 표에는 특정 관리형 런타임 버전에 적용되는 App Runner 빌드 버전이 나와 있습니다. 현재 런타임에 대한 최신 정보를 제공하기 위해이 문서를 계속 업데이트할 예정입니다.

플랫폼	런타임 버전	빌드 프로세스	App Runner 지원 종료 날짜
Python – 릴리스 정보	Python 3.11(!)	개정	
	Python 3.8	원본	2025년 12월 1일
	Python 3.7	원본	2025년 12월 1일
Node.js – 릴리스 정보	Node.js 22	개정	
	Node.js 18	개정	2025년 12월 1일
	Node.js 16	원본	2025년 12월 1일
	Node.js 14	원본	2025년 12월 1일
	Node.js 12	원본	2025년 12월 1일
Corretto - 릴리스 정보	Corretto 11	원본	

플랫폼	런타임 버전	빌드 프로세스	App Runner 지원 종료 날짜
	Corretto 8	원본	
.NET – 릴리스 정보	.NET 6*	원본	2025년 12월 1일
PHP – 릴리스 정보	PHP 8.1*	원본	2025년 12월 31일
Ruby – 릴리스 정보	Ruby 3.1*	원본	2025년 12월 1일
Go – 릴리스 정보	Go 1*	원본	2025년 12월 1일

Note

나열된 런타임 중 일부에는 지원 종료 날짜가 포함됩니다. 자세한 내용은 the section called “관리형 런타임 버전에 대한 지원 종료” 단원을 참조하십시오.

Important

Python 3.11 - Python 3.11 관리형 런타임을 사용하는 서비스의 빌드 구성에 대한 특정 권장 사항이 있습니다. 자세한 내용은 Python 플랫폼 주제특정 런타임 버전에 대한 콜아웃의 섹션을 참조하세요.

App Runner 빌드 및 마이그레이션에 대해 자세히 알아보기

수정된 빌드를 사용하는 최신 런타임으로 애플리케이션을 마이그레이션할 때 빌드 구성을 약간 수정해야 할 수 있습니다.

마이그레이션 고려 사항에 대한 컨텍스트를 제공하기 위해 먼저 원래 App Runner 빌드와 수정된 빌드 모두에 대한 상위 수준 프로세스를 설명합니다. 다음 단원에서는 일부 구성 업데이트가 필요할 수 있는 서비스에 대한 특정 속성을 설명합니다.

원래 App Runner 빌드

원래 App Runner 애플리케이션 빌드 프로세스는 AWS CodeBuild 서비스를 활용합니다. 초기 단계는 CodeBuild 서비스에서 큐레이션한 이미지를 기반으로 합니다. 해당하는 App Runner 관리형 런타임 이미지를 기본 이미지로 사용하는 Docker 빌드 프로세스가 이어집니다.

일반적인 단계는 다음과 같습니다.

1. CodeBuild에서 선별한 이미지에서 `pre-build` 명령을 실행합니다.

   `pre-build` 명령은 선택 사항입니다. `apprunner.yaml` 구성 파일에서만 지정할 수 있습니다.
2. 이전 단계의 동일한 이미지에서 CodeBuild를 사용하여 `build` 명령을 실행합니다.

   `build` 명령이 필요합니다. App Runner 콘솔, App Runner API 또는 `apprunner.yaml` 구성 파일에서 지정할 수 있습니다.
3. Docker 빌드를 실행하여 특정 플랫폼 및 런타임 버전에 대한 App Runner 관리형 런타임 이미지를 기반으로 이미지를 생성합니다.
4. 2단계에서 생성한 이미지에서 `/app` 디렉터리를 복사합니다. 대상은 3단계에서 생성한 App Runner 관리형 런타임 이미지를 기반으로 하는 이미지입니다.
5. 생성된 App Runner 관리형 런타임 이미지에서 `build` 명령을 다시 실행합니다. 빌드 명령을 다시 실행하여 4단계에서 복사한 `/app` 디렉터리의 소스 코드에서 빌드 아티팩트를 생성합니다. 이 이미지는 나중에 App Runner에서 배포하여 컨테이너에서 웹 서비스를 실행합니다.

   `build` 명령이 필요합니다. App Runner 콘솔, App Runner API 또는 `apprunner.yaml` 구성 파일에서 지정할 수 있습니다.
6. 2단계의 CodeBuild 이미지에서 `post-build` 명령을 실행합니다.

   `post-build` 명령은 선택 사항입니다. `apprunner.yaml` 구성 파일에서만 지정할 수 있습니다.

빌드가 완료되면 App Runner는 5단계에서 생성된 App Runner 관리형 런타임 이미지를 배포하여 컨테이너에서 웹 서비스를 실행합니다.

수정된 App Runner 빌드

수정된 빌드 프로세스는 이전 섹션에 설명된 원래 빌드 프로세스보다 빠르고 효율적입니다. 이전 버전 빌드에서 발생하는 빌드 명령의 중복을 제거합니다. 또한 애플리케이션을 실행하는 데 필요한 소스 코드, 빌드 아티팩트 및 런타임만 포함하는 더 작은 공간으로 최종 이미지를 생성합니다.

이 빌드 프로세스는 Docker 다단계 빌드를 사용합니다. 일반적인 프로세스 단계는 다음과 같습니다.

1. 빌드 단계 - App Runner 빌드 이미지 위에 `pre-build` 및 `build` 명령을 실행하는 Docker 빌드 프로세스를 시작합니다.

   a. 애플리케이션 소스 코드를 `/app` 디렉터리에 복사합니다.

   > **Note**
   >
   > 이 `/app` 디렉터리는 Docker 빌드의 모든 단계에서 작업 디렉터리로 지정됩니다.

   b. `pre-build` 명령 실행

   명령은`pre-build` 선택 사항입니다. `apprunner.yaml` 구성 파일에서만 지정할 수 있습니다.

   c. `build` 명령을 실행합니다.

   `build` 명령이 필요합니다. App Runner 콘솔, App Runner API 또는 `apprunner.yaml` 구성 파일에서 지정할 수 있습니다.

2. 패키징 단계 - App Runner 실행 이미지를 기반으로 하는 최종 고객 컨테이너 이미지를 생성합니다.

   a. 이전 빌드 단계에서 새 실행 이미지로 `/app` 디렉터리를 복사합니다. 여기에는 애플리케이션 소스 코드와 이전 단계의 빌드 아티팩트가 포함됩니다.

   b. `pre-run` 명령을 실행합니다. `build` 명령을 사용하여 `/app` 디렉터리 외부에서 런타임 이미지를 수정해야 하는 경우 `apprunner.yaml` 구성 파일의이 세그먼트에 동일하거나 필요한 명령을 추가합니다.

   이는 수정된 App Runner 빌드를 지원하기 위해 도입된 새로운 파라미터입니다.

   `pre-run` 명령은 선택 사항입니다. `apprunner.yaml` 구성 파일에서만 지정할 수 있습니다.

   > **참고**
   >
   > - `pre-run` 명령은 수정된 빌드에서만 지원됩니다. 서비스에서 원래 빌드를 사용하는 런타임 버전을 사용하는 경우 구성 파일에 추가하지 마십시오.
   > - `build` 명령을 사용하여 `/app` 디렉터리 외부의 어떤 것도 수정할 필요가 없는 경우 `pre-run` 명령을 지정할 필요가 없습니다.

3. 빌드 후 단계 -이 단계는 빌드 단계에서 재개되고 `post-build` 명령을 실행합니다.

   a. `/app` 디렉터리 내에서 `post-build` 명령을 실행합니다.

   `post-build` 명령은 선택 사항입니다. `apprunner.yaml` 구성 파일에서만 지정할 수 있습니다.

빌드가 완료되면 App Runner는 실행 이미지를 배포하여 컨테이너에서 웹 서비스를 실행합니다.

> **Note**
>
> 빌드 프로세스를 구성할 `apprunner.yaml` 때의 실행 섹션에 있는 env 항목을 오해하지 마세요. 2(b)단계에서 참조된 `pre-run` 명령 파라미터가 실행 섹션에 있더라도 실행 섹션의 env 파라미터를 사용하여 빌드를 구성하지 마십시오. `pre-run` 명령은 구성 파일의 빌드 섹션에 정의된 env 변수만 참조합니다. 자세한 내용은 App Runner 구성 파일 실행 섹션 장의 섹션을 참조하세요.

마이그레이션 고려 사항에 대한 서비스 요구 사항

애플리케이션 환경에 이러한 두 요구 사항 중 하나가 있는 경우 `pre-run` 명령을 추가하여 빌드 구성을 수정해야 합니다.

- `build` 명령을 사용하여 `/app` 디렉터리 외부의 항목을 수정해야 하는 경우.
- `build` 명령을 두 번 실행하여 필요한 환경을 생성해야 하는 경우. 이는 매우 드문 요구 사항입니다. 대부분의 빌드는이 작업을 수행하지 않습니다.

`/app` 디렉터리 외부의 수정 사항

- 수정된 App Runner 빌드는 애플리케이션에 `/app` 디렉터리 외부의 종속성이 없다고 가정합니다.
- `apprunner.yaml` 파일, App Runner API 또는 App Runner 콘솔과 함께 제공하는 명령은 `/app` 디렉터리에서 빌드 아티팩트를 생성해야 합니다.
- `pre-build`, `build`및 `post-build` 명령을 수정하여 모든 빌드 아티팩트가 `/app` 디렉터리에 있는지 확인할 수 있습니다.
- 애플리케이션에 서비스에 대해 생성된 이미지를 추가로 수정하기 위한 빌드가 필요한 경우 `/app` 디렉터리 외부에서의 새 `pre-run` 명령을 사용할 수 있습니다`apprunner.yaml`. 자세한 내용은 구성 파일을 사용하여 App Runner 서비스 옵션 설정 단원을 참조하십시오.

`build` 명령을 두 번 실행

- 원래 App Runner 빌드는 `build` 명령을 두 번 실행합니다. 먼저 2단계에서 실행한 다음 5단계에서 다시 실행합니다. 수정된 App Runner 빌드는 이러한 중복성을 해결하고 `build` 명령을 한 번만 실행합니다. 애플리케이션이 `build` 명령을 두 번 실행해야 하는 비정상적인 요구 사항이 있는 경우 수정

된 App Runner 빌드는 `pre-run` 파라미터를 사용하여 동일한 명령을 다시 지정하고 실행할 수 있는 옵션을 제공합니다. 이렇게 하면 동일한 이중 빌드 동작이 유지됩니다.

Python 플랫폼 사용

> **Important**
>
> App Runner는 2025년 12월 1일에 Python 3.7 및 Python 3.8에 대한 지원을 종료합니다. 권장 사항 및 자세한 내용은 섹션을 참조하세요the section called “관리형 런타임 버전에 대한 지원 종료”.

AWS App Runner Python 플랫폼은 관리형 런타임을 제공합니다. 각 런타임을 사용하면 Python 버전을 기반으로 웹 애플리케이션을 사용하여 컨테이너를 쉽게 빌드하고 실행할 수 있습니다. Python 런타임을 사용하는 경우 App Runner는 관리형 Python 런타임 이미지로 시작합니다. 이 이미지는 Amazon Linux Docker 이미지를 기반으로 하며 Python 버전에 대한 런타임 패키지와 일부 도구 및 인기 있는 종속성 패키지를 포함합니다. App Runner는이 관리형 런타임 이미지를 기본 이미지로 사용하고 애플리케이션 코드를 추가하여 도커 이미지를 빌드합니다. 그런 다음이 이미지를 배포하여 컨테이너에서 웹 서비스를 실행합니다.

App Runner 콘솔 또는 CreateService API 작업을 사용하여 서비스를 생성할 때 App Runner 서비스에 대한 런타임을 지정합니다. 런타임을 소스 코드의 일부로 지정할 수도 있습니다. 코드 리포지토리에 포함하는 App Runner 구성 파일의 `runtime` 키워드를 사용합니다. 관리형 런타임의 이름 지정 규칙은 *`<language-name><major-version>`*입니다.

유효한 Python 런타임 이름 및 버전은 섹션을 참조하세요the section called “릴리스 정보”.

App Runner는 모든 배포 또는 서비스 업데이트 시 서비스의 런타임을 최신 버전으로 업데이트합니다. 애플리케이션에 특정 버전의 관리형 런타임이 필요한 경우 App Runner 구성 파일의 `runtime-version` 키워드를 사용하여 지정할 수 있습니다. 메이저 또는 마이너 버전을 포함하여 모든 수준의 버전으로를 잠글 수 있습니다. App Runner는 서비스의 런타임만 하위 수준으로 업데이트합니다.

Python 런타임의 버전 구문: *`major`*`[.`*`minor`*`[.`*`patch`*`]]`

예: `3.8.5`

다음 예제에서는 버전 잠금을 보여줍니다.

- `3.8` - 메이저 및 마이너 버전을 잠급니다. App Runner는 패치 버전만 업데이트합니다.

- `3.8.5` - 특정 패치 버전으로 잠급니다. App Runner는 런타임 버전을 업데이트하지 않습니다.

주제

Python 런타임 구성

관리형 런타임을 선택할 때는 최소한 빌드 및 실행 명령도 구성해야 합니다. App Runner 서비스를 생성하거나 업데이트하는 동안 이를 구성합니다. 다음 방법 중 하나를 사용하여이 작업을 수행할 수 있습니다.

- App Runner 콘솔 사용 - 생성 프로세스 또는 구성 탭의 빌드 구성 섹션에서 명령을 지정합니다.
- App Runner API 사용 - CreateService 또는 UpdateService API 작업을 호출합니다. CodeConfigurationValues 데이터 형식의 `BuildCommand` 및 `StartCommand` 멤버를 사용하여 명령을 지정합니다.
- 구성 파일 사용 - 최대 3개의 빌드 단계에서 하나 이상의 빌드 명령을 지정하고 애플리케이션을 시작하는 데 사용되는 단일 실행 명령을 지정합니다. 추가 선택적 구성 설정이 있습니다.

구성 파일을 제공하는 것은 선택 사항입니다. 콘솔 또는 API를 사용하여 App Runner 서비스를 생성할 때 App Runner가 구성 설정을 생성할 때 직접 가져오거나 구성 파일에서 가져올지 지정합니다.

특정 런타임 버전에 대한 콜아웃

> **Note**
>
> App Runner는 이제 Python 3.11, Node.js 22 및 Node.js 18 런타임 버전을 기반으로 애플리케이션에 대해 업데이트된 빌드 프로세스를 실행합니다. 애플리케이션이 이러한 런타임 버전 중 하나에서 실행되는 경우 수정된 빌드 프로세스에 대한 자세한 내용은 관리형 런타임 버전 및 App Runner 빌드 섹션을 참조하세요. 다른 모든 런타임 버전을 사용하는 애플리케이션은 영향을 받지 않으며 원래 빌드 프로세스를 계속 사용합니다.

Python 3.11(개정된 App Runner 빌드)

관리형 Python 3.11 런타임에 대해 apprunner.yaml에서 다음 설정을 사용합니다.

- 상단 섹션의 `runtime` 키를 로 설정 `python311`

  Example

  ```
  runtime: python311
  ```

- 대신를 사용하여 종속성을 `pip3` `pip` 설치합니다.
- 대신 인터`python3`프리터를 사용합니다`python`.
- `pip3` 설치 관리자를 `pre-run`명령으로 실행합니다. Python은 `/app` 디렉터리 외부에 종속성을 설치합니다. App Runner는 Python 3.11용 수정된 App Runner 빌드를 실행하므로 `apprunner.yaml` 파일의 빌드 섹션에 있는 명령을 통해 `/app` 디렉터리 외부에 설치된 모든 항목이 손실됩니다. 자세한 내용은 수정된 App Runner 빌드 단원을 참조하십시오.

  Example

  ```
  run:
    runtime-version: 3.11
    pre-run:
      - pip3 install pipenv
      - pipenv install
      - python3 copy-global-files.py
    command: pipenv run gunicorn django_apprunner.wsgi --log-file -
  ```

자세한 내용은이 주제 뒷부분의 Python 3.11에 대한 확장 구성 파일의 예제도 참조하세요.

Python 런타임 예제

다음 예제에서는 Python 서비스를 빌드하고 실행하기 위한 App Runner 구성 파일을 보여줍니다. 마지막 예제는 Python 런타임 서비스에 배포할 수 있는 전체 Python 애플리케이션의 소스 코드입니다.

> **Note**
>
> 이 예제에서 사용되는 런타임 버전은 *3.7.7* 및 *3.11*입니다. 사용하려는 버전으로 바꿀 수 있습니다. 지원되는 최신 Python 런타임 버전은 섹션을 참조하세요the section called “릴리스 정보”.

미니멀 Python 구성 파일

이 예제는 Python 관리형 런타임과 함께 사용할 수 있는 최소 구성 파일을 보여줍니다. App Runner가 최소 구성 파일로 가정하는 내용은 섹션을 참조하세요the section called “구성 파일 예제”.

Python 3.11은 `pip3` 및 `python3` 명령을 사용합니다. 자세한 내용은이 주제의 뒷부분에서 Python 3.11용 확장 구성 파일의 예를 참조하세요.

Example apprunner.yaml

```
version: 1.0
runtime: python3
build:
  commands:
    build:
      - pip install pipenv
      - pipenv install
run:
  command: python app.py
```

확장 Python 구성 파일

이 예제에서는 Python 관리형 런타임에서 모든 구성 키를 사용하는 방법을 보여줍니다.

> **Note**
>
> 이 예제에서 사용되는 런타임 버전은 *3.7.7*입니다. 사용하려는 버전으로 바꿀 수 있습니다. 지원되는 최신 Python 런타임 버전은 섹션을 참조하세요the section called “릴리스 정보”. Python 3.11은 `pip3` 및 `python3` 명령을 사용합니다. 자세한 내용은이 주제의 뒷부분에서 Python 3.11용 확장 구성 파일의 예를 참조하세요.

Example apprunner.yaml

```
version: 1.0
runtime: python3
build:
  commands:
    pre-build:
      - wget -c https://s3.amazonaws.com/amzn-s3-demo-bucket/test-lib.tar.gz -O - | tar
 -xz
```

```
    build:
      - pip install pipenv
      - pipenv install
    post-build:
      - python manage.py test
  env:
    - name: DJANGO_SETTINGS_MODULE
      value: "django_apprunner.settings"
    - name: MY_VAR_EXAMPLE
      value: "example"
run:
  runtime-version: 3.7.7
  command: pipenv run gunicorn django_apprunner.wsgi --log-file -
  network:
    port: 8000
    env: MY_APP_PORT
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
  secrets:
    - name: my-secret
      value-from: "arn:aws:secretsmanager:us-
east-1:123456789012:secret:testingstackAppRunnerConstr-kJFXde2ULKbT-S7t8xR:username::"
    - name: my-parameter
      value-from: "arn:aws:ssm:us-east-1:123456789012:parameter/parameter-name"
    - name: my-parameter-only-name
      value-from: "parameter-name"
```

확장 Python 구성 파일 - Python 3.11(개정된 빌드 사용)

이 예제는에서 Python 3.11 관리형 런타임과 함께 모든 구성 키를 사용하는 방법을 보여줍니다`apprunner.yaml`. 이 Python 버전은 수정된 App Runner 빌드를 사용하기 때문에이 예제에는 `pre-run` 섹션이 포함되어 있습니다.

`pre-run` 파라미터는 수정된 App Runner 빌드에서만 지원됩니다. 애플리케이션이 원래 App Runner 빌드에서 지원하는 런타임 버전을 사용하는 경우 구성 파일에이 파라미터를 삽입하지 마십시오. 자세한 내용은 관리형 런타임 버전 및 App Runner 빌드 단원을 참조하십시오.

> **Note**
>
> 이 예제에서 사용되는 런타임 버전은 *3.11*입니다. 사용하려는 버전으로 바꿀 수 있습니다. 지원되는 최신 Python 런타임 버전은 섹션을 참조하세요the section called “릴리스 정보”.

Example apprunner.yaml

```
version: 1.0
runtime: python311
build:
  commands:
    pre-build:
      - wget -c https://s3.amazonaws.com/amzn-s3-demo-bucket/test-lib.tar.gz -O - | tar
 -xz
    build:
      - pip3 install pipenv
      - pipenv install
    post-build:
      - python3 manage.py test
  env:
    - name: DJANGO_SETTINGS_MODULE
      value: "django_apprunner.settings"
    - name: MY_VAR_EXAMPLE
      value: "example"
run:
  runtime-version: 3.11
  pre-run:
    - pip3 install pipenv
    - pipenv install
    - python3 copy-global-files.py
  command: pipenv run gunicorn django_apprunner.wsgi --log-file -
  network:
    port: 8000
    env: MY_APP_PORT
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
  secrets:
    - name: my-secret
      value-from: "arn:aws:secretsmanager:us-
east-1:123456789012:secret:testingstackAppRunnerConstr-kJFXde2ULKbT-S7t8xR:username::"
    - name: my-parameter
      value-from: "arn:aws:ssm:us-east-1:123456789012:parameter/parameter-name"
    - name: my-parameter-only-name
      value-from: "parameter-name"
```

전체 Python 애플리케이션 소스

이 예제는 Python 런타임 서비스에 배포할 수 있는 전체 Python 애플리케이션의 소스 코드를 보여줍니다.

Example requirements.txt

```
pyramid==2.0
```

Example server.py

```
from wsgiref.simple_server import make_server
from pyramid.config import Configurator
from pyramid.response import Response
import os

def hello_world(request):
    name = os.environ.get('NAME')
    if name == None or len(name) == 0:
        name = "world"
    message = "Hello, " + name + "!\n"
    return Response(message)

if __name__ == '__main__':
    port = int(os.environ.get("PORT"))
    with Configurator() as config:
        config.add_route('hello', '/')
        config.add_view(hello_world, route_name='hello')
        app = config.make_wsgi_app()
    server = make_server('0.0.0.0', port, app)
    server.serve_forever()
```

Example apprunner.yaml

```
version: 1.0
runtime: python3
build:
  commands:
    build:
      - pip install -r requirements.txt
run:
  command: python server.py
```

Python 런타임 릴리스 정보

> **Important**
>
> App Runner는 2025년 12월 1일에 Python 3.7 및 Python 3.8에 대한 지원을 종료합니다. 권장 사항 및 자세한 내용은 섹션을 참조하세요the section called “관리형 런타임 버전에 대한 지원 종료”.

이 주제에서는 App Runner가 지원하는 Python 런타임 버전에 대한 전체 세부 정보를 나열합니다.

지원되는 런타임 버전 - 수정된 App Runner 빌드

실행 시간 이름	마이너 버전	포함된 패키지
Python 3.11(python311)	3.11.14	SQLite 3.50.2
	3.11.13	SQLite 3.50.2
	3.11.13	SQLite 3.50.1
	3.11.12	SQLite 3.50.0
	3.11.11	SQLite 3.49.1
	3.11.10	SQLite 3.46.1
	3.11.9	SQLite 3.46.1
	3.11.8	SQLite 3.45.2
	3.11.7	SQLite 3.44.2

> **참고**
>
> - Python 3.11 - Python 3.11 관리형 런타임을 사용하는 서비스의 빌드 구성에 대한 특정 권장 사항이 있습니다. 자세한 내용은 Python 플랫폼 주제특정 런타임 버전에 대한 콜아웃의 섹션을 참조하세요.

- App Runner는 최근에 릴리스된 특정 주요 런타임에 대해 수정된 빌드 프로세스를 제공합니다. 따라서이 문서의 특정 섹션에 수정된 App Runner 빌드 및 원래 App Runner 빌드에 대한 참조가 표시됩니다. 자세한 내용은 단원을 참조하십시오관리형 런타임 버전 및 App Runner 빌드.

지원되는 런타임 버전 - 원래 App Runner 빌드

실행 시간 이름	마이너 버전	포함된 패키지
Python 3(python3)	3.8.20	SQLite 3.50.2
	3.8.20	SQLite 3.50.1
	3.8.20	SQLite 3.50.0
	3.8.16	SQLite 3.46.1
	3.8.15	SQLite 3.40.0
	3.7.16	SQLite 3.50.2
	3.7.16	SQLite 3.50.0
	3.7.15	SQLite 3.40.0
	3.7.10	SQLite 3.40.0

Note

App Runner는 최근에 릴리스된 특정 주요 런타임에 대해 수정된 빌드 프로세스를 제공합니다. 따라서이 문서의 특정 섹션에 수정된 App Runner 빌드 및 원래 App Runner 빌드에 대한 참조가 표시됩니다. 자세한 내용은 단원을 참조하십시오관리형 런타임 버전 및 App Runner 빌드.

Node.js 플랫폼 사용

> **Important**
>
> App Runner는 2025년 12월 1일에 Node.js 12, Node.js 14, Node.js 16 및 Node.js 18에 대한 지원을 종료합니다. 권장 사항 및 자세한 내용은 섹션을 참조하세요the section called “관리형 런타임 버전에 대한 지원 종료”.

AWS App Runner Node.js 플랫폼은 관리형 런타임을 제공합니다. 각 런타임을 사용하면 Node.js 버전을 기반으로 웹 애플리케이션을 사용하여 컨테이너를 쉽게 빌드하고 실행할 수 있습니다. Node.js 런타임을 사용하는 경우 App Runner는 관리형 Node.js 런타임 이미지로 시작합니다. 이 이미지는 Amazon Linux Docker 이미지를 기반으로 하며 Node.js 버전 및 일부 도구에 대한 런타임 패키지를 포함합니다. App Runner는이 관리형 런타임 이미지를 기본 이미지로 사용하고 애플리케이션 코드를 추가하여 도커 이미지를 빌드합니다. 그런 다음이 이미지를 배포하여 컨테이너에서 웹 서비스를 실행합니다.

App Runner 콘솔 또는 CreateService API 작업을 사용하여 서비스를 생성할 때 App Runner 서비스에 대한 런타임을 지정합니다. 런타임을 소스 코드의 일부로 지정할 수도 있습니다. 코드 리포지토리에 포함하는 App Runner 구성 파일의 `runtime` 키워드를 사용합니다. 관리형 런타임의 이름 지정 규칙은 *`<language-name><major-version>`*입니다.

유효한 Node.js 런타임 이름 및 버전은 섹션을 참조하세요the section called “릴리스 정보”.

App Runner는 모든 배포 또는 서비스 업데이트 시 서비스의 런타임을 최신 버전으로 업데이트합니다. 애플리케이션에 특정 버전의 관리형 런타임이 필요한 경우 App Runner 구성 파일의 `runtime-version` 키워드를 사용하여 지정할 수 있습니다. 메이저 또는 마이너 버전을 포함하여 모든 수준의 버전으로를 잠글 수 있습니다. App Runner는 서비스의 런타임만 하위 수준으로 업데이트합니다.

Node.js 런타임의 버전 구문: *`major`*`[.`*`minor`*`[.`*`patch`*`]]`

예: `22.14.0`

다음 예제에서는 버전 잠금을 보여줍니다.

- `22.14` - 메이저 및 마이너 버전을 잠급니다. App Runner는 패치 버전만 업데이트합니다.
- `22.14.0` - 특정 패치 버전으로 잠급니다. App Runner는 런타임 버전을 업데이트하지 않습니다.

주제

- Node.js 런타임 구성

- 특정 런타임 버전에 대한 호출
- Node.js 런타임 예제
- Node.js 런타임 릴리스 정보

Node.js 런타임 구성

관리형 런타임을 선택할 때는 최소한 빌드 및 실행 명령도 구성해야 합니다. App Runner 서비스를 생성하거나 업데이트하는 동안 이를 구성합니다. 다음 방법 중 하나를 사용하여이 작업을 수행할 수 있습니다.

- App Runner 콘솔 사용 - 생성 프로세스 또는 구성 탭의 빌드 구성 섹션에서 명령을 지정합니다.
- App Runner API 사용 - CreateService 또는 UpdateService API 작업을 호출합니다. CodeConfigurationValues 데이터 형식의 `BuildCommand` 및 `StartCommand` 멤버를 사용하여 명령을 지정합니다.
- 구성 파일 사용 - 최대 3개의 빌드 단계에서 하나 이상의 빌드 명령을 지정하고 애플리케이션을 시작하는 데 사용되는 단일 실행 명령을 지정합니다. 추가 선택적 구성 설정이 있습니다.

구성 파일을 제공하는 것은 선택 사항입니다. 콘솔 또는 API를 사용하여 App Runner 서비스를 생성할 때 App Runner가 구성 설정을 생성할 때 직접 가져올지 또는 구성 파일에서 가져올지 지정합니다.

특히 Node.js 런타임을 사용하면 소스 리포지토리의 루트`package.json`에 이름이 인 JSON 파일을 사용하여 빌드 및 런타임을 구성할 수도 있습니다. 이 파일을 사용하여 Node.js 엔진 버전, 종속성 패키지 및 다양한 명령(명령줄 애플리케이션)을 구성할 수 있습니다. npm 또는 yarn와 같은 패키지 관리자는이 파일을 명령의 입력으로 해석합니다.

예제:

- npm install는의 `dependencies` 및 `devDependencies` 노드에서 정의한 패키지를 설치합니다package.json.
- npm start 또는는의 `scripts/start` 노드에서 정의한 명령을 npm run start 실행합니다package.json.

다음은 예 `package.json` 파일입니다.

package.json

```
{
```

```
  "name": "node-js-getting-started",
  "version": "0.3.0",
  "description": "A sample Node.js app using Express 4",
  "engines": {
    "node": "22.14.0"
  },
  "scripts": {
    "start": "node index.js",
    "test": "node test.js"
  },
  "dependencies": {
    "cool-ascii-faces": "^1.3.4",
    "ejs": "^2.5.6",
    "express": "^4.15.2"
  },
  "devDependencies": {
    "got": "^11.3.0",
    "tape": "^4.7.0"
  }
}
```

에 대한 자세한 내용은 npm Docs 웹 사이트에서 package.json 파일 생성을 `package.json`참조하세요.

팁

- `package.json` 파일이 start 명령을 정의하는 경우 다음 예제와 같이 App Runner 구성 파일에서 run 명령으로 사용할 수 있습니다.

  Example

  package.json

  ```
  {
    "scripts": {
      "start": "node index.js"
    }
  }
  ```

  apprunner.yaml

```
run:
  command: npm start
```

- 개발 환경에서 npm install를 실행하면 npm이 파일을 생성합니다`package-lock.json`. 이 파일에는 방금 설치된 패키지 버전 npm의 스냅샷이 포함되어 있습니다. 그런 다음 npm이 종속성을 설치할 때 이러한 정확한 버전을 사용합니다. 얀을 설치하면 `yarn.lock` 파일이 생성됩니다. 이러한 파일을 소스 코드 리포지토리에 커밋하여 애플리케이션이 개발 및 테스트한 종속성 버전으로 설치되도록 합니다.
- App Runner 구성 파일을 사용하여 Node.js 버전 및 시작 명령을 구성할 수도 있습니다. 이렇게 하면 이러한 정의가의 정의보다 우선합니다`package.json`. 의 node 버전`package.json`과 App Runner 구성 파일의 `runtime-version` 값이 충돌하면 App Runner 빌드 단계가 실패합니다.

특정 런타임 버전에 대한 호출

Node.js 22 및 Node.js 18(개정된 App Runner 빌드)

App Runner는 이제 Python 3.11, Node.js 22 및 Node.js 18 런타임 버전을 기반으로 애플리케이션에 대해 업데이트된 빌드 프로세스를 실행합니다. 애플리케이션이 이러한 런타임 버전 중 하나에서 실행되는 경우 수정된 빌드 프로세스에 대한 자세한 내용은 관리형 런타임 버전 및 App Runner 빌드 섹션을 참조하세요. 다른 모든 런타임 버전을 사용하는 애플리케이션은 영향을 받지 않으며 원래 빌드 프로세스를 계속 사용합니다.

Node.js 런타임 예제

다음 예제에서는 Node.js 서비스를 빌드하고 실행하기 위한 App Runner 구성 파일을 보여줍니다.

> **Note**
>
> 이 예제에서 사용되는 런타임 버전은 *`22.14.0`*입니다. 이를 사용하려는 버전으로 바꿀 수 있습니다. 지원되는 최신 Node.js 런타임 버전은 섹션을 참조하세요the section called “릴리스 정보”.

미니멀 Node.js 구성 파일

이 예제는 Node.js 관리형 런타임과 함께 사용할 수 있는 최소 구성 파일을 보여줍니다. App Runner가 최소 구성 파일로 가정하는 내용은 섹션을 참조하세요the section called “구성 파일 예제”.

Example apprunner.yaml

```
version: 1.0
runtime: nodejs22
build:
  commands:
    build:
      - npm install --production
run:
  command: node app.js
```

확장 Node.js 구성 파일

이 예제에서는 Node.js 관리형 런타임에서 모든 구성 키를 사용하는 방법을 보여줍니다.

> **Note**
>
> 이 예제에서 사용되는 런타임 버전은 *22.14.0*입니다. 이를 사용하려는 버전으로 바꿀 수 있습니다. 지원되는 최신 Node.js 런타임 버전은 섹션을 참조하세요the section called “릴리스 정보”.

Example apprunner.yaml

```
version: 1.0
runtime: nodejs22
build:
  commands:
    pre-build:
      - npm install --only=dev
      - node test.js
    build:
      - npm install --production
    post-build:
      - node node_modules/ejs/postinstall.js
  env:
    - name: MY_VAR_EXAMPLE
```

```
      value: "example"
run:
  runtime-version: 22.14.0
  command: node app.js
  network:
    port: 8000
    env: APP_PORT
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
```

확장 Node.js 구성 파일 - Node.js 22(개정된 빌드 사용)

이 예제는에서 Node.js 관리형 런타임과 함께 모든 구성 키를 사용하는 방법을 보여줍니다apprunner.yaml. 이 Node.js 버전은 수정된 App Runner 빌드를 사용하기 때문에이 예제에는 pre-run 섹션이 포함되어 있습니다.

pre-run 파라미터는 수정된 App Runner 빌드에서만 지원됩니다. 애플리케이션이 원래 App Runner 빌드에서 지원하는 런타임 버전을 사용하는 경우 구성 파일에이 파라미터를 삽입하지 마십시오. 자세한 내용은 관리형 런타임 버전 및 App Runner 빌드 단원을 참조하십시오.

> **Note**
>
> 이 예제에서 사용되는 런타임 버전은 *22.14.0*입니다. 이를 사용하려는 버전으로 바꿀 수 있습니다. 지원되는 최신 Node.js 런타임 버전은 섹션을 참조하세요the section called “릴리스 정보”.

Example apprunner.yaml

```
version: 1.0
runtime: nodejs22
build:
  commands:
    pre-build:
      - npm install --only=dev
      - node test.js
    build:
      - npm install --production
    post-build:
      - node node_modules/ejs/postinstall.js
  env:
```

```
    - name: MY_VAR_EXAMPLE
      value: "example"
run:
  runtime-version: 22.14.0
  pre-run:
    - node copy-global-files.js
  command: node app.js
  network:
    port: 8000
    env: APP_PORT
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
```

Grunt가 있는 Node.js 앱

이 예제에서는 Grunt로 개발된 Node.js 애플리케이션을 구성하는 방법을 보여줍니다. Grunt는 명령 줄 JavaScript 작업 실행기입니다. 반복 작업을 실행하고 프로세스 자동화를 관리하여 인적 오류를 줄입니다. Grunt 및 Grunt 플러그인은 npm을 사용하여 설치 및 관리됩니다. 소스 리포지토리의 루트에 `Gruntfile.js` 파일을 포함하여 Grunt를 구성합니다.

Example package.json

```
{
  "scripts": {
    "build": "grunt uglify",
    "start": "node app.js"
  },
  "devDependencies": {
    "grunt": "~0.4.5",
    "grunt-contrib-jshint": "~0.10.0",
    "grunt-contrib-nodeunit": "~0.4.1",
    "grunt-contrib-uglify": "~0.5.0"
  },
  "dependencies": {
    "express": "^4.15.2"
  },
}
```

Example Gruntfile.js

```
module.exports = function(grunt) {
```

```
  // Project configuration.
  grunt.initConfig({
    pkg: grunt.file.readJSON('package.json'),
    uglify: {
      options: {
        banner: '/*! <%= pkg.name %> <%= grunt.template.today("yyyy-mm-dd") %> */\n'
      },
      build: {
        src: 'src/<%= pkg.name %>.js',
        dest: 'build/<%= pkg.name %>.min.js'
      }
    }
  });

  // Load the plugin that provides the "uglify" task.
  grunt.loadNpmTasks('grunt-contrib-uglify');

  // Default task(s).
  grunt.registerTask('default', ['uglify']);

};
```

Example apprunner.yaml

> **Note**
>
> 이 예제에서 사용되는 런타임 버전은 *22.14.0*입니다. 이를 사용하려는 버전으로 바꿀 수 있습니다. 지원되는 최신 Node.js 런타임 버전은 섹션을 참조하세요the section called “릴리스 정보”.

```
version: 1.0
runtime: nodejs22
build:
  commands:
    pre-build:
      - npm install grunt grunt-cli
      - npm install --only=dev
      - npm run build
    build:
      - npm install --production
```

```
run:
  runtime-version: 22.14.0
  command: node app.js
  network:
    port: 8000
    env: APP_PORT
```

Node.js 런타임 릴리스 정보

> **Important**
>
> App Runner는 2025년 12월 1일에 Node.js 12, Node.js 14, Node.js 16 및 Node.js 18에 대한 지원을 종료합니다. 권장 사항 및 자세한 내용은 섹션을 참조하세요the section called “관리형 런타임 버전에 대한 지원 종료”.

> **Note**
>
> App Runner의 표준 사용 중단 정책은 런타임의 주요 구성 요소가 커뮤니티 장기 지원(LTS) 종료에 도달하고 보안 업데이트를 더 이상 사용할 수 없는 경우 런타임을 사용 중지하는 것입니다. 경우에 따라 App Runner는 런타임에서 지원하는 언어 버전의 end-of-support 이후 제한된 기간 동안 런타임 사용 중단을 지연시킬 수 있습니다. 이러한 사례의 예로는 고객이 마이그레이션 시간을 확보할 수 있도록 런타임에 대한 지원을 확장하는 것이 있습니다.

이 주제에서는 App Runner가 지원하는 Node.js 런타임 버전의 전체 세부 정보를 나열합니다.

지원되는 런타임 버전 - 수정된 App Runner 빌드

실행 시간 이름	마이너 버전	포함된 패키지
Node.js 22(nodejs22)	22.21.1	npm 10.9.4, 얀 1.22.22
	22.20.0	npm 10.9.3, 얀 1.22.22
	22.17.0	npm 10.9.2, 얀 1.22.22
	22.16.0	npm 10.9.2, 얀 1.22.22
	22.14.0	npm 10.9.2, 얀 1.22.22

> **Note**
>
> App Runner는 최근에 릴리스된 특정 주요 런타임에 대해 수정된 빌드 프로세스를 제공합니다. 따라서이 문서의 특정 섹션에 수정된 App Runner 빌드 및 원래 App Runner 빌드에 대한 참조가 표시됩니다. 자세한 내용은 단원을 참조하십시오관리형 런타임 버전 및 App Runner 빌드.

지원되는 런타임 버전 - 수정된 App Runner 빌드

실행 시간 이름	마이너 버전	포함된 패키지
Node.js 18(nodejs18)	18.20.8	npm 10.8.2, 얀 1.22.22
	18.20.7	npm 10.8.2, 얀 1.22.22
	18.20.6	npm 10.8.2, 얀 1.22.22
	18.20.5	npm 10.8.2, 얀 1.22.22
	18.20.4	npm 10.7.0, 얀 1.22.22
	18.20.3	npm 10.7.0, 얀 1.22.22
	18.20.2	npm 10, 얀*
	18.19.1	npm 10, 얀*
	18.19.0	npm 10, 얀*

지원되는 런타임 버전 - 원래 App Runner 빌드

실행 시간 이름	마이너 버전	포함된 패키지
Node.js 16(nodejs16)	16.20.2	npm 8.19.4, 얀 1.22.22
	16.20.1	npm 8.19.4, 얀*
	16.20.0	npm 8.19.4, 얀*
	16.19.1	npm 8.19.4, 얀*

실행 시간 이름	마이너 버전	포함된 패키지
	16.19.0	npm 8.19.4, 얀*
	16.18.1	npm 8.19.4, 얀*
	16.17.1	npm 8.19.4, 얀*
	16.17.0	npm 8.19.4, 얀*
Node.js 14(nodejs14)	14.21.3	npm 6.14.18, 얀 1.22.22
	14.21.2	npm 6.14.18, 얀*
	14.21.1	npm 6.14.18, 얀*
	14.20.1	npm 6.14.18, 얀*
	14.19.0	npm 6.14.18, 얀*
Node.js 12(nodejs12)	12.22.12	npm 6.14.16, 얀 1.22.22
	12.21.0	npm 6.14.16, 얀*

Java 플랫폼 사용

AWS App Runner Java 플랫폼은 관리형 런타임을 제공합니다. 각 런타임을 사용하면 Java 버전을 기반으로 웹 애플리케이션을 사용하여 컨테이너를 쉽게 빌드하고 실행할 수 있습니다. Java 런타임을 사용하는 경우 App Runner는 관리형 Java 런타임 이미지로 시작합니다. 이 이미지는 Amazon Linux Docker 이미지를 기반으로 하며 Java 버전 및 일부 도구에 대한 런타임 패키지를 포함합니다. App Runner는이 관리형 런타임 이미지를 기본 이미지로 사용하고 애플리케이션 코드를 추가하여 도커 이미지를 빌드합니다. 그런 다음이 이미지를 배포하여 컨테이너에서 웹 서비스를 실행합니다.

App Runner 콘솔 또는 CreateService API 작업을 사용하여 서비스를 생성할 때 App Runner 서비스에 대한 런타임을 지정합니다. 런타임을 소스 코드의 일부로 지정할 수도 있습니다. 코드 리포지토리에 포함하는 App Runner 구성 파일의 `runtime` 키워드를 사용합니다. 관리형 런타임의 이름 지정 규칙은 *`<language-name><major-version>`*입니다.

현재 지원되는 모든 Java 런타임은 Amazon Corretto를 기반으로 합니다. 유효한 Java 런타임 이름 및 버전은 섹션을 참조하세요the section called “릴리스 정보”.

App Runner는 모든 배포 또는 서비스 업데이트 시 서비스의 런타임을 최신 버전으로 업데이트합니다. 애플리케이션에 특정 버전의 관리형 런타임이 필요한 경우 App Runner 구성 파일의 `runtime-version` 키워드를 사용하여 지정할 수 있습니다. 메이저 또는 마이너 버전을 포함하여 모든 수준의 버전으로를 잠글 수 있습니다. App Runner는 서비스의 런타임만 하위 수준으로 업데이트합니다.

Amazon Corretto 런타임의 버전 구문:

런타임	구문	예제
corretto11	`11.0[.`*`openjdk-update`* `[.`*`openjdk-build`* `[.`*`corretto-specific-revision`* `]]]`	`11.0.13.08.1`
corretto8	`8[.`*`openjdk-update`* `[.`*`openjdk-build`* `[.`*`corretto-specific-revision`* `]]]`	`8.312.07.1`

다음 예제에서는 버전 잠금을 보여줍니다.

- `11.0.13` - Open JDK 업데이트 버전을 잠급니다. App Runner는 Open JDK 및 Amazon Corretto 하위 수준 빌드만 업데이트합니다.
- `11.0.13.08.1` - 특정 버전으로 잠급니다. App Runner는 런타임 버전을 업데이트하지 않습니다.

주제

- Java 런타임 구성
- Java 런타임 예제
- Java 런타임 릴리스 정보

Java 런타임 구성

관리형 런타임을 선택할 때는 최소한 빌드 및 실행 명령도 구성해야 합니다. App Runner 서비스를 생성하거나 업데이트하는 동안 이를 구성합니다. 다음 방법 중 하나를 사용하여이 작업을 수행할 수 있습니다.

- App Runner 콘솔 사용 - 생성 프로세스 또는 구성 탭의 빌드 구성 섹션에서 명령을 지정합니다.

- App Runner API 사용 - CreateService 또는 UpdateService API 작업을 호출합니다. CodeConfigurationValues 데이터 형식의 `BuildCommand` 및 `StartCommand` 멤버를 사용하여 명령을 지정합니다.
- 구성 파일 사용 - 최대 3개의 빌드 단계에서 하나 이상의 빌드 명령을 지정하고 애플리케이션을 시작하는 데 사용되는 단일 실행 명령을 지정합니다. 추가 선택적 구성 설정이 있습니다.

구성 파일을 제공하는 것은 선택 사항입니다. 콘솔 또는 API를 사용하여 App Runner 서비스를 생성할 때 App Runner가 구성 설정을 생성할 때 직접 가져오거나 구성 파일에서 가져올지 지정합니다.

Java 런타임 예제

다음 예제에서는 Java 서비스를 빌드하고 실행하기 위한 App Runner 구성 파일을 보여줍니다. 마지막 예제는 Corretto 11 런타임 서비스에 배포할 수 있는 전체 Java 애플리케이션의 소스 코드입니다.

> **Note**
>
> 이 예제에서 사용되는 런타임 버전은 *`11.0.13.08.1`*입니다. 사용하려는 버전으로 바꿀 수 있습니다. 지원되는 최신 Java 런타임 버전은 섹션을 참조하세요the section called “릴리스 정보”.

최소 Corretto 11 구성 파일

이 예제는 Corretto 11 관리형 런타임과 함께 사용할 수 있는 최소 구성 파일을 보여줍니다. App Runner가 최소 구성 파일로 가정하는 내용은 섹션을 참조하세요.

Example apprunner.yaml

```
version: 1.0
runtime: corretto11
build:
  commands:
    build:
      - mvn clean package
run:
  command: java -Xms256m -jar target/MyApp-1.0-SNAPSHOT.jar .
```

확장 Corretto 11 구성 파일

이 예제에서는 Corretto 11 관리형 런타임에서 모든 구성 키를 사용하는 방법을 보여줍니다.

> **Note**
>
> 이 예제에서 사용되는 런타임 버전은 *11.0.13.08.1*입니다. 사용하려는 버전으로 바꿀 수 있습니다. 지원되는 최신 Java 런타임 버전은 섹션을 참조하세요the section called “릴리스 정보”.

Example apprunner.yaml

```
version: 1.0
runtime: corretto11
build:
  commands:
    pre-build:
      - yum install some-package
      - scripts/prebuild.sh
    build:
      - mvn clean package
    post-build:
      - mvn clean test
  env:
    - name: M2
      value: "/usr/local/apache-maven/bin"
    - name: M2_HOME
      value: "/usr/local/apache-maven/bin"
run:
  runtime-version: 11.0.13.08.1
  command: java -Xms256m -jar target/MyApp-1.0-SNAPSHOT.jar .
  network:
    port: 8000
    env: APP_PORT
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
```

Corretto 11 애플리케이션 소스 완료

이 예제는 Corretto 11 런타임 서비스에 배포할 수 있는 전체 Java 애플리케이션의 소스 코드를 보여줍니다.

Example src/main/java/com/HelloWorld/HelloWorld.java

```
package com.HelloWorld;

import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HelloWorld {

    @RequestMapping("/")
    public String index(){
        String s = "Hello World";
        return s;
    }
}
```

Example src/main/java/com/HelloWorld/Main.java

```
package com.HelloWorld;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Main {

    public static void main(String[] args) {

        SpringApplication.run(Main.class, args);
    }
}
```

Example apprunner.yaml

```
version: 1.0
runtime: corretto11
build:
  commands:
    build:
      - mvn clean package
run:
```

```
  command: java -Xms256m -jar target/HelloWorldJavaApp-1.0-SNAPSHOT.jar .
  network:
    port: 8080
```

Example pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/
maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.3.1.RELEASE</version>
    <relativePath/>
  </parent>
  <groupId>com.HelloWorld</groupId>
  <artifactId>HelloWorldJavaApp</artifactId>
  <version>1.0-SNAPSHOT</version>

  <properties>
    <java.version>11</java.version>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-rest</artifactId>
    </dependency>

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
      <exclusions>
        <exclusion>
          <groupId>org.junit.vintage</groupId>
          <artifactId>junit-vintage-engine</artifactId>
        </exclusion>
      </exclusions>
    </dependency>
```

```
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.8.0</version>
        <configuration>
          <release>11</release>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```

Java 런타임 릴리스 정보

이 주제에서는 App Runner가 지원하는 Java 런타임 버전에 대한 전체 세부 정보를 나열합니다.

지원되는 런타임 버전 - 원래 App Runner 빌드

실행 시간 이름	마이너 버전	포함된 패키지
Corretto 11(corretto11)	11.0.28.6.1	Maven 3.9.11, Gradle 6.9.4
	11.0.27.6.1	Maven 3.9.10, Gradle 6.9.4
	11.0.27.6.1	Maven 3.9.9, Gradle 6.9.4
	11.0.26.4.1	Maven 3.9.9, Gradle 6.9.4
	11.0.25.9.1	Maven 3.9.9, Gradle 6.9.4
	11.0.24.8.1	Maven 3.9.9, Gradle 6.9.4
	11.0.23.9.1	Maven 3.9.8, Gradle 6.9.4
	11.0.22.7.1	Maven 3.9.6, Gradle 6.9.4

실행 시간 이름	마이너 버전	포함된 패키지
	11.0.21.9.1	Maven 3.9.6, Gradle 6.9.4
	11.0.21.9.1	Maven 3.9.5, Gradle 6.9.4
	11.0.20.8.1	Maven 3.9.3, Gradle 6.9.4
	11.0.19.7.1	Maven 3.9.3, Gradle 6.9.4
	11.0.18.10.1	Maven 3.9.1, Gradle 6.9.4
	11.0.17.8.1	Maven 3.8.6, Gradle 6.9.3
	11.0.16.9.1	Maven 3.8.6, Gradle 6.9.2
	11.0.13.08.1	Maven 3.6.3, Gradle 6.5
Corretto 8(corretto8)	8.472.08.1	Maven 3.9.11, Gradle 6.9.4
	8.462.08.1	Maven 3.9.11, Gradle 6.9.4
	8.452.09.2	Maven 3.9.10, Gradle 6.9.4
	8.452.09.2	Maven 3.9.9, Gradle 6.9.4
	8.452.09.1	Maven 3.9.9, Gradle 6.9.4
	8.442.06.1	Maven 3.9.9, Gradle 6.9.4
	8.432.06.1	Maven 3.9.9, Gradle 6.9.4
	8.422.05.1	Maven 3.9.9, Gradle 6.9.4
	8.412.08.1	Maven 3.9.8, Gradle 6.9.4
	8.402.08.1	Maven 3.9.6, Gradle 6.9.4
	8.392.08.1	Maven 3.9.6, Gradle 6.9.4
	8.382.05.1	Maven 3.9.4, Gradle 6.9.4

실행 시간 이름	마이너 버전	포함된 패키지
	8.372.07.1	Maven 3.9.3, Gradle 6.9.4
	8.362.08.1	Maven 3.9.1, Gradle 6.9.4
	8.352.08.1	Maven 3.8.6, Gradle 6.9.3
	8.342.07.4	Maven 3.8.6, Gradle 6.9.2
	8.312.07.1	Maven 3.6.3, Gradle 6.5

> **Note**
>
> App Runner는 최근에 릴리스된 특정 주요 런타임에 대해 수정된 빌드 프로세스를 제공합니다. 따라서이 문서의 특정 섹션에 수정된 App Runner 빌드 및 원래 App Runner 빌드에 대한 참조가 표시됩니다. 자세한 내용은 단원을 참조하십시오관리형 런타임 버전 및 App Runner 빌드.

.NET 플랫폼 사용

> **Important**
>
> App Runner는 2025년 12월 1일에 .NET 6에 대한 지원을 종료합니다. 권장 사항 및 자세한 내용은 섹션을 참조하세요the section called “관리형 런타임 버전에 대한 지원 종료”.

AWS App Runner .NET 플랫폼은 관리형 런타임을 제공합니다. 각 런타임을 사용하면 .NET 버전을 기반으로 웹 애플리케이션을 사용하여 컨테이너를 쉽게 빌드하고 실행할 수 있습니다. .NET 런타임을 사용하는 경우 App Runner는 관리형 .NET 런타임 이미지로 시작합니다. 이 이미지는 Amazon Linux Docker 이미지를 기반으로 하며 .NET 버전용 런타임 패키지와 일부 도구 및 인기 있는 종속성 패키지를 포함합니다. App Runner는이 관리형 런타임 이미지를 기본 이미지로 사용하고 애플리케이션 코드를 추가하여 도커 이미지를 빌드합니다. 그런 다음이 이미지를 배포하여 컨테이너에서 웹 서비스를 실행합니다.

App Runner 콘솔 또는 CreateService API 작업을 사용하여 서비스를 생성할 때 App Runner 서비스에 대한 런타임을 지정합니다. 런타임을 소스 코드의 일부로 지정할 수도 있습니다. 코드 리포지토리에 포

함하는 App Runner 구성 파일의 `runtime` 키워드를 사용합니다. 관리형 런타임의 이름 지정 규칙은 *`<language-name><major-version>`*입니다.

유효한 .NET 런타임 이름 및 버전은 섹션을 참조하세요the section called “릴리스 정보”.

App Runner는 모든 배포 또는 서비스 업데이트 시 서비스의 런타임을 최신 버전으로 업데이트합니다. 애플리케이션에 특정 버전의 관리형 런타임이 필요한 경우 App Runner 구성 파일의 `runtime-version` 키워드를 사용하여 지정할 수 있습니다. 메이저 또는 마이너 버전을 포함하여 모든 수준의 버전으로를 잠글 수 있습니다. App Runner는 서비스의 런타임만 하위 수준으로 업데이트합니다.

.NET 런타임용 버전 구문: *`major`*`[.`*`minor`*`[.`*`patch`*`]]`

예: `6.0.9`

다음 예제에서는 버전 잠금을 보여줍니다.

- `6.0` - 메이저 및 마이너 버전을 잠급니다. App Runner는 패치 버전만 업데이트합니다.
- `6.0.9` - 특정 패치 버전으로 잠급니다. App Runner는 런타임 버전을 업데이트하지 않습니다.

주제

- .NET 런타임 구성
- .NET 런타임 예제
- .NET 런타임 릴리스 정보

.NET 런타임 구성

관리형 런타임을 선택할 때는 최소한 빌드 및 실행 명령도 구성해야 합니다. App Runner 서비스를 생성하거나 업데이트하는 동안 이를 구성합니다. 다음 방법 중 하나를 사용하여이 작업을 수행할 수 있습니다.

- App Runner 콘솔 사용 - 생성 프로세스 또는 구성 탭의 빌드 구성 섹션에서 명령을 지정합니다.
- App Runner API 사용 - CreateService 또는 UpdateService API 작업을 호출합니다. CodeConfigurationValues 데이터 형식의 `BuildCommand` 및 `StartCommand` 멤버를 사용하여 명령을 지정합니다.
- 구성 파일 사용 - 최대 3개의 빌드 단계에서 하나 이상의 빌드 명령을 지정하고 애플리케이션을 시작하는 데 사용되는 단일 실행 명령을 지정합니다. 추가 선택적 구성 설정이 있습니다.

구성 파일을 제공하는 것은 선택 사항입니다. 콘솔 또는 API를 사용하여 App Runner 서비스를 생성할 때 App Runner가 구성 설정을 생성할 때 직접 가져올지 또는 구성 파일에서 가져올지 지정합니다.

.NET 런타임 예제

다음 예제에서는 .NET 서비스를 빌드하고 실행하기 위한 App Runner 구성 파일을 보여줍니다. 마지막 예제는 .NET 런타임 서비스에 배포할 수 있는 전체 .NET 애플리케이션의 소스 코드입니다.

> **Note**
>
> 이 예제에서 사용되는 런타임 버전은 *6.0.9*입니다. 사용하려는 버전으로 바꿀 수 있습니다. 지원되는 최신 .NET 런타임 버전은 섹션을 참조하세요the section called “릴리스 정보”.

미니멀 .NET 구성 파일

이 예제는 .NET 관리형 런타임과 함께 사용할 수 있는 최소 구성 파일을 보여줍니다. App Runner가 최소 구성 파일로 가정하는 내용은 섹션을 참조하세요the section called “구성 파일 예제”.

Example apprunner.yaml

```
version: 1.0
runtime: dotnet6
build:
  commands:
    build:
      - dotnet publish -c Release -o out
run:
  command: dotnet out/HelloWorldDotNetApp.dll
```

확장 .NET 구성 파일

이 예제에서는 .NET 관리형 런타임에서 모든 구성 키를 사용하는 방법을 보여줍니다.

> **Note**
>
> 이 예제에서 사용되는 런타임 버전은 *6.0.9*입니다. 사용하려는 버전으로 바꿀 수 있습니다. 지원되는 최신 .NET 런타임 버전은 섹션을 참조하세요the section called “릴리스 정보”.

Example apprunner.yaml

```
version: 1.0
runtime: dotnet6
build:
  commands:
    pre-build:
      - scripts/prebuild.sh
    build:
      - dotnet publish -c Release -o out
    post-build:
      - scripts/postbuild.sh
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
run:
  runtime-version: 6.0.9
  command: dotnet out/HelloWorldDotNetApp.dll
  network:
    port: 5000
    env: APP_PORT
  env:
    - name: ASPNETCORE_URLS
      value: "http://*:5000"
```

전체 .NET 애플리케이션 소스

이 예제는 .NET 런타임 서비스에 배포할 수 있는 전체 .NET 애플리케이션의 소스 코드를 보여줍니다.

> **Note**
>
> - 다음 명령을 실행하여 간단한 .NET 6 웹 앱을 생성합니다. `dotnet new web --name HelloWorldDotNetApp -f net6.0`
> - 생성된 .NET 6 웹 앱`apprunner.yaml`에를 추가합니다.

Example HelloWorldDotNetApp

```
version: 1.0
runtime: dotnet6
build:
```

```
  commands:
    build:
      - dotnet publish -c Release -o out
run:
  command: dotnet out/HelloWorldDotNetApp.dll
  network:
    port: 5000
    env: APP_PORT
  env:
    - name: ASPNETCORE_URLS
      value: "http://*:5000"
```

.NET 런타임 릴리스 정보

> **Important**
>
> App Runner는 2025년 12월 1일에 .NET 6에 대한 지원을 종료합니다. 권장 사항 및 자세한 내용은 섹션을 참조하세요the section called “관리형 런타임 버전에 대한 지원 종료”.

이 주제에서는 App Runner가 지원하는 .NET 런타임 버전에 대한 전체 세부 정보를 나열합니다.

지원되는 런타임 버전 - 원래 App Runner 빌드

실행 시간 이름	마이너 버전	포함된 패키지
.NET 6(dotnet6)	6.0.36	.NET SDK 6.0.428
	6.0.33	.NET SDK 6.0.425
	6.0.32	.NET SDK 6.0.424
	6.0.31	.NET SDK 6.0.423
	6.0.30	.NET SDK 6.0.422
	6.0.29	.NET SDK 6.0.421
	6.0.28	.NET SDK 6.0.420
	6.0.26	.NET SDK 6.0.418

실행 시간 이름	마이너 버전	포함된 패키지
	6.0.25	.NET SDK 6.0.417
	6.0.24	.NET SDK 6.0.416
	6.0.22	.NET SDK 6.0.414
	6.0.21	.NET SDK 6.0.413
	6.0.20	.NET SDK 6.0.412
	6.0.19	.NET SDK 6.0.411
	6.0.16	.NET SDK 6.0.408
	6.0.15	.NET SDK 6.0.407
	6.0.14	.NET SDK 6.0.406
	6.0.13	.NET SDK 6.0.405
	6.0.12	.NET SDK 6.0.404
	6.0.11	.NET SDK 6.0.403
	6.0.10	.NET SDK 6.0.402
	6.0.9	.NET SDK 6.0.401

Note

App Runner는 최근에 릴리스된 특정 주요 런타임에 대해 수정된 빌드 프로세스를 제공합니다. 따라서이 문서의 특정 섹션에 수정된 App Runner 빌드 및 원래 App Runner 빌드에 대한 참조가 표시됩니다. 자세한 내용은 단원을 참조하십시오관리형 런타임 버전 및 App Runner 빌드.

PHP 플랫폼 사용

> **Important**
>
> App Runner는 2025년 12월 31일에 PHP 8.1에 대한 지원을 종료합니다. 권장 사항 및 자세한 내용은 섹션을 참조하세요the section called “관리형 런타임 버전에 대한 지원 종료”.

AWS App Runner PHP 플랫폼은 관리형 런타임을 제공합니다. 각 런타임을 사용하여 PHP 버전을 기반으로 웹 애플리케이션으로 컨테이너를 빌드하고 실행할 수 있습니다. PHP 런타임을 사용하는 경우 App Runner는 관리형 PHP 런타임 이미지로 시작합니다. 이 이미지는 Amazon Linux Docker 이미지를 기반으로 하며 PHP 버전 및 일부 도구에 대한 런타임 패키지를 포함합니다. App Runner는이 관리형 런타임 이미지를 기본 이미지로 사용하고 애플리케이션 코드를 추가하여 도커 이미지를 빌드합니다. 그런 다음이 이미지를 배포하여 컨테이너에서 웹 서비스를 실행합니다.

App Runner 콘솔 또는 CreateService API 작업을 사용하여 서비스를 생성할 때 App Runner 서비스에 대한 런타임을 지정합니다. 런타임을 소스 코드의 일부로 지정할 수도 있습니다. 코드 리포지토리에 포함하는 App Runner 구성 파일의 `runtime` 키워드를 사용합니다. 관리형 런타임의 이름 지정 규칙은 *`<language-name><major-version>`*입니다.

유효한 PHP 런타임 이름 및 버전은 섹션을 참조하세요the section called “릴리스 정보”.

App Runner는 모든 배포 또는 서비스 업데이트 시 서비스의 런타임을 최신 버전으로 업데이트합니다. 애플리케이션에 특정 버전의 관리형 런타임이 필요한 경우 App Runner 구성 파일의 `runtime-version` 키워드를 사용하여 지정할 수 있습니다. 메이저 또는 마이너 버전을 포함하여 모든 수준의 버전으로를 잠글 수 있습니다. App Runner는 서비스의 런타임만 하위 수준으로 업데이트합니다.

PHP 런타임의 버전 구문: *`major`*`[.`*`minor`*`[.`*`patch`*`]]`

예: `8.1.10`

다음은 버전 잠금의 예입니다.

- `8.1` - 메이저 및 마이너 버전을 잠급니다. App Runner는 패치 버전만 업데이트합니다.
- `8.1.10` - 특정 패치 버전으로 잠급니다. App Runner는 런타임 버전을 업데이트하지 않습니다.

> **Important**
>
> 기본 리포지토리 루트 디렉터리 이외의 위치에서 App Runner 서비스의 코드 리포지토리 소스 디렉터리를 지정하려면 PHP 관리형 런타임 버전이 PHP `8.1.22` 이상이어야 합니다. 이전 PHP 런타임 버전은 기본 루트 소스 디렉터리만 사용할 `8.1.22` 수 있습니다.

주제

PHP 런타임 구성

관리형 런타임을 선택할 때는 최소한 빌드 및 실행 명령도 구성해야 합니다. App Runner 서비스를 생성하거나 업데이트하는 동안 이를 구성합니다. 다음 방법 중 하나를 사용하여이 작업을 수행할 수 있습니다.

- App Runner 콘솔 사용 - 생성 프로세스 또는 구성 탭의 빌드 구성 섹션에서 명령을 지정합니다.
- App Runner API 사용 - CreateService 또는 UpdateService API 작업을 호출합니다. CodeConfigurationValues 데이터 형식의 `BuildCommand` 및 `StartCommand` 멤버를 사용하여 명령을 지정합니다.
- 구성 파일 사용 - 최대 3개의 빌드 단계에서 하나 이상의 빌드 명령을 지정하고 애플리케이션을 시작하는 데 사용되는 단일 실행 명령을 지정합니다. 추가 선택적 구성 설정이 있습니다.

구성 파일을 제공하는 것은 선택 사항입니다. 콘솔 또는 API를 사용하여 App Runner 서비스를 생성할 때 App Runner가 구성 설정을 생성할 때 직접 가져올지 또는 구성 파일에서 가져올지 지정합니다.

호환성

다음 웹 서버 중 하나를 사용하여 PHP 플랫폼에서 App Runner 서비스를 실행할 수 있습니다.

- Apache HTTP Server
- NGINX

Apache HTTP Server 및 NGINX는 PHP-FPM과 호환됩니다. 다음 중 하나를 NGINX 사용하여 Apache HTTP Server 및를 시작할 수 있습니다.

- 감독자 - 실행에 대한 자세한 내용은 감독자 실행을 supervisord참조하세요. http://supervisord.org/running.html#running-supervisord
- 시작 스크립트

Apache HTTP Server 또는 NGINX를 사용하여 PHP 플랫폼으로 App Runner 서비스를 구성하는 방법에 대한 예는 섹션을 참조하세요the section called “전체 PHP 애플리케이션 소스”.

파일 구조

는 웹 서버의 `root` 디렉터리 아래에 있는 `public` 폴더에 설치해야 `index.php` 합니다.

> **Note**
>
> `startup.sh` 또는 `supervisord.conf` 파일은 웹 서버의 루트 디렉터리에 저장하는 것이 좋습니다. `start` 명령이 `startup.sh` 또는 `supervisord.conf` 파일이 저장된 위치를 가리키는지 확인합니다.

다음은를 사용하는 경우 파일 구조의 예입니다supervisord.

```
/
## public/
#  ## index.php
## apprunner.yaml
## supervisord.conf
```

다음은 시작 스크립트를 사용하는 경우 파일 구조의 예입니다.

```
/
## public/
#  ## index.php
## apprunner.yaml
## startup.sh
```

App Runner 서비스에 지정된 코드 리포지토리 소스 디렉터리에 이러한 파일 구조를 저장하는 것이 좋습니다.

```
/<sourceDirectory>/
## public/
#  ## index.php
## apprunner.yaml
## startup.sh
```

> **Important**
>
> 기본 리포지토리 루트 디렉터리 이외의 위치에서 App Runner 서비스의 코드 리포지토리 소스 디렉터리를 지정하려면 PHP 관리형 런타임 버전이 PHP 8.1.22 이상이어야 합니다. 이전의 PHP 런타임 버전은 기본 루트 소스 디렉터리만 사용할 8.1.22 수 있습니다.
> App Runner는 모든 배포 또는 서비스 업데이트 시 서비스의 런타임을 최신 버전으로 업데이트합니다. App Runner 구성 파일의 `runtime-version` 키워드를 사용하여 버전 잠금을 지정하지 않는 한 서비스는 기본적으로 최신 런타임을 사용합니다.

PHP 런타임 예제

다음은 PHP 서비스를 빌드하고 실행하는 데 사용되는 App Runner 구성 파일의 예입니다.

최소 PHP 구성 파일

다음 예제는 PHP 관리형 런타임과 함께 사용할 수 있는 최소 구성 파일입니다. 최소 구성 파일에 대한 자세한 내용은 섹션을 참조하세요the section called “구성 파일 예제”.

Example apprunner.yaml

```
version: 1.0
runtime: php81
build:
  commands:
    build:
      - echo example build command for PHP
run:
  command: ./startup.sh
```

확장 PHP 구성 파일

다음 예제에서는 PHP 관리형 런타임과 함께 모든 구성 키를 사용합니다.

> **Note**
>
> 이 예제에서 사용되는 런타임 버전은 *8.1.10*입니다. 이를 사용하려는 버전으로 바꿀 수 있습니다. 지원되는 최신 PHP 런타임 버전은 섹션을 참조하세요the section called “릴리스 정보”.

Example apprunner.yaml

```
version: 1.0
runtime: php81
build:
  commands:
     pre-build:
      - scripts/prebuild.sh
    build:
      - echo example build command for PHP
    post-build:
      - scripts/postbuild.sh
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
run:
  runtime-version: 8.1.10
  command: ./startup.sh
  network:
    port: 5000
    env: APP_PORT
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
```

전체 PHP 애플리케이션 소스

다음 예제는 Apache HTTP Server 또는를 사용하여 PHP 런타임 서비스에 배포하는 데 사용할 수 있는 PHP 애플리케이션 소스 코드입니다NGINX. 이 예제에서는 기본 파일 구조를 사용한다고 가정합니다.

를 Apache HTTP Server 사용하여에서 PHP 플랫폼 실행 supervisord

Example파일 구조

> **Note**
>
> - `supervisord.conf` 파일은 리포지토리의 어디에나 저장할 수 있습니다. `start` 명령이 `supervisord.conf` 파일이 저장되는 위치를 가리키는지 확인합니다.
> - 는 `root` 디렉터리 아래의 `public` 폴더에 설치해야 `index.php` 합니다.

```
/
## public/
#  ## index.php
## apprunner.yaml
## supervisord.conf
```

Example supervisord.conf

```
[supervisord]
nodaemon=true

[program:httpd]
command=httpd -DFOREGROUND
autostart=true
autorestart=true
stdout_logfile=/dev/stdout
stdout_logfile_maxbytes=0
stderr_logfile=/dev/stderr
stderr_logfile_maxbytes=0

[program:php-fpm]
command=php-fpm -F
autostart=true
autorestart=true
stdout_logfile=/dev/stdout
stdout_logfile_maxbytes=0
stderr_logfile=/dev/stderr
stderr_logfile_maxbytes=0
```

Example apprunner.yaml

```
version: 1.0
runtime: php81
build:
  commands:
    build:
      - PYTHON=python2 amazon-linux-extras install epel
      - yum -y install supervisor
run:
  command: supervisord
  network:
    port: 8080
    env: APP_PORT
```

Example index.php

```
<html>
<head> <title>First PHP App</title> </head>
<body>
<?php
    print("Hello World!");
    print("<br>");
?>
</body>
</html>
```

를 Apache HTTP Server 사용하여에서 PHP 플랫폼 실행 startup script

Example파일 구조

> **Note**
>
> - `startup.sh` 파일은 리포지토리의 모든 위치에 저장할 수 있습니다. `start` 명령이 `startup.sh` 파일이 저장되는 위치를 가리키는지 확인합니다.
> - 는 `root` 디렉터리 아래의 `public` 폴더에 설치해야 `index.php` 합니다.

```
/
## public/
```

```
#  ## index.php
## apprunner.yaml
## startup.sh
```

Example startup.sh

```
#!/bin/bash

set -o monitor

trap exit SIGCHLD

# Start apache
httpd -DFOREGROUND &

# Start php-fpm
php-fpm -F &

wait
```

> **Note**
>
> - `startup.sh` 파일을 Git 리포지토리에 커밋하기 전에 파일을 실행 파일로 저장해야 합니다. `chmod +x startup.sh`를 사용하여 `startup.sh` 파일에 대한 실행 권한을 설정합니다.
> - `startup.sh` 파일을 실행 파일로 저장하지 않는 경우를 `apprunner.yaml` 파일에 `build` 명령`chmod +x startup.sh`으로 입력합니다.

Example apprunner.yaml

```
version: 1.0
runtime: php81
build:
  commands:
    build:
      - echo example build command for PHP
run:
  command: ./startup.sh
  network:
    port: 8080
```

```
    env: APP_PORT
```

Example index.php

```
<html>
<head> <title>First PHP App</title> </head>
<body>
<?php
    print("Hello World!");
    print("<br>");
?>
</body>
</html>
```

를 NGINX 사용하여에서 PHP 플랫폼 실행 supervisord

Example파일 구조

> **Note**
>
> - `supervisord.conf` 파일은 리포지토리의 어디에나 저장할 수 있습니다. `start` 명령이 `supervisord.conf` 파일이 저장되는 위치를 가리키는지 확인합니다.
> - 는 `root` 디렉터리 아래의 `public` 폴더에 설치해야 `index.php` 합니다.

```
/
## public/
#  ## index.php
## apprunner.yaml
## supervisord.conf
```

Example supervisord.conf

```
[supervisord]
nodaemon=true

[program:nginx]
command=nginx -g "daemon off;"
autostart=true
```

```
autorestart=true
stdout_logfile=/dev/stdout
stdout_logfile_maxbytes=0
stderr_logfile=/dev/stderr
stderr_logfile_maxbytes=0

[program:php-fpm]
command=php-fpm -F
autostart=true
autorestart=true
stdout_logfile=/dev/stdout
stdout_logfile_maxbytes=0
stderr_logfile=/dev/stderr
stderr_logfile_maxbytes=0
```

Example apprunner.yaml

```
version: 1.0
runtime: php81
build:
  commands:
    build:
      - PYTHON=python2 amazon-linux-extras install epel
      - yum -y install supervisor
run:
  command: supervisord
  network:
    port: 8080
    env: APP_PORT
```

Example index.php

```
<html>
<head> <title>First PHP App</title> </head>
<body>
<?php
    print("Hello World!");
    print("<br>");
?>
</body>
</html>
```

를 NGINX 사용하여에서 PHP 플랫폼 실행 startup script

Example파일 구조

> **Note**
>
> - `startup.sh` 파일은 리포지토리의 모든 위치에 저장할 수 있습니다. `start` 명령이 `startup.sh` 파일이 저장되는 위치를 가리키는지 확인합니다.
> - 는 `root` 디렉터리 아래의 `public` 폴더에 설치해야 `index.php` 합니다.

```
/
## public/
#  ## index.php
## apprunner.yaml
## startup.sh
```

Example startup.sh

```
#!/bin/bash

set -o monitor

trap exit SIGCHLD

# Start nginx
nginx -g 'daemon off;' &

# Start php-fpm
php-fpm -F &

wait
```

> **Note**
>
> - `startup.sh` 파일을 Git 리포지토리에 커밋하기 전에 파일을 실행 파일로 저장해야 합니다. `chmod +x startup.sh`를 사용하여 `startup.sh` 파일에 대한 실행 권한을 설정합니다.

- `startup.sh` 파일을 실행 파일로 저장하지 않는 경우를 `apprunner.yaml` 파일에 `build` 명령`chmod +x startup.sh`으로 입력합니다.

Example apprunner.yaml

```
version: 1.0
runtime: php81
build:
  commands:
    build:
      - echo example build command for PHP
run:
  command: ./startup.sh
  network:
    port: 8080
    env: APP_PORT
```

Example index.php

```
<html>
<head> <title>First PHP App</title> </head>
<body>
<?php
    print("Hello World!");
    print("<br>");
?>
</body>
</html>
```

PHP 런타임 릴리스 정보

Important

App Runner는 2025년 12월 31일에 PHP 8.1에 대한 지원을 종료합니다. 권장 사항 및 자세한 내용은 섹션을 참조하세요the section called “관리형 런타임 버전에 대한 지원 종료”.

이 주제에서는 App Runner가 지원하는 PHP 런타임 버전에 대한 전체 세부 정보를 나열합니다.

지원되는 런타임 버전 - 원래 App Runner 빌드

실행 시간 이름	마이너 버전	포함된 패키지
PHP 8.1(php81)	8.1.33	
	8.1.32	
	8.1.31	
	8.1.29	
	8.1.28	
	8.1.27	
	8.1.26	
	8.1.24	
	8.1.22	
	8.1.21	
	8.1.20	
	8.1.19	
	8.1.17	
	8.1.16	
	8.1.14	
	8.1.13	
	8.1.12	
	8.1.10	

> **Note**
>
> App Runner는 최근에 릴리스된 특정 주요 런타임에 대해 수정된 빌드 프로세스를 제공합니다. 따라서이 문서의 특정 섹션에 수정된 App Runner 빌드 및 원래 App Runner 빌드에 대한 참조가 표시됩니다. 자세한 내용은 단원을 참조하십시오관리형 런타임 버전 및 App Runner 빌드.

Ruby 플랫폼 사용

> **Important**
>
> App Runner는 2025년 12월 1일에 Ruby 3.1에 대한 지원을 종료합니다. 권장 사항 및 자세한 내용은 섹션을 참조하세요the section called “관리형 런타임 버전에 대한 지원 종료”.

AWS App Runner Ruby 플랫폼은 관리형 런타임을 제공합니다. 각 런타임을 사용하면 Ruby 버전을 기반으로 웹 애플리케이션을 사용하여 컨테이너를 쉽게 빌드하고 실행할 수 있습니다. Ruby 런타임을 사용하는 경우 App Runner는 관리형 Ruby 런타임 이미지로 시작합니다. 이 이미지는 Amazon Linux Docker 이미지를 기반으로 하며 Ruby 버전 및 일부 도구에 대한 런타임 패키지를 포함합니다. App Runner는이 관리형 런타임 이미지를 기본 이미지로 사용하고 애플리케이션 코드를 추가하여 도커 이미지를 빌드합니다. 그런 다음이 이미지를 배포하여 컨테이너에서 웹 서비스를 실행합니다.

App Runner 콘솔 또는 CreateService API 작업을 사용하여 서비스를 생성할 때 App Runner 서비스에 대한 런타임을 지정합니다. 런타임을 소스 코드의 일부로 지정할 수도 있습니다. 코드 리포지토리에 포함하는 App Runner 구성 파일의 `runtime` 키워드를 사용합니다. 관리형 런타임의 이름 지정 규칙은 *`<language-name><major-version>`*입니다.

유효한 Ruby 런타임 이름 및 버전은 섹션을 참조하세요the section called “릴리스 정보”.

App Runner는 모든 배포 또는 서비스 업데이트 시 서비스의 런타임을 최신 버전으로 업데이트합니다. 애플리케이션에 특정 버전의 관리형 런타임이 필요한 경우 App Runner 구성 파일의 `runtime-version` 키워드를 사용하여 지정할 수 있습니다. 메이저 또는 마이너 버전을 포함하여 모든 수준의 버전으로를 잠글 수 있습니다. App Runner는 서비스의 런타임만 하위 수준으로 업데이트합니다.

Ruby 런타임용 버전 구문: *`major`*`[.`*`minor`*`[.`*`patch`*`]]`

예: `3.1.2`

다음 예제에서는 버전 잠금을 보여줍니다.

- `3.1` - 메이저 및 마이너 버전을 잠급니다. App Runner는 패치 버전만 업데이트합니다.
- `3.1.2` - 특정 패치 버전으로 잠급니다. App Runner는 런타임 버전을 업데이트하지 않습니다.

주제

Ruby 런타임 구성

관리형 런타임을 선택할 때는 최소한 빌드 및 실행 명령도 구성해야 합니다. App Runner 서비스를 생성하거나 업데이트하는 동안 이를 구성합니다. 다음 방법 중 하나를 사용하여이 작업을 수행할 수 있습니다.

- App Runner 콘솔 사용 - 생성 프로세스 또는 구성 탭의 빌드 구성 섹션에서 명령을 지정합니다.
- App Runner API 사용 - CreateService 또는 UpdateService API 작업을 호출합니다. CodeConfigurationValues 데이터 형식의 `BuildCommand` 및 `StartCommand` 멤버를 사용하여 명령을 지정합니다.
- 구성 파일 사용 - 최대 3개의 빌드 단계에서 하나 이상의 빌드 명령을 지정하고 애플리케이션을 시작하는 데 사용되는 단일 실행 명령을 지정합니다. 추가 선택적 구성 설정이 있습니다.

구성 파일을 제공하는 것은 선택 사항입니다. 콘솔 또는 API를 사용하여 App Runner 서비스를 생성할 때 App Runner가 구성 설정을 생성할 때 직접 가져오거나 구성 파일에서 가져올지 지정합니다.

Ruby 런타임 예제

다음 예제에서는 Ruby 서비스를 빌드하고 실행하기 위한 App Runner 구성 파일을 보여줍니다.

최소 Ruby 구성 파일

이 예제는 Ruby 관리형 런타임과 함께 사용할 수 있는 최소 구성 파일을 보여줍니다. App Runner가 최소 구성 파일로 가정하는 내용은 섹션을 참조하세요the section called “구성 파일 예제”.

Example apprunner.yaml

```
version: 1.0
```

```
runtime: ruby31
build:
  commands:
    build:
      - bundle install
run:
  command: bundle exec rackup --host 0.0.0.0 -p 8080
```

확장 Ruby 구성 파일

이 예제에서는 Ruby 관리형 런타임에서 모든 구성 키를 사용하는 방법을 보여줍니다.

> **Note**
>
> 이 예제에서 사용되는 런타임 버전은 *3.1.2*입니다. 이를 사용하려는 버전으로 바꿀 수 있습니다. 지원되는 최신 Ruby 런타임 버전은 섹션을 참조하세요the section called “릴리스 정보”.

Example apprunner.yaml

```
version: 1.0
runtime: ruby31
build:
  commands:
     pre-build:
      - scripts/prebuild.sh
    build:
      - bundle install
    post-build:
      - scripts/postbuild.sh
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
run:
  runtime-version: 3.1.2
  command: bundle exec rackup --host 0.0.0.0 -p 4567
  network:
    port: 4567
    env: APP_PORT
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
```

전체 Ruby 애플리케이션 소스

이 예제에서는 Ruby 런타임 서비스에 배포할 수 있는 전체 Ruby 애플리케이션의 소스 코드를 보여줍니다.

Example server.rb

```
# server.rb
require 'sinatra'

get '/' do
  'Hello World!'
end
```

Example config.ru

```
# config.ru

require './server'

run Sinatra::Application
```

Example Gemfile

```
# Gemfile
source 'https://rubygems.org (https://rubygems.org/)'

gem 'sinatra'
gem 'puma'
```

Example apprunner.yaml

```
version: 1.0
runtime: ruby31
build:
  commands:
    build:
      - bundle install
run:
  command: bundle exec rackup --host 0.0.0.0 -p 4567
  network:
```

```
    port: 4567
    env: APP_PORT
```

Ruby 런타임 릴리스 정보

> **Important**
>
> App Runner는 2025년 12월 1일에 Ruby 3.1에 대한 지원을 종료합니다. 권장 사항 및 자세한 내용은 섹션을 참조하세요the section called “관리형 런타임 버전에 대한 지원 종료”.

이 주제에서는 App Runner가 지원하는 Ruby 런타임 버전의 전체 세부 정보를 나열합니다.

지원되는 런타임 버전 - 원래 App Runner 빌드

실행 시간 이름	마이너 버전	포함된 패키지
Ruby 3.1(ruby31)	3.1.7	SQLite 3.50.2
	3.1.7	SQLite 3.50.1
	3.1.7	SQLite 3.50.0
	3.1.6	SQLite 3.49.1
	3.1.4	SQLite 3.46.0
	3.1.3	SQLite 3.41.0
	3.1.2	SQLite 3.39.4

> **Note**
>
> App Runner는 최근에 릴리스된 특정 주요 런타임에 대해 수정된 빌드 프로세스를 제공합니다. 따라서이 문서의 특정 섹션에 수정된 App Runner 빌드 및 원래 App Runner 빌드에 대한 참조가 표시됩니다. 자세한 내용은 단원을 참조하십시오관리형 런타임 버전 및 App Runner 빌드.

Go 플랫폼 사용

> **Important**
>
> App Runner는 2025년 12월 1일에 Go 1.18에 대한 지원을 종료합니다. 권장 사항 및 자세한 내용은 섹션을 참조하세요the section called “관리형 런타임 버전에 대한 지원 종료”.

AWS App Runner Go 플랫폼은 관리형 런타임을 제공합니다. 각 런타임을 사용하면 Go 버전을 기반으로 웹 애플리케이션을 사용하여 컨테이너를 쉽게 빌드하고 실행할 수 있습니다. Go 런타임을 사용하는 경우 App Runner는 관리형 Go 런타임 이미지로 시작합니다. 이 이미지는 Amazon Linux Docker 이미지를 기반으로 하며 Go 버전 및 일부 도구에 대한 런타임 패키지를 포함합니다. App Runner는이 관리형 런타임 이미지를 기본 이미지로 사용하고 애플리케이션 코드를 추가하여 도커 이미지를 빌드합니다. 그런 다음이 이미지를 배포하여 컨테이너에서 웹 서비스를 실행합니다.

App Runner 콘솔 또는 CreateService API 작업을 사용하여 서비스를 생성할 때 App Runner 서비스에 대한 런타임을 지정합니다. 런타임을 소스 코드의 일부로 지정할 수도 있습니다. 코드 리포지토리에 포함하는 App Runner 구성 파일의 `runtime` 키워드를 사용합니다. 관리형 런타임의 이름 지정 규칙은 *`<language-name><major-version>`*입니다.

유효한 Go 런타임 이름 및 버전은 섹션을 참조하세요the section called “릴리스 정보”.

App Runner는 모든 배포 또는 서비스 업데이트 시 서비스의 런타임을 최신 버전으로 업데이트합니다. 애플리케이션에 특정 버전의 관리형 런타임이 필요한 경우 App Runner 구성 파일의 `runtime-version` 키워드를 사용하여 지정할 수 있습니다. 메이저 또는 마이너 버전을 포함하여 모든 수준의 버전으로를 잠글 수 있습니다. App Runner는 서비스의 런타임만 하위 수준으로 업데이트합니다.

Go 런타임용 버전 구문: *`major`*`[.`*`minor`*`[.`*`patch`*`]]`

예: `1.18.7`

다음 예제에서는 버전 잠금을 보여줍니다.

- `1.18` - 메이저 및 마이너 버전을 잠급니다. App Runner는 패치 버전만 업데이트합니다.
- `1.18.7` - 특정 패치 버전으로 잠급니다. App Runner는 런타임 버전을 업데이트하지 않습니다.

주제

- Go 런타임 구성

- Go 런타임 예제
- Go 런타임 릴리스 정보

Go 런타임 구성

관리형 런타임을 선택할 때는 최소한 빌드 및 실행 명령도 구성해야 합니다. App Runner 서비스를 생성하거나 업데이트하는 동안 이를 구성합니다. 다음 방법 중 하나를 사용하여이 작업을 수행할 수 있습니다.

- App Runner 콘솔 사용 - 생성 프로세스 또는 구성 탭의 빌드 구성 섹션에서 명령을 지정합니다.
- App Runner API 사용 - CreateService 또는 UpdateService API 작업을 호출합니다. CodeConfigurationValues 데이터 형식의 `BuildCommand` 및 `StartCommand` 멤버를 사용하여 명령을 지정합니다.
- 구성 파일 사용 - 최대 3개의 빌드 단계에서 하나 이상의 빌드 명령을 지정하고 애플리케이션을 시작하는 데 사용되는 단일 실행 명령을 지정합니다. 추가 선택적 구성 설정이 있습니다.

구성 파일을 제공하는 것은 선택 사항입니다. 콘솔 또는 API를 사용하여 App Runner 서비스를 생성할 때 App Runner가 구성 설정을 생성할 때 직접 가져오거나 구성 파일에서 가져올지 지정합니다.

Go 런타임 예제

다음 예제에서는 Go 서비스를 빌드하고 실행하기 위한 App Runner 구성 파일을 보여줍니다.

Minimal Go 구성 파일

이 예제는 Go 관리형 런타임과 함께 사용할 수 있는 최소 구성 파일을 보여줍니다. App Runner가 최소 구성 파일로 가정하는 내용은 섹션을 참조하세요the section called “구성 파일 예제”.

Example apprunner.yaml

```
version: 1.0
runtime: go1
build:
  commands:
    build:
      - go build main.go
run:
  command: ./main
```

확장 Go 구성 파일

이 예제에서는 Go 관리형 런타임에서 모든 구성 키를 사용하는 방법을 보여줍니다.

> **Note**
>
> 이 예제에서 사용되는 런타임 버전은 *1.18.7*입니다. 이를 사용하려는 버전으로 바꿀 수 있습니다. 지원되는 최신 Go 런타임 버전은 섹션을 참조하세요the section called “릴리스 정보”.

Example apprunner.yaml

```
version: 1.0
runtime: go1
build:
  commands:
     pre-build:
      - scripts/prebuild.sh
    build:
      - go build main.go
    post-build:
      - scripts/postbuild.sh
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
run:
  runtime-version: 1.18.7
  command: ./main
  network:
    port: 3000
    env: APP_PORT
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
```

전체 Go 애플리케이션 소스

이 예제에서는 Go 런타임 서비스에 배포할 수 있는 전체 Go 애플리케이션의 소스 코드를 보여줍니다.

Example main.go

```
package main
import (
```

```
    "fmt"
    "net/http"
)

func main() {
    http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
        fmt.Fprint(w, "<h1>Welcome to App Runner</h1>")
    })
    fmt.Println("Starting the server on :3000...")
    http.ListenAndServe(":3000", nil)
}
```

Example apprunner.yaml

```
version: 1.0
runtime: go1
build:
  commands:
    build:
      - go build main.go
run:
  command: ./main
  network:
    port: 3000
    env: APP_PORT
```

Go 런타임 릴리스 정보

> **Important**
>
> App Runner는 2025년 12월 1일에 Go 1.18에 대한 지원을 종료합니다. 권장 사항 및 자세한 내용은 섹션을 참조하세요the section called “관리형 런타임 버전에 대한 지원 종료”.

이 주제에서는 App Runner가 지원하는 Go 런타임 버전에 대한 전체 세부 정보를 나열합니다.

지원되는 런타임 버전 - 원래 App Runner 빌드

실행 시간 이름	마이너 버전	포함된 패키지
Go 1(go1)	1.18.10	

실행 시간 이름	마이너 버전	포함된 패키지
	1.18.9	
	1.18.8	
	1.18.7	

> **Note**
>
> App Runner는 최근에 릴리스된 특정 주요 런타임에 대해 수정된 빌드 프로세스를 제공합니다. 따라서이 문서의 특정 섹션에 수정된 App Runner 빌드 및 원래 App Runner 빌드에 대한 참조가 표시됩니다. 자세한 내용은 단원을 참조하십시오관리형 런타임 버전 및 App Runner 빌드.

App Runner용 애플리케이션 코드 개발

이 장에서는 배포용 애플리케이션 코드를 개발하거나 마이그레이션할 때 고려해야 할 런타임 정보 및 개발 지침에 대해 설명합니다 AWS App Runner.

런타임 정보

컨테이너 이미지를 제공하든 App Runner가 컨테이너 이미지를 빌드하든 App Runner는 컨테이너 인스턴스에서 애플리케이션 코드를 실행합니다. 다음은 컨테이너 인스턴스 런타임 환경의 몇 가지 주요 측면입니다.

- 프레임워크 지원 - App Runner는 웹 애플리케이션을 구현하는 모든 이미지를 지원합니다. 선택한 프로그래밍 언어와 사용하는 웹 애플리케이션 서버 또는 프레임워크에 관계없이 사용할 수 있습니다. 편의를 위해 다양한 프로그래밍 플랫폼을 위한 플랫폼별 관리형 런타임을 제공하여 애플리케이션 빌드 프로세스를 간소화하고 이미지를 추상화합니다.
- 웹 요청 - App Runner는 컨테이너 인스턴스에 HTTP 1.0 및 HTTP 1.1에 대한 지원을 제공합니다. 서비스 구성에 대한 자세한 내용은 섹션을 참조하세요the section called “구성”. HTTPS 보안 트래픽 처리를 구현할 필요가 없습니다. App Runner는 모든 수신 HTTP 요청을 해당 HTTPS 엔드포인트로 리디렉션합니다. HTTP 웹 요청 리디렉션을 활성화하도록 설정을 구성할 필요가 없습니다. App Runner는 요청을 애플리케이션 컨테이너 인스턴스에 전달하기 전에 TLS를 종료합니다.

  > **Note**
  >
  > - HTTP 요청에는 총 120초의 요청 제한 시간이 있습니다. 120초에는 애플리케이션이 본문을 포함하여 요청을 읽고 HTTP 응답 작성을 완료하는 데 걸리는 시간이 포함됩니다.
  > - 요청 읽기 및 응답 제한 시간은 사용하는 애플리케이션에 따라 다릅니다. 이러한 애플리케이션에는 Python용 HTTP 서버, Gunicorn과 같은 자체 내부 제한 시간이 있을 수 있으며 기본 제한 시간은 30초입니다. 이 경우 애플리케이션의 제한 시간이 App Runner 120초 제한 시간을 재정의합니다.
  > - App Runner가 완전 관리형 서비스이므로 TLS 암호 제품군 또는 기타 파라미터를 구성할 필요가 없으며 TLS 종료를 관리합니다.

- 상태 비저장 앱 - 현재 App Runner는 상태 저장 앱을 지원하지 않습니다. 따라서 App Runner는 단일 수신 웹 요청을 처리하는 기간 이후의 상태 지속성을 보장하지 않습니다.
- 스토리지 - App Runner는 수신 트래픽 볼륨에 따라 App Runner 애플리케이션의 인스턴스를 자동으로 확장하거나 축소합니다. App Runner 애플리케이션에 대한 Auto Scaling 옵션을 구성할 수 있습니

다. 웹 요청을 처리하는 현재 활성 인스턴스 수는 수신 트래픽 볼륨을 기반으로 하므로 App Runner는 단일 요청 처리 후에도 파일이 지속될 수 있다고 보장할 수 없습니다. 따라서 App Runner는 컨테이너 인스턴스의 파일 시스템을 임시 스토리지로 구현하므로 파일이 일시적입니다. 예를 들어 App Runner 서비스를 일시 중지했다가 다시 시작할 때 파일이 지속되지 않습니다.

App Runner는 3GB의 임시 스토리지를 제공하며 인스턴스에서 풀링, 압축 및 압축되지 않은 컨테이너 이미지에 3GB의 임시 스토리지의 일부를 사용합니다. App Runner 서비스에서 나머지 임시 스토리지를 사용할 수 있습니다. 그러나 상태 비저장 특성으로 인해 영구 스토리지는 아닙니다.

> **Note**
>
> 스토리지 파일이 요청 간에 지속되는 시나리오가 있을 수 있습니다. 예를 들어 다음 요청이 동일한 인스턴스에 도착하면 스토리지 파일이 유지됩니다. 요청 간 스토리지 파일의 지속성은 특정 상황에서 유용할 수 있습니다. 예를 들어 요청을 처리할 때 향후 요청에 필요할 경우 애플리케이션이 다운로드하는 파일을 캐싱할 수 있습니다. 이렇게 하면 향후 요청 처리 속도가 빨라질 수 있지만 속도 증가를 보장할 수는 없습니다. 코드는 이전 요청에서 다운로드한 파일이 여전히 존재한다고 가정해서는 안 됩니다.
> 높은 처리량, 짧은 지연 시간 인 메모리 데이터 스토어를 사용하여 캐싱을 보장하려면 Amazon ElastiCache와 같은 서비스를 사용합니다.

- 환경 변수 - 기본적으로 App Runner는 컨테이너 인스턴스에서 `PORT` 환경 변수를 사용할 수 있도록 합니다. 포트 정보로 변수 값을 구성하고 사용자 지정 환경 변수 및 값을 추가할 수 있습니다. AWS Secrets Manager 또는 AWS Systems Manager 파라미터 스토어에 저장된 민감한 데이터를 환경 변수로 참조할 수도 있습니다. 환경 변수 생성에 대한 자세한 내용은 섹션을 참조하세요참조 환경 변수.
- 인스턴스 역할 - 애플리케이션 코드가 AWS 서비스 APIs 또는 AWS SDKs 중 하나를 사용하여 서비스를 호출하는 경우 AWS Identity and Access Management (IAM)을 사용하여 인스턴스 역할을 생성합니다. 그런 다음 생성할 때 App Runner 서비스에 연결합니다. 코드에 필요한 모든 AWS 서비스 작업 권한을 인스턴스 역할에 포함합니다. 자세한 내용은 the section called “인스턴스 역할” 단원을 참조하십시오.

코드 개발 지침

App Runner 웹 애플리케이션용 코드를 개발할 때 다음 지침을 고려하세요.

- 컨테이너 이미지 패치 적용 - 컨테이너 이미지를 제공할 때 사용자는 이러한 이미지를 정기적으로 업데이트하고 패치를 적용할 책임이 있습니다. App Runner가 인프라를 관리하는 동안 제공된 컨테이

너 이미지의 보안 및 up-to-date 상태를 확인해야 합니다. 자세한 내용은 AWS App Runner 설명서를 참조하세요.

- 상태 비저장 코드 설계 - App Runner 서비스에 배포하는 웹 애플리케이션을 상태 비저장으로 설계합니다. 코드는 단일 수신 웹 요청을 처리하는 기간 이후에도 상태가 지속되지 않는다고 가정해야 합니다.
- 임시 파일 삭제 - 파일을 생성하면 파일 시스템에 저장되고 서비스의 스토리지 할당에 포함됩니다. out-of-storage 오류를 방지하려면 임시 파일을 장기간 보관하지 마세요. 파일 캐싱 결정을 내릴 때 스토리지 크기와 요청 처리 속도의 균형을 맞춥니다.
- 인스턴스 시작 - App Runner는 5분의 인스턴스 시작 시간을 제공합니다. 인스턴스는 구성된 수신 포트에서 요청을 수신해야 하며 시작 후 5분 이내에 정상 상태여야 합니다. 시작 시간 동안 App Runner 인스턴스에는 vCPU 구성에 따라 가상 CPU(vCPU)가 할당됩니다. 사용 가능한 vCPU 구성에 대한 자세한 내용은 섹션을 참조하세요the section called “App Runner 지원 구성”.

  인스턴스가 성공적으로 시작되면 유휴 상태로 전환되고 요청을 기다립니다. 인스턴스 시작 기간을 기준으로 비용을 지불하며, 인스턴스 시작당 최소 요금은 1분입니다. 요금에 대한 자세한 정보는 AWS App Runner 요금을 참조하세요.

App Runner 콘솔 사용

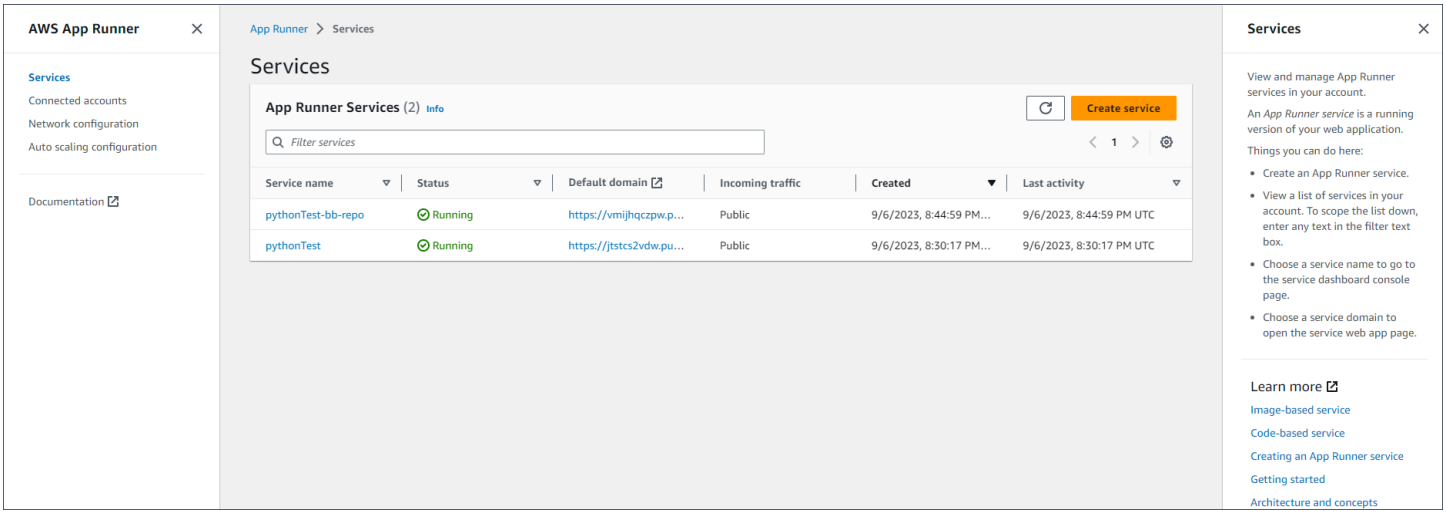
AWS App Runner 콘솔을 사용하여 App Runner 서비스 및 연결된 계정과 같은 관련 리소스를 생성, 관리 및 모니터링합니다. 기존 서비스를 보고, 새 서비스를 생성하고, 서비스를 구성할 수 있습니다. App Runner 서비스의 상태를 보고 로그를 보고 활동을 모니터링하며 지표를 추적할 수 있습니다. 서비스 웹 사이트 또는 소스 리포지토리로 이동할 수도 있습니다.

다음 섹션에서는 콘솔의 레이아웃과 기능에 대해 설명하고 관련 정보를 가리킵니다.

전체 콘솔 레이아웃

App Runner 콘솔에는 세 가지 영역이 있습니다. 왼쪽에서 오른쪽으로:

- 탐색 창 - 축소하거나 확장할 수 있는 측면 창입니다. 이를 사용하여 사용하려는 최상위 콘솔 페이지를 선택합니다.
- 콘텐츠 창 - 콘솔 페이지의 주요 부분입니다. 이를 사용하여 정보를 보고 작업을 수행할 수 있습니다.
- 도움말 창 - 자세한 내용을 볼 수 있는 측면 창입니다. 확장하여 현재 페이지에 대한 도움말을 확인합니다. 또는 콘솔 페이지에서 정보 링크를 선택하여 상황에 맞는 도움을 받을 수 있습니다.



서비스 페이지

서비스 페이지에는 계정의 App Runner 서비스가 나열됩니다. 필터 텍스트 상자를 사용하여 목록의 범위를 좁힐 수 있습니다.

서비스 페이지로 이동하려면

1. App Runner 콘솔을 열고 리전 목록에서를 선택합니다 AWS 리전.
2. 탐색 창에서 서비스를 선택합니다.

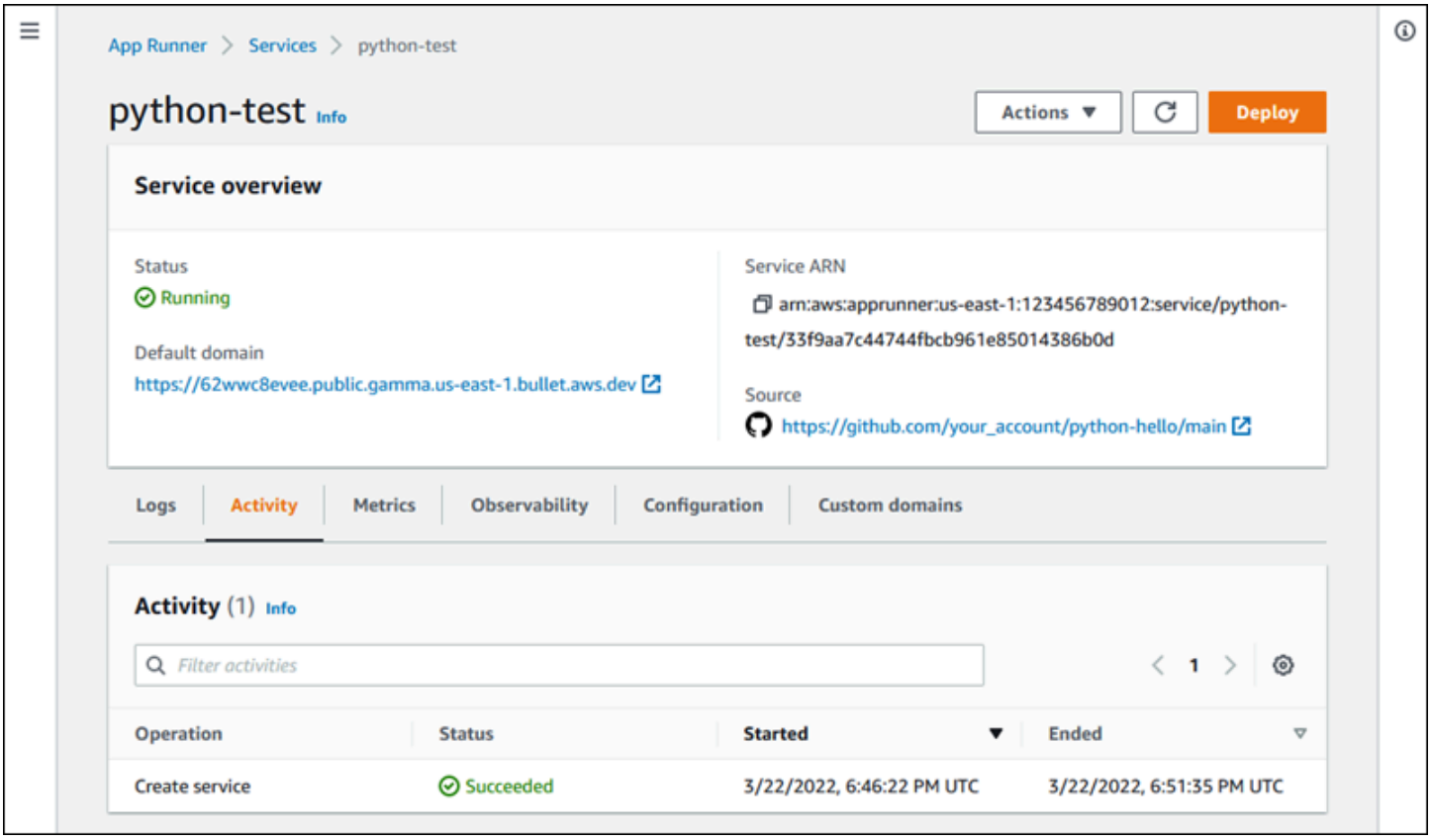
여기에서 수행할 수 있는 작업:

- App Runner 서비스를 생성합니다. 자세한 내용은 the section called “만들기” 단원을 참조하십시오.
- 서비스 이름을 선택하여 서비스 대시보드 콘솔 페이지로 이동합니다.
- 서비스 도메인을 선택하여 서비스 웹 앱 페이지를 엽니다.

서비스 대시보드 페이지

App Runner 서비스에 대한 정보를 보고 서비스 대시보드 페이지에서 관리할 수 있습니다. 페이지 상단에서 서비스 이름을 볼 수 있습니다.

서비스 대시보드로 이동하려면 서비스 페이지(이전 섹션 참조)로 이동한 다음 App Runner 서비스를 선택합니다.



서비스 개요 섹션에서는 App Runner 서비스 및 애플리케이션에 대한 기본 세부 정보를 제공합니다. 여기에서 수행할 수 있는 작업:

- 상태, 상태 및 ARN과 같은 서비스 세부 정보를 봅니다.
- App Runner가 서비스에서 실행되는 웹 애플리케이션에 제공하는 도메인인 기본 도메인으로 이동합니다. App Runner가 소유한 도메인의 하위 `awsapprunner.com` 도메인입니다.
- 서비스에 배포된 소스 리포지토리로 이동합니다.
- 서비스에 대한 소스 리포지토리 배포를 시작합니다.
- 서비스를 일시 중지, 재개 및 삭제합니다.

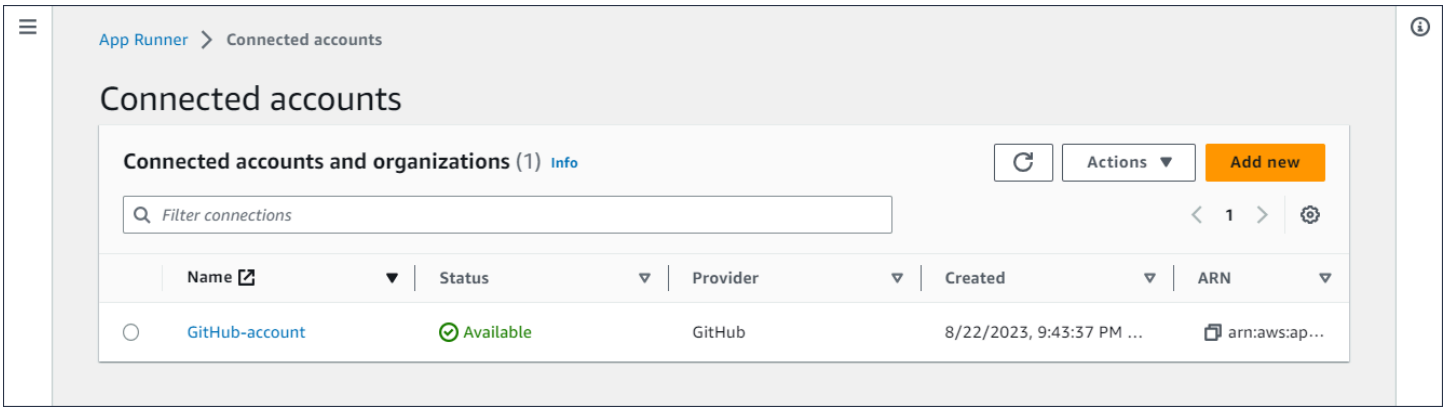
서비스 개요 아래의 탭은 서비스 관리 및 관찰성을 위한 것입니다.

연결된 계정 페이지

연결된 계정 페이지에는 계정의 소스 코드 리포지토리 공급자에 대한 App Runner 연결이 나열됩니다. 필터 텍스트 상자를 사용하여 목록의 범위를 좁힐 수 있습니다. 연결된 계정에 대한 자세한 내용은 섹션을 참조하세요the section called “연결”.

연결된 계정 페이지로 이동하려면

1. App Runner 콘솔을 열고 리전 목록에서를 선택합니다 AWS 리전.
2. 탐색 창에서 연결된 계정을 선택합니다.



여기에서 수행할 수 있는 작업:

- 계정의 리포지토리 공급자 연결 목록을 봅니다. 목록의 범위를 좁히려면 필터 텍스트 상자에 텍스트를 입력합니다.

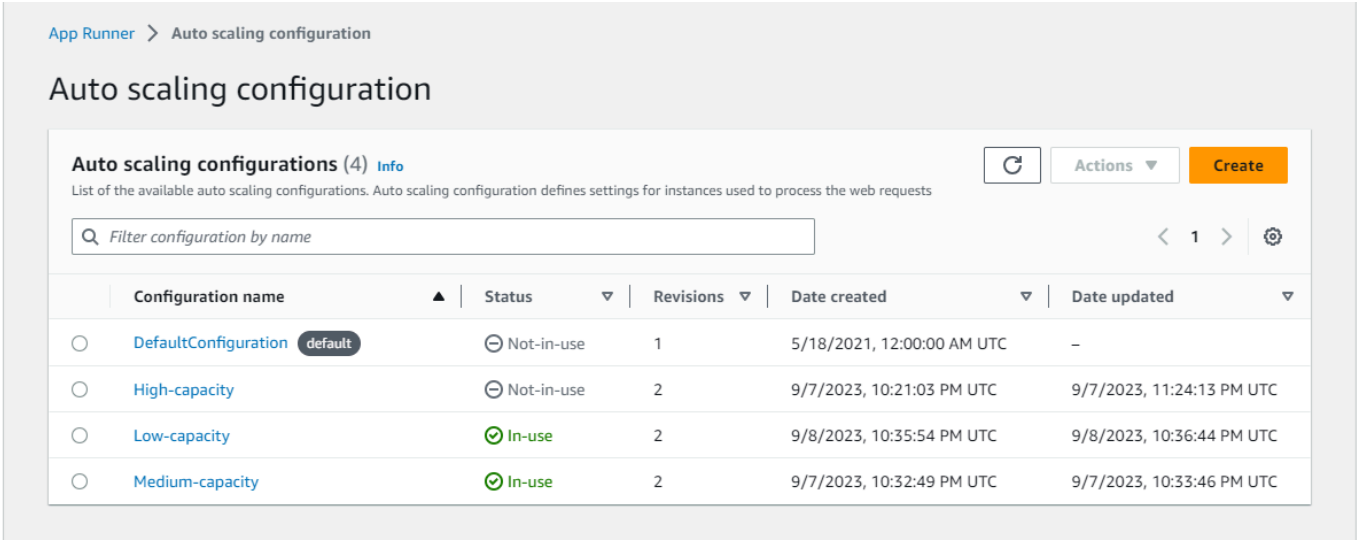
- 연결 이름을 선택하여 관련 공급자 계정 또는 조직으로 이동합니다.
- 연결을 선택하여 방금 설정한 연결에 대한 핸드셰이크를 완료하거나(서비스 생성의 일부로) 연결을 삭제합니다.

Auto Scaling 구성 페이지

Auto Scaling 구성 페이지에는 계정에서 설정한 Auto Scaling 구성이 나열됩니다. 몇 가지 파라미터를 구성하여 Auto Scaling 동작을 조정하고 나중에 하나 이상의 App Runner 서비스에 할당할 수 있는 다양한 구성에 저장할 수 있습니다. 필터 텍스트 상자를 사용하여 목록의 범위를 좁힐 수 있습니다. Auto Scaling 구성에 대한 자세한 내용은 섹션을 참조하세요서비스의 Auto Scaling 관리.

Auto Scaling 구성 페이지로 이동하려면

1. App Runner 콘솔을 열고 리전 목록에서를 선택합니다 AWS 리전.
2. 탐색 창에서 Auto Scaling 구성을 선택합니다.



여기에서 수행할 수 있는 작업:

- 계정의 기존 Auto Scaling 구성 목록을 봅니다.
- 기존 Auto Scaling 구성 또는 개정을 새로 생성합니다.
- 자동 조정 구성을 생성한 새 서비스의 기본값으로 설정합니다.
- 구성을 삭제합니다.
- 구성 이름을 선택하여 Auto Scaling 개정 패널로 이동하여 개정을 관리합니다.

App Runner 서비스 관리

이 장에서는 AWS App Runner 서비스를 관리하는 방법을 설명합니다. 이 장에서는 서비스 생성, 구성 및 삭제, 서비스에 새 애플리케이션 버전 배포, 서비스 일시 중지 및 재개를 통한 웹 서비스 가용성 제어 등 서비스의 수명 주기를 관리하는 방법을 알아봅니다. 또한 연결 및 Auto Scaling과 같은 서비스의 다른 측면을 관리하는 방법도 알아봅니다.

주제

App Runner 서비스 생성

AWS App Runner 는 컨테이너 이미지 또는 소스 코드 리포지토리에서 자동으로 확장되는 실행 중인 웹 서비스로의 전환을 자동화합니다. App Runner는 소수의 필수 설정만 지정하여 소스 이미지 또는 코드를 가리킵니다. App Runner는 필요한 경우 애플리케이션을 빌드하고, 컴퓨팅 리소스를 프로비저닝하고, 애플리케이션을 배포하여 애플리케이션을 실행합니다.

서비스를 생성하면 App Runner가 서비스 리소스를 생성합니다. 경우에 따라 연결 리소스를 제공해야 할 수 있습니다. App Runner 콘솔을 사용하는 경우 콘솔은 암시적으로 연결 리소스를 생성합니다. App Runner 리소스 유형에 대한 자세한 내용은 섹션을 참조하세요the section called “App Runner 리소스”. 이러한 리소스 유형에는 각의 계정과 연결된 할당량이 있습니다 AWS 리전. 자세한 내용은 the section called “App Runner 리소스 할당량” 단원을 참조하십시오.

소스 유형 및 공급자에 따라 서비스 생성 절차에 미묘한 차이가 있습니다. 이 주제에서는 상황에 적합한 유형을 따를 수 있도록 이러한 소스 유형을 생성하는 다양한 절차를 다룹니다. 코드 예제로 기본 절차를 시작하려면 섹션을 참조하세요시작하기.

사전 조건

App Runner 서비스를 생성하기 전에 다음 작업을 완료해야 합니다.

- 의 설정 단계를 완료합니다설정.
- 애플리케이션 소스가 준비되었는지 확인합니다. GitHub, Bitbucket의 코드 리포지토리 또는 Amazon Elastic Container Registry(Amazon ECR)의 컨테이너 이미지를 사용하여 App Runner 서비스를 생성할 수 있습니다.

서비스 생성

이 섹션에서는 소스 코드와 컨테이너 이미지를 기반으로 하는 두 가지 App Runner 서비스 유형에 대한 생성 프로세스를 안내합니다.

> **Note**
>
> 서비스에 대한 아웃바운드 트래픽 VPC 커넥터를 생성하는 경우 다음 서비스 시작 프로세스에는 일회성 지연 시간이 발생합니다. 새 서비스를 생성할 때 또는 이후에 서비스 업데이트를 통해이 구성을 설정할 수 있습니다. 자세한 내용은이 가이드일회성 지연 시간의 App Runner를 사용한 네트워킹 장의 섹션을 참조하세요.

코드 리포지토리에서 서비스 생성

다음 섹션에서는 소스가 GitHub 또는 Bitbucket의 코드 리포지토리인 경우 App Runner 서비스를 생성하는 방법을 보여줍니다. 코드 리포지토리를 사용하는 경우 App Runner는 공급자 조직 또는 계정에 연결해야 합니다. 따라서이 연결을 설정하는 데 도움이 필요합니다. App Runner 연결에 대한 자세한 내용은 섹션을 참조하세요the section called “연결”.

서비스를 생성하면 App Runner는 애플리케이션 코드와 종속성이 포함된 Docker 이미지를 빌드합니다. 그런 다음이 이미지의 컨테이너 인스턴스를 실행하는 서비스를 시작합니다.

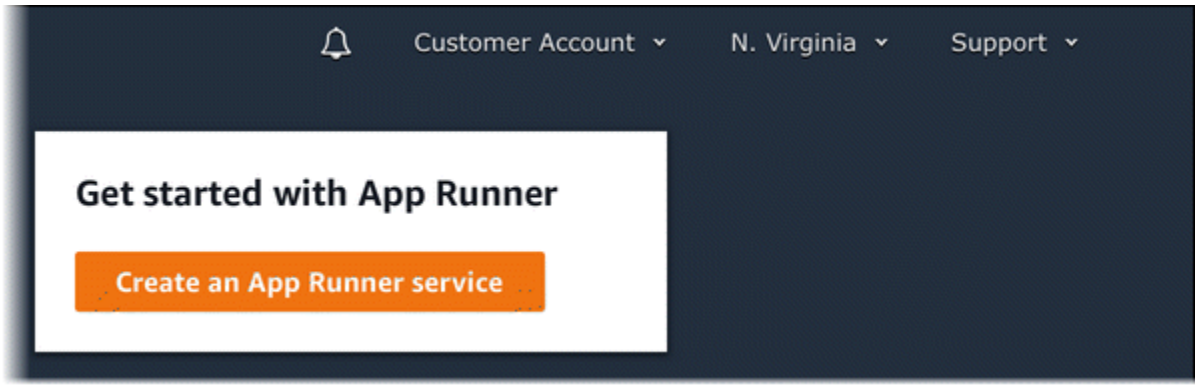
App Runner 콘솔을 사용하여 코드에서 서비스 생성

콘솔을 사용하여 App Runner 서비스를 생성하려면

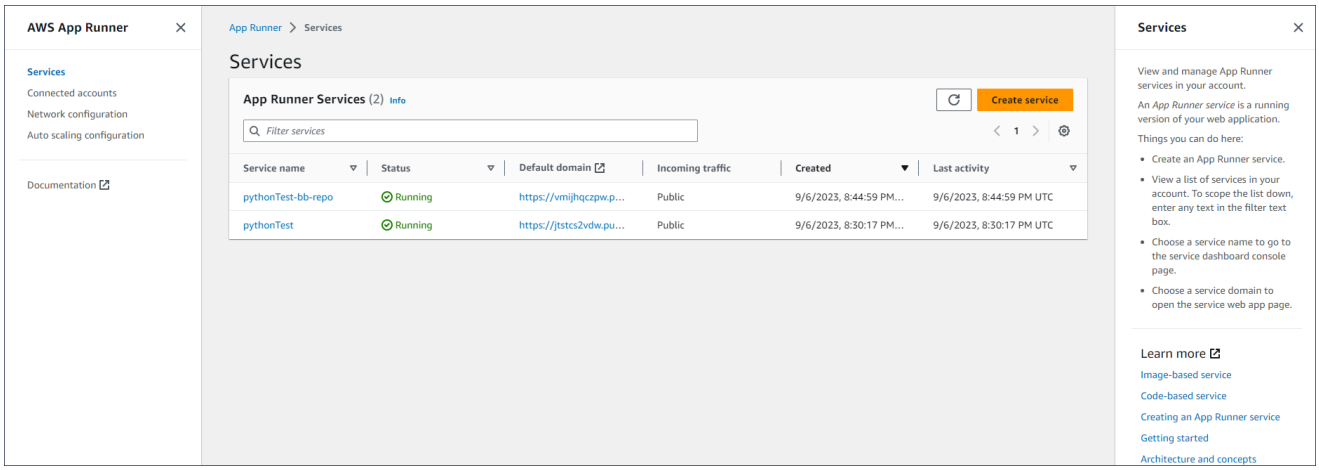
1. 소스 코드를 구성합니다.

   a. App Runner 콘솔을 열고 리전 목록에서를 선택합니다 AWS 리전.

b. 에 아직 App Runner 서비스가 AWS 계정 없는 경우 콘솔 홈 페이지가 표시됩니다. App Runner 서비스 생성을 선택합니다.



에 기존 서비스가 AWS 계정 있는 경우 서비스 목록과 함께 서비스 페이지가 표시됩니다. 서비스 생성을 선택합니다.



c. 소스 및 배포 페이지의 소스 섹션의 리포지토리 유형에서 소스 코드 리포지토리를 선택합니다.

d. 공급자 유형을 선택합니다. GitHub 또는 Bitbucket을 선택합니다.

e. 그런 다음 이전에 사용한 공급자의 계정 또는 조직을 선택하거나 새로 추가를 선택합니다. 그런 다음 코드 리포지토리 자격 증명을 제공하고 연결할 계정 또는 조직을 선택하는 프로세스를 살펴봅니다.

f. 리포지토리에서 애플리케이션 코드가 포함된 리포지토리를 선택합니다.

g. 브랜치에서 배포하려는 브랜치를 선택합니다.

h. 소스 디렉터리에 애플리케이션 코드와 구성 파일을 저장하는 소스 리포지토리의 디렉터리를 입력합니다.

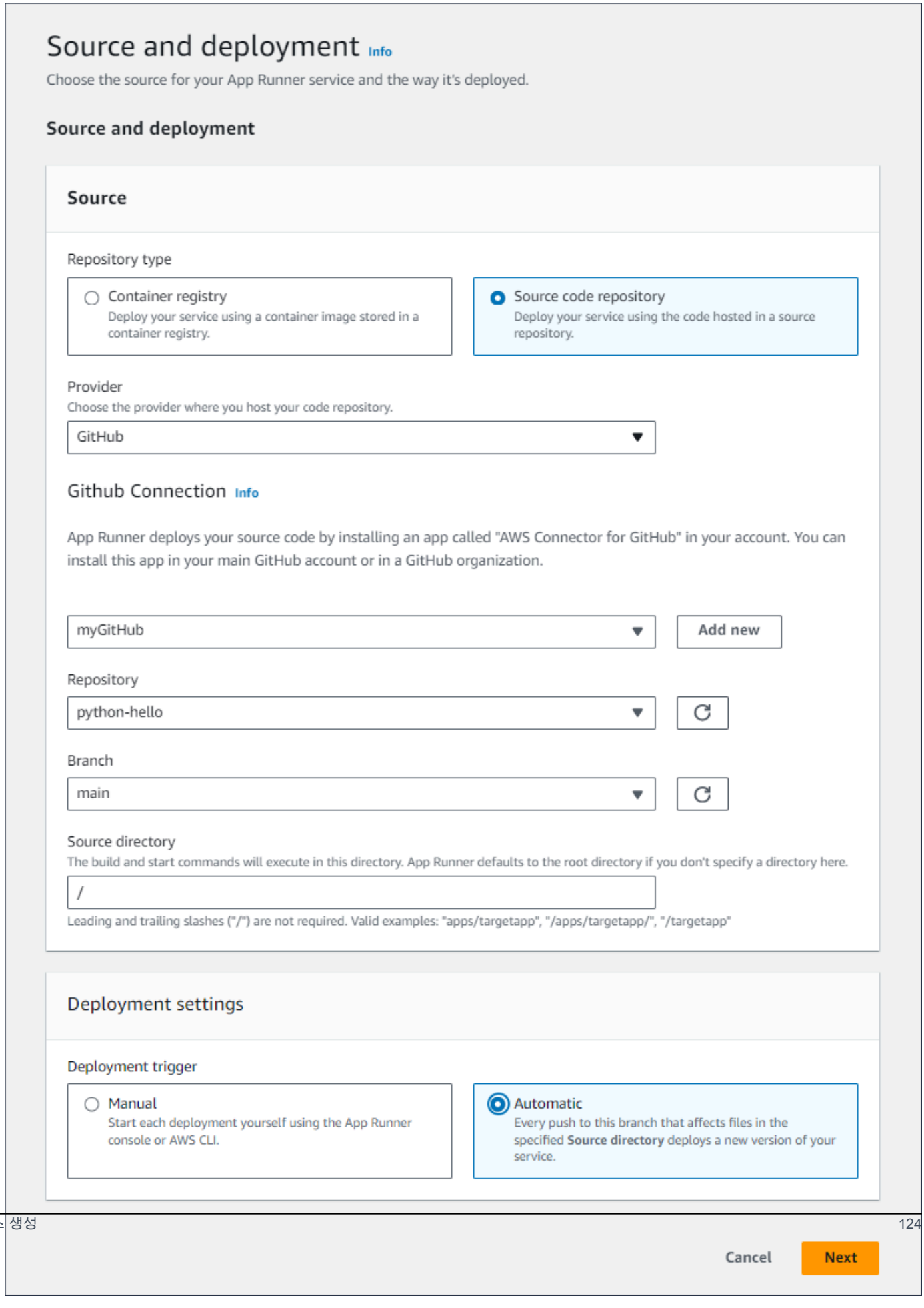
> **Note**
>
> 빌드 및 시작 명령은 지정한 소스 디렉터리에서 실행됩니다. App Runner는 경로를 루트에서 절대 경로로 처리합니다. 여기서 값을 지정하지 않으면 디렉터리는 기본적으로 리포지토리 루트로 설정됩니다.

2. 배포를 구성합니다.

   a. 배포 설정 섹션에서 수동 또는 자동을 선택합니다.

      배포 방법에 대한 자세한 내용은 섹션을 참조하세요the section called “배포 방법”.

   b. 다음을 선택합니다.


Source and deployment Info
Choose the source for your App Runner service and the way it's deployed.
Source and deployment
Source
Repository type
Container registry
Deploy your service using a container image stored in a container registry.
Source code repository
Deploy your service using the code hosted in a source repository.
Provider
Choose the provider where you host your code repository.
GitHub
Github Connection Info
App Runner deploys your source code by installing an app called "AWS Connector for GitHub" in your account. You can install this app in your main GitHub account or in a GitHub organization.
myGitHub
Add new
Repository
python-hello
Branch
main
Source directory
The build and start commands will execute in this directory. App Runner defaults to the root directory if you don't specify a directory here.
/
Leading and trailing slashes ("/") are not required. Valid examples: "apps/targetapp", "/apps/targetapp/", "/targetapp"
Deployment settings
Deployment trigger
Manual
Start each deployment yourself using the App Runner console or AWS CLI.
Automatic
Every push to this branch that affects files in the specified Source directory deploys a new version of your service.
Cancel
Next

3. 애플리케이션 빌드를 구성합니다.

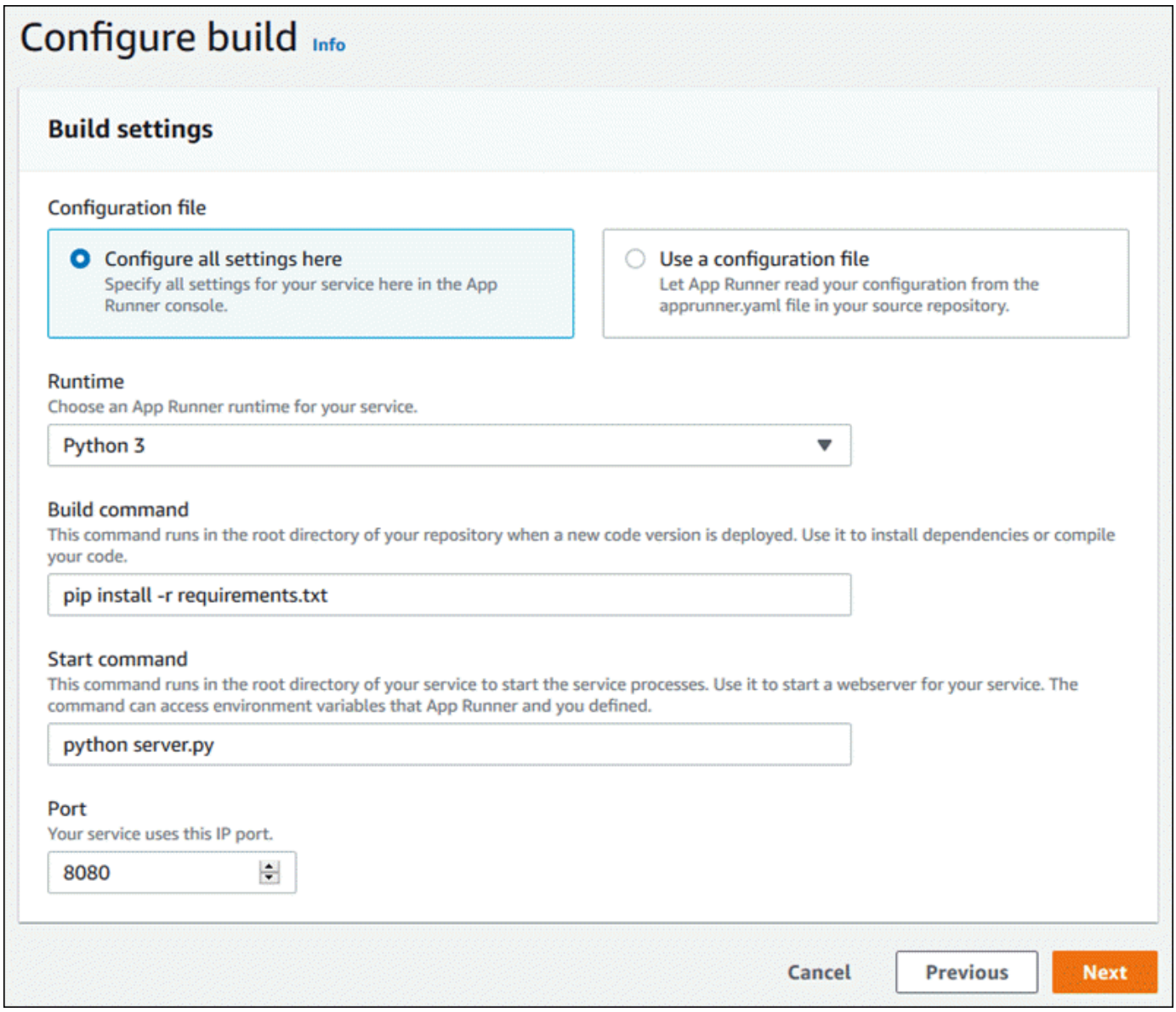
   a. 빌드 구성 페이지의 구성 파일에서 리포지토리에 App Runner 구성 파일이 포함되어 있지 않은 경우 모든 설정 구성을 선택하고, 포함되어 있는 경우 구성 파일 사용을 선택합니다.

   > **Note**
   >
   > App Runner 구성 파일은 애플리케이션 소스의 일부로 빌드 구성을 유지하는 방법입니다. 제공하면 App Runner는 파일에서 일부 값을 읽고 콘솔에서 설정하도록 허용하지 않습니다.

   b. 다음 빌드 설정을 제공합니다.

   - 런타임 - 애플리케이션의 특정 관리형 런타임을 선택합니다.
   - 빌드 명령 - 소스 코드에서 애플리케이션을 빌드하는 명령을 입력합니다. 이는 코드와 함께 제공되는 언어별 도구 또는 스크립트일 수 있습니다.
   - 시작 명령 - 웹 서비스를 시작하는 명령을 입력합니다.
   - 포트 - 웹 서비스가 수신 대기하는 IP 포트를 입력합니다.

   c. 다음을 선택합니다.



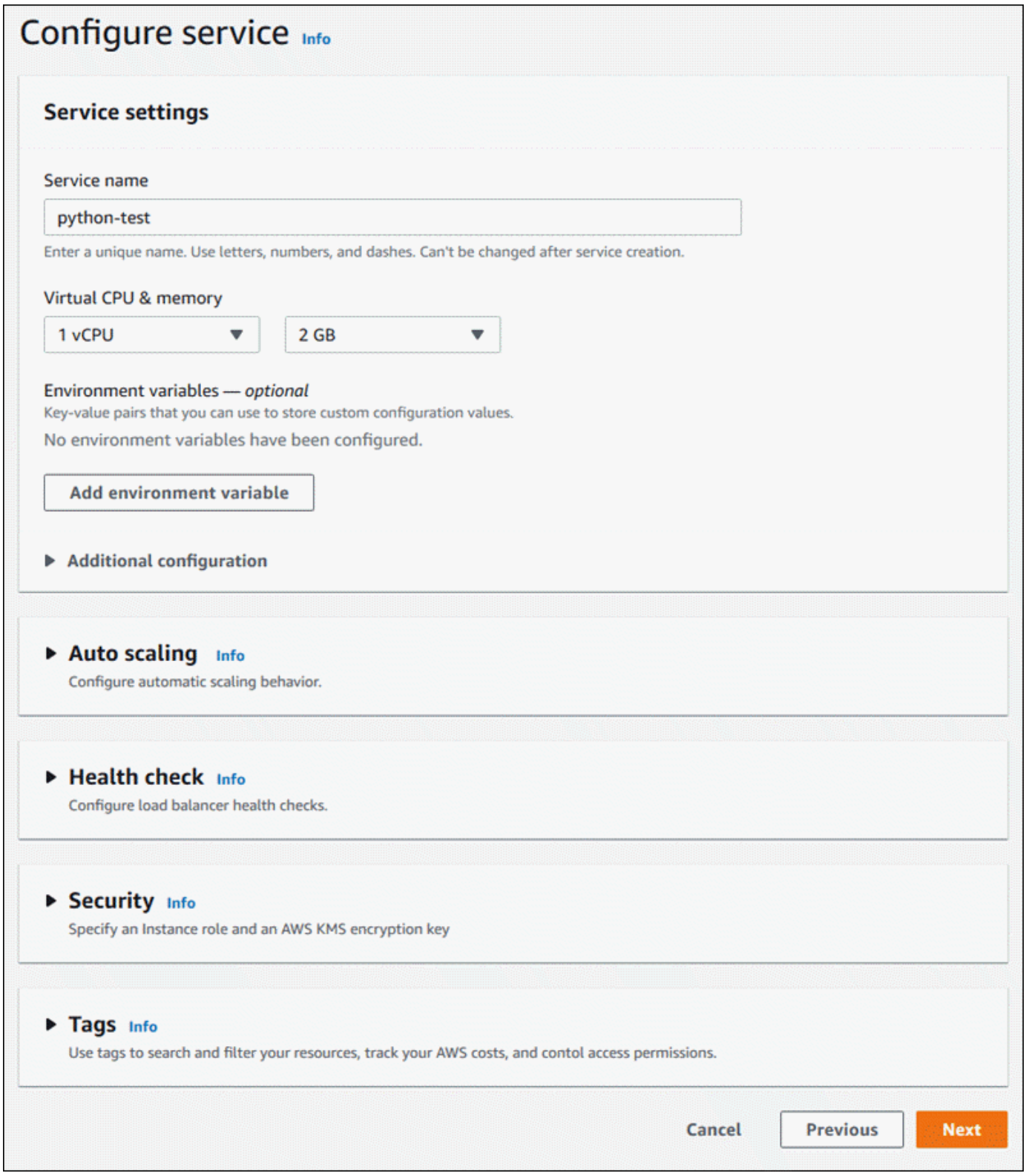
4. 서비스를 구성합니다.

   a. 서비스 구성 페이지의 서비스 설정 섹션에서 서비스 이름을 입력합니다.

   > **Note**
   >
   > 다른 모든 서비스 설정은 선택 사항이거나 콘솔에서 제공하는 기본값입니다.

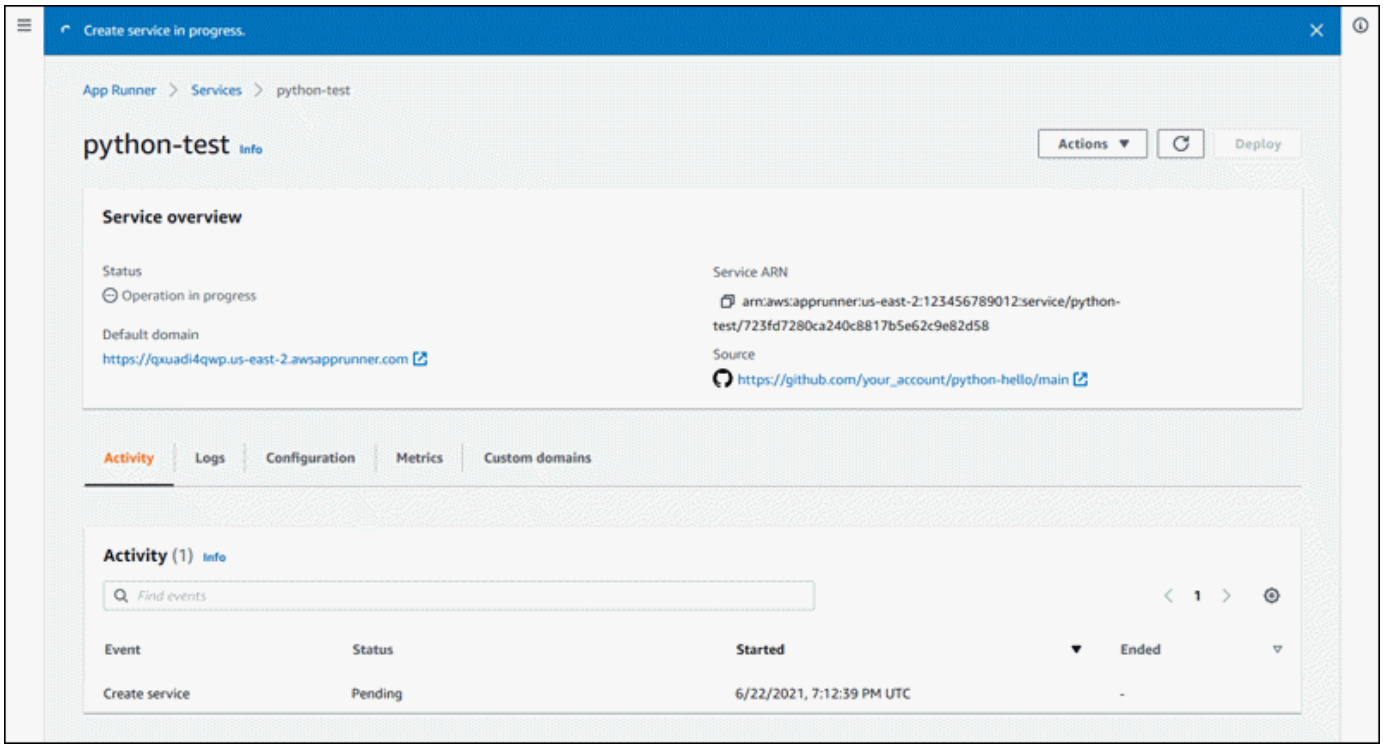
   b. 필요에 따라 애플리케이션 요구 사항에 맞게 다른 설정을 변경하거나 추가할 수 있습니다.

   c. 다음을 선택합니다.



5. 검토 및 생성 페이지에서 입력한 모든 세부 정보를 확인한 다음 생성 및 배포를 선택합니다.

결과: 서비스가 성공적으로 생성되면 콘솔에 새 서비스의 서비스 개요와 함께 서비스 대시보드가 표시됩니다.



6. 서비스가 실행 중인지 확인합니다.

   a. 서비스 대시보드 페이지에서 서비스 상태가 실행 중일 때까지 기다립니다.

   b. 기본 도메인 값을 선택합니다. 서비스 웹 사이트의 URL입니다.

   c. 웹 사이트를 사용하여 제대로 실행되고 있는지 확인합니다.

App Runner API 또는를 사용하여 코드에서 서비스 생성 AWS CLI

App Runner API를 사용하여 서비스를 생성하려면 `CreateService` API 작업을 AWS CLI호출합니다. 자세한 내용과 예제는 CreateService를 참조하세요. 소스 코드 리포지토리(GitHub 또는 Bitbucket)의 특정 조직 또는 계정을 사용하여 서비스를 처음 생성하는 경우 CreateConnection을 호출하여 시작합니다. 이렇게 하면 App Runner와 리포지토리 공급자의 조직 또는 계정 간에 연결이 설정됩니다. App Runner 연결에 대한 자세한 내용은 섹션을 참조하세요the section called “연결”.

호출이를 표시하는 서비스 객체와 함께 성공적인 응답을 반환하면 `"Status": "CREATING"`서비스가 생성을 시작합니다.

예제 호출은 API 참조의 소스 코드 리포지토리 서비스 생성을 참조하세요. AWS App Runner

Amazon ECR 이미지에서 서비스 생성

다음 섹션에서는 소스가 Amazon ECR에 저장된 컨테이너 이미지인 경우 App Runner 서비스를 생성하는 방법을 보여줍니다. Amazon ECR은 입니다 AWS 서비스. 따라서 Amazon ECR 이미지를 기반으로 서비스를 생성하려면 App Runner에 필요한 Amazon ECR 작업 권한이 포함된 액세스 역할을 제공합니다.

> **Note**
>
> Amazon ECR Public에 저장된 이미지는 공개적으로 사용할 수 있습니다. 따라서 이미지가 Amazon ECR Public에 저장된 경우 액세스 역할이 필요하지 않습니다.

서비스가 생성되면 App Runner는 사용자가 제공한 이미지의 컨테이너 인스턴스를 실행하는 서비스를 시작합니다. 이 경우에는 빌드 단계가 없습니다.

자세한 내용은 이미지 기반 서비스 단원을 참조하십시오.

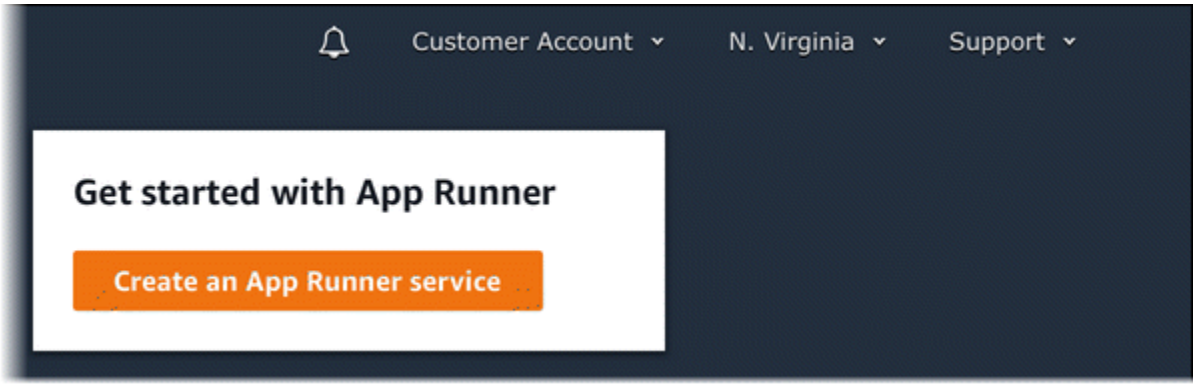
App Runner 콘솔을 사용하여 이미지에서 서비스 생성

콘솔을 사용하여 App Runner 서비스를 생성하려면

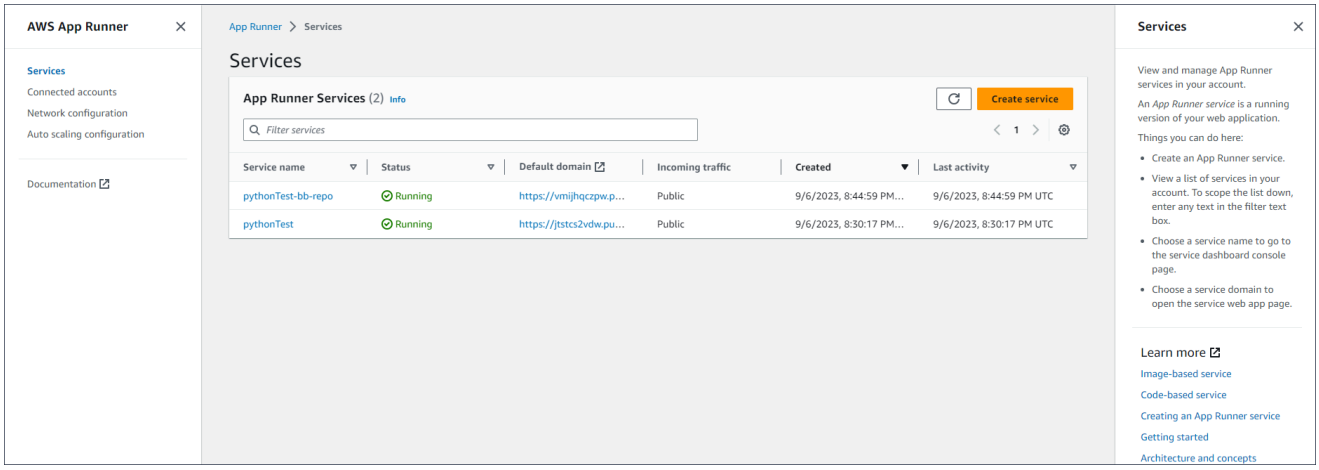
1. 소스 코드를 구성합니다.

   a. App Runner 콘솔을 열고 리전 목록에서를 선택합니다 AWS 리전.

   b. 에 아직 App Runner 서비스가 AWS 계정 없는 경우 콘솔 홈 페이지가 표시됩니다. App Runner 서비스 생성을 선택합니다.



   에 기존 서비스가 AWS 계정 있는 경우 서비스 목록과 함께 서비스 페이지가 표시됩니다. 서비스 생성을 선택합니다.



c. 소스 및 배포 페이지의 소스 섹션의 리포지토리 유형에서 컨테이너 레지스트리를 선택합니다.

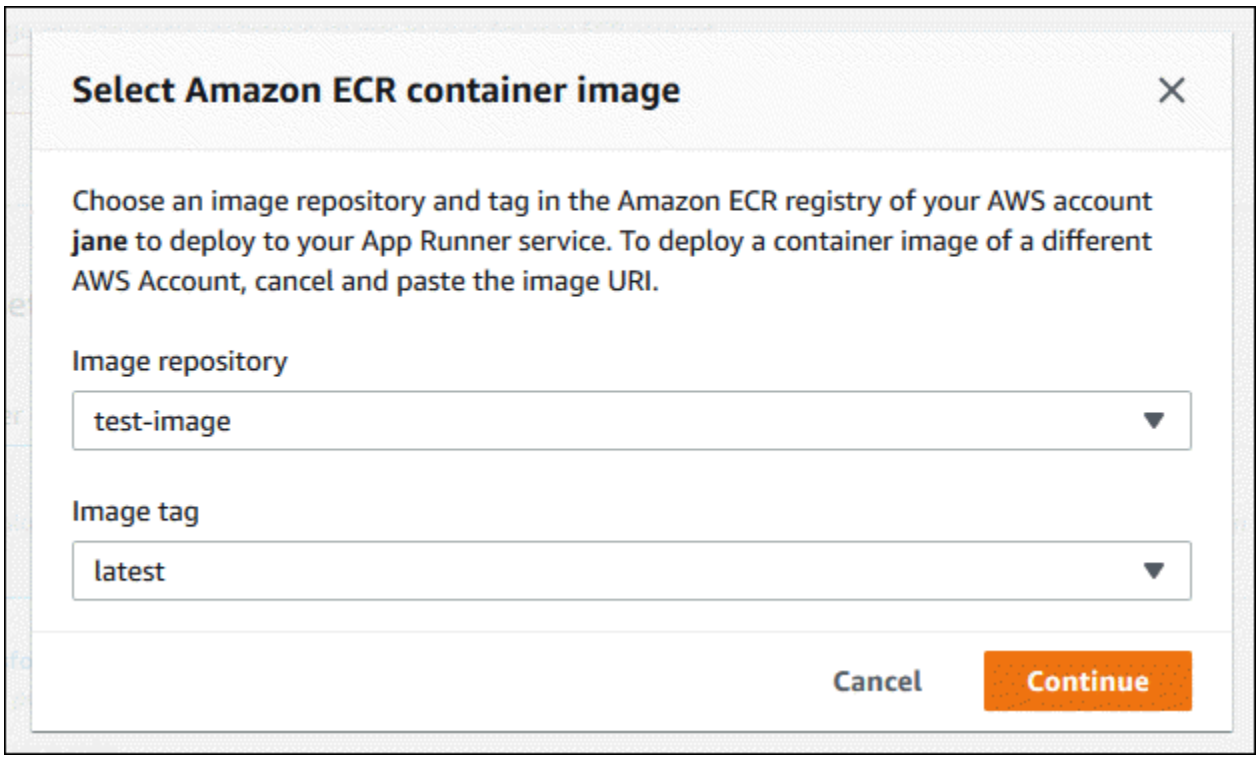
d. 공급자에서 이미지가 저장되는 공급자를 선택합니다.

- Amazon ECR - Amazon ECR에 저장된 프라이빗 이미지입니다.
- Amazon ECR 퍼블릭 - Amazon ECR 퍼블릭에 저장된 공개적으로 읽을 수 있는 이미지입니다.

e. 컨테이너 이미지 URI에서 찾아보기를 선택합니다.

f. Amazon ECR 컨테이너 이미지 선택 대화 상자의 이미지 리포지토리에서 이미지가 포함된 리포지토리를 선택합니다.

g. 이미지 태그에서 배포하려는 특정 이미지 태그(예: 최신)를 선택한 다음 계속을 선택합니다.



2. 배포를 구성합니다.

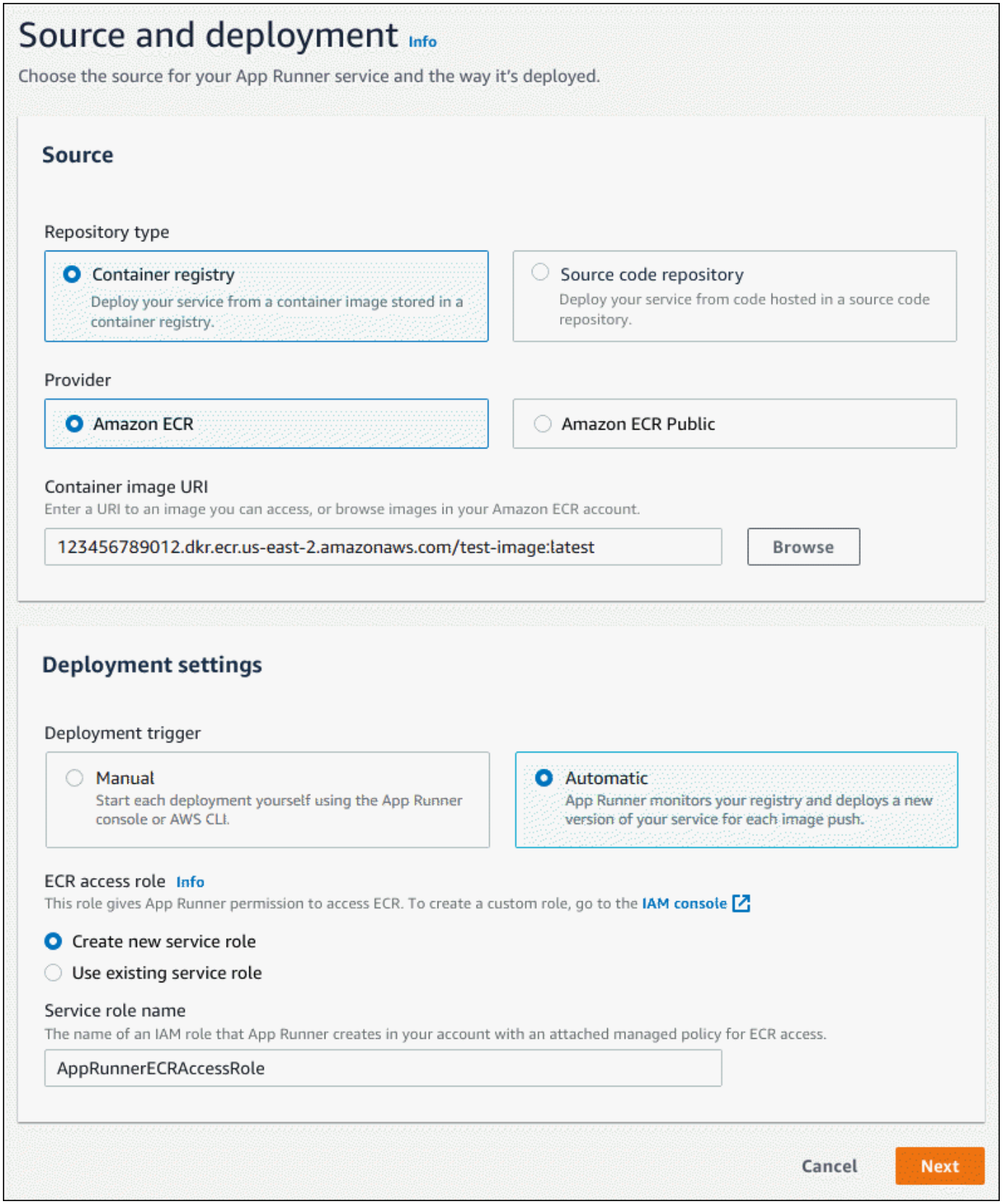
   a. 배포 설정 섹션에서 수동 또는 자동을 선택합니다.

   > **Note**
   >
   > App Runner는 Amazon ECR 퍼블릭 이미지 및 서비스가 속한 계정과 다른 AWS 계정에 속하는 Amazon ECR 리포지토리의 이미지에 대한 자동 배포를 지원하지 않습니다.

   배포 방법에 대한 자세한 내용은 섹션을 참조하세요the section called “배포 방법”.

   b. [Amazon ECR 공급자] ECR 액세스 역할에서 계정의 기존 서비스 역할을 선택하거나 새 역할을 생성하도록 선택합니다. 수동 배포를 사용하는 경우 배포 시 IAM 사용자 역할을 사용하도록 선택할 수도 있습니다.

   c. 다음을 선택합니다.


Source and deployment Info
Choose the source for your App Runner service and the way it's deployed.
Source
Repository type
Container registry
Deploy your service from a container image stored in a container registry.
Source code repository
Deploy your service from code hosted in a source code repository.
Provider
Amazon ECR
Amazon ECR Public
Container image URI
Enter a URI to an image you can access, or browse images in your Amazon ECR account.
123456789012.dkr.ecr.us-east-2.amazonaws.com/test-image:latest
Browse
Deployment settings
Deployment trigger
Manual
Start each deployment yourself using the App Runner console or AWS CLI.
Automatic
App Runner monitors your registry and deploys a new version of your service for each image push.
ECR access role Info
This role gives App Runner permission to access ECR. To create a custom role, go to the IAM console
Create new service role
Use existing service role
Service role name
The name of an IAM role that App Runner creates in your account with an attached managed policy for ECR access.
AppRunnerECRAccessRole
Cancel
Next

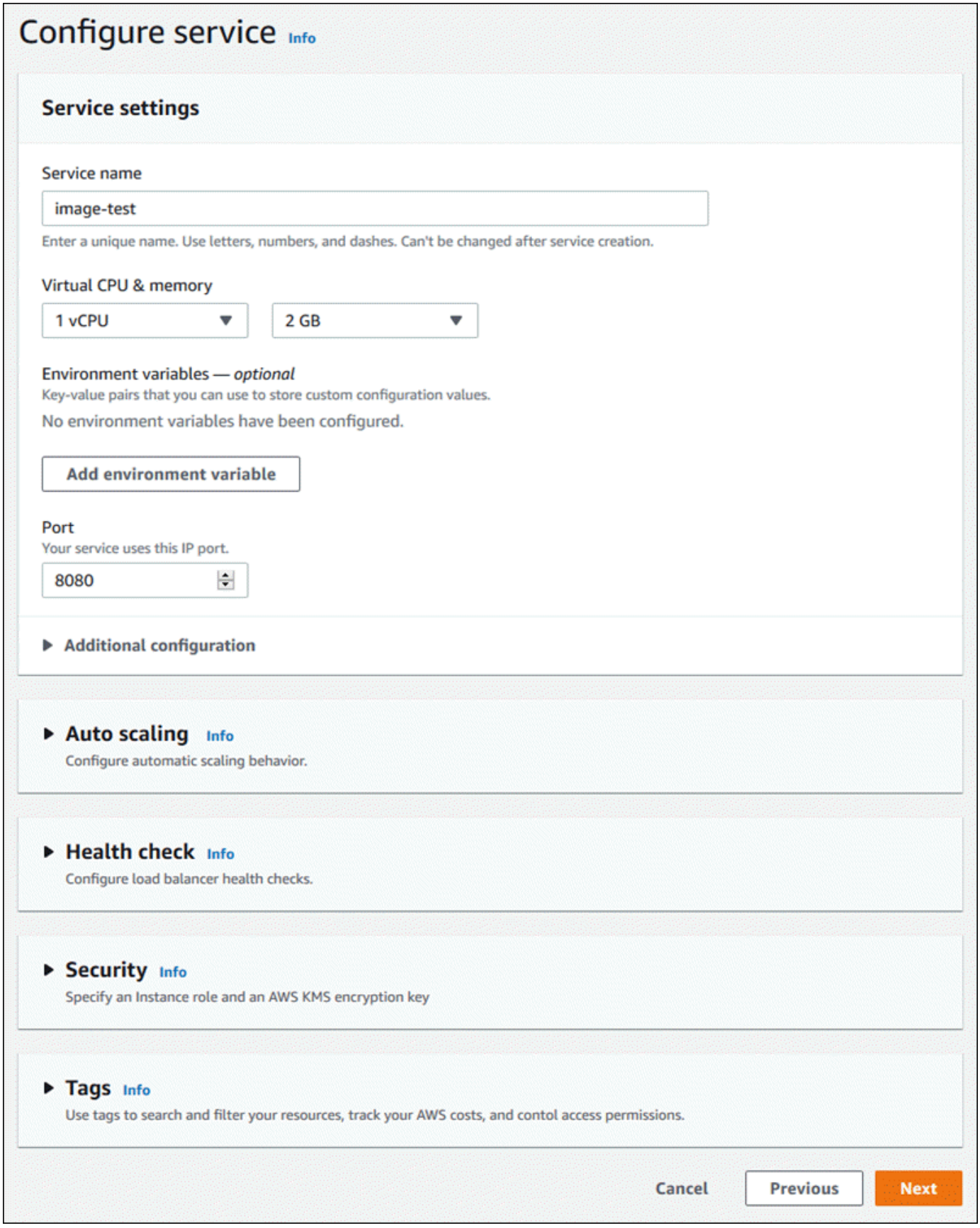
3. 서비스를 구성합니다.

   a. 서비스 구성 페이지의 서비스 설정 섹션에 서비스 이름과 서비스 웹 사이트에서 수신 대기하는 IP 포트를 입력합니다.

   > **Note**
   >
   > 다른 모든 서비스 설정은 선택 사항이거나 콘솔에서 제공하는 기본값입니다.

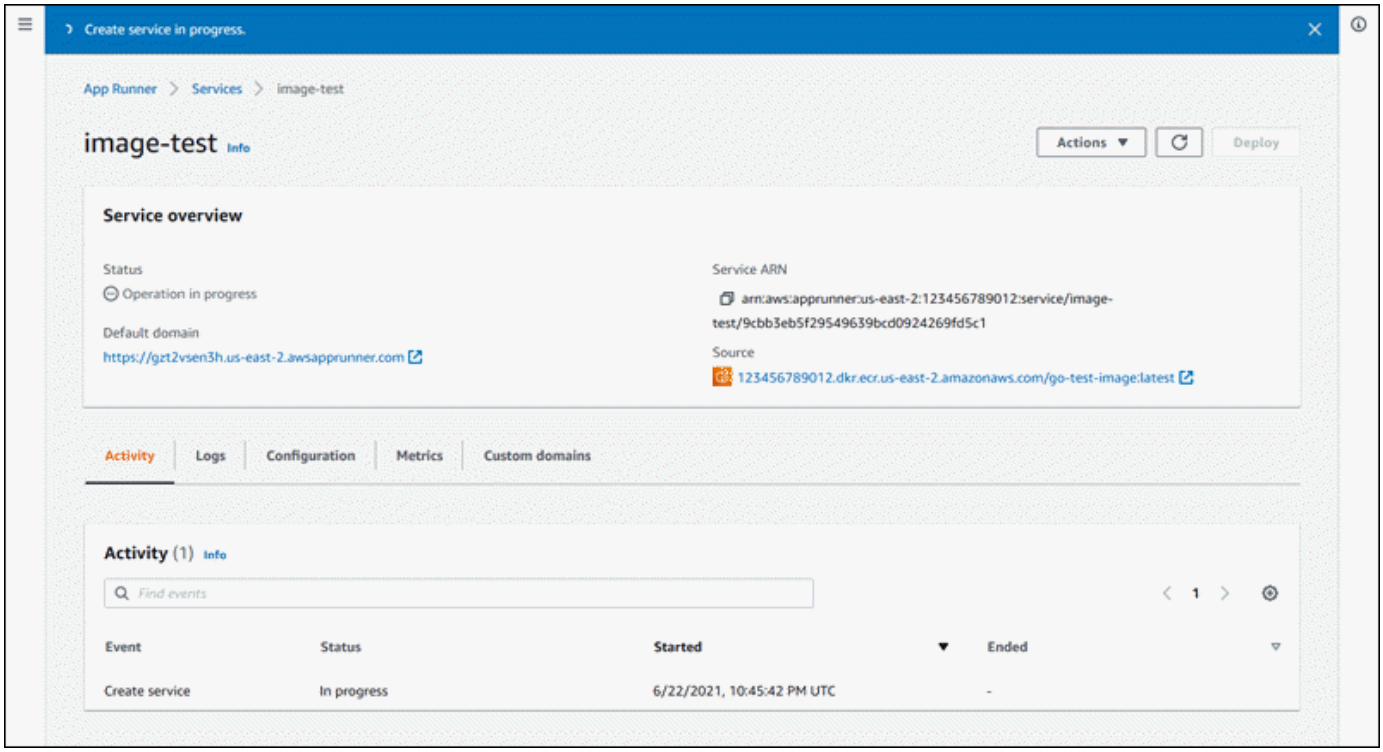
   b. (선택 사항) 애플리케이션의 요구 사항에 맞게 다른 설정을 변경하거나 추가합니다.

   c. 다음을 선택합니다.


Configure service Info
Service settings
Service name
image-test
Enter a unique name. Use letters, numbers, and dashes. Can't be changed after service creation.
Virtual CPU & memory
1 vCPU
2 GB
Environment variables — optional
Key-value pairs that you can use to store custom configuration values.
No environment variables have been configured.
Add environment variable
Port
Your service uses this IP port.
8080
Additional configuration
Auto scaling Info
Configure automatic scaling behavior.
Health check Info
Configure load balancer health checks.
Security Info
Specify an Instance role and an AWS KMS encryption key
Tags Info
Use tags to search and filter your resources, track your AWS costs, and contol access permissions.
Cancel
Previous
Next

4. 검토 및 생성 페이지에서 입력한 모든 세부 정보를 확인한 다음 생성 및 배포를 선택합니다.

   결과: 서비스가 성공적으로 생성되면 콘솔에 새 서비스의 서비스 개요와 함께 서비스 대시보드가 표시됩니다.



5. 서비스가 실행 중인지 확인합니다.

   a. 서비스 대시보드 페이지에서 서비스 상태가 실행 중일 때까지 기다립니다.
   b. 기본 도메인 값을 선택합니다. 서비스 웹 사이트의 URL입니다.
   c. 웹 사이트를 사용하여 제대로 실행되고 있는지 확인합니다.

App Runner API 또는를 사용하여 이미지에서 서비스 생성 AWS CLI

App Runner API를 사용하여 서비스를 생성하려면 CreateService API 작업을 AWS CLI호출합니다.

호출이를 표시하는 서비스 객체와 함께 성공적인 응답을 반환하면 서비스 생성이 시작됩니다"Status": "CREATING".

호출 예제는 API 참조의 소스 이미지 리포지토리 서비스 생성을 참조하세요. AWS App Runner

실패한 App Runner 서비스 재구축

App Runner 서비스를 생성할 때 생성 실패 오류가 표시되면 다음 중 하나를 수행할 수 있습니다.

- 의 단계에 따라 오류의 원인을 the section called “서비스 생성 실패” 식별합니다.
- 소스 또는 구성에서 오류가 발견되면 필요한 사항을 변경한 다음 서비스를 다시 빌드합니다.
- App Runner에서 일시적인 문제로 인해 서비스가 실패하는 경우 소스 또는 구성을 변경하지 않고 실패한 서비스를 다시 빌드합니다.

App Runner 콘솔 또는 App Runner API 또는 AWS CLI를 통해 실패한 서비스를 다시 빌드할 수 있습니다.

App Runner 콘솔을 사용하여 실패한 App Runner 서비스 재구축

Rebuild with updates

서비스 생성은 다양한 이유로 실패할 수 있습니다. 이 경우 서비스를 다시 빌드하기 전에 문제의 근본 원인을 식별하고 수정하는 것이 중요합니다. 자세한 내용은 the section called “서비스 생성 실패” 단원을 참조하십시오.

업데이트를 사용하여 실패한 서비스를 다시 빌드하려면

1. 서비스 페이지의 구성 탭으로 이동하여 편집을 선택합니다.

   페이지에서 모든 업데이트 목록을 표시하는 요약 패널이 열립니다.
2. 필요한 사항을 변경하고 요약 패널에서 검토합니다.
3. 저장 및 재구축을 선택합니다.

   서비스 페이지의 로그 탭에서 진행 상황을 모니터링할 수 있습니다.

Rebuild without updates

일시적인 문제로 인해 서비스 생성이 실패하는 경우 소스 또는 구성 설정을 수정하지 않고 서비스를 다시 빌드할 수 있습니다.

업데이트 없이 실패한 서비스를 다시 빌드하려면

- 서비스 페이지의 오른쪽 상단 모서리에서 재구축을 선택합니다.

  서비스 페이지의 로그 탭에서 진행 상황을 모니터링할 수 있습니다.
- 서비스가 다시 생성되지 않는 경우의 문제 해결 지침을 따릅니다the section called “서비스 생성 실패”. 필요한 사항을 변경한 다음 서비스를 다시 빌드합니다.

App Runner API 또는를 사용하여 실패한 App Runner 서비스 재구축 AWS CLI

Rebuild with updates

실패한 서비스를 다시 빌드하려면:

1. 의 지침에 따라 오류의 원인을 the section called “서비스 생성 실패” 찾습니다.
2. 브랜치 또는 소스 리포지토리의 이미지 또는 오류를 일으킨 구성을 필요에 따라 변경합니다.
3. 새 소스 코드 리포지토리 또는 소스 이미지 리포지토리 파라미터를 사용하여 UpdateService API 작업을 호출하여 재구축합니다. App Runner는 소스 코드 리포지토리에서 최신 커밋을 검색합니다.

Example업데이트로 재구축

다음 예제에서는 이미지 기반 서비스의 소스 구성을 업데이트하고 있습니다. 의 값이 로 `Port` 변경됩니다`80`.

이미지 기반 App Runner 서비스용 `input.json` 파일 업데이트

```
{
  "ServiceArn": "arn:aws:apprunner:us-east-1:123456789012:service/python-
app/8fe1e10304f84fd2b0df550fe98a71fa",
  "SourceConfiguration": {
    "ImageRepository": {
      "ImageConfiguration": {
        "Port": "80"
      }
    }
  }
}
```

`UpdateService` API 작업을 호출합니다.

```
aws apprunner update-service
--cli-input-json file://input.json
```

Rebuild without updates

App Runner API를 사용하여 실패한 서비스를 다시 빌드하려면 서비스의 소스 또는 구성을 변경하지 않고 UpdateService API 작업을 AWS CLI호출합니다. App Runner의 일시적인 문제로 인해 서비스 생성에 실패한 경우에만 업데이트하지 않고 재구축하도록 선택합니다.

App Runner에 새 애플리케이션 버전 배포

에서 서비스를 생성할 때 컨테이너 이미지 또는 소스 리포지토리와 같은 애플리케이션 소스를 AWS App Runner구성합니다. App Runner는 서비스를 실행하기 위해 리소스를 프로비저닝하고 해당 리소스에 애플리케이션을 배포합니다.

이 주제에서는 새 버전을 사용할 수 있을 때 애플리케이션 소스를 App Runner 서비스에 재배포하는 방법을 설명합니다. 이미지 리포지토리의 새 이미지 버전이거나 코드 리포지토리의 새 커밋일 수 있습니다. App Runner는 서비스에 배포하는 두 가지 방법인 자동 및 수동을 제공합니다.

배포 방법

App Runner는 애플리케이션 배포가 시작되는 방식을 제어할 수 있는 다음과 같은 방법을 제공합니다.

자동 배포

서비스에 대한 지속적 통합 및 배포(CI/CD) 동작을 원하는 경우 자동 배포를 사용합니다. App Runner는 이미지 또는 코드 리포지토리의 변경 사항을 모니터링합니다.

이미지 리포지토리 - 이미지 리포지토리에 새 이미지 버전을 푸시하거나 코드 리포지토리에 새 커밋을 푸시할 때마다 App Runner는 추가 작업 없이 서비스에 자동으로 배포합니다.

코드 리포지토리 - 소스 디렉터리를 변경하는 새 커밋을 코드 리포지토리에 푸시할 때마다 App Runner는 전체 리포지토리를 배포합니다. 소스 디렉터리의 변경 사항만 자동 배포를 트리거하므로 소스 디렉터리 위치가 자동 배포 범위에 미치는 영향을 이해하는 것이 중요합니다.

- 최상위 디렉터리(리포지토리 루트) - 서비스를 생성할 때 소스 디렉터리에 대해 설정된 기본값입니다. 소스 디렉터리가이 값으로 설정된 경우 전체 리포지토리가 소스 디렉터리 내에 있음을 의미합니다. 따라서 소스 리포지토리로 푸시하는 모든 커밋은이 경우 배포를 트리거합니다.
- 리포지토리 루트가 아닌 디렉터리 경로(기본값 아님) - 소스 디렉터리 내에서 푸시된 변경 사항만 자동 배포를 트리거하므로 소스 디렉터리에 없는 리포지토리로 푸시된 변경 사항은 자동 배포를 트리거하지 않습니다. 따라서 수동 배포를 사용하여 소스 디렉터리 외부로 푸시하는 변경 사항을 배포해야 합니다.

> **Note**
>
> App Runner는 Amazon ECR 퍼블릭 이미지 및 서비스가 속한 계정과 다른 AWS 계정에 속하는 Amazon ECR 리포지토리의 이미지에 대한 자동 배포를 지원하지 않습니다.

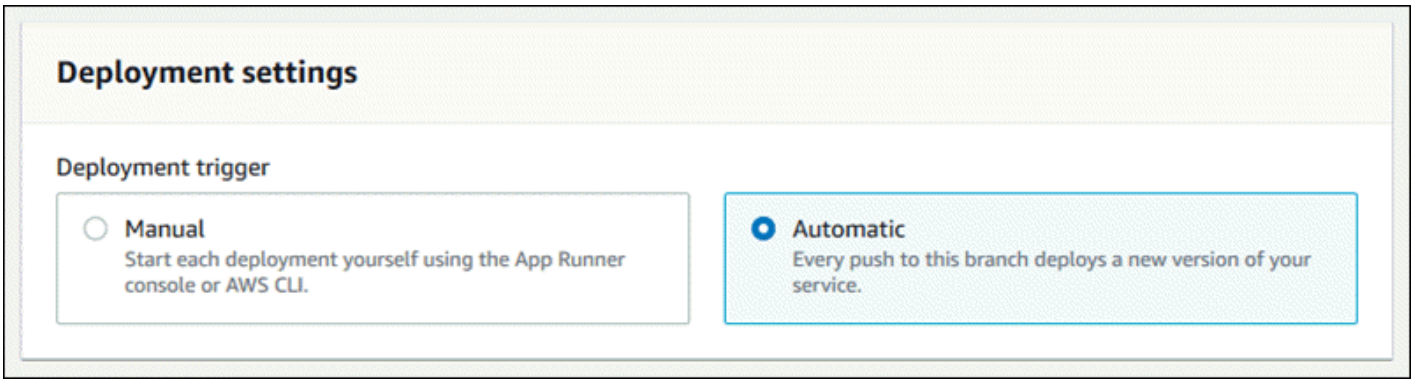
수동 배포

서비스에 대한 각 배포를 명시적으로 시작하려는 경우 수동 배포를 사용합니다. 서비스에 대해 구성한 리포지토리에 배포하려는 새 버전이 있는 경우 배포를 시작합니다. 자세한 내용은 the section called “수동 배포” 단원을 참조하십시오.

> **Note**
>
> 수동 배포를 실행하면 App Runner가 전체 리포지토리에서 소스를 배포합니다.

다음과 같은 방법으로 서비스에 대한 배포 방법을 구성할 수 있습니다.

- 콘솔 - 생성 중인 새 서비스 또는 기존 서비스의 경우 소스 및 배포 구성 페이지의 배포 설정 섹션에서 수동 또는 자동을 선택합니다.



- API 또는 AWS CLI - CreateService 또는 UpdateService 작업을 호출할 때 수동 배포 또는 자동 배포를 `True` 위해 SourceConfiguration 파라미터의 `AutoDeploymentsEnabled` 멤버`False`를 로 설정합니다.

> **자동 및 수동 배포 비교**
>
> 자동 배포와 수동 배포 모두 동일한 결과를 얻습니다. 두 방법 모두 전체 리포지토리를 배포합니다.
> 두 방법의 차이점은 트리거 메커니즘입니다.
>
> - 수동 배포는 콘솔의 배포,에 대한 호출 AWS CLI또는 App Runner API에 대한 호출에 의해 트리거됩니다. 다음 수동 배포 섹션에서는 이에 대한 절차를 제공합니다.
> - 자동 배포는 소스 디렉터리의 내용 내에서 변경하여 트리거됩니다.

수동 배포

수동 배포를 사용하면 서비스에 대한 각 배포를 명시적으로 시작해야 합니다. 새 버전의 애플리케이션 이미지 또는 코드를 배포할 준비가 되면 다음 섹션을 참조하여 콘솔 및 API를 사용하여 배포를 수행하는 방법을 알아볼 수 있습니다.

> **Note**
>
> 수동 배포를 실행하면 App Runner가 전체 리포지토리에서 소스를 배포합니다.

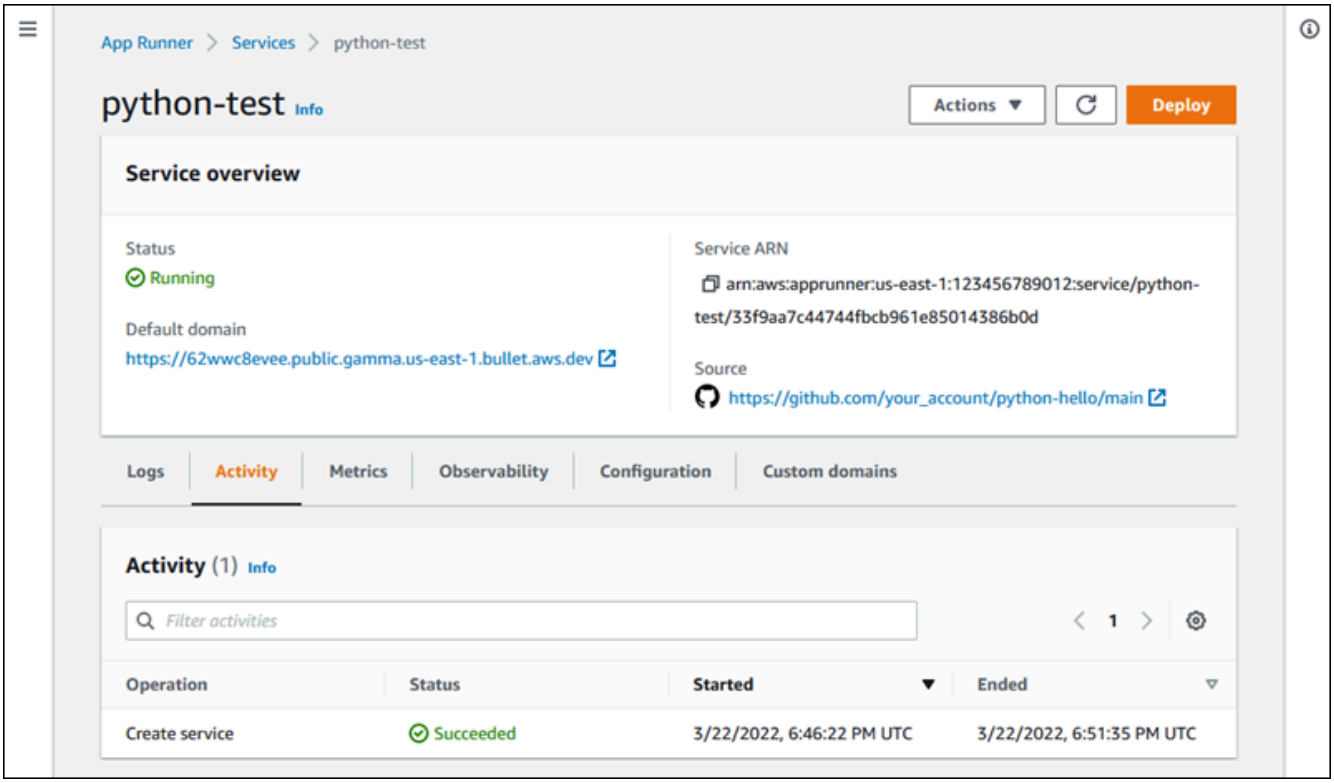
다음 방법 중 하나를 사용하여 애플리케이션 버전을 배포합니다.

App Runner console

App Runner 콘솔을 사용하여 배포하려면

1. App Runner 콘솔을 열고 리전 목록에서를 선택합니다 AWS 리전.
2. 탐색 창에서 서비스를 선택한 다음 App Runner 서비스를 선택합니다.

   콘솔에 서비스 개요와 함께 서비스 대시보드가 표시됩니다.



3. 배포(Deploy)를 선택합니다.

   결과: 새 버전의 배포가 시작됩니다. 서비스 대시보드 페이지에서 서비스 상태가 진행 중인 작업으로 변경됩니다.

4. 배포가 종료될 때까지 기다립니다. 서비스 대시보드 페이지에서 서비스 상태가 다시 실행 중으로 변경되어야 합니다.

5. 배포가 성공했는지 확인하려면 서비스 대시보드 페이지에서 기본 도메인 값인 서비스 웹 사이트의 URL을 선택합니다. 웹 애플리케이션을 검사하거나 상호 작용하고 버전 변경을 확인합니다.

> **Note**
>
> App Runner 애플리케이션의 보안을 강화하기 위해 *.awsapprunner.com 도메인은 퍼블릭 접미사 목록(PSL)에 등록됩니다. 보안을 강화하기 위해 App Runner 애플리케이션의 기본 도메인 이름에 민감한 쿠키를 설정해야 하는 경우 `__Host-` 접두사가 있는 쿠키를 사용하는 것이 좋습니다. 이렇게 쿠키를 설정하면 교차 사이트 요청 위조 시도(CSRF)로부터 도메인을 보호하는 데 도움이 됩니다. 자세한 내용은 Mozilla 개발자 네트워크의 Set-Cookie 페이지를 참조하세요.

App Runner API or AWS CLI

App Runner API를 사용하여 배포하려면 StartDeployment API 작업을 AWS CLI호출합니다. 전달할 유일한 파라미터는 서비스 ARN입니다. 서비스를 생성할 때 애플리케이션 소스 위치를 이미 구성했으며 App Runner에서 새 버전을 찾을 수 있습니다. 호출이 성공적인 응답을 반환하면 배포가 시작됩니다.

App Runner 서비스 구성

AWS App Runner 서비스를 생성할 때 다양한 구성 값을 설정합니다. 서비스를 생성한 후 이러한 구성 설정 중 일부를 변경할 수 있습니다. 다른 설정은 서비스를 생성하는 동안에만 적용할 수 있으며 이후에는 변경할 수 없습니다. 이 주제에서는 App Runner API, App Runner 콘솔 및 App Runner 구성 파일을 사용하여 서비스를 구성하는 방법에 대해 설명합니다.

주제

- App Runner API 또는를 사용하여 서비스 구성 AWS CLI
- App Runner 콘솔을 사용하여 서비스 구성
- App Runner 구성 파일을 사용하여 서비스 구성
- 서비스에 대한 관찰성 구성
- 공유 가능한 리소스를 사용하여 서비스 설정 구성
- 서비스에 대한 상태 확인 구성

App Runner API 또는를 사용하여 서비스 구성 AWS CLI

API는 서비스 생성 후 변경할 수 있는 설정을 정의합니다. 다음 목록에서는 관련 작업, 유형 및 제한 사항에 대해 설명합니다.

- UpdateService 작업 - 생성 후 호출하여 일부 구성 설정을 업데이트할 수 있습니다.
  - 업데이트 가능 - `SourceConfiguration`, `InstanceConfiguration`및 `HealthCheckConfiguration` 파라미터에서 설정을 업데이트할 수 있습니다. 그러나 에서는 소스 유형을 코드에서 이미지로 또는 반대로 전환할 `SourceConfiguration`수 없습니다. 서비스를 생성할 때 제공한 것과 동일한 리포지토리파라미터를 제공해야 합니다. `CodeRepository` 또는 입니다`ImageRepository`.

    서비스와 연결된 별도의 구성 리소스의 다음 ARNs을 업데이트할 수도 있습니다.

    - `AutoScalingConfigurationArn`
    - `VpcConnectorArn`
- 업데이트할 수 없음 - CreateService 작업에서 사용할 수 있는 `ServiceName` 및 `EncryptionConfiguration` 파라미터를 변경할 수 없습니다. 생성 후에는 변경할 수 없습니다. UpdateService 작업에는 이러한 파라미터가 포함되지 않습니다.
- API와 파일 비교 - CodeConfiguration 유형(의 일부로 소스 코드 리포지토리에 사용`SourceConfiguration`됨)의 `ConfigurationSource` 파라미터를 로 설정할 수 있습니다`Repository`. 이 경우 App Runner는의 구성 설정을 무시하고 `CodeConfigurationValues`리포지토리의 구성 파일에서 이러한 설정을 읽습니다. 를 `ConfigurationSource`로 설정하면 `API`App Runner는 API 호출에서 모든 구성 설정을 가져오고 구성 파일이 있더라도 해당 구성 파일을 무시합니다.
- TagResource 작업 - 서비스에 태그를 추가하거나 기존 태그의 값을 업데이트하기 위해 서비스가 생성된 후 호출할 수 있습니다.
- UntagResource 작업 - 서비스에서 태그를 제거하기 위해 서비스가 생성된 후 호출할 수 있습니다.

> **Note**
>
> 서비스에 대한 아웃바운드 트래픽 VPC 커넥터를 생성하는 경우 다음 서비스 시작 프로세스에는 일회성 지연 시간이 발생합니다. 새 서비스를 생성할 때 또는 이후에 서비스 업데이트를 통해이 구성을 설정할 수 있습니다. 자세한 내용은이 가이드일회성 지연 시간의 App Runner를 사용한 네트워킹 장의 섹션을 참조하세요.

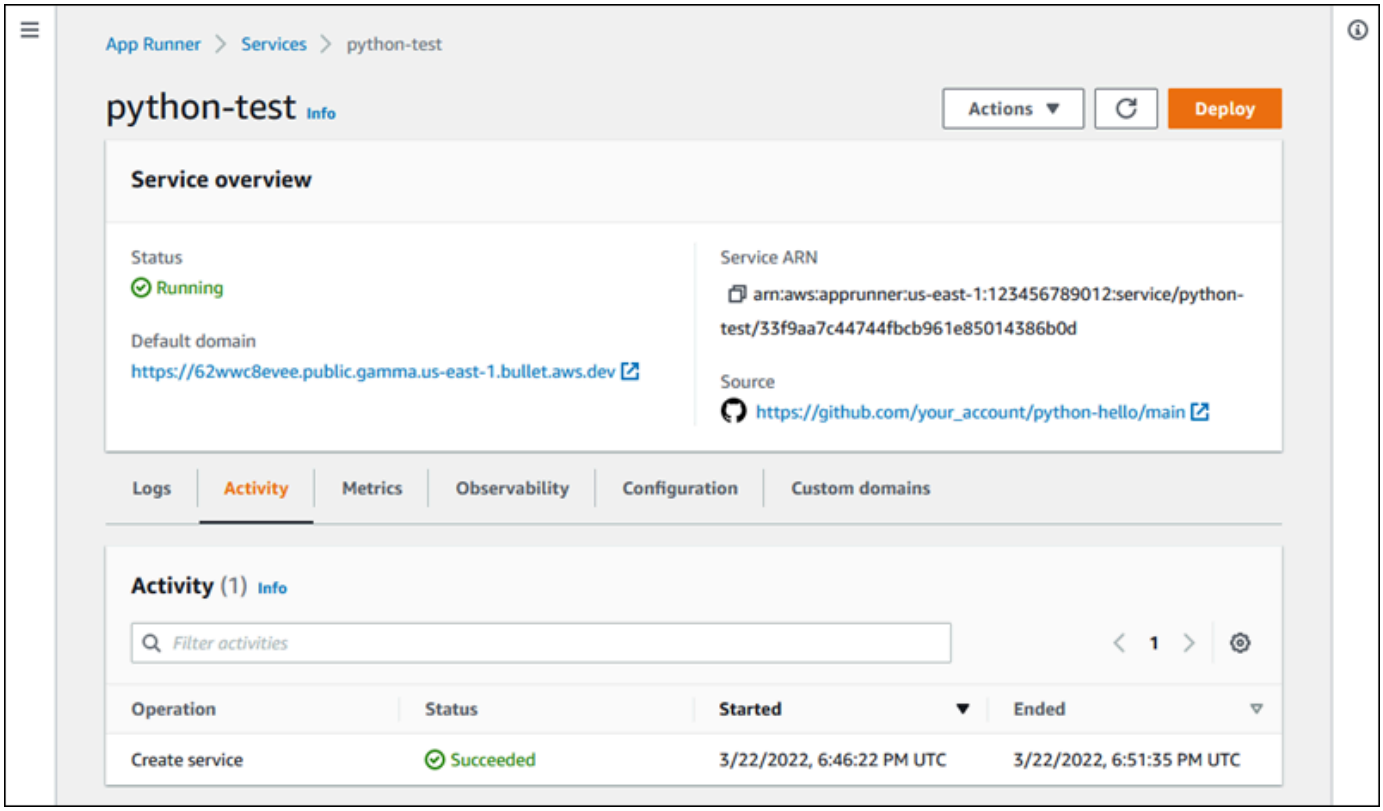
App Runner 콘솔을 사용하여 서비스 구성

콘솔은 App Runner API를 사용하여 구성 업데이트를 적용합니다. 이전 섹션에 정의된 대로 API가 부과하는 업데이트 규칙은 콘솔을 사용하여 구성할 수 있는 항목을 결정합니다. 서비스 생성 중에 사용 가능한 일부 설정은 나중에 수정할 수 없습니다. 또한 구성 파일을 사용하기로 결정하면 콘솔에 추가 설정이 숨겨지고 App Runner가 파일에서 해당 설정을 읽습니다.

서비스를 구성하려면

1. App Runner 콘솔을 열고 리전 목록에서를 선택합니다 AWS 리전.
2. 탐색 창에서 서비스를 선택한 다음 App Runner 서비스를 선택합니다.

   콘솔에 서비스 개요와 함께 서비스 대시보드가 표시됩니다.



3. 서비스 대시보드 페이지에서 구성 탭을 선택합니다.

   결과: 콘솔에는 서비스의 현재 구성 설정이 소스 및 배포, 빌드 구성 및 서비스 구성의 여러 섹션에 표시됩니다.
4. 모든 범주의 설정을 업데이트하려면 편집을 선택합니다.
5. 구성 편집 페이지에서 원하는 대로 변경한 다음 변경 사항 저장을 선택합니다.

> **Note**
>
> 서비스에 대한 아웃바운드 트래픽 VPC 커넥터를 생성하는 경우 다음 서비스 시작 프로세스에는 일회성 지연 시간이 발생합니다. 새 서비스를 생성할 때 또는 이후에 서비스 업데이트를 통해이 구성을 설정할 수 있습니다. 자세한 내용은이 가이드일회성 지연 시간의 App Runner를 사용한 네트워킹 장의 섹션을 참조하세요.

App Runner 구성 파일을 사용하여 서비스 구성

App Runner 서비스를 생성하거나 업데이트할 때 소스 리포지토리의 일부로 제공하는 구성 파일에서 일부 구성 설정을 읽도록 App Runner에 지시할 수 있습니다. 이렇게 하면 소스 제어에서 소스 코드와

관련된 설정을 코드 자체와 함께 관리할 수 있습니다. 구성 파일은 콘솔 또는 API를 사용하여 설정할 수 없는 특정 고급 설정도 제공합니다. 자세한 내용은 App Runner 구성 파일 단원을 참조하십시오.

> **Note**
>
> 서비스에 대한 아웃바운드 트래픽 VPC 커넥터를 생성하는 경우 다음 서비스 시작 프로세스에는 일회성 지연 시간이 발생합니다. 새 서비스를 생성할 때 또는 이후에 서비스 업데이트를 통해이 구성을 설정할 수 있습니다. 자세한 내용은이 가이드일회성 지연 시간의 App Runner를 사용한 네트워킹 장의 섹션을 참조하세요.

서비스에 대한 관찰성 구성

AWS App Runner 는 여러 AWS 서비스와 통합되어 App Runner 서비스를 위한 광범위한 관찰성 도구 모음을 제공합니다. 자세한 내용은 관찰성 단원을 참조하십시오.

App Runner는 ObservabilityConfiguration 기능을 활성화하고 동작을 구성할 수 있도록 지원합니다. 서비스를 생성하거나 업데이트할 때 관찰성 구성 리소스를 제공할 수 있습니다. App Runner 콘솔은 새 App Runner 서비스를 생성할 때 하나를 생성합니다. 관찰성 구성을 제공하는 것은 선택 사항입니다. 제공하지 않으면 App Runner는 기본 관찰성 구성을 제공합니다.

여러 App Runner 서비스에서 단일 관찰성 구성을 공유하여 관찰성 동작이 동일한지 확인할 수 있습니다. 자세한 내용은 the section called “구성 리소스” 단원을 참조하십시오.

관찰성 구성을 사용하여 다음 관찰성 기능을 구성할 수 있습니다.

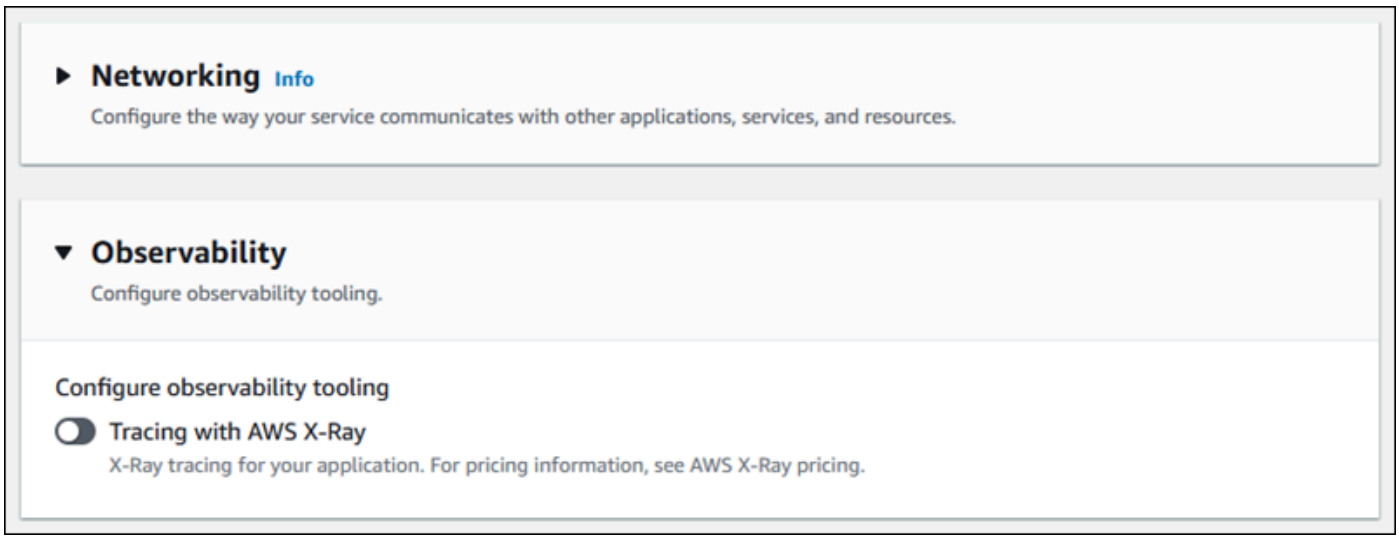
- 추적 구성 - 애플리케이션이 처리하는 요청 및 애플리케이션이 수행하는 다운스트림 호출을 추적하기 위한 설정입니다. 추적에 대한 자세한 내용은 the section called “추적(X-Ray)”의 내용을 참조하세요.

관찰성 관리

다음 방법 중 하나를 사용하여 App Runner 서비스의 관찰성을 관리합니다.

App Runner console

App Runner 콘솔을 사용하여 서비스를 생성하거나 나중에 구성을 업데이트할 때 서비스에 대한 관찰성 기능을 구성할 수 있습니다. 콘솔 페이지에서 관찰성 구성 섹션을 찾습니다.



App Runner API or AWS CLI

CreateService 또는 UpdateService App Runner API 작업을 호출할 때 `ObservabilityConfiguration` 파라미터 객체를 사용하여 관찰성 기능을 활성화하고 서비스에 대한 관찰성 구성 리소스를 지정할 수 있습니다.

다음 App Runner API 작업을 사용하여 관찰성 구성 리소스를 관리합니다.

- CreateObservabilityConfiguration - 새 관찰성 구성 또는 기존 구성에 대한 개정을 생성합니다.
- ListObservabilityConfigurations -와 연결된 관찰성 구성 목록을 요약 정보와 AWS 계정함께 반환합니다.
- DescribeObservabilityConfiguration – 관찰성 구성에 대한 전체 설명을 반환합니다.
- DeleteObservabilityConfiguration – 관찰성 구성을 삭제합니다. 특정 개정 또는 최신 활성 개정을 삭제할 수 있습니다. 에 대한 관찰성 구성 할당량에 도달하면 불필요한 관찰성 구성을 삭제해야 할 수 있습니다 AWS 계정.

공유 가능한 리소스를 사용하여 서비스 설정 구성

일부 기능의 경우 AWS App Runner 서비스 간에 구성을 공유하는 것이 좋습니다. 예를 들어 서비스 세트가 동일한 Auto Scaling 동작을 갖도록 할 수 있습니다. 또는 모든 서비스에 대해 동일한 관찰성 설정을 원할 수 있습니다. App Runner를 사용하면 별도의 공유 가능한 리소스를 사용하여 설정을 공유할 수 있습니다. 기능에 대한 구성 설정 세트를 정의하는 리소스를 생성한 다음이 구성 리소스의 Amazon 리소스 이름(ARN)을 하나 이상의 App Runner 서비스에 제공합니다.

App Runner는 다음 기능에 대해 공유 가능한 구성 리소스를 구현합니다.

- Auto Scaling
- Observability
- VPC 액세스

이러한 각 기능에 대한 문서 페이지에서는 사용 가능한 설정 및 관리 절차에 대한 정보를 제공합니다.

별도의 구성 리소스를 사용하는 기능은 몇 가지 설계 특성과 고려 사항을 공유합니다.

- 개정 - 일부 구성 리소스에는 개정이 있을 수 있습니다. Auto Scaling 및 관찰성은 개정을 사용하는 두 가지 구성 리소스의 예입니다. 이러한 경우 각 구성에는 이름과 숫자 개정이 있습니다. 구성의 여러 개정은 동일한 이름과 다른 개정 번호를 갖습니다. 시나리오마다 다른 구성 이름을 사용할 수 있습니다. 각 이름에 대해 여러 개정을 추가하여 특정 시나리오에 대한 설정을 미세 조정할 수 있습니다.

  이름으로 생성한 첫 번째 구성은 개정 번호 1을 가져옵니다. 동일한 이름의 후속 구성은 연속된 개정 번호(2로 시작)를 가져옵니다. App Runner 서비스를 특정 구성 개정 또는 최신 구성 개정과 연결할 수 있습니다.
- 공유 - 여러 App Runner 서비스에서 단일 구성 리소스를 공유할 수 있습니다. 이는 이러한 서비스에서 동일한 구성을 유지하려는 경우에 유용합니다. 특히 리소스가 개정을 지원하는 경우 구성의 최신 개정을 사용하도록 여러 서비스를 구성할 수 있습니다. 이렇게 하려면 구성 이름만 지정하고 개정은 지정할 수 없습니다. 이렇게 구성한 모든 서비스는 서비스를 업데이트할 때 구성 업데이트를 수신합니다. 구성 변경에 대한 자세한 내용은 섹션을 참조하세요the section called “구성”.
- 리소스 관리 - App Runner를 사용하여 구성을 생성하고 삭제할 수 있습니다. 구성을 직접 업데이트할 수는 없습니다. 대신 개정을 지원하는 리소스의 경우 기존 구성 이름에 대한 새 개정을 생성하여 구성을 효과적으로 업데이트할 수 있습니다.

  > **Note**
  >
  > Auto Scaling의 경우 App Runner 콘솔과 App Runner API를 모두 사용하여 구성과 여러 개정을 생성할 수 있습니다. App Runner 콘솔과 App Runner API 모두 구성 및 개정을 삭제할 수도 있습니다. 자세한 내용은 App Runner 자동 조정 관리 섹션을 참조하세요.
  > 관찰성 구성과 같은 다른 구성 유형의 경우 App Runner 콘솔을 사용하여 단일 개정으로만 구성을 생성할 수 있습니다. 더 많은 개정을 생성하고 구성을 삭제하려면 App Runner API를 사용해야 합니다.
- 리소스 할당량 - 각의 구성 리소스에 대해 가질 수 있는 고유한 구성 이름 및 개정의 수에 대해 설정된 할당량이 있습니다 AWS 리전. 이러한 할당량에 도달하면 구성 이름이나 수정 사항 중 일부를 삭

제해야 더 생성할 수 있습니다. Auto Scaling 구성 개정의 경우 App Runner 콘솔 또는 App Runner API를 사용하여 삭제할 수 있습니다. 자세한 내용은 App Runner 자동 조정 관리 섹션을 참조하세요. App Runner API를 사용하여 다른 리소스를 삭제해야 합니다. 할당량에 대한 자세한 내용은 the section called “App Runner 리소스 할당량” 섹션을 참조하세요.

- 리소스 비용 없음 - 구성 리소스를 생성하는 데 추가 비용이 발생하지 않습니다. 기능 자체에 대한 비용이 발생할 수 있지만(예: X-Ray 추적을 켤 때 일반 AWS X-Ray 비용이 청구됨) App Runner 서비스에 대한 기능을 구성하는 App Runner 구성 리소스에 대해서는 청구되지 않습니다.

서비스에 대한 상태 확인 구성

AWS App Runner 는 상태 확인을 수행하여 서비스의 상태를 모니터링합니다. 기본 상태 확인 프로토콜은 TCP입니다. App Runner는 서비스에 할당된 도메인을 ping합니다. 또는 상태 확인 프로토콜을 HTTP로 설정할 수 있습니다. App Runner는 웹 애플리케이션에 상태 확인 HTTP 요청을 보냅니다.

상태 확인과 관련된 몇 가지 설정을 구성할 수 있습니다. 다음 표에서는 상태 확인 설정과 기본값을 설명합니다.

설정	설명	기본값
프로토콜	App Runner가 서비스에 대한 상태 확인을 수행하는 데 사용하는 IP 프로토콜입니다. 프로토콜을 로 설정하면 TCPApp Runner는 애플리케이션이 수신하는 포트에서 서비스에 할당된 기본 도메인을 ping합니다. 프로토콜을 로 설정하면 HTTPApp Runner는 구성된 경로로 상태 확인 요청을 보냅니다.	TCP
경로	App Runner가 HTTP 상태 확인 요청을 보내는 URL입니다. HTTP 검사에만 적용됩니다.	/
간격	상태 확인 간격(초)입니다.	5
제한 시간	상태 확인 응답이 실패했다고 결정하기 전에 대기하는 시간(초)입니다.	2
정상 임계값	App Runner가 서비스가 정상이라고 판단하기 전에 성공해야 하는 연속 확인 횟수입니다.	1

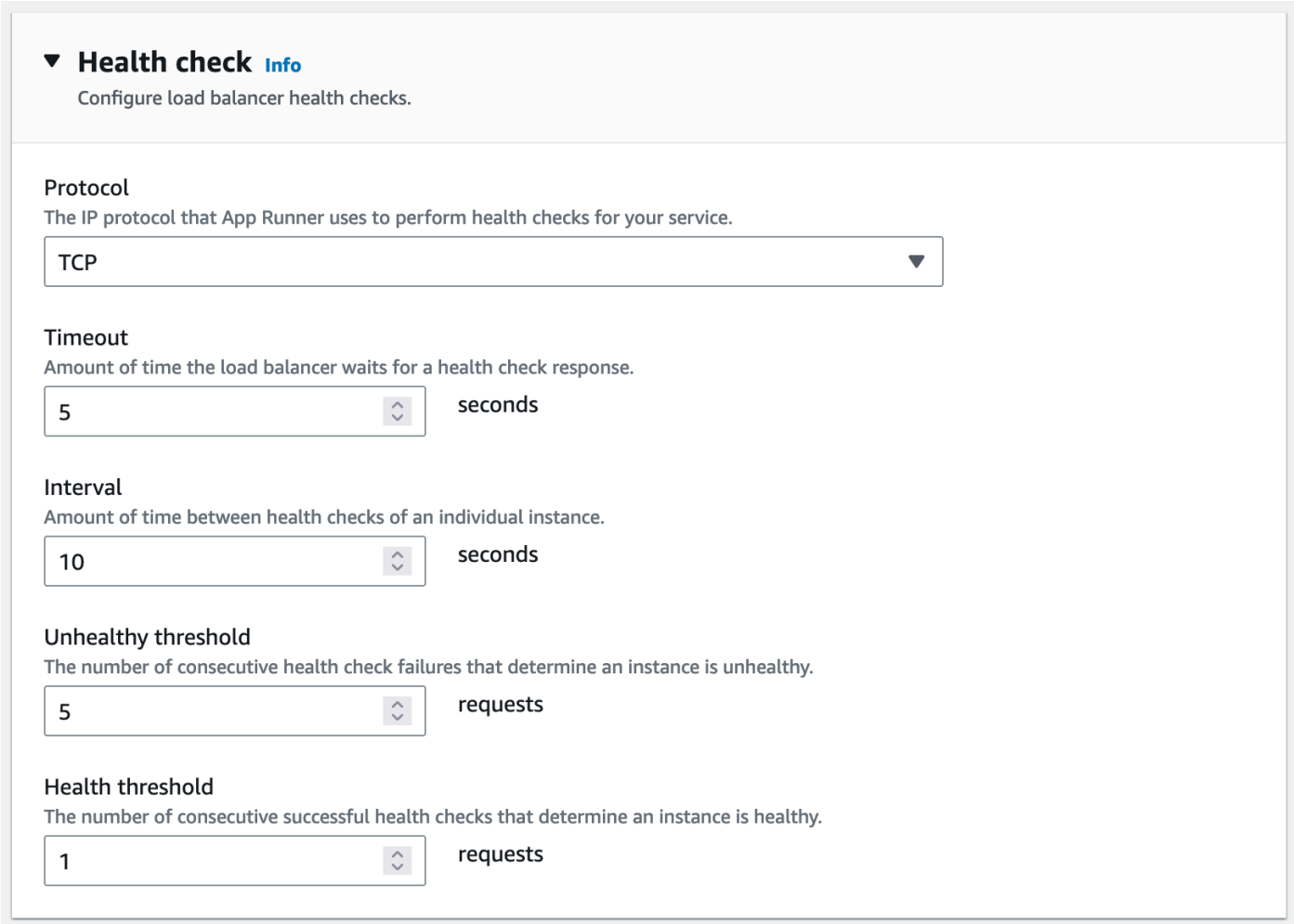
설정	설명	기본값
비정상 임계값	App Runner가 서비스가 비정상이라고 판단하기 전에 실패해야 하는 연속 확인 횟수입니다.	5

상태 확인 구성

다음 방법 중 하나를 사용하여 App Runner 서비스에 대한 상태 확인을 구성합니다.

App Runner console

App Runner 콘솔을 사용하여 App Runner 서비스를 생성하거나 나중에 구성을 업데이트할 때 상태 확인 설정을 구성할 수 있습니다. 전체 콘솔 절차는 the section called “만들기” 및 단원을 참조하십시오the section called “구성”. 두 경우 모두 콘솔 페이지에서 상태 확인 구성 섹션을 찾습니다.



App Runner API or AWS CLI

CreateService 또는 UpdateService API 작업을 호출할 때 `HealthCheckConfiguration` 파라미터를 사용하여 상태 확인 설정을 지정할 수 있습니다.

파라미터의 구조에 대한 자세한 내용은 API 참조의 HealthCheckConfiguration을 참조하세요. AWS App Runner

App Runner 연결 관리

에서 서비스를 생성할 때 컨테이너 이미지 또는 공급자와 함께 저장된 소스 리포지토리와 같은 애플리케이션 소스를 AWS App Runner구성합니다. App Runner는 공급자와의 인증되고 승인된 연결을 설정해야 합니다. 그런 다음 App Runner는 리포지토리를 읽고 서비스에 배포할 수 있습니다. App Runner는에 저장된 코드에 액세스하는 서비스를 생성할 때 연결 설정이 필요하지 않습니다 AWS 계정.

App Runner는 연결이라는 리소스에 연결 정보를 유지합니다. App Runner 콘솔과이 안내서에서는 연결도 연결된 계정이라고 합니다. App Runner는 타사 연결 정보가 필요한 서비스를 생성할 때 연결 리소스가 필요합니다. 다음은 연결에 대한 몇 가지 중요한 정보입니다.

- 공급자 - App Runner에는 현재 GitHub 또는 Bitbucket과의 연결 리소스가 필요합니다.
- 공유 - 연결 리소스를 사용하여 동일한 리포지토리 공급자 계정을 사용하는 여러 App Runner 서비스를 생성할 수 있습니다.
- 리소스 관리 - App Runner에서 연결을 생성하고 삭제할 수 있습니다. 그러나 기존 연결은 수정할 수 없습니다.
- 리소스 할당량 - 연결 리소스에는 각 AWS 계정 의와 연결된 설정된 할당량이 있습니다 AWS 리전. 이 할당량에 도달하면 새 공급자 계정에 연결하기 전에 연결을 삭제해야 할 수 있습니다. 다음 섹션에 설명된 대로 App Runner 콘솔 또는 API를 사용하여 연결을 삭제할 수 있습니다the section called “연결 관리”. 자세한 내용은 the section called “App Runner 리소스 할당량” 단원을 참조하십시오.

연결 관리

다음 방법 중 하나를 사용하여 App Runner 연결을 관리합니다.

App Runner console

App Runner 콘솔을 사용하여 서비스를 생성할 때 연결 세부 정보를 제공합니다. 연결 리소스를 명시적으로 생성할 필요는 없습니다. 콘솔에서 이전에 연결한 GitHub 또는 Bitbucket 계정에 연결하

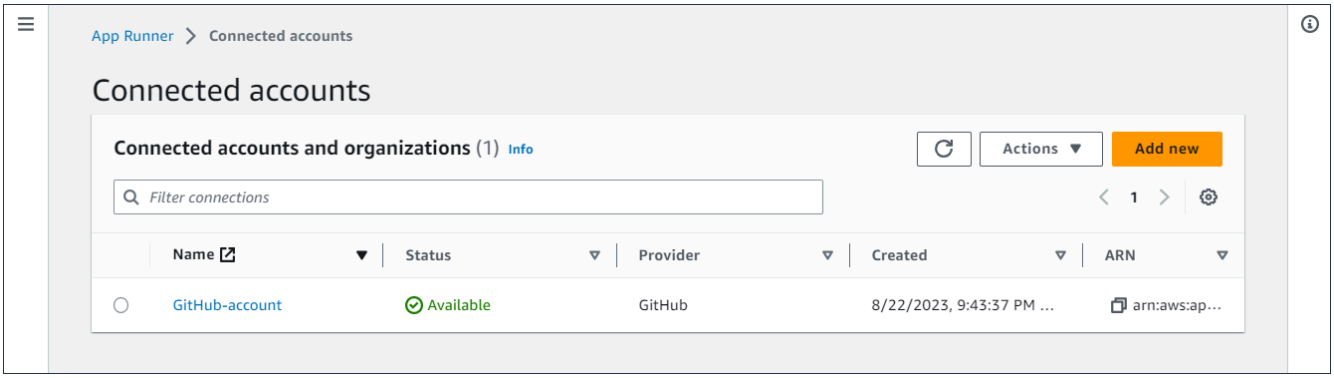
거나 새 계정에 연결하도록 선택할 수 있습니다. 필요한 경우 App Runner는 연결 리소스를 생성합니다. 새 연결의 경우 일부 공급자는 연결을 사용하기 전에 인증 핸드셰이크를 완료해야 합니다. 콘솔에서이 프로세스를 안내합니다.

콘솔에는 기존 연결을 관리하기 위한 페이지도 있습니다. 서비스를 생성할 때 인증 핸드셰이크를 수행하지 않은 경우 연결에 대한 인증 핸드셰이크를 완료할 수 있습니다. 더 이상 사용하지 않는 연결을 삭제할 수도 있습니다. 다음 절차에서는 리포지토리 공급자 연결을 관리하는 방법을 보여줍니다.

계정에서 연결을 관리하려면

1. App Runner 콘솔을 열고 리전 목록에서를 선택합니다 AWS 리전.
2. 탐색 창에서 연결된 계정을 선택합니다.

   그러면 콘솔에 계정의 리포지토리 공급자 연결 목록이 표시됩니다.



3. 이제 목록에 있는 연결로 다음 작업 중 하나를 수행할 수 있습니다.

   - GitHub/Bitbucket 계정 또는 조직 열기 - 연결 이름을 선택합니다.
   - 인증 핸드셰이크 완료 - 연결을 선택한 다음 작업 메뉴에서 핸드셰이크 완료를 선택합니다. 콘솔은 인증 핸드셰이크 프로세스를 안내합니다.
   - 연결 삭제 - 연결을 선택한 다음 작업 메뉴에서 삭제를 선택합니다. 삭제 프롬프트의 지침을 따릅니다.

App Runner API or AWS CLI

다음 App Runner API 작업을 사용하여 연결을 관리할 수 있습니다.

- CreateConnection - 리포지토리 공급자 계정에 대한 연결을 생성합니다. 연결이 생성된 후에는 App Runner 콘솔을 사용하여 인증 핸드셰이크를 수동으로 완료해야 합니다. 이 프로세스는 이전 단원에서 설명합니다.

- ListConnections -와 연결된 App Runner 연결 목록을 반환합니다 AWS 계정.
- DeleteConnection - 연결을 삭제합니다. 의 연결 할당량에 도달하면 불필요한 연결을 삭제해야 할 수 있습니다 AWS 계정.

App Runner 자동 조정 관리

AWS App Runner 는 App Runner 애플리케이션에 맞게 컴퓨팅 리소스, 특히 인스턴스를 자동으로 확장 또는 축소합니다. 자동 조정은 트래픽이 많을 때 적절한 요청 처리를 제공하고 트래픽이 느려질 때 비용을 절감합니다.

오토 스케일링 구성

서비스의 Auto Scaling 동작을 조정하도록 몇 가지 파라미터를 구성할 수 있습니다. App Runner는 AutoScalingConfiguration Scaling 설정을 유지합니다. 서비스에 할당하기 전에 독립 실행형 Auto Scaling 구성을 생성하고 유지 관리할 수 있습니다. 서비스에 연결한 후에도 구성을 계속 유지할 수 있습니다. 새 서비스를 생성하거나 기존 서비스를 구성하는 동안 새 Auto Scaling 구성을 생성하도록 선택할 수도 있습니다. 새 Auto Scaling 구성이 생성되면 서비스에 연결하고 서비스 생성 또는 구성 프로세스를 계속할 수 있습니다.

이름 지정 및 개정

Auto Scaling 구성에는 이름과 숫자 개정이 있습니다. 구성의 여러 개정은 동일한 이름과 다른 개정 번호를 갖습니다. 고가용성 또는 저비용과 같은 다양한 Auto Scaling 시나리오에 대해 다양한 구성 이름을 사용할 수 있습니다. 각 이름에 대해 여러 개정을 추가하여 특정 시나리오에 대한 설정을 미세 조정할 수 있습니다. 각 구성에 대해 최대 10개의 고유한 Auto Scaling 구성 이름과 최대 5개의 개정을 가질 수 있습니다. 한도에 도달하여 더 생성해야 하는 경우 하나를 삭제한 다음 다른 하나를 생성할 수 있습니다. App Runner는 기본값으로 설정되거나 활성 서비스에서 사용 중인 구성을 삭제할 수 없습니다. 할당량에 대한 자세한 내용은 the section called “App Runner 리소스 할당량” 섹션을 참조하세요.

기본 구성 설정

App Runner 서비스를 생성하거나 업데이트할 때 Auto Scaling 구성 리소스를 제공할 수 있습니다. Auto Scaling 구성을 제공하는 것은 선택 사항입니다. 제공하지 않으면 App Runner는 권장 값이 포함된 기본 Auto Scaling 구성을 제공합니다. Auto Scaling 구성 기능은 App Runner에서 제공하는 기본 설정을 사용하는 대신 자체 기본 Auto Scaling 구성을 설정하는 옵션을 제공합니다. 다른 Auto Scaling 구성을 기본값으로 지정하면 해당 구성이 나중에 생성하는 새 서비스에 기본값으로 자동 할당됩니다. 새 기본 지정은 기존 서비스에 대해 이전에 설정된 연결에는 영향을 주지 않습니다.

Auto Scaling을 사용하여 서비스 구성

여러 App Runner 서비스에서 단일 Auto Scaling 구성을 공유하여 서비스가 동일한 Auto Scaling 동작을 갖도록 할 수 있습니다. App Runner 콘솔 또는 App Runner API를 사용하여 Auto Scaling 구성을 구성하는 방법에 대한 자세한 내용은이 주제의 다음 섹션을 참조하세요. 공유 가능한 리소스에 대한 자세한 내용은 섹션을 참조하세요the section called “구성 리소스”.

구성 가능한 설정

다음과 같은 Auto Scaling 설정을 구성할 수 있습니다.

- 최대 동시성 - 인스턴스가 처리하는 최대 동시 요청 수입니다. 동시 요청 수가이 할당량을 초과하면 App Runner가 서비스를 확장합니다.
- 최대 크기 - 서비스가 확장할 수 있는 최대 인스턴스 수입니다. 이는 서비스 트래픽을 동시에 처리할 수 있는 가장 많은 수의 인스턴스입니다.
- 최소 크기 - App Runner가 서비스에 프로비저닝할 수 있는 최소 인스턴스 수입니다. 이 서비스에는 항상이 수 이상의 프로비저닝된 인스턴스가 있습니다. 이러한 인스턴스 중 일부는 트래픽을 적극적으로 처리합니다. 나머지는 빠르게 활성화할 준비가 된 비용 효율적인 컴퓨팅 용량 예약의 일부입니다. 프로비저닝된 모든 인스턴스의 메모리 사용량에 대해 비용을 지불합니다. 활성 하위 집합의 CPU 사용량에 대해서만 비용을 지불합니다.

> **Note**
>
> vCPU 리소스 수는 App Runner가 서비스에 제공할 수 있는 인스턴스 수를 결정합니다. 서비스에 있는 Fargate 온디맨드 vCPU 리소스 수에 대한 조정 가능한 할당량 값입니다 AWS Fargate . 계정의 vCPU 할당량 설정을 보거나 할당량 증가를 요청하려면에서 Service Quotas 콘솔을 사용합니다 AWS Management Console. 자세한 내용은 Amazon Elastic Container Service 개발자 안내서의 AWS Fargate 서비스 할당량을 참조하세요.

서비스의 Auto Scaling 관리

다음 방법 중 하나를 사용하여 App Runner 서비스의 Auto Scaling을 관리합니다.

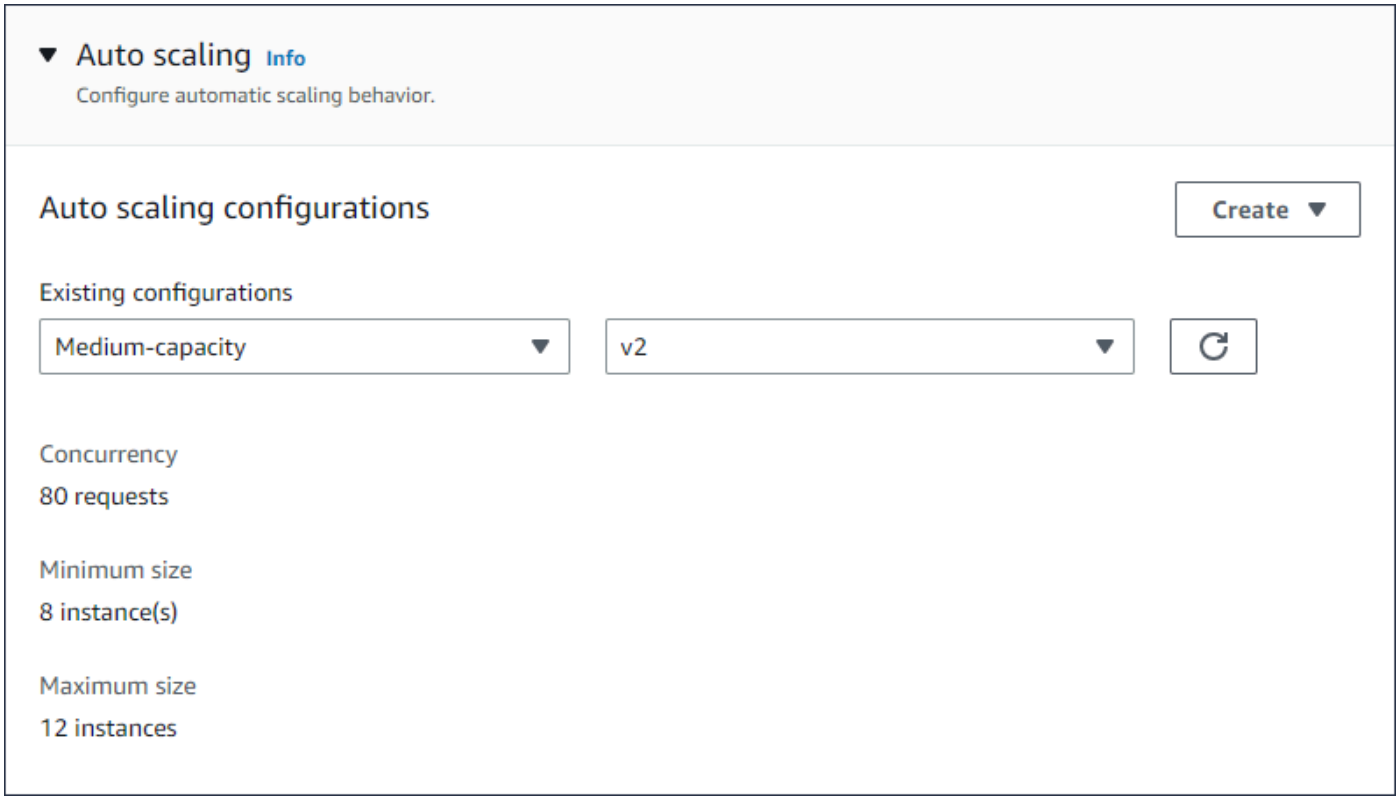
App Runner console

App Runner 콘솔을 사용하여 서비스를 생성하거나 서비스 구성을 업데이트할 때 Auto Scaling 구성을 지정할 수 있습니다.

> **Note**
>
> 서비스와 연결된 Auto Scaling 구성 또는 개정을 변경하면 서비스가 다시 배포됩니다.

Auto Scaling 구성 페이지에서는 서비스에 대한 Auto Scaling을 구성하는 몇 가지 옵션을 제공합니다.

- 기존 구성 및 개정을 할당하려면 - 기존 구성 드롭다운에서 값을 선택합니다. 최신 개정 버전은 인접 드롭다운에서 기본적으로 사용됩니다. 선택하려는 다른 개정이 있는 경우 개정 드롭다운에서 수정합니다. 개정 버전의 구성 값이 표시됩니다.
- 새 Auto Scaling 구성을 생성하고 할당하려면 - 생성 메뉴에서 새 ASC 생성을 선택합니다. 그러면 사용자 지정 Auto Scaling 구성 추가 페이지가 시작됩니다. Auto Scaling 파라미터의 구성 이름과 값을 입력합니다. 그런 다음 추가를 선택합니다. App Runner는 새 Auto Scaling 구성 리소스를 생성하고 새 구성이 선택되고 표시된 Auto Scaling 섹션으로 돌아갑니다.
- 새 개정을 생성하고 할당하려면 - 먼저 기존 구성 드롭다운에서 구성 이름을 선택합니다. 그런 다음 생성 메뉴에서 ASC 개정 생성을 선택합니다. 그러면 사용자 지정 Auto Scaling 구성 추가 페이지가 시작됩니다. Auto Scaling 파라미터의 값을 입력합니다. 그런 다음 추가를 선택합니다. App Runner는 사용자를 위해 새 Auto Scaling 구성 개정을 생성하고 새 개정이 선택되고 표시된 Auto Scaling 섹션으로 돌아갑니다.



App Runner API or AWS CLI

CreateService 또는 UpdateService App Runner API 작업을 호출할 때 `AutoScalingConfigurationArn` 파라미터를 사용하여 서비스에 대한 Auto Scaling 구성 리소스를 지정할 수 있습니다.

다음 섹션에서는 Auto Scaling 구성 리소스를 관리하기 위한 지침을 제공합니다.

Auto Scaling 구성 리소스 관리

다음 방법 중 하나를 사용하여 계정의 App Runner Auto Scaling 구성 및 개정을 관리합니다.

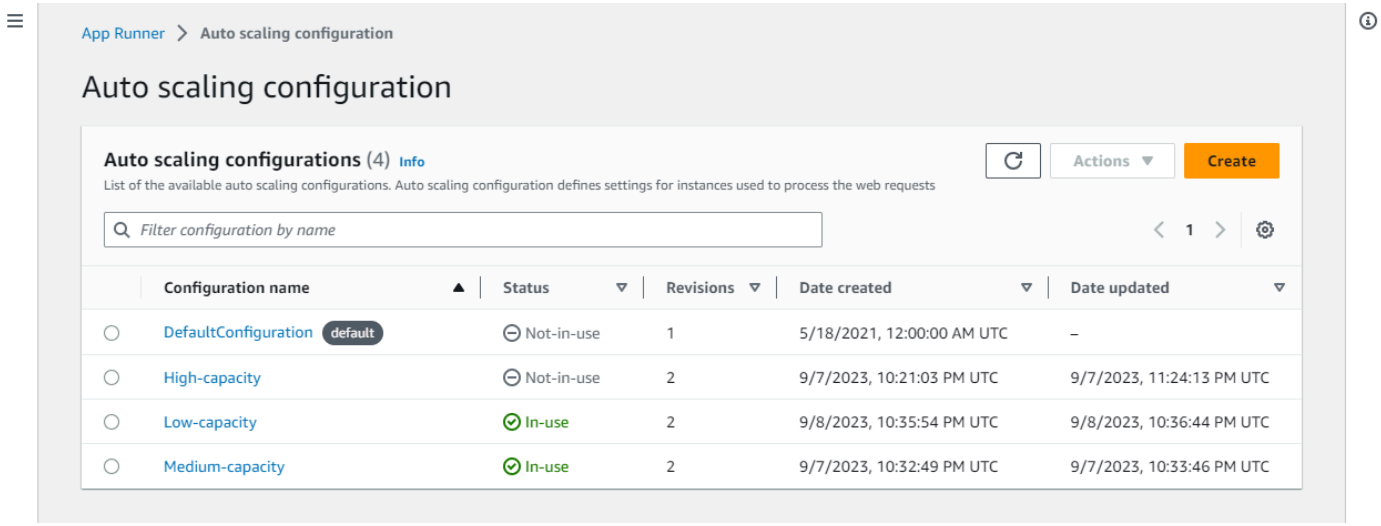
App Runner console

Auto Scaling 구성 관리

Auto Scaling 구성 페이지에는 계정에서 설정한 Auto Scaling 구성이 나열됩니다. 이 페이지에서 Auto Scaling 구성을 생성 및 관리하고 나중에 하나 이상의 App Runner 서비스에 할당할 수 있습니다.

이 페이지에서 다음 중 하나를 수행할 수 있습니다.

- 새 Auto Scaling 구성을 생성합니다.
- 기존 Auto Scaling 구성에 대한 새 개정을 생성합니다.
- Auto Scaling 구성을 삭제합니다.
- Auto Scaling 구성을 기본값으로 설정합니다.



계정에서 Auto Scaling 구성을 관리하려면

1. App Runner 콘솔을 열고 리전 목록에서를 선택합니다 AWS 리전.
2. 탐색 창에서 Auto Scaling 구성을 선택합니다. 콘솔에 계정의 Auto Scaling 구성 목록이 표시됩니다.

   이제 다음 중 하나를 수행할 수 있습니다.

   - 새 Auto Scaling 구성을 생성하려면 다음 단계를 따릅니다.

     a. Auto Scaling 구성 페이지에서 생성을 선택합니다.

        Auto Scaling 구성 생성 페이지가 표시됩니다.

     b. 구성 이름, 동시성, 최소 크기 및 최대 크기 값을 입력합니다.

     c. (선택 사항) 태그를 추가하려면 새 태그 자동을 선택합니다. 그런 다음 표시되는 필드에 이름과 값(선택 사항)을 입력합니다.

     d. 생성을 선택합니다.

   - 기존 Auto Scaling 구성에 대한 새 개정을 생성하려면 다음 단계를 따릅니다.

a. Auto Scaling 구성 페이지에서 새 개정이 필요한 구성 옆에 있는 라디오 버튼을 선택합니다. 그런 다음 작업 메뉴에서 개정 생성을 선택합니다.

   개정 생성 페이지가 표시됩니다.

b. 에 동시성, 최소 크기 및 최대 크기 값을 입력합니다.

c. (선택 사항) 태그를 추가하려면 새 태그 자동을 선택합니다. 그런 다음 표시되는 필드에 이름과 값(선택 사항)을 입력합니다.

d. 생성을 선택합니다.

- Auto Scaling 구성을 삭제하려면 다음 단계를 따릅니다.

  a. Auto Scaling 구성 페이지에서 삭제해야 하는 구성 옆에 있는 라디오 버튼을 선택합니다.

  b. 작업 메뉴에서 삭제를 선택합니다.

  c. 삭제를 계속하려면 확인 대화 상자에서 삭제를 선택합니다. 그렇지 않으면 취소를 선택합니다.

  > **Note**
  >
  > App Runner는 삭제 선택이 기본값으로 설정되지 않았거나 현재 활성 서비스에서 사용 중인지 확인합니다.

- Auto Scaling 구성을 기본값으로 설정하려면 다음 단계를 따릅니다.

  a. Auto Scaling 구성 페이지에서 기본값으로 설정해야 하는 구성 옆에 있는 라디오 버튼을 선택합니다.

  b. 작업 메뉴에서 기본값으로 설정을 선택합니다.

  c. App Runner가 최신 개정을 생성한 모든 새 서비스의 기본 구성으로 사용할 것임을 알리는 대화 상자가 표시됩니다. 계속하려면 확인을 선택합니다. 그렇지 않으면 취소를 선택합니다.

  > **Note**
  >
  > - Auto Scaling 구성을 기본값으로 설정하면 나중에 생성하는 새 서비스에 기본 구성으로 자동 할당됩니다.

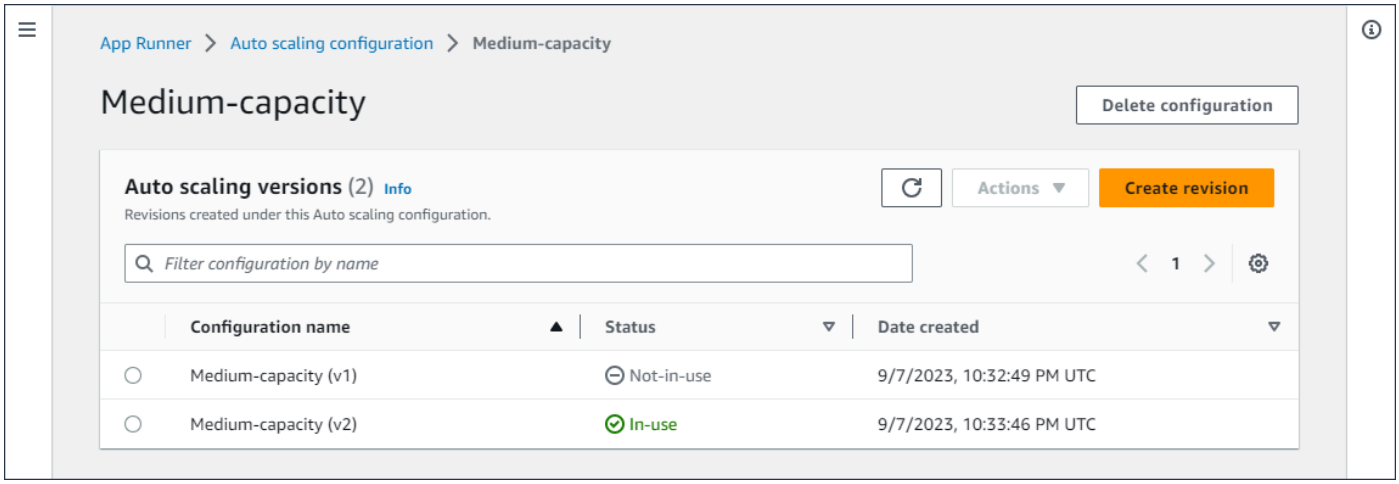
- 새 기본 지정은 기존 서비스에 대해 이전에 설정된 연결에는 영향을 주지 않습니다.
- 지정된 기본 Auto Scaling 구성에 개정이 있는 경우 App Runner는 최신 개정을 기본값으로 할당합니다.

개정 관리

콘솔에는 Auto Scaling 개정이라는 기존 Auto Scaling 개정을 생성하고 관리하기 위한 페이지도 있습니다. Auto Scaling 구성 페이지에서 구성 이름을 선택하여이 페이지에 액세스합니다.

Auto Scaling 개정 페이지에서 다음 중 하나를 수행할 수 있습니다.

- 새 Auto Scaling 개정을 생성합니다.
- Auto Scaling 구성 개정을 기본값으로 설정합니다.
- 개정을 삭제합니다.
- 연결된 모든 개정을 포함하여 전체 Auto Scaling 구성을 삭제합니다.
- 개정에 대한 구성 세부 정보를 봅니다.
- 개정과 연결된 서비스 목록을 봅니다.
- 나열된 서비스의 개정을 변경합니다.



계정에서 Auto Scaling 개정을 관리하려면

1. App Runner 콘솔을 열고 리전 목록에서를 선택합니다 AWS 리전.

2. 탐색 창에서 Auto Scaling 구성을 선택합니다. 콘솔에 계정의 Auto Scaling 구성 목록이 표시됩니다. ??? 섹션의 이전 절차 세트에는이 페이지의 화면 이미지가 포함되어 있습니다.
3. 이제 특정 Auto Scaling 구성으로 드릴다운하여 모든 개정을 보고 관리할 수 있습니다. Auto Scaling 구성 창의 구성 이름 열에서 Auto Scaling 구성 이름을 선택합니다. 라디오 버튼 대신 실제 이름을 선택합니다. 그러면 Auto Scaling 개정 페이지에서 해당 구성에 대한 모든 개정 목록으로 이동합니다.
4. 이제 다음 중 하나를 수행할 수 있습니다.
   - 기존 Auto Scaling 구성에 대한 새 개정을 생성하려면 다음 단계를 따릅니다.
     a. Auto Scaling 개정 페이지에서 개정 생성을 선택합니다.

        개정 생성 페이지가 표시됩니다.
     b. 동시성, 최소 크기 및 최대 크기 값을 입력합니다.
     c. (선택 사항) 태그를 추가하려면 새 태그 자동을 선택합니다. 그런 다음 표시되는 필드에 이름과 값(선택 사항)을 입력합니다.
     d. 생성을 선택합니다.
   - 연결된 모든 개정을 포함하여 전체 Auto Scaling 구성을 삭제하려면 다음 단계를 따릅니다.
     a. 페이지 오른쪽 상단에서 구성 삭제를 선택합니다.
     b. 삭제를 계속하려면 확인 대화 상자에서 삭제를 선택합니다. 그렇지 않으면 취소를 선택합니다.

        > **Note**
        >
        > App Runner는 삭제 선택이 기본값으로 설정되지 않았거나 현재 활성 서비스에서 사용 중인지 확인합니다.
   - Auto Scaling 개정을 기본값으로 설정하려면 다음 단계를 따릅니다.
     a. 기본값으로 설정해야 하는 개정 옆의 라디오 버튼을 선택합니다.
     b. 작업 메뉴에서 기본값으로 설정을 선택합니다.

> **Note**
>
> - Auto Scaling 구성을 기본값으로 설정하면 나중에 생성하는 새 서비스에 기본 구성으로 자동 할당됩니다.
> - 새 기본 지정은 기존 서비스에 대해 이전에 설정된 연결에는 영향을 주지 않습니다.

- 개정의 구성 세부 정보를 보려면 다음 단계를 따르세요.
  - 개정 옆의 라디오 버튼을 선택합니다.

    ARN을 포함한 개정의 구성 세부 정보가 하단 분할 패널에 표시됩니다. 이 절차의 끝에 있는 화면 이미지를 참조하세요.
- 개정과 연결된 서비스 목록을 보려면 다음 단계를 따르세요.
  - 개정 옆의 라디오 버튼을 선택합니다.

    서비스 패널은 개정 구성 세부 정보 아래의 하단 분할 패널에 표시됩니다. 패널에는이 Auto Scaling 구성 개정을 사용하는 모든 서비스가 나열됩니다. 이 절차의 끝에 있는 화면 이미지를 참조하세요.
- 나열된 서비스의 개정을 변경하려면 다음 단계를 따릅니다.
  a. 아직 수정하지 않았다면 개정 옆의 라디오 버튼을 선택합니다.

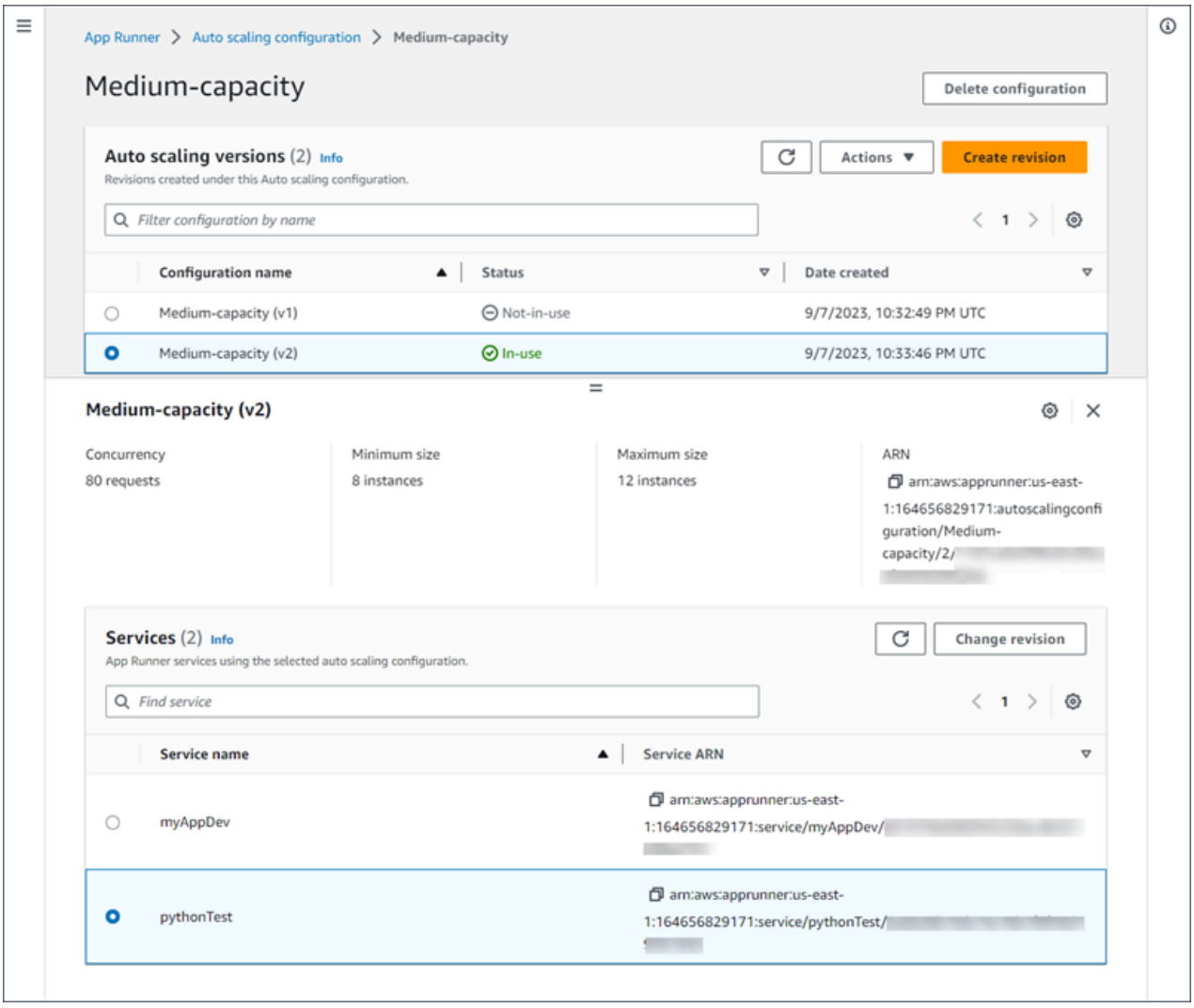
     서비스 패널은 개정 구성 세부 정보 아래의 하단 분할 패널에 표시됩니다. 패널에는이 Auto Scaling 구성 개정을 사용하는 모든 서비스가 나열됩니다. 이 절차의 끝에 있는 화면 이미지를 참조하세요.

  b. 서비스 패널에서 수정하려는 서비스 옆에 있는 라디오 버튼을 선택합니다. 그런 다음 개정 변경을 선택합니다.

  c. ASC 개정 변경 패널이 표시됩니다. 드롭다운에서 사용 가능한 개정 중에서 선택합니다. 이전에 선택한 Auto Scaling 구성의 개정 버전만 사용할 수 있습니다. 다른 Auto Scaling 구성으로 변경해야 하는 경우 이전 섹션의 절차를 따르세요the section called “서비스의 Auto Scaling 관리”.

     업데이트를 선택하여 변경을 진행합니다. 그렇지 않으면 취소를 선택합니다.

> **Note**
>
> 서비스와 연결된 개정을 변경하면 서비스가 다시 배포됩니다.
> 업데이트된 연결을 보려면이 패널에서 새로 고침을 선택해야 합니다.
> 진행 중인 활동 및 서비스 재배포 상태를 보려면 패널 브레드크럼을 사용하여 App Runner > 서비스로 이동하고 서비스를 선택한 다음 서비스 개요 패널에서 로그 탭을 확인합니다.



App Runner API or AWS CLI

다음 App Runner API 작업을 사용하여 Auto Scaling 구성 리소스를 관리합니다.

- CreateAutoScalingConfiguration - 새 Auto Scaling 구성 또는 기존 구성에 대한 개정을 생성합니다.
- UpdateDefaultAutoScalingConfiguration - Auto Scaling 구성을 기본값으로 설정합니다. 기존 기본 Auto Scaling 구성은 기본값이 아닌 구성으로 자동 설정됩니다.
- ListAutoScalingConfigurations -와 연결된 Auto Scaling 구성 목록을 요약 정보와 AWS 계정함께 반환합니다.
- ListServicesForAutoScalingConfiguration - Auto Scaling 구성을 사용하여 연결된 App Runner 서비스 목록을 반환합니다.
- DescribeAutoScalingConfiguration - Auto Scaling 구성에 대한 전체 설명을 반환합니다.
- DeleteAutoScalingConfiguration - Auto Scaling 구성을 삭제합니다. 최상위 Auto Scaling 구성, 특정 개정 또는 최상위 구성과 연결된 모든 개정을 삭제할 수 있습니다. 선택적 `DeleteAllRevisions` 파라미터를 사용하여 모든 개정을 삭제합니다. 의 Auto Scaling 구성 리소스 할당량에 도달 AWS 계정하면 불필요한 Auto Scaling 구성을 삭제해야 할 수 있습니다.

App Runner 서비스의 사용자 지정 도메인 이름 관리

AWS App Runner 서비스를 생성할 때 App Runner는 서비스에 대한 도메인 이름을 할당합니다. App Runner가 소유한 도메인의 하위 `awsapprunner.com` 도메인입니다. 도메인 이름을 사용하여 서비스에서 실행 중인 웹 애플리케이션에 액세스할 수 있습니다.

> **Note**
>
> App Runner 애플리케이션의 보안을 강화하기 위해 *.awsapprunner.com 도메인은 퍼블릭 접미사 목록(PSL)에 등록됩니다. 보안을 강화하기 위해 App Runner 애플리케이션의 기본 도메인 이름에 민감한 쿠키를 설정해야 하는 경우 `__Host-` 접두사가 있는 쿠키를 사용하는 것이 좋습니다. 이렇게 쿠키를 설정하면 교차 사이트 요청 위조 시도(CSRF)로부터 도메인을 보호하는 데 도움이 됩니다. 자세한 내용은 Mozilla 개발자 네트워크의 Set-Cookie 페이지를 참조하세요.

도메인 이름을 소유한 경우 이를 App Runner 서비스에 연결할 수 있습니다. App Runner가 새 도메인을 검증한 후 도메인을 사용하여 App Runner 도메인 외에도 애플리케이션에 액세스할 수 있습니다. 최대 5개의 사용자 지정 도메인을 연결할 수 있습니다.

> **Note**
>
> 도메인의 www 하위 도메인을 선택적으로 포함할 수 있습니다. 그러나 이는 현재 API에서만 지원됩니다. App Runner 콘솔은 도메인의 www 하위 도메인 포함을 지원하지 않습니다.

> **Note**
>
> AWS App Runner 는 Route 53 프라이빗 호스팅 영역 사용을 지원하지 않습니다. 프라이빗 호스팅 영역은 Amazon VPC 트래픽에 대한 도메인 이름 확인을 사용자 지정합니다. 프라이빗 호스팅 영역에 대한 자세한 내용은 Route 53 설명서의 프라이빗 호스팅 영역 작업을 참조하세요.

사용자 지정 도메인을 서비스에 연결(링크)

사용자 지정 도메인을 서비스에 연결할 때 CNAME 레코드와 DNS 대상 레코드를 DNS 서버에 추가해야 합니다. 다음 섹션에서는 CNAME 레코드 및 DNS 대상 레코드와 이를 사용하는 방법에 대한 정보를 제공합니다.

> **Note**
>
> Amazon Route 53를 DNS 공급자로 사용하는 경우 App Runner는 App Runner 웹 애플리케이션에 연결하는 데 필요한 인증서 검증 및 DNS 레코드로 사용자 지정 도메인을 자동으로 구성합니다. 이는 App Runner 콘솔을 사용하여 사용자 지정 도메인을 서비스에 연결할 때 발생합니다. 다음 사용자 지정 도메인 관리 주제에서는 자세한 정보를 제공합니다.

CNAME 레코드

사용자 지정 도메인을 서비스와 연결하면 App Runner는 인증서 검증을 위한 인증서 검증 레코드 세트를 제공합니다. 이러한 인증서 검증 레코드를 도메인 이름 시스템(DNS) 서버에 추가해야 합니다. App Runner에서 제공하는 인증서 검증 레코드를 DNS 서버에 추가합니다. 이렇게 하면 App Runner가 도메인을 소유하거나 제어하는지 검증할 수 있습니다.

> **Note**
>
> 사용자 지정 도메인 인증서를 자동 갱신하려면 DNS 서버에서 인증서 검증 레코드를 삭제하지 않아야 합니다. 인증서 갱신과 관련된 문제를 해결하는 방법에 대한 자세한 내용은 섹션을 참조하세요the section called “사용자 지정 도메인 인증서 갱신”.

App Runner는 ACM을 사용하여 도메인을 확인합니다. DNS 레코드에서 CAA 레코드를 사용하는 경우 하나 이상의 CAA 레코드가를 참조하는지 확인합니다`amazon.com`. 그렇지 않으면 ACM이 도메인을 확인하고 도메인을 성공적으로 생성할 수 없습니다.

CAA와 관련된 오류가 발생하는 경우 다음 링크를 참조하여 문제를 해결하는 방법을 알아봅니다.

- 인증 기관 인증(CAA) 문제
- ACM 인증서 발급 또는 갱신에 대한 CAA 오류를 해결하려면 어떻게 해야 합니까?
- 사용자 지정 도메인 이름

> **Note**
>
> Amazon Route 53를 DNS 공급자로 사용하는 경우 App Runner는 App Runner 웹 애플리케이션에 연결하는 데 필요한 인증서 검증 및 DNS 레코드로 사용자 지정 도메인을 자동으로 구성합니다. 이는 App Runner 콘솔을 사용하여 사용자 지정 도메인을 서비스에 연결할 때 발생합니다. 다음 사용자 지정 도메인 관리 주제에서는 자세한 정보를 제공합니다.

DNS 대상 레코드

DNS 대상 레코드를 DNS 서버에 추가하여 App Runner 도메인을 대상으로 지정합니다. 이 옵션을 선택한 경우 사용자 지정 도메인에 대한 레코드 하나와 `www` 하위 도메인에 대한 레코드 하나를 추가합니다. 그런 다음 App Runner 콘솔에서 사용자 지정 도메인 상태가 활성이 될 때까지 기다립니다. 일반적으로 몇 분 정도 걸리지만 최대 24~48시간(1~2일)이 걸릴 수 있습니다. 사용자 지정 도메인이 검증되면 App Runner는이 도메인에서 웹 애플리케이션으로 트래픽을 라우팅하기 시작합니다.

> **Note**
>
> App Runner 서비스와의 호환성을 높이려면 Amazon Route 53을 DNS 공급자로 사용하는 것이 좋습니다. Amazon Route 53를 사용하여 퍼블릭 DNS 레코드를 관리하지 않는 경우 DNS 공급자에게 문의하여 레코드를 추가하는 방법을 알아보세요.

> Amazon Route 53를 DNS 공급자로 사용하는 경우 하위 도메인에 대한 CNAME 또는 별칭 레코드를 추가할 수 있습니다. 루트 도메인의 경우 별칭 레코드를 사용해야 합니다.

Amazon Route 53 또는 다른 공급자로부터 도메인 이름을 구매할 수 있습니다. Amazon Route 53에서 도메인 이름을 구매하려면 Amazon Route 53 개발자 안내서의 새 도메인 등록을 참조하세요.

Route 53에서 DNS 대상을 구성하는 방법에 대한 지침은 Amazon Route 53 개발자 안내서의 리소스로 트래픽 라우팅을 참조하세요.

GoDaddy, Shopify, Hover 등과 같은 다른 등록 기관에서 DNS 대상을 구성하는 방법에 대한 지침은 DNS 대상 레코드 추가에 대한 특정 설명서를 참조하세요.

App Runner 서비스와 연결할 도메인 지정

다음과 같은 방법으로 App Runner 서비스와 연결할 도메인을 지정할 수 있습니다.

- 루트 도메인 - DNS에는 루트 도메인 이름에 대한 CNAME 레코드 생성을 차단할 수 있는 몇 가지 고유한 제한 사항이 있습니다. 예를 들어 도메인 이름이 인 경우 `example.com`에 대한 트래픽을 App Runner 서비스로 라우팅하는 CNAME 레코드`acme.example.com`를 생성할 수 있습니다. 그러나에 대한 트래픽을 App Runner 서비스로 라우팅하는 CNAME 레코드`example.com`는 생성할 수 없습니다. 루트 도메인을 생성하려면 별칭 레코드를 추가해야 합니다.

  별칭 레코드는 Route 53에 고유하며 CNAME 레코드에 비해 다음과 같은 이점이 있습니다.

  - Route 53는 루트 도메인 또는 하위 도메인에 대해 별칭 레코드를 생성할 수 있으므로 더 유연하게 사용할 수 있습니다. 예를 들어 도메인 이름이 인 경우 `example.com` 또는에 대한 요청을 App Runner 서비스로 라우팅하는 레코드`acme.example.com`를 생성할 `example.com`수 있습니다.
  - 더 비용 효율적입니다. 이는 Route 53가 별칭 레코드를 사용하여 트래픽을 라우팅하는 요청에 대해 요금을 부과하지 않기 때문입니다.
- 하위 도메인 - 예: `login.example.com` 또는 `admin.login.example.com`. 필요에 따라 `www` 하위 도메인을 동일한 작업의 일부로 연결할 수도 있습니다. 하위 도메인에 대한 CNAME 또는 별칭 레코드를 추가할 수 있습니다.
- 와일드카드 - 예: `*.example.com`. 이 경우에는 `www` 옵션을 사용할 수 없습니다. 와일드카드는 루트 도메인의 즉시 하위 도메인으로만 지정하고 자체적으로만 지정할 수 있습니다. 이는 유효한 사양이 아닙니다. `login*.example.com`, `*.login.example.com`. 이 와일드카드 사양은 모든 직계 하위 도메인을 연결하고 루트 도메인 자체를 연결하지 않습니다. 루트 도메인은 별도의 작업으로 연결되어야 합니다.

보다 구체적인 도메인 연결은 덜 구체적인 도메인 연결을 재정의합니다. 예를 들어는를 `login.example.com` 재정의합니다`*.example.com`. 보다 구체적인 연결의 인증서와 CNAME이 사용됩니다.

다음 예제에서는 여러 사용자 지정 도메인 연결을 사용하는 방법을 보여줍니다.

1. 서비스의 홈 페이지에 `example.com` 연결합니다. 를 활성화`www`하여를 연결합니다`www.example.com`.
2. 서비스의 로그인 페이지에 `login.example.com` 연결합니다.
3. 사용자 지정 "찾을 수 없음" 페이지`*.example.com`와 연결합니다.

사용자 지정 도메인 연결 해제(연결 해제)

App Runner 서비스에서 사용자 지정 도메인의 연결을 해제(연결 해제)할 수 있습니다. 도메인 연결을 해제하면 App Runner는이 도메인에서 웹 애플리케이션으로의 트래픽 라우팅을 중지합니다.

> **Note**
>
> DNS 서버에서 연결 해제한 도메인의 레코드를 삭제해야 합니다.

App Runner는 도메인 유효성을 추적하는 인증서를 내부적으로 생성합니다. 이러한 인증서는 AWS Certificate Manager (ACM)에 저장됩니다. App Runner는 도메인이 서비스에서 연결 해제된 후 또는 서비스가 삭제된 후 7일 동안 이러한 인증서를 삭제하지 않습니다.

사용자 지정 도메인 관리

다음 방법 중 하나를 사용하여 App Runner 서비스의 사용자 지정 도메인을 관리합니다.

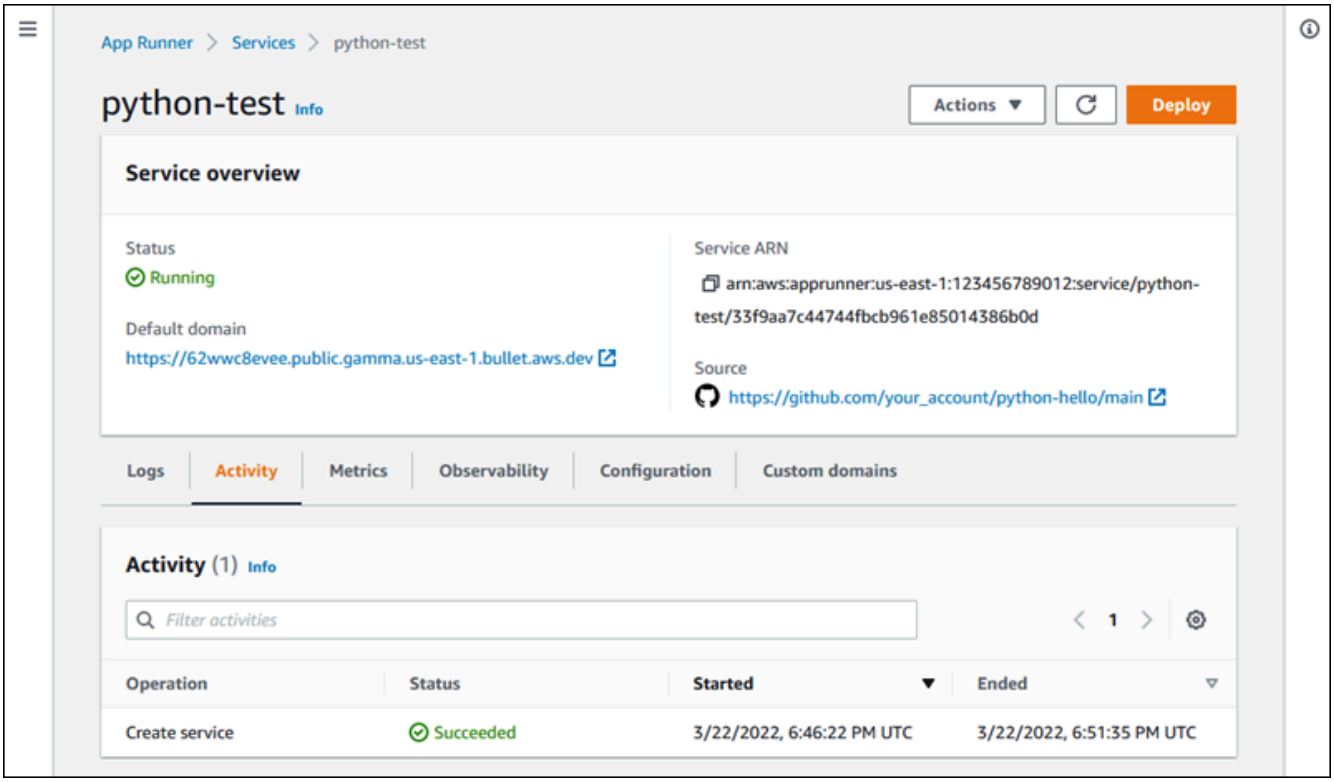
> **Note**
>
> App Runner 서비스와의 호환성을 높이려면 Amazon Route 53을 DNS 공급자로 사용하는 것이 좋습니다. Amazon Route 53를 사용하여 퍼블릭 DNS 레코드를 관리하지 않는 경우 DNS 공급자에게 문의하여 레코드를 추가하는 방법을 알아보세요.
> Amazon Route 53를 DNS 공급자로 사용하는 경우 하위 도메인에 대한 CNAME 또는 별칭 레코드를 추가할 수 있습니다. 루트 도메인의 경우 별칭 레코드를 사용해야 합니다.

App Runner console

App Runner 콘솔을 사용하여 사용자 지정 도메인을 연결(링크)하려면

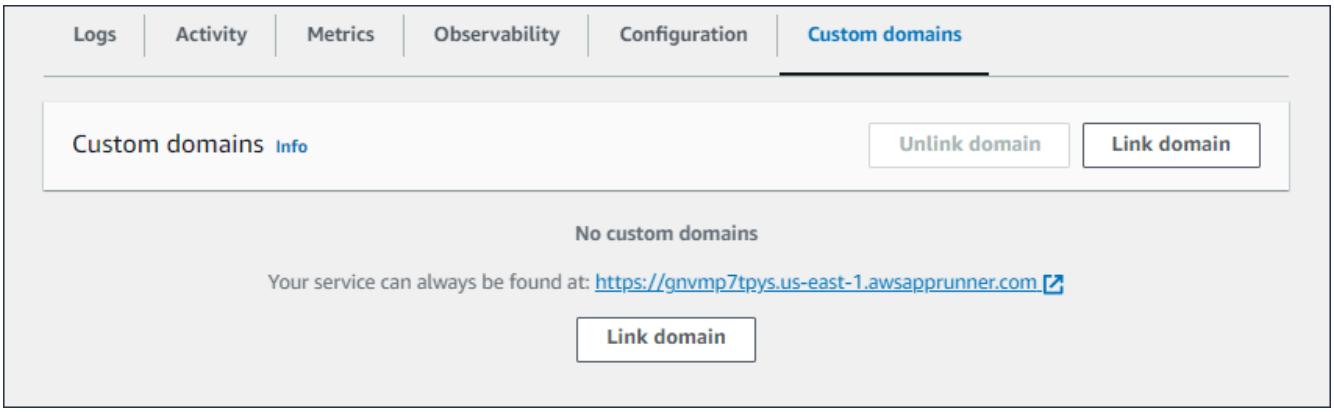
1. App Runner 콘솔을 열고 리전 목록에서를 선택합니다 AWS 리전.
2. 탐색 창에서 서비스를 선택한 다음 App Runner 서비스를 선택합니다.

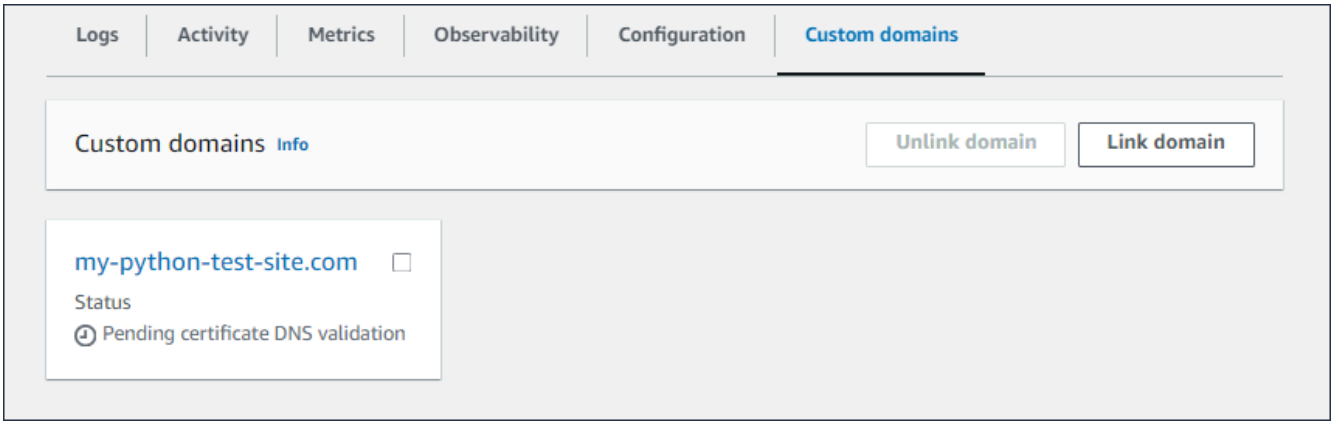
   콘솔에 서비스 개요와 함께 서비스 대시보드가 표시됩니다.



3. 서비스 대시보드 페이지에서 사용자 지정 도메인 탭을 선택합니다.

   콘솔에는 서비스와 연결된 사용자 지정 도메인 또는 사용자 지정 도메인 없음이 표시됩니다.





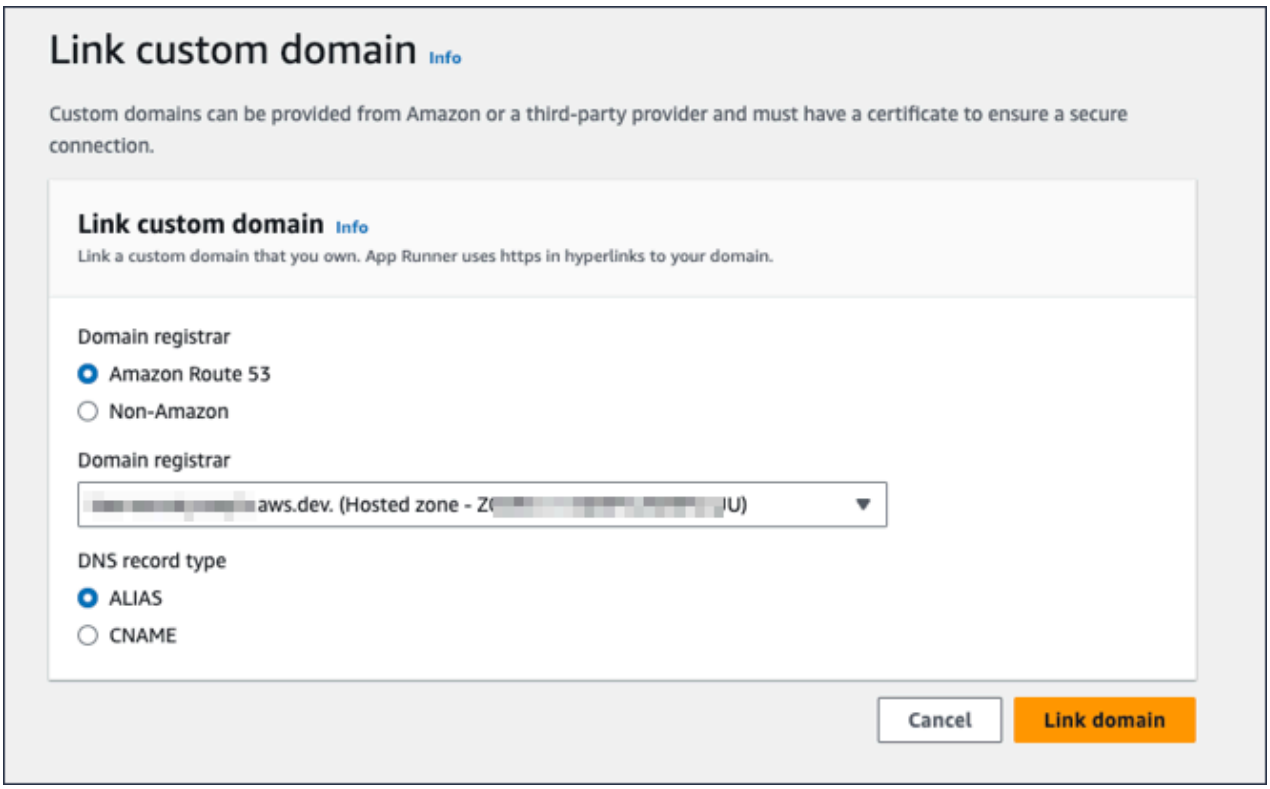
4. 사용자 지정 도메인 탭에서 도메인 연결을 선택합니다.
5. 사용자 지정 도메인 연결 페이지가 표시됩니다.
   - 사용자 지정 도메인이 Amazon Route 53에 등록된 경우 도메인 등록 대행자로 Amazon Route 53을 선택합니다.

     a. 드롭다운 목록에서 도메인 이름을 선택합니다. 이 목록에는 Route 53 도메인 이름의 이름과 호스팅 영역 ID가 표시됩니다.

     > **Note**
     >
     > 먼저 다른 App Runner 리소스를 관리하는 데 사용하는 것과 동일한 AWS 계정에서 Amazon Route 53 서비스를 사용하여 Route 53 도메인을 생성해야 합니다.

     b. DNS 레코드 유형을 선택합니다.

     c. 도메인 연결을 선택합니다.

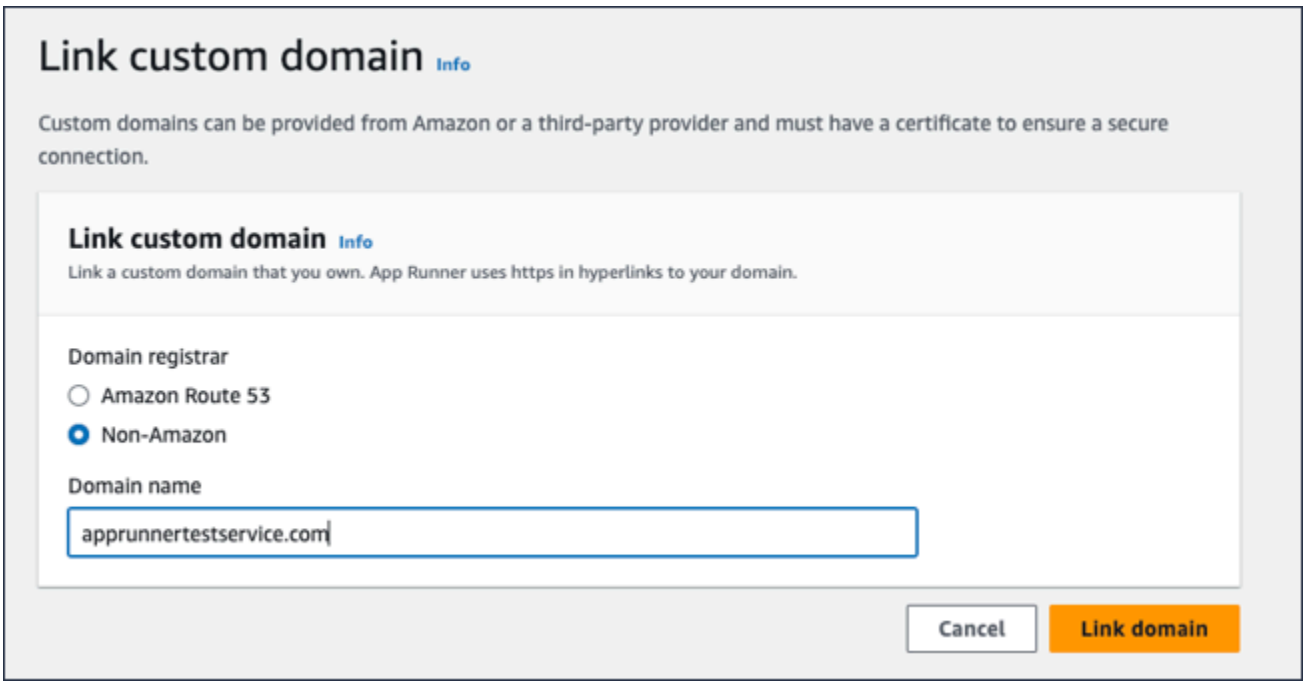


> **Note**
>
> App Runner에 자동 구성 시도가 실패했다는 오류 메시지가 표시되면 DNS 레코드를 수동으로 구성하여 진행할 수 있습니다. 이 문제는 이후에 삭제되는 서비스를 가리키는 DNS 공급자 레코드 없이 동일한 도메인 이름이 이전에 서비스에서 연결 해제된 경우 발생할 수 있습니다. 이 경우 App Runner는 이러한 레코드를 자동으로 덮어쓰지 못하도록 차단됩니다. DNS 구성을 완료하려면이 절차의 나머지 단계를 건너뛰고의 지침을 따릅니다Amazon Route 53 별칭 레코드 구성.

- 사용자 지정 도메인이 다른 도메인 등록 대행자에 등록된 경우 도메인 등록 대행자에 대해 비 Amazon을 선택합니다.

  a. 도메인 이름을 입력합니다.

  b. 도메인 연결을 선택합니다.



6. DNS 구성 페이지가 표시됩니다.

   - Amazon Route 53 가 DNS 공급자인 경우이 단계는 선택 사항입니다.

     이때 App Runner는 필요한 인증서 검증 및 DNS 레코드로 Route 53 도메인을 자동으로 구성했습니다.

     > **Note**
     >
     > 이 동일한 도메인 이름이 이전에 서비스에서 연결 해제된 경우 DNS 공급자가 나중에 삭제되는 서비스를 가리키는 레코드 없이 App Runner가 시도한 자동 구성이 실패했을 수 있습니다. 이 문제를 해결하고 DNS 연결을 완료하려면 DNS 구성 페이지의 (1) 및 (2)단계로 진행하여 현재 대상 및 인증서 레코드를 DNS 공급자에게 복사합니다.

   - 인증서 검증 레코드와 DNS 대상 레코드를 복사하여 DNS 서버에 추가합니다. 그런 다음 App Runner는 도메인을 소유하거나 제어하는지 확인할 수 있습니다.

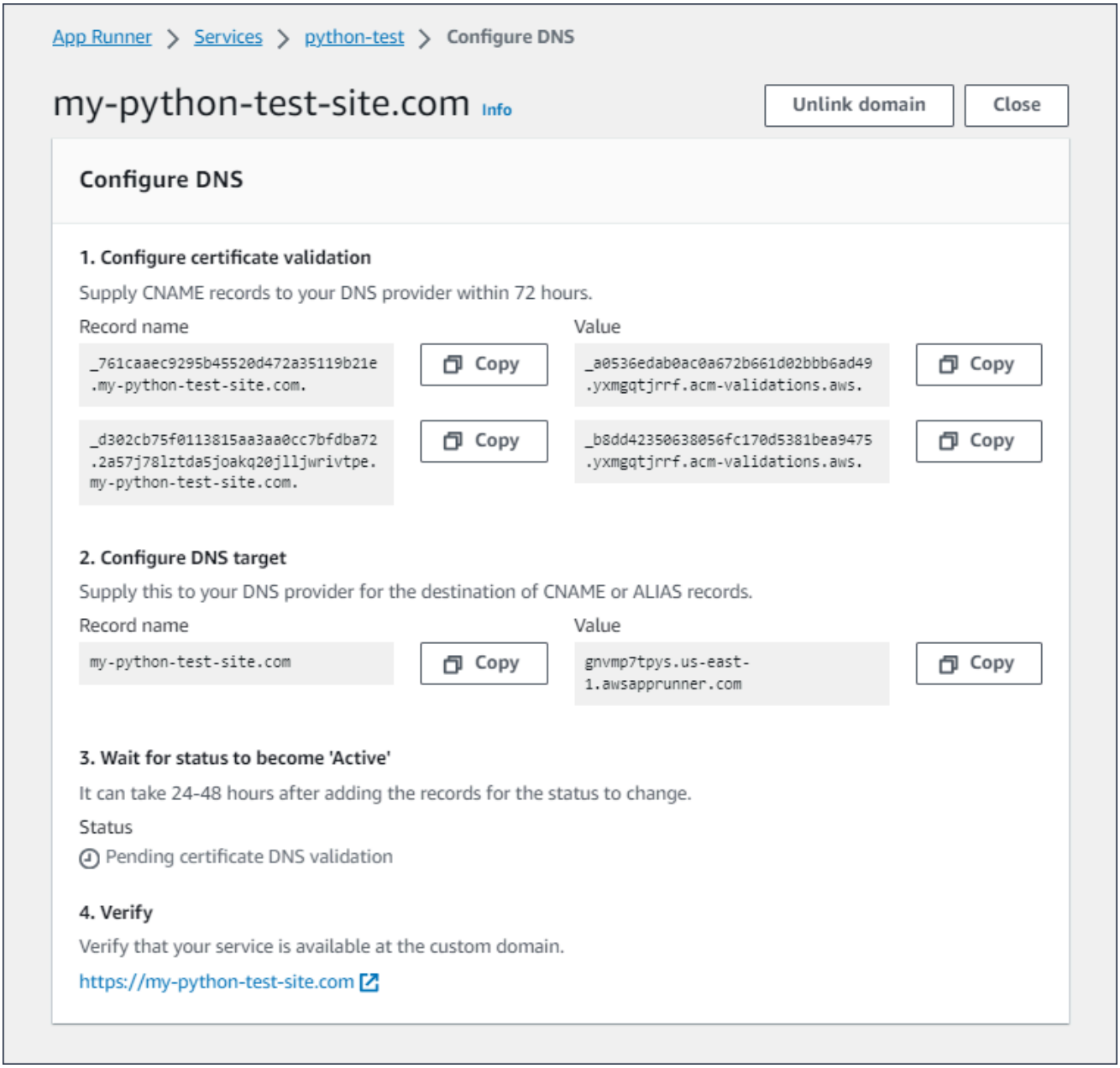
     > **Note**
     >
     > 사용자 지정 도메인 인증서를 자동 갱신하려면 DNS 서버에서 인증서 검증 레코드를 삭제하지 않아야 합니다.

- 인증서 검증 구성에 대한 자세한 내용은 AWS Certificate Manager 사용 설명서의 DNS 검증을 참조하세요.
- Amazon Route 53 별칭 레코드를 사용하여 DNS 대상을 구성하는 방법에 대한 자세한 내용은 섹션을 참조하세요the section called “Amazon Route 53 별칭 레코드 구성”.

- Amazon Route 53 이외의 DNS 공급자를 사용하는 경우 다음 단계를 따르세요.
  - 인증서 검증 레코드와 DNS 대상 레코드를 복사하여 DNS 서버에 추가합니다. 그런 다음 App Runner는 도메인을 소유하거나 제어하는지 확인할 수 있습니다.

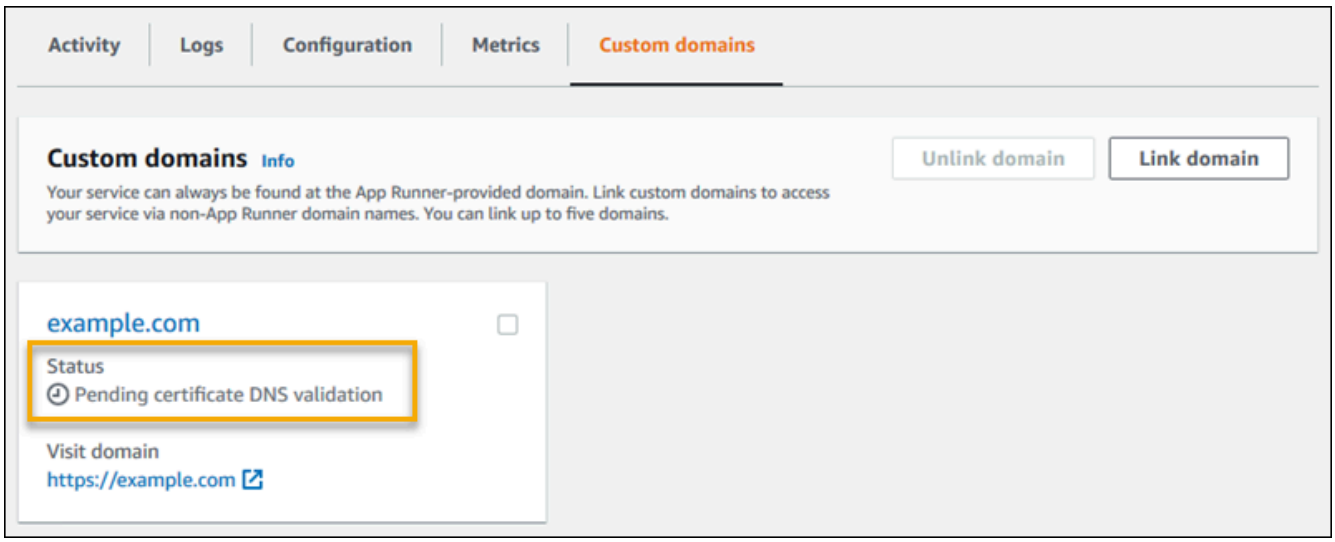
    > **Note**
    >
    > 사용자 지정 도메인 인증서를 자동 갱신하려면 DNS 서버에서 인증서 검증 레코드를 삭제하지 않아야 합니다.

    - 인증서 검증 구성에 대한 자세한 내용은 AWS Certificate Manager 사용 설명서의 DNS 검증을 참조하세요.
    - GoDaddy, Shopify, Hover 등과 같은 다른 등록 기관에서 DNS 대상을 구성하는 방법에 대한 지침은 DNS 대상 추가에 대한 특정 설명서를 참조하세요.

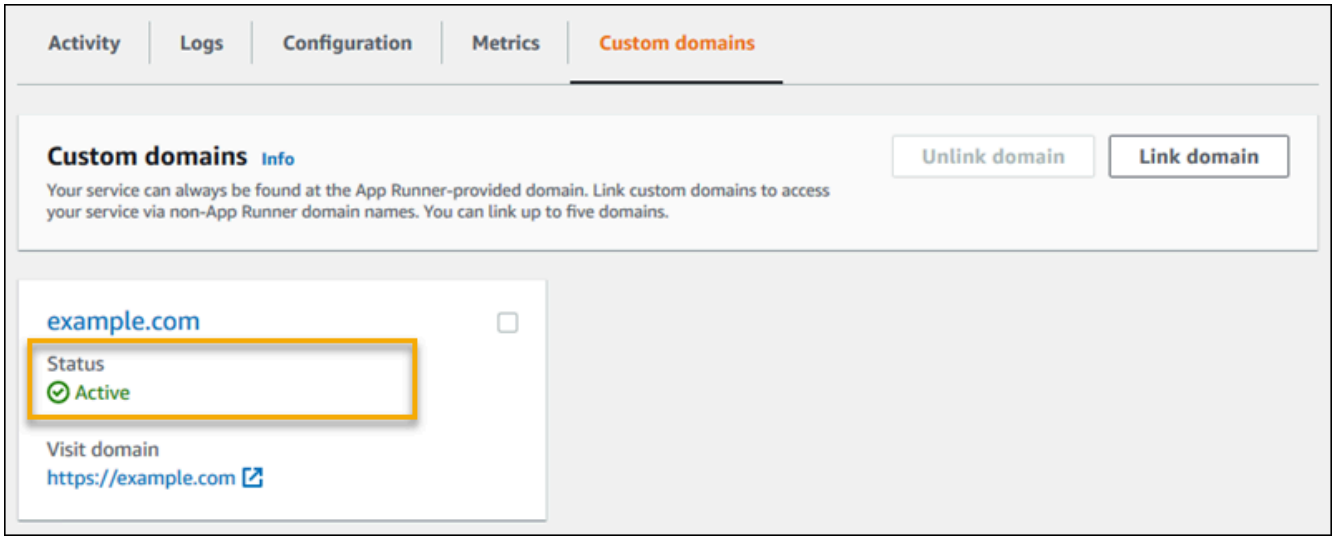


7. 닫기를 선택합니다.

   콘솔에 대시보드가 다시 표시됩니다. 사용자 지정 도메인 탭에는 보류 중인 인증서 DNS 검증 상태에서 방금 연결한 도메인을 보여주는 새 타일이 있습니다.



8. 도메인 상태가 활성으로 변경되면 도메인을 탐색하여 트래픽을 라우팅하는 데 도메인이 작동하는지 확인합니다.



> **Note**
>
> 사용자 지정 도메인과 관련된 오류를 해결하는 방법에 대한 지침은 섹션을 참조하세요the section called “사용자 지정 도메인 이름”.

App Runner 콘솔을 사용하여 사용자 지정 도메인의 연결을 해제(연결 해제)하려면

1. 사용자 지정 도메인 탭에서 연결 해제하려는 도메인의 타일을 선택한 다음 도메인 연결 해제를 선택합니다.

2. 도메인 연결 해제 대화 상자에서 도메인 연결 해제를 선택하여 작업을 확인합니다.

> **Note**
>
> DNS 서버에서 연결 해제한 도메인의 레코드를 삭제해야 합니다.

App Runner API or AWS CLI

App Runner API를 사용하여 사용자 지정 도메인을 서비스에 연결하려면 AssociateCustomDomain API 작업을 AWS CLI호출합니다. 호출이 성공하면 서비스와 연결된 사용자 지정 도메인을 설명하는 CustomDomain 객체가 반환됩니다. 객체는 `CREATING` 상태를 표시하고 CertificateValidationRecord 객체 목록을 포함합니다. 호출은 DNS 대상을 구성하는 데 사용할 수 있는 대상 별칭도 반환합니다. 이는 DNS에 추가할 수 있는 레코드입니다.

App Runner API를 사용하여 서비스에서 사용자 지정 도메인의 연결을 해제하려면 DisassociateCustomDomain API 작업을 AWS CLI호출합니다. 호출이 성공하면 서비스에서 연결 해제되는 사용자 지정 도메인을 설명하는 CustomDomain 객체가 반환됩니다. 객체에 `DELETING` 상태가 표시됩니다.

주제

- 대상 DNS에 대한 Amazon Route 53 별칭 레코드 구성

대상 DNS에 대한 Amazon Route 53 별칭 레코드 구성

> **Note**
>
> Amazon Route 53가 DNS 공급자인 경우이 절차를 따를 필요가 없습니다. 이 경우 App Runner는 App Runner 웹 애플리케이션에 연결하기 위해 필요한 인증서 검증 및 DNS 레코드로 Route 53 도메인을 자동으로 구성합니다.
> App Runner의 자동 구성 시도가 실패한 경우 다음 절차에 따라 DNS 구성을 완료합니다. 이전에 동일한 도메인 이름이 서비스에서 연결 해제된 경우 DNS 공급자 레코드가 나중에 서비스가 삭제되는 것을 가리키지 않으면 App Runner는 이러한 레코드를 자동으로 덮어쓰지 못하도록 차단됩니다. 이 절차에서는 Route 53 DNS에 수동으로 복사하는 방법을 설명합니다.

Amazon Route 53를 DNS 공급자로 사용하여 트래픽을 App Runner 서비스로 라우팅할 수 있습니다. 가용성과 확장성이 뛰어난 도메인 이름 시스템(DNS) 웹 서비스입니다. Amazon Route 53 레코드에는 트래픽이 App Runner 서비스로 라우팅되는 방식을 제어하는 설정이 포함되어 있습니다. CNAME 레코드 또는 ALIAS 레코드를 생성합니다. CNAME 및 별칭 레코드에 대한 비교는 Amazon Route 53 개발자 안내서의 별칭 레코드와 비별칭 레코드 선택 섹션을 참조하세요.

> **Note**
>
> Amazon Route 53는 현재 2022년 8월 1일 이후에 생성된 서비스에 대한 별칭 레코드를 지원합니다.

Amazon Route 53 console

Amazon Route 53 별칭 레코드를 구성하려면

1. 에 로그인 AWS Management Console 하고 Route 53 콘솔을 엽니다.
2. 탐색 창에서 호스팅 영역(Hosted zones)을 선택합니다.
3. 트래픽을 App Runner 서비스로 라우팅하는 데 사용할 호스팅 영역의 이름을 선택합니다.
4. 레코드 세트 생성을 선택합니다.
5. 다음 값을 지정하세요.
   - 라우팅 정책: 해당 라우팅 정책을 선택합니다. 자세한 내용은 라우팅 정책 선택을 참조하세요.
   - 레코드 이름: 트래픽을 App Runner 서비스로 라우팅하는 데 사용할 도메인 이름을 입력합니다. 기본값은 호스팅 영역 이름입니다. 예를 들어 호스팅 영역의 이름이 `example.com`이고를 사용하여 트래픽을 환경으로 라우팅하려는 경우 `acme.example.com`를 입력합니다`acme`.
   - 값/트래픽 라우팅 대상: App Runner 애플리케이션에 대한 별칭을 선택한 다음 엔드포인트가 있는 리전을 선택합니다. 트래픽을 라우팅할 애플리케이션의 도메인 이름을 선택합니다.
   - 레코드 유형: 기본값인 A – IPv4 주소를 수락합니다.
   - 대상 상태 평가: 기본값인 예를 수락합니다.
6. 레코드 생성을 선택합니다.

생성한 Route 53 별칭 레코드는 60초 이내에 모든 Route 53 서버에 전파됩니다. Route 53 서버가 별칭 레코드와 함께 전파되면 생성한 별칭 레코드의 이름을 사용하여 트래픽을 App Runner 서비스로 라우팅할 수 있습니다.

DNS 변경 사항이 전파되는 데 시간이 너무 오래 걸리는 경우 문제를 해결하는 방법에 대한 자세한 내용은 Route 53 및 퍼블릭 해석기에서 DNS 변경 사항이 전파되는 데 시간이 오래 걸리는 이유는 무엇입니까?를 참조하세요.

Amazon Route 53 API or AWS CLI

Amazon Route 53 API를 사용하여 Amazon Route 53 별칭 레코드를 구성하거나 ChangeResourceRecordSets API 작업을 AWS CLI 호출합니다. Route 53의 대상 호스팅 영역 ID에 대한 자세한 내용은 서비스 엔드포인트를 참조하세요.

App Runner 서비스 일시 중지 및 재개

웹 애플리케이션을 일시적으로 비활성화하고 코드 실행을 중지해야 하는 경우 AWS App Runner 서비스를 일시 중지할 수 있습니다. App Runner는 서비스에 대한 컴퓨팅 용량을 0으로 줄입니다.

애플리케이션을 다시 실행할 준비가 되면 App Runner 서비스를 재개할 수 있습니다. App Runner는 새로운 컴퓨팅 파워를 프로비저닝하고, 애플리케이션을 배포한 후 애플리케이션을 실행합니다. 애플리케이션 소스는 재배포되지 않으므로 빌드할 필요가 없습니다. 대신 App Runner는 현재 배포된 버전으로 재개합니다. 애플리케이션은 App Runner 도메인을 유지합니다.

Important

- 서비스를 일시 중지하면 애플리케이션이 상태를 잃게 됩니다. 예를 들어 코드가 사용한 임시 스토리지는 손실됩니다. 코드의 경우 서비스를 일시 중지했다가 다시 시작하는 것은 새 서비스에 배포하는 것과 같습니다.
- 코드의 결함(예: 발견된 버그 또는 보안 문제)으로 인해 서비스를 일시 중지하는 경우 서비스를 재개하기 전에 새 버전을 배포할 수 없습니다.

  따라서 서비스를 계속 실행하고 마지막으로 안정적인 애플리케이션 버전으로 롤백하는 것이 좋습니다.
- 서비스를 재개하면 App Runner는 서비스를 일시 중지하기 전에 사용된 마지막 애플리케이션 버전을 배포합니다. 서비스 일시 중지 이후 새 소스 버전을 추가한 경우 자동 배포가 선택되어 있더라도 App Runner는 새 소스 버전을 자동으로 배포하지 않습니다. 예를 들어 이미

지 리포지토리에 새 이미지 버전이 있거나 코드 리포지토리에 새 커밋이 있다고 가정합니다. 이러한 버전은 자동으로 배포되지 않습니다.

최신 버전을 배포하려면 App Runner 서비스를 재개한 후 수동 배포를 수행하거나 소스 리포지토리에 다른 버전을 추가합니다.

비교 일시 중지 및 삭제

App Runner 서비스를 일시 중지하여 일시적으로 비활성화합니다. 컴퓨팅 리소스만 종료되고 저장된 데이터(예: 애플리케이션 버전이 있는 컨테이너 이미지)는 그대로 유지됩니다. 서비스 재개가 빠릅니다. 애플리케이션을 새 컴퓨팅 리소스에 배포할 준비가 되었습니다. App Runner 도메인은 동일하게 유지됩니다.

App Runner 서비스를 삭제하여 영구적으로 제거합니다. 저장된 데이터가 삭제됩니다. 서비스를 다시 생성해야 하는 경우 App Runner는 소스를 다시 가져와야 하며 코드 리포지토리인 경우에도 빌드해야 합니다. 웹 애플리케이션은 새로운 App Runner 도메인을 가져옵니다.

서비스가 일시 중지된 경우

서비스를 일시 중지하고 일시 중지됨 상태인 경우 API 호출 또는 콘솔 작업을 포함한 작업 요청에 다르게 응답합니다. 서비스가 일시 중지된 경우에도 실행 시간에 영향을 미치는 방식으로 서비스의 정의 또는 구성을 수정하지 않는 App Runner 작업을 수행할 수 있습니다. 즉, 작업이 실행 중인 서비스의 동작, 규모 또는 기타 특성을 변경하는 경우 일시 중지된 서비스에서 해당 작업을 수행할 수 없습니다.

다음 목록은 일시 중지된 서비스에서 수행할 수 있는 API 작업에 대한 정보를 제공합니다. 동등한 콘솔 작업이 비슷하게 허용되거나 거부됩니다.

일시 중지된 서비스에 대해 수행할 수 있는 작업

- *`List*`* 및 *`Describe*`* 작업 - 정보만 읽는 작업입니다.
- *`DeleteService`* - 언제든지 서비스를 삭제할 수 있습니다.
- *`TagResource`*, *`UntagResource`* - 태그는 서비스와 연결되지만 정의의 일부가 아니며 런타임 동작에 영향을 주지 않습니다.

일시 중지된 서비스에서 수행할 수 없는 작업

- *`StartDeployment`* 작업(또는 콘솔을 사용한 수동 배포)

- *UpdateService* (또는 태깅 변경 사항을 제외하고 콘솔을 사용한 구성 변경)
- *CreateCustomDomainAssociations*, *DeleteCustomDomainAssociations*
- *CreateConnection*, *DeleteConnection*

서비스 일시 중지 및 재개

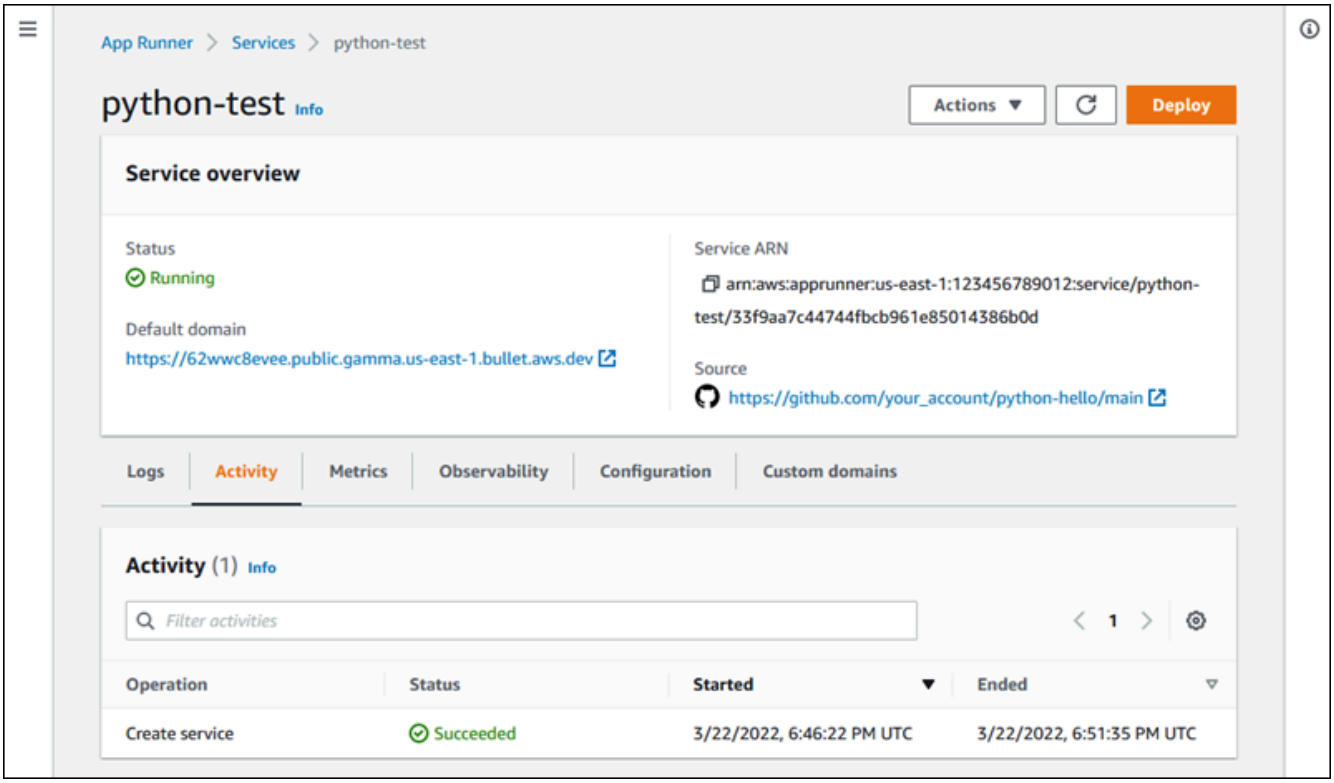
다음 방법 중 하나를 사용하여 App Runner 서비스를 일시 중지했다가 다시 시작합니다.

App Runner console

App Runner 콘솔을 사용하여 서비스를 일시 중지하려면

1. App Runner 콘솔을 열고 리전 목록에서를 선택합니다 AWS 리전.
2. 탐색 창에서 서비스를 선택한 다음 App Runner 서비스를 선택합니다.

   콘솔에 서비스 개요와 함께 서비스 대시보드가 표시됩니다.



3. 작업을 선택한 다음 일시 중지를 선택합니다.

   서비스 대시보드 페이지에서 서비스 상태가 진행 중인 작업으로 변경된 다음 일시 중지됨으로 변경됩니다. 이제 서비스가 일시 중지되었습니다.

App Runner 콘솔을 사용하여 서비스를 재개하려면

1. 작업을 선택한 다음 재개를 선택합니다.

   서비스 대시보드 페이지에서 서비스 상태가 진행 중인 작업으로 변경됩니다.
2. 서비스가 재개될 때까지 기다립니다. 서비스 대시보드 페이지에서 서비스 상태가 다시 실행 중으로 변경됩니다.
3. 서비스 재개가 성공했는지 확인하려면 서비스 대시보드 페이지에서 App Runner 도메인 값을 선택합니다. 서비스 웹 사이트의 URL입니다. 웹 애플리케이션이 올바르게 실행되고 있는지 확인합니다.

App Runner API or AWS CLI

App Runner API를 사용하여 서비스를 일시 중지하려면 PauseService API 작업을 AWS CLI 호출합니다. 호출이를 표시하는 서비스 객체와 함께 성공적인 응답을 반환하면 `"Status": "OPERATION_IN_PROGRESS"`App Runner가 서비스 일시 중지를 시작합니다.

App Runner API를 사용하여 서비스를 재개하려면 ResumeService API 작업을 AWS CLI호출합니다. 호출이를 표시하는 서비스 객체와 함께 성공적인 응답을 반환하면 `"Status": "OPERATION_IN_PROGRESS"`App Runner는 서비스 재개를 시작합니다.

App Runner 서비스 삭제

AWS App Runner 서비스에서 실행 중인 웹 애플리케이션을 종료하려는 경우 서비스를 삭제할 수 있습니다. 서비스를 삭제하면 실행 중인 웹 서비스가 중지되고, 기본 리소스가 제거되며, 연결된 데이터가 삭제됩니다.

다음 이유 중 하나 이상으로 App Runner 서비스를 삭제할 수 있습니다.

- 웹 애플리케이션이 더 이상 필요하지 않습니다. 예를 들어 사용 중지되었거나 사용한 개발 버전입니다.
- App Runner 서비스 할당량에 도달했습니다 - 동일한에서 새 서비스를 생성하려고 AWS 리전 하는데 계정과 연결된 할당량에 도달했습니다. 자세한 내용은 the section called “App Runner 리소스 할당량” 단원을 참조하십시오.
- 보안 또는 개인 정보 보호 고려 사항 - App Runner가 서비스에 대해 저장하는 데이터를 삭제하도록 합니다.

비교 일시 중지 및 삭제

App Runner 서비스를 일시 중지하여 일시적으로 비활성화합니다. 컴퓨팅 리소스만 종료되고 저장된 데이터(예: 애플리케이션 버전이 있는 컨테이너 이미지)는 그대로 유지됩니다. 서비스 재개가 빠릅니다. 애플리케이션을 새 컴퓨팅 리소스에 배포할 준비가 되었습니다. App Runner 도메인은 동일하게 유지됩니다.

App Runner 서비스를 삭제하여 영구적으로 제거합니다. 저장된 데이터가 삭제됩니다. 서비스를 다시 생성해야 하는 경우 App Runner는 소스를 다시 가져와야 하며 코드 리포지토리인 경우에도 빌드해야 합니다. 웹 애플리케이션은 새로운 App Runner 도메인을 가져옵니다.

App Runner는 무엇을 삭제하나요?

서비스를 삭제하면 App Runner는 일부 연결된 항목을 삭제하고 다른 항목은 삭제하지 않습니다. 다음 목록은 세부 정보를 제공합니다.

App Runner가 삭제하는 항목:

- 컨테이너 이미지 - 배포한 이미지 또는 App Runner가 소스 코드에서 빌드한 이미지의 사본입니다. App Runner가 소유 AWS 계정 한 내부를 사용하여 Amazon Elastic Container Registry(Amazon ECR)에 저장됩니다.
- 서비스 구성 - App Runner 서비스와 연결된 구성 설정입니다. App Runner가 소유 AWS 계정 한 내부를 사용하여 Amazon DynamoDB에 저장됩니다.

App Runner가 삭제하지 않는 항목:

- 연결 - 서비스와 연결된 연결이 있을 수 있습니다. App Runner 연결은 여러 App Runner 서비스 간에 공유될 수 있는 별도의 리소스입니다. 연결이 더 이상 필요하지 않은 경우 명시적으로 삭제할 수 있습니다. 자세한 내용은 the section called “연결” 단원을 참조하십시오.
- 사용자 지정 도메인 인증서 - 사용자 지정 도메인을 App Runner 서비스에 연결하는 경우 App Runner는 도메인 유효성을 추적하는 인증서를 내부적으로 생성합니다. (AWS Certificate Manager ACM)에 저장됩니다. App Runner는 도메인이 서비스에서 연결 해제된 후 또는 서비스가 삭제된 후 7일 동안 인증서를 삭제하지 않습니다. 자세한 내용은 the section called “사용자 지정 도메인 이름” 단원을 참조하십시오.

서비스 삭제

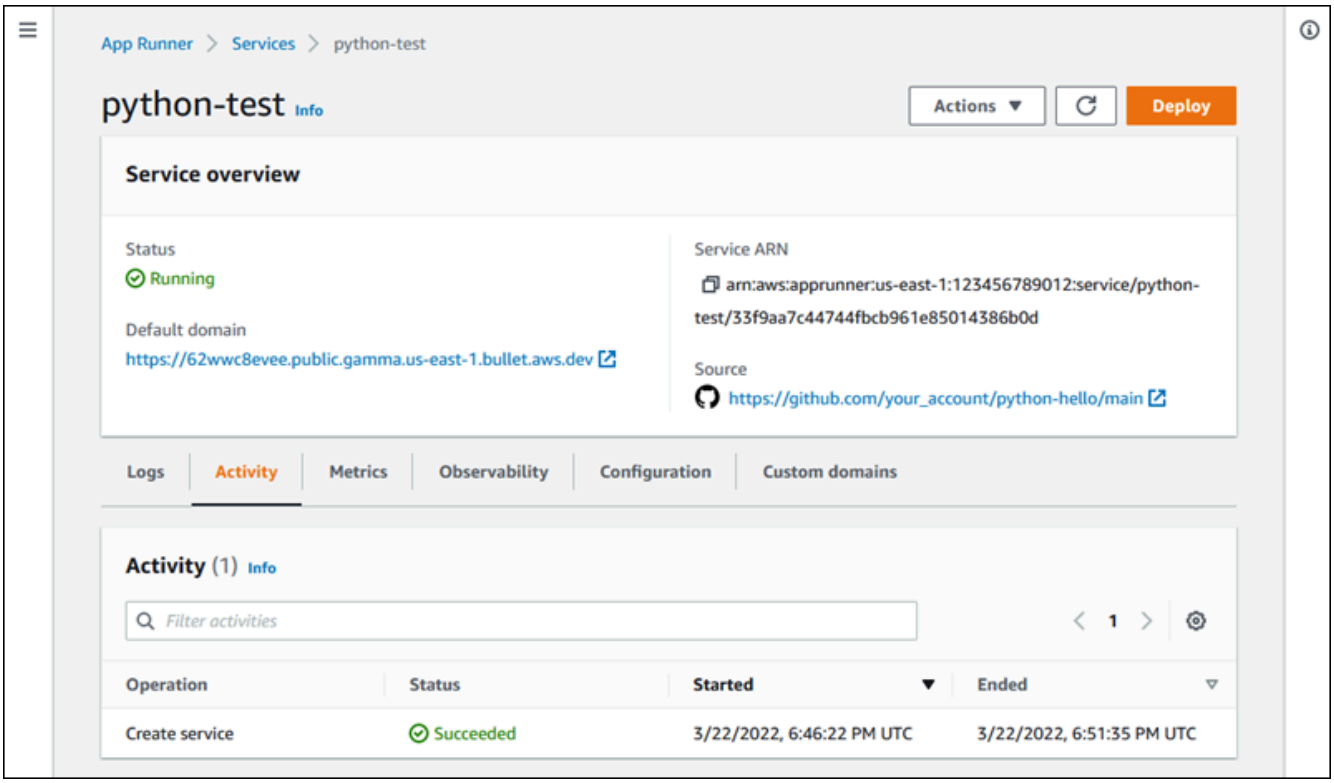
다음 방법 중 하나를 사용하여 App Runner 서비스를 삭제합니다.

App Runner console

App Runner 콘솔을 사용하여 서비스를 삭제하려면

1. App Runner 콘솔을 열고 리전 목록에서를 선택합니다 AWS 리전.
2. 탐색 창에서 서비스를 선택한 다음 App Runner 서비스를 선택합니다.

   콘솔에 서비스 개요와 함께 서비스 대시보드가 표시됩니다.



3. 작업을 선택한 후 삭제를 선택합니다.

   콘솔에서 서비스 페이지로 이동합니다. 삭제된 서비스는 작업 진행 중 상태로 표시된 후 목록에서 사라집니다. 이제 서비스가 삭제됩니다.

App Runner API or AWS CLI

App Runner API를 사용하여 서비스를 삭제하려면 DeleteService API 작업을 AWS CLI호출합니다. 호출이를 표시하는 서비스 객체와 함께 성공적인 응답을 반환하면 `"Status": "OPERATION_IN_PROGRESS"`App Runner가 서비스 삭제를 시작합니다.

환경 변수 참조

App Runner를 사용하면 서비스를 생성하거나 업데이트할 때 보안 암호 및 구성을 서비스의 환경 변수로 참조할 수 있습니다.

일반 텍스트의 제한 시간 및 재시도 횟수와 같은 민감하지 않은 구성 데이터를 키-값 페어로 참조할 수 있습니다. 일반 텍스트에서 참조하는 구성 데이터는 암호화되지 않으며 App Runner 서비스 구성 및 애플리케이션 로그에서 다른 사용자에게 표시됩니다.

> **Note**
>
> 보안상의 이유로 App Runner 서비스의 일반 텍스트에서 민감한 데이터를 참조하지 마세요.

민감한 데이터를 환경 변수로 참조

App Runner는 민감한 데이터를 서비스에서 환경 변수로 안전하게 참조할 수 있도록 지원합니다. AWS Secrets Manager 또는 AWS Systems Manager 파라미터 스토어에서 참조하려는 민감한 데이터를 저장하는 것이 좋습니다. 그런 다음 App Runner 콘솔에서 또는 API를 호출하여 서비스에서 환경 변수로 안전하게 참조할 수 있습니다. 이렇게 하면 보안 암호 및 파라미터 관리가 애플리케이션 코드 및 서비스 구성과 효과적으로 분리되어 App Runner에서 실행되는 애플리케이션의 전반적인 보안이 향상됩니다.

> **Note**
>
> App Runner는 Secrets Manager 및 SSM 파라미터 스토어를 환경 변수로 참조하는 데 따른 요금을 부과하지 않습니다. 하지만 Secrets Manager 및 SSM Parameter Store 사용에 대한 표준 요금을 지불합니다.
> 요금에 대한 자세한 내용은 다음을 참조하세요.
>
> - AWS Secrets Manager 요금
> - AWS SSM 파라미터 스토어 요금

다음은 민감한 데이터를 환경 변수로 참조하는 프로세스입니다.

1. API 키, 데이터베이스 자격 증명, 데이터베이스 연결 파라미터 또는 애플리케이션 버전과 같은 민감한 데이터를 AWS Secrets Manager 또는 AWS Systems Manager 파라미터 스토어에 보안 암호 또는 파라미터로 저장합니다.
2. App Runner가 Secrets Manager 및 SSM 파라미터 스토어에 저장된 보안 암호 및 파라미터에 액세스할 수 있도록 인스턴스 역할의 IAM 정책을 업데이트합니다. 자세한 내용은 권한 단원을 참조하십시오.
3. 이름을 할당하고 Amazon 리소스 이름(ARN)을 제공하여 보안 암호와 파라미터를 환경 변수로 안전하게 참조합니다. 서비스를 생성하거나 서비스 구성을 업데이트할 때 환경 변수를 추가할 수 있습니다. 다음 옵션 중 하나를 사용하여 환경 변수를 추가할 수 있습니다.
   - App Runner 콘솔
   - App Runner API
   - `apprunner.yaml` 구성 파일

   > **Note**
   >
   > App Runner 서비스를 생성하거나 업데이트할 때 환경 변수의 `PORT` 이름으로를 할당할 수 없습니다. App Runner 서비스를 위한 예약 환경 변수입니다.

   보안 암호 및 파라미터를 참조하는 방법에 대한 자세한 내용은 환경 변수 관리를 참조하세요.

> **Note**
>
> App Runner는 보안 암호 및 파라미터 ARNs에 대한 참조만 저장하므로 민감한 데이터는 App Runner 서비스 구성 및 애플리케이션 로그에서 다른 사용자에게 표시되지 않습니다.

고려 사항

- AWS Secrets Manager 또는 파라미터 스토어의 보안 암호 및 파라미터에 AWS Systems Manager 액세스할 수 있는 적절한 권한으로 인스턴스 역할을 업데이트해야 합니다. 자세한 내용은 권한 단원을 참조하십시오.
- AWS Systems Manager 파라미터 스토어가 시작 또는 업데이트하려는 서비스와 동일한 AWS 계정에 있는지 확인합니다. 현재는 계정 간에 SSM Parameter Store 파라미터를 참조할 수 없습니다.

- 보안 암호와 파라미터 값이 교체되거나 변경되면 App Runner 서비스에서 자동으로 업데이트되지 않습니다. App Runner는 배포 중에 보안 암호와 파라미터만 가져오므로 App Runner 서비스를 재배포합니다.
- App Runner 서비스의 SDK를 통해 AWS Secrets Manager 및 AWS Systems Manager 파라미터 스토어를 직접 호출할 수도 있습니다.
- 오류를 방지하려면 환경 변수로 참조할 때 다음 사항을 확인하세요.
  - 보안 암호의 올바른 ARN을 지정합니다.
  - 파라미터의 올바른 이름 또는 ARN을 지정합니다.

권한

AWS Secrets Manager 또는 SSM 파라미터 스토어에 저장된 보안 암호 및 파라미터 참조를 활성화하려면 인스턴스 역할의 IAM 정책에 적절한 권한을 추가하여 Secrets Manager 및 SSM 파라미터 스토어에 액세스합니다.

> **Note**
>
> App Runner는 권한 없이 계정의 리소스에 액세스할 수 없습니다. IAM 정책을 업데이트하여 권한을 제공합니다.

다음 정책 템플릿을 사용하여 IAM 콘솔에서 인스턴스 역할을 업데이트할 수 있습니다. 특정 요구 사항에 맞게 이러한 정책 템플릿을 수정할 수 있습니다. 인스턴스 역할 업데이트에 대한 자세한 내용은 IAM 사용 설명서의 역할 수정을 참조하세요.

> **Note**
>
> 환경 변수를 생성할 때 App Runner 콘솔에서 다음 템플릿을 복사할 수도 있습니다.

다음 템플릿을 인스턴스 역할에 복사하여의 보안 암호를 참조할 수 있는 권한을 추가합니다AWS Secrets Manager.

JSON

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

```
    "Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "secretsmanager:GetSecretValue",
            "kms:Decrypt*"
        ],
        "Resource": [
            "arn:aws:secretsmanager:us-east-1:111122223333:secret:my-secret",
            "arn:aws:kms:us-east-1:111122223333:key/my-key"
         ]
     }
   ]
}
```

다음 템플릿을 인스턴스 역할에 복사하여 AWS Systems Manager Parameter Store에서 파라미터를 참조할 수 있는 권한을 추가합니다.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ssm:GetParameters"
            ],
            "Resource": [
            "arn:aws:ssm:us-east-1:111122223333:parameter/my-parameter"
            ]
        }
    ]
}
```

환경 변수 관리

다음 방법 중 하나를 사용하여 App Runner 서비스의 환경 변수를 관리합니다.

- the section called “App Runner 콘솔”

- the section called “App Runner API 또는 AWS CLI”

App Runner 콘솔

App Runner 콘솔에서 서비스를 생성하거나 업데이트할 때 환경 변수를 추가할 수 있습니다.

환경 변수 추가

환경 변수를 추가하려면

1. App Runner 콘솔을 열고 리전 목록에서를 선택합니다 AWS 리전.
2. 서비스 생성 또는 업데이트 여부에 따라 다음 단계 중 하나를 수행합니다.
   - 새 서비스를 생성하는 경우 App Runner 서비스 생성을 선택하고 서비스 구성으로 이동합니다.
   - 기존 서비스를 업데이트하는 경우 업데이트할 서비스를 선택하고 서비스의 구성 탭으로 이동합니다.
3. 서비스 설정에서 환경 변수 - 선택 사항으로 이동합니다.
4. 요구 사항에 따라 다음 옵션 중 하나를 선택합니다.
   - 환경 변수 소스에서 일반 텍스트를 선택하고 환경 변수 이름 및 환경 변수 값 아래에 각각 해당 키-값 페어를 입력합니다.

     > **Note**
     >
     > 민감하지 않은 데이터를 참조하려면 일반 텍스트를 선택합니다. 이 데이터는 암호화되지 않으며 App Runner 서비스 구성 및 애플리케이션 로그에서 다른 사용자에게 표시됩니다.

   - 환경 변수 소스에서 Secrets Manager를 선택하여에 저장된 보안 암호를 서비스의 환경 변수 AWS Secrets Manager 로 참조합니다. 환경 변수 이름 및 환경 변수 값 아래에 참조하려는 보안 암호의 환경 변수 이름과 Amazon 리소스 이름(ARN)을 각각 입력합니다.
   - 환경 변수 소스에서 SSM 파라미터 스토어를 선택하여 SSM 파라미터 스토어에 저장된 파라미터를 서비스의 환경 변수로 참조합니다. 환경 변수 이름 및 환경 변수 값 아래에 참조하려는 파라미터의 환경 변수 이름과 ARN을 각각 입력합니다.

> **Note**
>
> - App Runner 서비스를 생성하거나 업데이트할 때 환경 변수의 `PORT` 이름으로를 할당할 수 없습니다. App Runner 서비스를 위한 예약 환경 변수입니다.
> - SSM Parameter Store 파라미터가 시작하려는 AWS 리전 서비스와 동일한에 있는 경우 전체 Amazon 리소스 이름(ARN) 또는 파라미터 이름을 지정할 수 있습니다. 파라미터가 다른 리전에 있는 경우 전체 ARN을 지정해야 합니다.
> - 참조하는 파라미터가 시작하거나 업데이트하는 서비스와 동일한 계정에 있는지 확인합니다. 현재는 계정 간에 SSM 파라미터 스토어 파라미터를 참조할 수 없습니다.

5. 환경 변수 추가를 선택하여 다른 환경 변수를 참조합니다.
6. IAM 정책 템플릿을 확장하여 AWS Secrets Manager 및 SSM 파라미터 스토어에 제공된 IAM 정책 템플릿을 보고 복사합니다. 필요한 권한으로 인스턴스 역할의 IAM 정책을 아직 업데이트하지 않은 경우에만이 작업을 수행하면 됩니다. 자세한 내용은 권한 단원을 참조하십시오.

환경 변수 제거

환경 변수를 삭제하기 전에 애플리케이션 코드가 동일하게 업데이트되었는지 확인합니다. 애플리케이션 코드가 업데이트되지 않으면 App Runner 서비스가 실패할 수 있습니다.

환경 변수를 제거하려면

1. App Runner 콘솔을 열고 리전 목록에서를 선택합니다 AWS 리전.
2. 업데이트하려는 서비스의 구성 탭으로 이동합니다.
3. 서비스 설정에서 환경 변수 - 선택 사항으로 이동합니다.
4. 제거하려는 환경 변수 옆에 있는 제거를 선택합니다. 삭제를 확인하는 메시지가 표시됩니다.
5. 삭제를 선택합니다.

App Runner API 또는 AWS CLI

서비스에 환경 변수로 추가하여 Secrets Manager 및 SSM 파라미터 스토어에 저장된 민감한 데이터를 참조할 수 있습니다.

> **Note**
>
> App Runner가 Secrets Manager 및 SSM 파라미터 스토어에 저장된 보안 암호 및 파라미터에 액세스할 수 있도록 인스턴스 역할의 IAM 정책을 업데이트합니다. 자세한 내용은 권한 단원을 참조하십시오.

보안 암호 및 구성을 환경 변수로 참조하려면

1. Secrets Manager 또는 SSM 파라미터 스토어에서 보안 암호 또는 구성을 생성합니다.

   다음 예제에서는 SSM 파라미터 스토어를 사용하여 보안 암호와 파라미터를 생성하는 방법을 보여줍니다.

   Example보안 암호 생성 - 요청

   다음 예제에서는 데이터베이스 자격 증명을 나타내는 보안 암호를 생성하는 방법을 보여줍니다.

   ```
   aws secretsmanager create-secret \
   –name DevRdsCredentials \
   –description "Rds credentials for development account." \
   –secret-string "{\"user\":\"diegor\",\"password\":\"EXAMPLE-PASSWORD\"}"
   ```

   Example보안 암호 생성 - 응답

   ```
   arn:aws:secretsmanager:<region>:<aws_account_id>:secret:DevRdsCredentials
   ```

   Example구성 생성 - 요청

   다음 예제에서는 RDS 연결 문자열을 나타내는 파라미터를 생성하는 방법을 보여줍니다.

   ```
   aws systemsmanager put-parameter \
   –name DevRdsConnectionString \
   –value "mysql2://dev-mysqlcluster-rds.com:3306/diegor" \
   –type "String" \
   –description "Rds connection string for development account."
   ```

Example구성 생성 - 응답

```
arn:aws:ssm:<region>:<aws_account_id>:parameter/DevRdsConnectionString
```

2. 환경 변수로 추가하여 Secrets Manager 및 SSM Parameter Store에 저장된 보안 암호 및 구성을 참조합니다. App Runner 서비스를 생성하거나 업데이트할 때 환경 변수를 추가할 수 있습니다.

   다음 예제에서는 암호 및 구성을 코드 기반 및 이미지 기반 App Runner 서비스의 환경 변수로 참조하는 방법을 보여줍니다.

   Example이미지 기반 App Runner 서비스용 Input.json 파일

```
{
  "ServiceName": "example-secrets",
  "SourceConfiguration": {
    "ImageRepository": {
      "ImageIdentifier": "<image-identifier>",
      "ImageConfiguration": {
        "Port": "<port>",
        "RuntimeEnvironmentSecrets": {

 "Credential1":"arn:aws:secretsmanager:<region>:<aws_account_id>:secret:XXXXXXXXXXXX",
          "Credential2":"arn:aws:ssm:<region>:<aws_account_id>:parameter/
<parameter-name>"
        }
      },
      "ImageRepositoryType": "ECR_PUBLIC"
    }
  },
  "InstanceConfiguration": {
    "Cpu": "1 vCPU",
    "Memory": "3 GB",
    "InstanceRoleArn": "<instance-role-arn>"
  }
}
```

   Example이미지 기반 App Runner 서비스 - 요청

```
aws apprunner create-service \
--cli-input-json file://input.json
```

Example이미지 기반 App Runner 서비스 - 응답

```
{
...
      "ImageRepository": {
         "ImageIdentifier":"<image-identifier>",
         "ImageConfiguration":{
            "Port": "<port>",
            "RuntimeEnvironmentSecrets":{
               "Credential1":
 "arn:aws:secretsmanager:<region>:<aws_account_id>:secret:XXXXXXXXXXXX",
               "Credential2": "arn:aws:ssm:<region>:<aws_account_id>:parameter/
<parameter-name>"
            },
            "ImageRepositoryType":"ECR"
      }
   },
   "InstanceConfiguration": {
        "CPU": "1 vCPU",
        "Memory": "3 GB",
        "InstanceRoleArn: "<instance-role-arn>"
   }
...
}
```

Example코드 기반 App Runner 서비스를 위한 Input.json 파일

```
{
  "ServiceName": "example-secrets",
  "SourceConfiguration": {
    "AuthenticationConfiguration": {
      "ConnectionArn": "arn:aws:apprunner:us-east-1:123456789012:connection/my-
github-connection/XXXXXXXXXX"
    },
    "AutoDeploymentsEnabled": false,
    "CodeRepository": {
      "RepositoryUrl": "<repository-url>",
      "SourceCodeVersion": {
        "Type": "BRANCH",
        "Value": "main"
      },
      "CodeConfiguration": {
```

```
        "ConfigurationSource": "API",
        "CodeConfigurationValues": {
          "Runtime": "<runtime>",
          "BuildCommand": "<build-command>",
          "StartCommand": "<start-command>",
          "Port": "<port>",
          "RuntimeEnvironmentSecrets": {

 "Credential1":"arn:aws:secretsmanager:<region>:<aws_account_id>:secret:XXXXXXXXXXXX",
            "Credential2":"arn:aws:ssm:<region>:<aws_account_id>:parameter/
<parameter-name>"
          }
        }
      }
    }
  },
  "InstanceConfiguration": {
    "Cpu": "1 vCPU",
    "Memory": "3 GB",
    "InstanceRoleArn": "<instance-role-arn>"
  }
}
```

Example코드 기반 App Runner 서비스 - 요청

```
aws apprunner create-service \
--cli-input-json file://input.json
```

Example코드 기반 App Runner 서비스 - 응답

```
{
...
   "SourceConfiguration":{
      "CodeRepository":{
         "RepositoryUrl":"<repository-url>",
         "SourceCodeVersion":{
            "Type":"Branch",
            "Value":"main"
         },
         "CodeConfiguration":{
            "ConfigurationSource":"API",
            "CodeConfigurationValues":{
```

```
                "Runtime":"<runtime>",
                "BuildCommand":"<build-command>",
                "StartCommand":"<start-command>",
                "Port":"<port>",
                "RuntimeEnvironmentSecrets":{
                   "Credential1" :
 "arn:aws:secretsmanager:<region>:<aws_account_id>:secret:XXXXXXXX",
                   "Credential2" : "arn:aws:ssm:<region>:<aws_account_id>:parameter/
<parameter-name>"
                }
             }
          }
      },
      "InstanceConfiguration": {
        "CPU": "1 vCPU",
        "Memory": "3 GB",
        "InstanceRoleArn: "<instance-role-arn>"
   }
...
}
```

3. 추가된 보안 암호를 반영하도록 apprunner.yaml 모델이 업데이트됩니다.

   다음은 업데이트된 apprunner.yaml모델의 예입니다.

   Example **apprunner.yaml**

```
version: 1.0
runtime: python3
build:
  commands:
    build:
      - python -m pip install flask
run:
  command: python app.py
  network:
    port: 8080
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
  secrets:
    - name: my-secret
      value-from:
 "arn:aws:secretsmanager:<region>:<aws_account_id>:secret:XXXXXXXXXXXX"
```

```
    - name: my-parameter
      value-from: "arn:aws:ssm:<region>:<aws_account_id>:parameter/<parameter-
name>"
    - name: my-parameter-only-name
      value-from: "parameter-name"
```

App Runner를 사용한 네트워킹

이 장에서는 AWS App Runner 서비스의 네트워킹 구성에 대해 설명합니다.

이 장에서는 다음 내용을 알아봅니다.

- 프라이빗 및 퍼블릭 엔드포인트에 대한 수신 트래픽을 구성하는 방법입니다. 자세한 내용은 수신 트래픽에 대한 네트워킹 구성 설정을 참조하세요.
- Amazon VPC에서 실행되는 다른 애플리케이션에 액세스하도록 발신 트래픽을 구성하는 방법입니다. 자세한 내용은 발신 트래픽에 대한 VPC 액세스 활성화를 참조하세요.

주제

용어

필요에 맞게 네트워크 트래픽을 사용자 지정하는 방법을 알아보기 위해이 장에서 사용되는 다음 용어를 살펴보겠습니다.

일반 용어

Amazon Virtual Private Cloud(VPC)와 연결하는 데 필요한 사항을 알아보기 위해 다음 용어를 이해해 보겠습니다.

- VPC: Amazon VPC는 리소스 배치, 연결 및 보안을 포함하여 가상 네트워킹 환경을 완벽하게 제어할 수 있는 논리적으로 격리된 가상 네트워크입니다. 자체 데이터 센터에서 운영하는 기존 네트워크와 매우 유사한 가상 네트워크입니다.
- VPC 인터페이스 엔드포인트: AWS PrivateLink 리소스인 VPC 인터페이스 엔드포인트는 VPC를 엔드포인트 서비스에 연결합니다. VPC 인터페이스 엔드포인트를 생성하여 Network Load Balancer를 사용하여 트래픽을 분산하는 엔드포인트 서비스로 트래픽을 전송합니다. 엔드포인트 서비스로 전송되는 트래픽은 DNS를 사용하여 확인됩니다.
- 리전: 각 리전은 App Runner 서비스를 호스팅할 수 있는 별도의 지리적 영역입니다.

- 가용 영역: 가용 영역은 AWS 리전 내의 격리된 위치입니다. 중복 전원, 네트워킹 및 연결이 있는 하나 이상의 개별 데이터 센터입니다. 가용 영역을 통해 프로덕션 애플리케이션은 고가용성, 내결함성 및 확장성을 갖춥니다.
- 서브넷: 서브넷은 VPC의 IP 주소 범위입니다. 서브넷은 단일 가용 영역에 상주해야 합니다. AWS 리소스를 지정된 서브넷으로 시작할 수 있습니다. 인터넷에 연결되어야 하는 리소스에는 퍼블릭 서브넷을 사용하고, 인터넷에 연결되지 않는 리소스에는 프라이빗 서브넷을 사용하십시오.
- 보안 그룹: 보안 그룹은 연결된 리소스에 도달하고 나갈 수 있는 트래픽을 제어합니다. 보안 그룹은 각 서브넷의 AWS 리소스를 보호하기 위한 추가 보안 계층을 제공하므로 네트워크 트래픽을 더 잘 제어할 수 있습니다. VPC를 생성할 경우 VPC는 기본 보안 그룹과 함께 제공됩니다. 각 VPC 대해 추가 보안 그룹을 생성할 수 있습니다. 보안 그룹은 보안 그룹이 생성된 VPC 내의 리소스에만 연결할 수 있습니다.
- 듀얼 스택: 듀얼 스택은 IPv4 및 IPv6 엔드포인트 모두의 네트워크 트래픽을 지원하는 주소 유형입니다.

발신 트래픽 구성과 관련된 기간

VPC 커넥터

VPC 커넥터는 App Runner 서비스가 프라이빗 Amazon VPC에서 실행되는 애플리케이션에 액세스할 수 있도록 하는 App Runner 리소스입니다.

수신 트래픽 구성과 관련된 용어

Amazon VPC 내에서만 서비스에 비공개로 액세스할 수 있도록 하는 방법을 알아보기 위해 다음 용어를 살펴보겠습니다.

- VPC 수신 연결: VPC 수신 연결은 수신 트래픽에 대한 App Runner 엔드포인트를 제공하는 App Runner 리소스입니다. App Runner 콘솔에서 수신 트래픽에 프라이빗 엔드포인트를 선택하면 App Runner가 백그라운드에서 VPC Ingress Connection 리소스를 할당합니다. VPC 수신 연결 리소스는 App Runner 서비스를 Amazon VPC의 VPC 인터페이스 엔드포인트에 연결합니다.

  > **Note**
  >
  > App Runner API를 사용하는 경우 VPC Ingress Connection 리소스가 자동으로 생성되지 않습니다.

- 프라이빗 엔드포인트: 프라이빗 엔드포인트는 Amazon VPC 내에서만 액세스할 수 있도록 수신 네트워크 트래픽을 구성하도록 선택하는 App Runner 콘솔 옵션입니다.

수신 트래픽에 대한 네트워킹 구성 설정

프라이빗 또는 퍼블릭 엔드포인트에서 들어오는 트래픽을 수신하도록 서비스를 구성할 수 있습니다.

퍼블릭 엔드포인트는 기본 구성입니다. 퍼블릭 인터넷에서 들어오는 모든 트래픽에 대한 서비스를 엽니다. 또한 서비스의 IPv4 또는 듀얼 스택(IPv4 및 IPv6) 주소 유형 중에서 선택할 수 있는 유연성도 제공합니다.

프라이빗 엔드포인트는 Amazon VPC의 트래픽만 App Runner 서비스에 액세스할 수 있도록 허용합니다. 이는 App Runner 서비스에 대한 리소스인 AWS PrivateLink VPC 인터페이스 엔드포인트를 설정하여 달성할 수 있습니다. 이렇게 하면 Amazon VPC와 App Runner 서비스 간에 프라이빗 연결이 생성됩니다. 또한 서비스의 IPv4 또는 듀얼 스택(IPv4 및 IPv6) 주소 유형 중에서 선택할 수 있는 유연성도 제공합니다.

다음은 수신 트래픽에 대한 네트워크 구성 설정의 일부로 다루는 주제입니다.

- Amazon VPC 내에서만 서비스를 비공개로 사용할 수 있도록 수신 트래픽을 구성하는 방법입니다. 자세한 내용은 수신 트래픽에 프라이빗 엔드포인트 활성화를 참조하세요.
- 듀얼 스택 주소 유형에서 인터넷 트래픽을 수신하도록 서비스를 구성하는 방법입니다. 자세한 내용은 퍼블릭 수신 트래픽에 듀얼 스택 활성화를 참조하세요.

헤더

App Runner를 사용하면 애플리케이션에 들어오는 트래픽의 원본 소스 IPv4 및 IPv6 주소에 액세스할 수 있습니다. 원본 소스 IP 주소는 `X-Forwarded-For` 요청 헤더를 할당하여 보존됩니다. 이렇게 하면 필요한 경우 애플리케이션이 원본 소스 IP 주소를 가져올 수 있습니다.

> **Note**
>
> 서비스가 프라이빗 엔드포인트를 사용하도록 구성된 경우 `X-Forwarded-For` 요청 헤더를 사용하여 원래 소스 IP 주소에 액세스할 수 없습니다. 사용되는 경우 false 값을 검색합니다.

수신 트래픽에 프라이빗 엔드포인트 활성화

기본적으로 AWS App Runner 서비스를 생성할 때 인터넷을 통해 서비스에 액세스할 수 있습니다. 그러나 App Runner 서비스를 프라이빗으로 설정하고 Amazon Virtual Private Cloud(Amazon VPC) 내에서만 액세스할 수 있습니다.

App Runner 서비스를 비공개로 사용하면 수신 트래픽을 완벽하게 제어하여 보안 계층을 추가할 수 있습니다. 이는 내부 APIs, 기업 웹 애플리케이션 또는 더 높은 수준의 개인 정보 보호 및 보안이 필요하거나 특정 규정 준수 요구 사항을 충족해야 하는 개발 중인 애플리케이션을 실행하는 등 다양한 사용 사례에 유용합니다.

> **Note**
>
> App Runner 애플리케이션에 소스 IP/CIDR 수신 트래픽 제어 규칙이 필요한 경우 WAF 웹 ACLs 대신 프라이빗 엔드포인트에 대한 보안 그룹 규칙을 사용해야 합니다. 이는 현재 요청 소스 IP 데이터를 WAF와 연결된 App Runner 프라이빗 서비스로 전달하는 것을 지원하지 않기 때문입니다. 따라서 WAF 웹 ACLs과 연결된 App Runner 프라이빗 서비스에 대한 소스 IP 규칙은 IP 기반 규칙을 준수하지 않습니다.
> 모범 사례를 포함하여 인프라 보안 및 보안 그룹에 대해 자세히 알아보려면 Amazon VPC 사용 설명서의 보안 그룹을 사용하여 네트워크 트래픽 제어 및 AWS 리소스에 대한 트래픽 제어를 참조하세요. https://docs.aws.amazon.com//vpc/latest/userguide/vpc-security-groups.html

App Runner 서비스가 프라이빗인 경우 Amazon VPC 내에서 서비스에 액세스할 수 있습니다. 인터넷 게이트웨이, NAT 디바이스 또는 VPN 연결은 필요하지 않습니다.

> **Note**
>
> App Runner는 수신 트래픽과 발신 트래픽 모두에 대해 IPv4 및 듀얼 스택(IPv4 및 IPv6 모두)을 지원합니다.

고려 사항

- App Runner용 VPC 인터페이스 엔드포인트를 설정하기 전에 AWS PrivateLink 가이드의 고려 사항을 검토하세요.
- VPC 엔드포인트 정책은 App Runner에서 지원되지 않습니다. 기본적으로 VPC 인터페이스 엔드포인트를 통해 App Runner에 대한 전체 액세스가 허용됩니다. 또는 보안 그룹을 엔드포인트 네트워크 인터페이스와 연결하여 VPC 인터페이스 엔드포인트를 통해 App Runner에 대한 트래픽을 제어할 수 있습니다.
- App Runner 애플리케이션에 소스 IP/CIDR 수신 트래픽 제어 규칙이 필요한 경우 WAF 웹 ACLs 대신 프라이빗 엔드포인트에 대한 보안 그룹 규칙을 사용해야 합니다. 이는 현재 요청 소스 IP 데이터를 WAF와 연결된 App Runner 프라이빗 서비스로 전달하는 것을 지원하지 않기 때문입니다. 따라서

WAF 웹 ACLs과 연결된 App Runner 프라이빗 서비스에 대한 소스 IP 규칙은 IP 기반 규칙을 준수하지 않습니다.

- 프라이빗 엔드포인트를 활성화한 후에는 VPC에서만 서비스에 액세스할 수 있으며 인터넷에서는 액세스할 수 없습니다.
- 가용성을 높이려면 가용 영역에서 VPC 인터페이스 엔드포인트에 대해 서로 다른 서브넷을 두 개 이상 선택하는 것이 좋습니다. 서브넷은 하나만 사용하지 않는 것이 좋습니다.
- IP 주소 유형에 대해 듀얼 스택 옵션을 선택하는 경우 서브넷이 듀얼 스택 트래픽을 지원할 수 있는지 확인합니다.
- 동일한 VPC 인터페이스 엔드포인트를 사용하여 VPC의 여러 App Runner 서비스에 액세스할 수 있습니다.

이 섹션에 사용되는 용어에 대한 자세한 내용은 용어를 참조하세요.

권한

다음은 프라이빗 엔드포인트를 활성화하는 데 필요한 권한 목록입니다.

- ec2:CreateTags
- ec2:CreateVpcEndpoint
- ec2:ModifyVpcEndpoint
- ec2:DeleteVpcEndpoints
- ec2:DescribeSubnets
- ec2:DescribeVpcEndpoints
- ec2:DescribeVpcs

VPC 인터페이스 엔드포인트

VPC 인터페이스 엔드포인트는 Amazon VPC를 엔드포인트 서비스에 연결하는 AWS PrivateLink 리소스입니다. VPC 인터페이스 엔드포인트를 전달하여 App Runner 서비스에 액세스할 Amazon VPC를 지정할 수 있습니다. VPC 인터페이스 엔드포인트를 생성하려면 다음을 지정합니다.

- 연결을 활성화하는 Amazon VPC입니다.
- 보안 그룹을 추가합니다. 기본적으로 보안 그룹은 VPC 인터페이스 엔드포인트에 할당됩니다. 사용자 지정 보안 그룹을 연결하여 수신 네트워크 트래픽을 추가로 제어할 수 있습니다.

- 서브넷을 추가합니다. 가용성을 높이려면 App Runner 서비스에 액세스할 각 가용 영역에 대해 최소 두 개의 서브넷을 선택하는 것이 좋습니다. 네트워크 인터페이스 엔드포인트는 VPC 인터페이스 엔드포인트에 대해 활성화하는 각 서브넷에 생성됩니다. 이는 App Runner로 향하는 트래픽의 진입점 역할을 하는 요청자 관리형 네트워크 인터페이스입니다. 요청자 관리형 네트워크 인터페이스는 AWS 서비스가 사용자를 대신하여 VPC에 생성하는 네트워크 인터페이스입니다.
- API를 사용하는 경우 App Runner VPC 인터페이스 엔드포인트를 추가합니다`Servicename`. 예:

```
com.amazonaws.region.apprunner.requests
```

다음 AWS 서비스 중 하나를 사용하여 VPC 인터페이스 엔드포인트를 생성할 수 있습니다.

- App Runner 콘솔. 자세한 내용은 프라이빗 엔드포인트 관리를 참조하세요.
- Amazon VPC 콘솔 또는 API, 및 AWS Command Line Interface (AWS CLI). 자세한 내용은 AWS PrivateLink 안내서의 AWS PrivateLink를 통해 AWS 서비스 에 액세스를 참조하세요.

> **Note**
>
> AWS PrivateLink 요금에 따라 사용하는 각 VPC 인터페이스 엔드포인트에 대해 요금이 부과됩니다. 따라서 비용 효율성을 높이기 위해 동일한 VPC 인터페이스 엔드포인트를 사용하여 VPC 내의 여러 App Runner 서비스에 액세스할 수 있습니다. 그러나 더 나은 격리를 위해 각 App Runner 서비스에 대해 서로 다른 VPC 인터페이스 엔드포인트를 연결하는 것이 좋습니다.

VPC Ingress Connection

VPC 수신 연결은 수신 트래픽에 대한 App Runner 엔드포인트를 지정하는 App Runner 리소스입니다. App Runner는 수신 트래픽에 대해 App Runner 콘솔에서 프라이빗 엔드포인트를 선택하면 백그라운드에서 VPC Ingress Connection 리소스를 할당합니다. Amazon VPC의 트래픽만 App Runner 서비스에 액세스하도록 허용하려면이 옵션을 선택합니다. VPC 수신 연결 리소스는 App Runner 서비스를 Amazon VPC의 VPC 인터페이스 엔드포인트에 연결합니다. API 작업을 사용하여 수신 트래픽에 대한 네트워크 설정을 구성하는 경우에만 VPC Ingress Connection 리소스를 생성할 수 있습니다. VPC Ingress Connection 리소스를 생성하는 방법에 대한 자세한 내용은 API 참조의 CreateVpcIngressConnection을 참조하세요. AWS App Runner

> **Note**
>
> App Runner의 VPC 수신 연결 리소스 하나는 Amazon VPC의 VPC 인터페이스 엔드포인트 하나에 연결할 수 있습니다. 또한 각 App Runner 서비스에 대해 하나의 VPC Ingress Connection 리소스만 생성할 수 있습니다.

프라이빗 엔드포인트

프라이빗 엔드포인트는 Amazon VPC에서 들어오는 트래픽만 수신하려는 경우 선택할 수 있는 App Runner 콘솔 옵션입니다. App Runner 콘솔에서 프라이빗 엔드포인트 옵션을 선택하면 VPC 인터페이스 엔드포인트를 구성하여 서비스를 VPC에 연결할 수 있는 옵션이 제공됩니다. 백그라운드에서 App Runner는 구성한 VPC 인터페이스 엔드포인트에 VPC Ingress Connection 리소스를 할당합니다.

요약

Amazon VPC의 트래픽만 App Runner 서비스에 액세스하도록 허용하여 서비스를 비공개로 설정합니다. 이를 위해 App Runner 또는 Amazon VPC를 사용하여 선택한 Amazon VPC에 대한 VPC 인터페이스 엔드포인트를 생성합니다. App Runner 콘솔에서 수신 트래픽에 프라이빗 엔드포인트를 활성화할 때 VPC 인터페이스 엔드포인트를 생성합니다. 그런 다음 App Runner는 VPC Ingress Connection 리소스를 자동으로 생성하고 VPC 인터페이스 엔드포인트 및 App Runner 서비스에 연결합니다. 이렇게 하면 선택한 VPC의 트래픽만 App Runner 서비스에 액세스할 수 있는 프라이빗 서비스 연결이 생성됩니다.

프라이빗 엔드포인트 관리

다음 방법 중 하나를 사용하여 수신 트래픽에 대한 프라이빗 엔드포인트를 관리합니다.

- the section called “App Runner 콘솔”
- the section called “App Runner API 또는 AWS CLI”

> **Note**
>
> App Runner 애플리케이션에 소스 IP/CIDR 수신 트래픽 제어 규칙이 필요한 경우 WAF 웹 ACLs 대신 프라이빗 엔드포인트에 대한 보안 그룹 규칙을 사용해야 합니다. 이는 현재 요청 소스 IP 데이터를 WAF와 연결된 App Runner 프라이빗 서비스로 전달하는 것을 지원하지 않기 때문입니다. 따라서 WAF 웹 ACLs과 연결된 App Runner 프라이빗 서비스에 대한 소스 IP 규칙은 IP 기반 규칙을 준수하지 않습니다.

모범 사례를 포함하여 인프라 보안 및 보안 그룹에 대해 자세히 알아보려면 Amazon VPC 사용 설명서의 보안 그룹을 사용하여 네트워크 트래픽 제어 및 AWS 리소스에 대한 트래픽 제어를 참조하세요. https://docs.aws.amazon.com//vpc/latest/userguide/vpc-security-groups.html

App Runner 콘솔

App Runner 콘솔을 사용하여 서비스를 생성하거나 나중에 구성을 업데이트할 때 수신 트래픽을 구성하도록 선택할 수 있습니다.

수신 트래픽을 구성하려면 다음 중 하나를 선택합니다.

- 퍼블릭 엔드포인트: 인터넷을 통해 모든 서비스에 액세스할 수 있도록 합니다. 기본적으로 퍼블릭 엔드포인트가 선택됩니다.
- 프라이빗 엔드포인트: Amazon VPC 내에서만 App Runner 서비스에 액세스할 수 있도록 합니다.

프라이빗 엔드포인트 활성화

액세스하려는 Amazon VPC의 VPC 인터페이스 엔드포인트와 연결하여 프라이빗 엔드포인트를 활성화합니다. 새 VPC 인터페이스 엔드포인트를 생성하거나 기존 엔드포인트를 선택할 수 있습니다.

VPC 인터페이스 엔드포인트를 생성하려면

1. App Runner 콘솔을 열고 리전 목록에서를 선택합니다 AWS 리전.
2. 서비스 구성 아래의 네트워킹 섹션으로 이동합니다.
3. 수신 네트워크 트래픽에 대해 프라이빗 엔드포인트를 선택합니다. VPC 인터페이스 엔드포인트를 사용하여 VCP에 연결하는 옵션이 열립니다.
4. 새 엔드포인트 생성을 선택합니다. 새 VPC 인터페이스 엔드포인트 생성 대화 상자가 열립니다.
5. VPC 인터페이스 엔드포인트의 이름을 입력합니다.
6. 사용 가능한 드롭다운 목록에서 필요한 VPC 인터페이스 엔드포인트를 선택합니다.
7. 드롭다운 목록에서 보안 그룹을 선택합니다. 보안 그룹을 추가하면 VPC 인터페이스 엔드포인트에 추가 보안 계층이 제공됩니다. 둘 이상의 보안 그룹을 선택하는 것이 좋습니다. 보안 그룹을 선택하지 않으면 App Runner는 VPC 인터페이스 엔드포인트에 기본 보안 그룹을 할당합니다. 보안 그룹 규칙이 App Runner 서비스와 통신하려는 리소스를 차단하지 않는지 확인합니다. 보안 그룹 규칙은 App Runner 서비스와 상호 작용하는 리소스를 허용해야 합니다.

> **Note**
>
> App Runner 애플리케이션에 소스 IP/CIDR 수신 트래픽 제어 규칙이 필요한 경우 WAF 웹 ACLs 대신 프라이빗 엔드포인트에 대한 보안 그룹 규칙을 사용해야 합니다. 이는 현재 요청 소스 IP 데이터를 WAF와 연결된 App Runner 프라이빗 서비스로 전달하는 것을 지원하지 않기 때문입니다. 따라서 WAF 웹 ACLs과 연결된 App Runner 프라이빗 서비스에 대한 소스 IP 규칙은 IP 기반 규칙을 준수하지 않습니다.
> 모범 사례를 포함하여 인프라 보안 및 보안 그룹에 대해 자세히 알아보려면 Amazon VPC 사용 설명서의 보안 그룹을 사용하여 네트워크 트래픽 제어 및 AWS 리소스에 대한 트래픽 제어를 참조하세요. https://docs.aws.amazon.com//vpc/latest/userguide/vpc-security-groups.html

8. 드롭다운 목록에서 필요한 서브넷을 선택합니다. App Runner 서비스에 액세스할 각 가용 영역에 대해 최소 두 개의 서브넷을 선택하는 것이 좋습니다.

> **Note**
>
> 듀얼 스택에 대한 엔드포인트를 구성하는 경우 인프라 및 VPC 엔드포인트가 듀얼 스택 트래픽을 지원하는지 확인합니다.

9. (선택 사항) 새 태그 추가를 선택하고 태그 키와 태그 값을 입력합니다.
10. 생성(Create)을 선택합니다. 서비스 구성 페이지가 열리고 상단 표시줄에 VPC 인터페이스 엔드포인트가 성공적으로 생성되었다는 메시지가 표시됩니다.

기존 VPC 인터페이스 엔드포인트를 선택하려면

1. App Runner 콘솔을 열고 리전 목록에서를 선택합니다 AWS 리전.
2. 서비스 구성 아래의 네트워킹 섹션으로 이동합니다.
3. 수신 네트워크 트래픽에 대해 프라이빗 엔드포인트를 선택합니다. VPC 인터페이스 엔드포인트를 사용하여 VPC에 연결하는 옵션이 열립니다. 사용 가능한 VPC 인터페이스 엔드포인트 목록이 표시됩니다.
4. VPC 인터페이스 엔드포인트 아래에 나열된 필수 VPC 인터페이스 엔드포인트를 선택합니다.
5. 다음을 선택하여 서비스를 생성합니다. App Runner는 프라이빗 엔드포인트를 활성화합니다.

> **Note**
>
> 서비스가 생성된 후 필요한 경우 VPC 인터페이스 엔드포인트와 연결된 보안 그룹 및 서브넷을 편집하도록 선택할 수 있습니다.

프라이빗 엔드포인트의 세부 정보를 확인하려면 서비스로 이동하여 구성 탭 아래의 네트워킹 섹션을 확장합니다. 프라이빗 엔드포인트와 연결된 VPC 및 VPC 인터페이스 엔드포인트의 세부 정보를 보여줍니다.

VPC 인터페이스 엔드포인트 업데이트

App Runner 서비스가 생성된 후 프라이빗 엔드포인트와 연결된 VPC 인터페이스 엔드포인트를 편집할 수 있습니다.

> **Note**
>
> 엔드포인트 이름과 VPC 필드는 업데이트할 수 없습니다.

VPC 인터페이스 엔드포인트를 업데이트하려면

1. App Runner 콘솔을 열고 리전 목록에서를 선택합니다 AWS 리전.
2. 서비스로 이동하여 왼쪽 패널에서 네트워킹 구성을 선택합니다.
3. 수신 트래픽을 선택하여 각 서비스와 연결된 VPC 인터페이스 엔드포인트를 확인합니다.
4. 편집하려는 VPC 인터페이스 엔드포인트를 선택합니다.
5. 편집을 선택합니다. VPC 인터페이스 엔드포인트를 편집하는 대화 상자가 열립니다.
6. 필요한 보안 그룹 및 서브넷을 선택하고 업데이트를 클릭합니다. VPC 인터페이스 엔드포인트 세부 정보가 표시된 페이지가 열리고 상단 표시줄에 VPC 인터페이스 엔드포인트가 성공적으로 업데이트되었다는 메시지가 표시됩니다.

> **Note**
>
> App Runner 애플리케이션에 소스 IP/CIDR 수신 트래픽 제어 규칙이 필요한 경우 WAF 웹 ACLs 대신 프라이빗 엔드포인트에 대한 보안 그룹 규칙을 사용해야 합니다. 이는 현재 요청 소스 IP 데이터를 WAF와 연결된 App Runner 프라이빗 서비스로 전달하는 것을 지원

하지 않기 때문입니다. 따라서 WAF 웹 ACLs과 연결된 App Runner 프라이빗 서비스에 대한 소스 IP 규칙은 IP 기반 규칙을 준수하지 않습니다.
모범 사례를 포함하여 인프라 보안 및 보안 그룹에 대해 자세히 알아보려면 Amazon VPC 사용 설명서의 보안 그룹을 사용하여 네트워크 트래픽 제어 및 AWS 리소스에 대한 트래픽 제어를 참조하세요. https://docs.aws.amazon.com//vpc/latest/userguide/vpc-security-groups.html

VPC 인터페이스 엔드포인트 삭제

App Runner 서비스에 비공개로 액세스하지 않으려면 수신 트래픽을 퍼블릭으로 설정할 수 있습니다. 퍼블릭으로 변경하면 프라이빗 엔드포인트가 제거되지만 VPC 인터페이스 엔드포인트는 삭제되지 않습니다.

VPC 인터페이스 엔드포인트를 삭제하려면

1. App Runner 콘솔을 열고 리전 목록에서를 선택합니다 AWS 리전.
2. 서비스로 이동하여 왼쪽 패널에서 네트워킹 구성을 선택합니다.
3. 수신 트래픽을 선택하여 각 서비스와 연결된 VPC 인터페이스 엔드포인트를 확인합니다.

> **Note**
>
> VPC 인터페이스 엔드포인트를 삭제하기 전에 서비스를 업데이트하여 연결된 모든 서비스에서 제거합니다.

4. 삭제를 선택합니다.

   VPC 인터페이스 엔드포인트에 연결된 서비스가 있는 경우 VPC 인터페이스 엔드포인트를 삭제할 수 없음 메시지가 표시됩니다. VPC 인터페이스 엔드포인트에 연결된 서비스가 없는 경우 삭제를 확인하는 메시지가 표시됩니다.

5. 삭제를 선택합니다. 수신 트래픽에 대한 네트워크 구성 페이지가 열리고 상단 표시줄에 VPC 인터페이스 엔드포인트가 성공적으로 삭제되었다는 메시지가 표시됩니다.

App Runner API 또는 AWS CLI

Amazon VPC 내에서만 액세스할 수 있는 애플리케이션을 App Runner에 배포할 수 있습니다.

서비스를 비공개로 만드는 데 필요한 권한에 대한 자세한 내용은 섹션을 참조하세요the section called "권한".

Amazon VPC에 대한 프라이빗 서비스 연결을 생성하려면

1. VPC 인터페이스 엔드포인트인 AWS PrivateLink 리소스를 생성하여 App Runner에 연결합니다. 이렇게 하려면 애플리케이션과 연결할 서브넷 및 보안 그룹을 지정합니다. 다음은 VPC 인터페이스 엔드포인트를 생성하는 예제입니다.

   > **Note**
   >
   > App Runner 애플리케이션에 소스 IP/CIDR 수신 트래픽 제어 규칙이 필요한 경우 WAF 웹 ACLs 대신 프라이빗 엔드포인트에 대한 보안 그룹 규칙을 사용해야 합니다. 이는 현재 요청 소스 IP 데이터를 WAF와 연결된 App Runner 프라이빗 서비스로 전달하는 것을 지원하지 않기 때문입니다. 따라서 WAF 웹 ACLs과 연결된 App Runner 프라이빗 서비스에 대한 소스 IP 규칙은 IP 기반 규칙을 준수하지 않습니다.
   > 모범 사례를 포함하여 인프라 보안 및 보안 그룹에 대해 자세히 알아보려면 Amazon VPC 사용 설명서의 보안 그룹을 사용하여 네트워크 트래픽 제어 및 AWS 리소스에 대한 트래픽 제어를 참조하세요. https://docs.aws.amazon.com//vpc/latest/userguide/vpc-security-groups.html

   Example

   ```
   aws ec2 create-vpc-endpoint
    --vpc-endpoint-type: Interface
    --service-name: com.amazonaws.us-east-1.apprunner.requests
    --subnets: subnet1, subnet2
    --security-groups: sg1
   ```

2. CLI를 통해 CreateService 또는 UpdateService App Runner API 작업을 사용하여 VPC 인터페이스 엔드포인트를 참조합니다. 공개적으로 액세스할 수 없도록 서비스를 구성합니다. `NetworkConfiguration` 파라미터의 `IngressConfiguration` 멤버`False`에서를 `IsPubliclyAccessible`로 설정합니다. 선택적으로 `IpAddressType` 필드를 `IPV4` 또는 로 설정할 수 있습니다`DUAL_STACK`. 설정하지 않으면이 값은 기본적으로 IPV4로 설정됩니다. 다음 예제에서는 VPC 인터페이스 엔드포인트를 참조합니다.

Example

```
aws apprunner create-service
 --network-configuration:
   {
   "IngressConfiguration":
      {
      "IsPubliclyAccessible": False
      },
      "IpAddressType": "IPV4"
   }
 --service-name: com.amazonaws.us-east-1.apprunner.requests
 --source-configuration: <source_configuration>
```

3. `create-vpc-ingress-connection` API 작업을 호출하여 App Runner에 대한 VPC Ingress Connection 리소스를 생성하고 이전 단계에서 생성한 VPC 인터페이스 엔드포인트와 연결합니다. 지정된 VPC에서 서비스에 액세스하는 데 사용되는 도메인 이름을 반환합니다. 다음은 VPC Ingress Connection 리소스를 생성하는 예제입니다.

   Example요청

   ```
   aws apprunner create-vpc-ingress-connection
    --service-arn: <apprunner_service_arn>
    --ingress-vpc-configuration: {"VpcId":<vpc_id>, "VpceId": <vpce_id>}
    --vpc-ingress-connection-name: <vic_connection_name>
   ```

   Example응답

   ```
   {
      "VpcIngressConnectionArn": <vpc_ingress_connection_arn>,
      "VpcIngressConnectionName": <vic_connection_name>,
      "ServiceArn": <apprunner_service_arn>,
      "Status": "PENDING_CREATION",
      "AccountId": <connection_owner_id>,
      "DomainName": <domain_name_associated_with_vpce>,
      "IngressVpcConfiguration": {"VpcId":<vpc_id>, "VpceId":<vpce_id>},
      "CreatedAt": <date_created>
   }
   ```

VPC 수신 연결 업데이트

VPC Ingress Connection 리소스를 업데이트할 수 있습니다. VPC 수신 연결을 업데이트하려면 다음 상태 중 하나여야 합니다.

- AVAILABLE
- FAILED_CREATION
- FAILED_UPDATE

다음은 VPC Ingress Connection 리소스를 업데이트하는 예입니다.

Example요청

```
aws apprunner update-vpc-ingress-connection
      --vpc-ingress-connection-arn: <vpc_ingress_connection_arn>
```

Example응답

```
{
    "VpcIngressConnectionArn":  <vpc_ingress_connection_arn>,
    "VpcIngressConnectionName":  <vic_connection_name>,
    "ServiceArn":  <apprunner_service_arn>,
    "Status": "FAILED_UPDATE",
    "AccountId":  <connection_owner_id>,
    "DomainName":  <domain_name_associated_with_vpce>,
    "IngressVpcConfiguration": {"VpcId":<vpc_id>, "VpceId":<vpce_id>},
    "CreatedAt":  <date_created>
}
```

VPC 수신 연결 삭제

Amazon VPC에 대한 프라이빗 연결이 더 이상 필요하지 않은 경우 VPC 수신 연결 리소스를 삭제할 수 있습니다.

VPC 수신 연결을 삭제하려면 다음 상태 중 하나여야 합니다.

- AVAILABLE
- 실패한 생성
- 업데이트 실패

- 실패한 삭제

다음은 VPC 수신 연결을 삭제하는 예입니다.

Example요청

```
aws apprunner delete-vpc-ingress-connection
      --vpc-ingress-connection-arn: <vpc_ingress_connection_arn>
```

Example응답

```
{
   "VpcIngressConnectionArn": <vpc_ingress_connection_arn>,
   "VpcIngressConnectionName": <vic_connection_name>,
   "ServiceArn": <apprunner_service_arn>,
   "Status": "PENDING_DELETION",
   "AccountId": <connection_owner_id>,
   "DomainName": <domain_name_associated_with_vpce>,
   "IngressVpcConfiguration": {"VpcId":<vpc_id>, "VpceId":<vpce_id>},
   "CreatedAt": <date_created>,
   "DeletedAt": <date_deleted>
}
```

다음 App Runner API 작업을 사용하여 서비스의 프라이빗 인바운드 트래픽을 관리합니다.

- CreateVpcIngressConnection - 새 VPC Ingress Connection 리소스를 생성합니다. App Runner는 App Runner 서비스를 Amazon VPC 엔드포인트에 연결하려는 경우 이 리소스가 필요합니다.
- ListVpcIngressConnections - AWS 계정과 연결된 AWS App Runner VPC Ingress Connection 엔드포인트 목록을 반환합니다.
- DescribeVpcIngressConnection - AWS App Runner VPC Ingress Connection 리소스에 대한 전체 설명을 반환합니다.
- UpdateVpcIngressConnection - AWS App Runner VPC Ingress Connection 리소스를 업데이트합니다.
- DeleteVpcIngressConnection - App Runner 서비스와 연결된 App Runner VPC Ingress Connection 리소스를 삭제합니다.

App Runner API 사용에 대한 자세한 내용은 App Runner API 참조 가이드를 참조하세요.

수신 트래픽에 대해 IPv6 활성화

서비스가 IPv6 주소 또는 IPv4 및 IPv6 주소 모두에서 수신 네트워크 트래픽을 수신하도록 하려면 엔드포인트의 듀얼 스택 주소 유형을 IPv6 선택합니다. 새 애플리케이션을 생성할 때 서비스 구성 > 네트워킹 섹션에서이 설정을 찾을 수 있습니다. 다음 절차에서는 App Runner 콘솔 또는 App Runner API를 사용하여 IPv4 또는 듀얼 스택(IPv6 및 IPv4)을 활성화하는 방법을 설명합니다.

수신 트래픽에 대한 듀얼 스택 관리

다음 방법 중 하나를 사용하여 수신 트래픽의 듀얼 스택 주소 유형을 관리합니다.

- the section called “App Runner 콘솔”
- the section called “App Runner API 또는 AWS CLI”

> **Note**
>
> 다음 절차에서는 퍼블릭 수신 트래픽의 네트워크 주소 유형을 관리하는 방법을 설명합니다. 프라이빗 엔드포인트의 듀얼 스택 또는 IPv4 주소 유형 관리에 대한 자세한 내용은 섹션을 참조하세요the section called “프라이빗 엔드포인트 관리”.

App Runner 콘솔

App Runner 콘솔을 사용하여 서비스를 생성하거나 나중에 구성을 업데이트할 때 수신 인터넷 트래픽에 대해 듀얼 스택 주소 유형을 선택할 수 있습니다.

듀얼 스택 주소 유형을 활성화하려면

1. 서비스를 생성하거나 업데이트할 때 서비스 구성 아래의 네트워킹 섹션을 확장합니다.
2. 수신 네트워크 트래픽에 대해 퍼블릭 엔드포인트를 선택합니다. 퍼블릭 엔드포인트를 선택하면 엔드포인트 IP 주소 유형 옵션이 열립니다.

   프라이빗 엔드포인트의 듀얼 스택 또는 IPv4 주소 유형을 관리하는 the section called “프라이빗 엔드포인트 관리” 절차는 섹션을 참조하세요.
3. 엔드포인트 IP 주소 유형을 확장하여 다음 IP 주소 유형을 확인합니다.

   - IPv4
   - 듀얼 스택(IPv4 및 IPv6)

> **Note**
>
> 엔드포인트 IP 주소 유형을 확장하여 선택하지 않으면 App Runner가 IPv4를 기본 구성으로 할당합니다.

4. 듀얼 스택(IPv4 및 IPv6)을 선택합니다.
5. 서비스를 생성하는 경우 다음을 선택한 후 생성 및 배포를 선택합니다. 또는 서비스를 업데이트하는 경우 변경 사항 저장을 선택합니다.

   서비스가 배포되면 애플리케이션이 IPv4 및 IPv6 엔드포인트 모두에서 네트워크 트래픽을 수신하기 시작합니다.

주소 유형을 변경하려면

1. 단계에 따라 서비스를 업데이트하고 네트워킹으로 이동합니다.
2. 수신 네트워크 트래픽에서 엔드포인트 IP 주소 유형으로 이동하여 필요한 주소 유형을 선택합니다.
3. 변경 사항 저장을 선택합니다. 선택한 항목으로 서비스가 업데이트됩니다.

App Runner API 또는 AWS CLI

CreateService 또는 UpdateService App Runner API 작업을 호출할 때 `NetworkConfiguration` 파라미터의 `IpAddressType` 멤버를 사용하여 주소 유형을 지정합니다. 지정할 수 있는 지원되는 값은 `IPv4` 및 입니다`DUAL_STACK`. 서비스가 IPv4 및 IPv6 엔드포인트에서 인터넷 트래픽을 수신할`DUAL_STACK`지 여부를 지정합니다. 에 값을 지정하지 않으면 `IpAddressType`기본적으로 IPv4가 적용됩니다.

> **Note**
>
> 프라이빗 엔드포인트 예제는 섹션을 참조하세요the section called “App Runner API 또는 AWS CLI”.

다음은 듀얼 스택을 IP 주소로 사용하여 서비스를 생성하는 예제입니다. 이 예제에서는 `input.json` 파일을 호출합니다.

Example듀얼 스택을 지원하는 서비스 생성 요청

```
aws apprunner create-service \
 --cli-input-json file://input.json
```

Example **input.json**의 콘텐츠

```
{
  "ServiceName": "example-service",
  "SourceConfiguration": {
    "ImageRepository": {
      "ImageIdentifier": "public.ecr.aws/aws-containers/hello-app-runner:latest",
      "ImageConfiguration": {
        "Port": "8000"
      },
      "ImageRepositoryType": "ECR_PUBLIC"
    },
    "NetworkConfiguration": {
      "IpAddressType": "DUAL_STACK"
    }
  }
}
```

Example응답

```
{
  "Service": {
    "ServiceName": "example-service",
    "ServiceId": "<service-id>",
    "ServiceArn": "arn:aws:apprunner:us-east-2:123456789012:service/example-
service/<service-id>",
    "ServiceUrl": "1234567890.us-east-2.awsapprunner.com",
    "CreatedAt": "2023-10-16T12:30:51.724000-04:00",
    "UpdatedAt": "2023-10-16T12:30:51.724000-04:00",
    "Status": "OPERATION_IN_PROGRESS",
    "SourceConfiguration": {
      "ImageRepository": {
        "ImageIdentifier": "public.ecr.aws/aws-containers/hello-app-runner:latest",
        "ImageConfiguration": {
          "Port": "8000"
        },
        "ImageRepositoryType": "ECR_PUBLIC"
```

```
      },
      "AutoDeploymentsEnabled": false
    },
    "InstanceConfiguration": {
      "Cpu": "1024",
      "Memory": "2048"
    },
    "HealthCheckConfiguration": {
      "Protocol": "TCP",
      "Path": "/",
      "Interval": 5,
      "Timeout": 2,
      "HealthyThreshold": 1,
      "UnhealthyThreshold": 5
    },
    "AutoScalingConfigurationSummary": {
      "AutoScalingConfigurationArn": "arn:aws:apprunner:us-
east-2:123456789012:autoscalingconfiguration/
DefaultConfiguration/1/00000000000000000000000000000001",
      "AutoScalingConfigurationName": "DefaultConfiguration",
      "AutoScalingConfigurationRevision": 1
    },
    "NetworkConfiguration": {
      "IpAddressType": "DUAL_STACK",
      "EgressConfiguration": {
        "EgressType": "DEFAULT"
      },
      "IngressConfiguration": {
        "IsPubliclyAccessible": true
      }
    }
  },
  "OperationId": "24bd100b1e111ae1a1f0e1115c4f11de"
}
```

API 파라미터에 대한 자세한 내용은 NetworkConfiguration을 참조하세요.

발신 트래픽에 대한 VPC 액세스 활성화

기본적으로 AWS App Runner 애플리케이션은 퍼블릭 엔드포인트로 메시지를 보낼 수 있습니다. 여기에는 자체 솔루션 AWS 서비스과 기타 퍼블릭 웹 사이트 또는 웹 서비스가 포함됩니다. 애플리케이션은 Amazon Virtual Private Cloud(VPC)에서 VPC에서 실행되는 애플리케이션의 퍼블릭 엔드포인트로

메시지를 보낼 수도 있습니다. 환경을 시작할 때 VPC를 구성하지 않으면 App Runner는 퍼블릭인 기본 VPC를 사용합니다.

사용자 지정 VPC에서 환경을 시작하도록 선택하여 송신 트래픽에 대한 네트워킹 및 보안 설정을 사용자 지정할 수 있습니다. AWS App Runner 서비스가 Amazon Virtual Private Cloud(Amazon VPC)의 프라이빗 VPC에서 실행되는 애플리케이션에 액세스하도록 할 수 있습니다. 이렇게 하면 애플리케이션이에 연결하여 Amazon Virtual Private Cloud(Amazon VPC)에서 호스팅되는 다른 애플리케이션에 메시지를 보낼 수 있습니다. 예를 들어 Amazon RDS 데이터베이스, Amazon ElastiCache 및 프라이빗 VPC에서 호스팅되는 기타 프라이빗 서비스가 있습니다.

VPC 커넥터

App Runner 콘솔에서 VPC 커넥터라는 VPC 엔드포인트를 생성하여 서비스를 VPC와 연결할 수 있습니다. VPC 커넥터를 생성하려면 VPC, 하나 이상의 서브넷 및 선택적으로 하나 이상의 보안 그룹을 지정합니다. VPC 커넥터를 구성한 후 하나 이상의 App Runner 서비스와 함께 사용할 수 있습니다.

일회성 지연 시간

아웃바운드 트래픽에 대한 사용자 지정 VPC 커넥터로 App Runner 서비스를 구성하는 경우 2~5분의 일회성 시작 지연 시간이 발생할 수 있습니다. 시작 프로세스는 VPC 커넥터가 다른 리소스에 연결할 준비가 될 때까지 기다렸다가 서비스 상태를 실행 중으로 설정합니다. 서비스를 처음 생성할 때 사용자 지정 VPC 커넥터를 사용하여 서비스를 구성하거나 나중에 서비스 업데이트를 수행하여 구성할 수 있습니다.

다른 서비스에 대해 동일한 VPC 커넥터 구성을 재사용하면 지연 시간이 발생하지 않습니다. VPC 커넥터 구성은 보안 그룹과 서브넷 조합을 기반으로 합니다. 지정된 VPC 커넥터 구성의 경우 지연 시간은 VPC 커넥터 Hyperplane ENIs(탄력적 네트워크 인터페이스)를 처음 생성하는 동안 한 번만 발생합니다.

사용자 지정 VPC 커넥터 및 AWS Hyperplane에 대해 자세히 알아보기

App Runner의 VPC 커넥터는 Network Load Balancer, NAT Gateway, AWS PrivateLink와 같은 여러 AWS 리소스 뒤에 있는 내부 Amazon 네트워크 시스템인 AWS Hyperplane을 기반으로 합니다. AWS Hyperplane 기술은 높은 공유 수준과 함께 높은 처리량과 짧은 지연 시간 기능을 제공합니다. VPC 커넥터를 생성하고 서비스와 연결할 때 서브넷에 Hyperplane ENI가 생성됩니다. VPC 커넥터 구성은 보안 그룹과 서브넷 조합을 기반으로 하며 여러 App Runner 서비스에서 동일한 VPC 커넥터를 참조할 수 있습니다. 따라서 기본 Hyperplane ENIs App Runner 서비스에서 공유됩니다. 요청 로드를 처리하는 데 필요한 작업 수를 확장하여 VPC의 IP 공간을 보다 효율적으로 사용할 수 있게 되더라도 이러한 공유가 가능합니다. 자세한 내용은 AWS 컨테이너 블로그의 AWS App Runner VPC 네트워킹에 대한 심층 분석을 참조하세요.

서브넷

각 서브넷은 특정 가용 영역에 있습니다. 고가용성을 위해 최소 3개의 가용 영역에서 서브넷을 선택하는 것이 좋습니다. 리전에 가용 영역이 3개 미만인 경우 지원되는 모든 가용 영역에서 서브넷을 선택하는 것이 좋습니다.

VPC의 서브넷을 선택할 때는 퍼블릭 서브넷이 아닌 프라이빗 서브넷을 선택해야 합니다. 이는 VPC 커넥터를 생성할 때 App Runner 서비스가 각 서브넷에 Hyperplane ENI를 생성하기 때문입니다. 각 Hyperplane ENI에는 프라이빗 IP 주소만 할당되고 AWSAppRunnerManaged 키의 태그로 태그가 지정됩니다. 퍼블릭 서브넷을 선택하면 App Runner 서비스를 실행할 때 오류가 발생합니다. 그러나 서비스가 인터넷 또는 다른 퍼블릭에 있는 일부 서비스에 액세스해야 하는 경우 섹션을 AWS 서비스참조하세요the section called “서브넷 선택 시 고려 사항”.

서브넷 선택 시 고려 사항

- 서비스를 VPC에 연결하면 아웃바운드 트래픽이 퍼블릭 인터넷에 액세스할 수 없습니다. 애플리케이션의 모든 아웃바운드 트래픽은 서비스가 연결된 VPC를 통해 전달됩니다. VPC에 대한 모든 네트워킹 규칙은 애플리케이션의 아웃바운드 트래픽에 적용됩니다. 즉, 서비스가 퍼블릭 인터넷 및 AWS APIs에 액세스할 수 없습니다. 액세스 권한을 얻으려면 다음 중 하나를 수행합니다.
  - NAT 게이트웨이를 통해 서브넷을 인터넷에 연결합니다.
  - 액세스 AWS 서비스 하려는에 대한 VPC 엔드포인트를 설정합니다. 를 사용하여 서비스가 Amazon VPC 내에 유지됩니다 AWS PrivateLink.
- 일부의 일부 가용 영역은 App Runner 서비스와 함께 사용할 수 있는 서브넷을 지원하지 AWS 리전 않습니다. 이러한 가용 영역에서 서브넷을 선택하면 서비스가 생성되거나 업데이트되지 않습니다. 이러한 상황에서 App Runner는 지원되지 않는 서브넷 및 가용 영역을 가리키는 자세한 오류 메시지를 제공합니다. 이 경우 요청에서 지원되지 않는 서브넷을 제거하여 문제를 해결한 다음 다시 시도하세요.
- 선택한 서브넷은 모두 IPv4 또는 듀얼 스택과 동일한 IP 주소 유형을 가져야 합니다.

보안 그룹

선택적으로 App Runner가 AWS 지정된 서브넷에서에 액세스하는 데 사용하는 보안 그룹을 지정할 수 있습니다. 보안 그룹을 지정하지 않으면 App Runner는 VPC의 기본 보안 그룹을 사용합니다. 기본 보안 그룹은 모든 아웃바운드 트래픽을 허용합니다.

보안 그룹을 추가하면 VPC 커넥터에 추가 보안 계층이 제공되므로 네트워크 트래픽을 더 잘 제어할 수 있습니다. VPC 커넥터는 애플리케이션의 아웃바운드 통신에만 사용됩니다. 아웃바운드 규칙을 사용

하여 원하는 대상 엔드포인트와의 통신을 허용합니다. 또한 대상 리소스와 연결된 모든 보안 그룹에 적절한 인바운드 규칙이 있는지 확인해야 합니다. 그렇지 않으면 이러한 리소스는 VPC 커넥터 보안 그룹에서 오는 트래픽을 수락할 수 없습니다.

> **Note**
>
> 서비스를 VPC와 연결해도 다음 트래픽은 영향을 받지 않습니다.
>
> - 인바운드 트래픽 - 애플리케이션이 수신하는 수신 메시지는 연결된 VPC의 영향을 받지 않습니다. 메시지는 서비스와 연결되어 있고 VPC와 상호 작용하지 않는 퍼블릭 도메인 이름을 통해 라우팅됩니다.
> - App Runner 트래픽 - App Runner는 소스 코드 및 이미지 가져오기, 로그 푸시, 보안 암호 검색 등 사용자를 대신하여 여러 작업을 관리합니다. 이러한 작업이 생성하는 트래픽은 VPC를 통해 라우팅되지 않습니다.

가 Amazon VPC와 AWS App Runner 통합되는 방법에 대한 자세한 내용은 AWS App Runner VPC 네트워킹을 참조하세요.

VPC 액세스 관리

> **Note**
>
> 서비스에 대한 아웃바운드 트래픽 VPC 커넥터를 생성하는 경우 다음 서비스 시작 프로세스에는 일회성 지연 시간이 발생합니다. 새 서비스를 생성할 때 또는 이후에 서비스 업데이트를 통해이 구성을 설정할 수 있습니다. 자세한 내용은이 가이드일회성 지연 시간의 App Runner를 사용한 네트워킹 장의 섹션을 참조하세요.

다음 방법 중 하나를 사용하여 App Runner 서비스에 대한 VPC 액세스를 관리합니다.

App Runner console

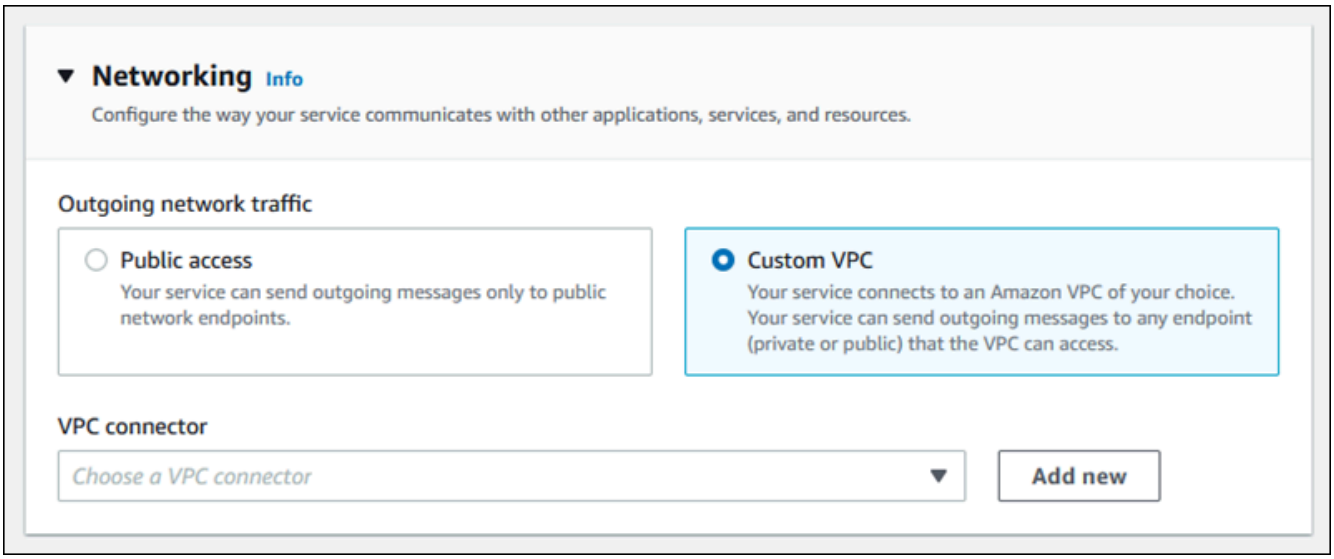
App Runner 콘솔을 사용하여 서비스를 생성하거나 나중에 구성을 업데이트할 때 발신 트래픽을 구성하도록 선택할 수 있습니다. 콘솔 페이지에서 네트워킹 구성 섹션을 찾습니다. 발신 네트워크 트래픽의 경우 다음에서를 선택합니다.

- 퍼블릭 액세스: 서비스를 다른의 퍼블릭 엔드포인트와 연결합니다 AWS 서비스.

- 사용자 지정 VPC: 서비스를 Amazon VPC의 VPC와 연결합니다. 애플리케이션은 Amazon VPC에서 호스팅되는 다른 애플리케이션과 연결하고 메시지를 보낼 수 있습니다.

사용자 지정 VPC를 활성화하려면

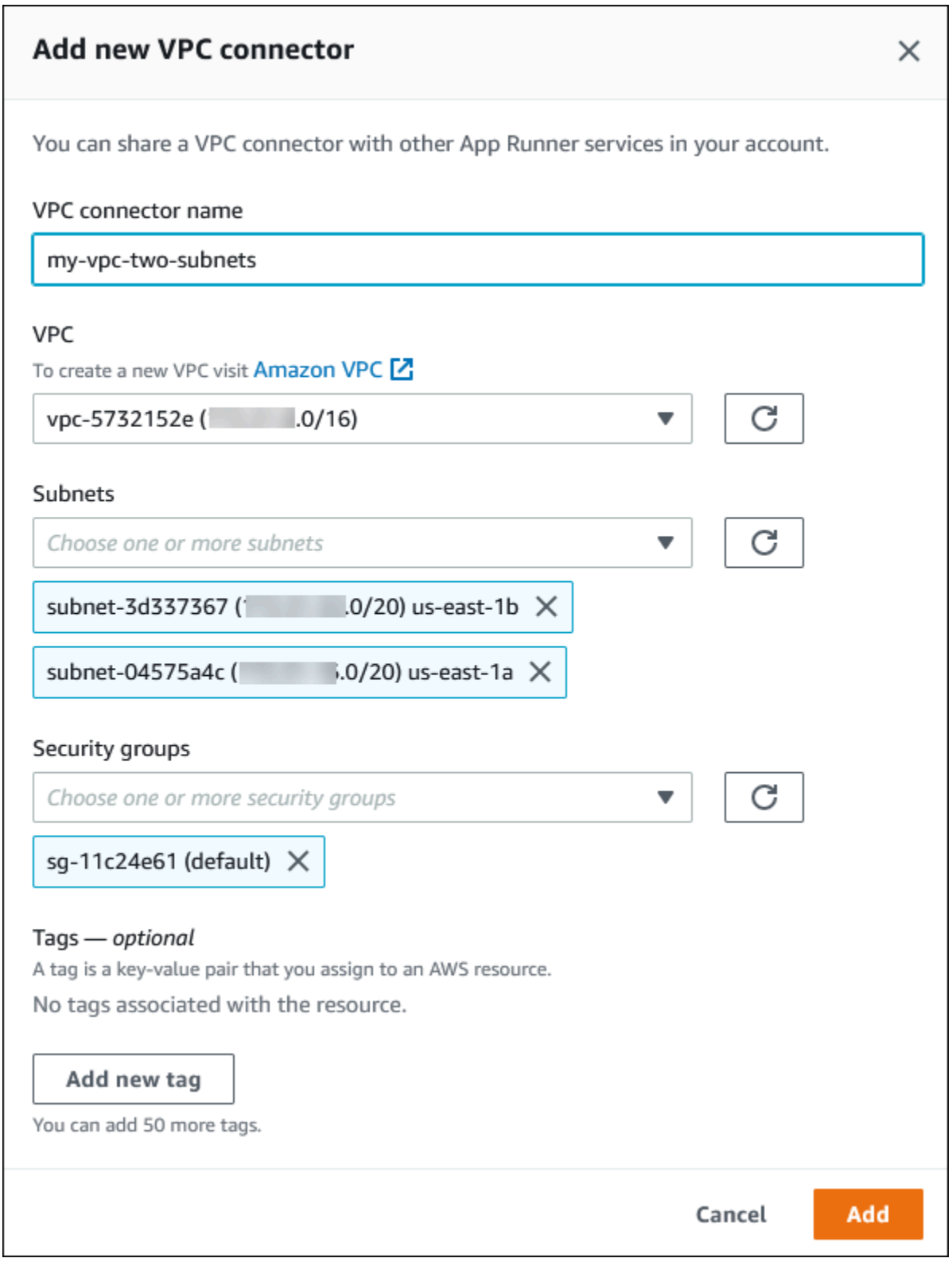
1. App Runner 콘솔을 열고 리전 목록에서를 선택합니다 AWS 리전.
2. 서비스 구성 아래의 네트워킹 섹션으로 이동합니다.



3. 발신 네트워크 트래픽에 대해 사용자 지정 VPC를 선택합니다.
4. 탐색 창에서 VPC 커넥터를 선택합니다.

   VPC 커넥터를 생성한 경우 콘솔에 계정에 VPC 커넥터 목록이 표시됩니다. 기존 VPC 커넥터를 선택하고 다음을 선택하여 구성을 검토할 수 있습니다. 그런 다음 마지막 단계로 이동합니다. 또는 다음 단계를 사용하여 새 VPC 커넥터를 추가할 수 있습니다.
5. 새로 추가를 선택하여 서비스에 대한 새 VPC 커넥터를 생성합니다.

   그러면 새 VPC 커넥터 추가 대화 상자가 열립니다.



6. VPC 커넥터의 이름을 입력하고 사용 가능한 목록에서 필요한 VPC를 선택합니다.

7. 서브넷에서 App Runner 서비스에 액세스하려는 각 가용 영역에 대해 하나의 서브넷을 선택합니다. 가용성을 높이려면 서브넷 3개를 선택합니다. 또는 서브넷이 3개 미만인 경우 사용 가능한 서브넷을 모두 선택합니다.

   > **Note**
   >
   > - VPC 커넥터에 프라이빗 서브넷을 할당해야 합니다. VPC 커넥터에 퍼블릭 서브넷을 할당하면 업데이트 중에 서비스가 자동으로 생성되지 않거나 롤백됩니다.
   > - 발신 트래픽이 듀얼 스택인 경우 선택한 모든 서브넷이 VPC 콘솔에서 듀얼 스택에 대해 구성되어 있는지 확인합니다.

8. (선택 사항) 보안 그룹에서 엔드포인트 네트워크 인터페이스와 연결할 보안 그룹을 선택합니다.
9. (선택 사항) 태그를 추가하려면 새 태그 추가를 선택하고 태그 키와 태그 값을 입력합니다.
10. 추가를 선택합니다.

    생성한 VPC 커넥터의 세부 정보는 VPC 커넥터 아래에 표시됩니다.
11. 다음을 선택하여 구성을 검토한 다음 생성 및 배포를 선택합니다.

    App Runner는 VPC 커넥터 리소스를 생성한 다음 서비스와 연결합니다. 서비스가 성공적으로 생성되면 콘솔에 새 서비스의 서비스 개요와 함께 서비스 대시보드가 표시됩니다.

App Runner API or AWS CLI

CreateService 또는 UpdateService App Runner API 작업을 호출할 때 `NetworkConfiguration` 파라미터의 `EgressConfiguration` 멤버를 사용하여 서비스에 대한 VPC 커넥터 리소스를 지정합니다.

다음 App Runner API 작업을 사용하여 VPC 커넥터 리소스를 관리합니다.

- CreateVpcConnector - 새 VPC 커넥터를 생성합니다.
- ListVpcConnectors -와 연결된 VPC 커넥터 목록을 반환합니다 AWS 계정. 목록에는 전체 설명이 포함되어 있습니다.
- DescribeVpcConnector - VPC 커넥터에 대한 전체 설명을 반환합니다.
- DeleteVpcConnector - VPC 커넥터를 삭제합니다. 의 VPC 커넥터 할당량에 도달 AWS 계정하면 불필요한 VPC 커넥터를 삭제해야 할 수 있습니다.

VPC에 대한 아웃바운드 액세스 권한이 있는 애플리케이션을 App Runner에 배포하려면 먼저 VPC 커넥터를 생성해야 합니다. 애플리케이션과 연결할 서브넷 및 보안 그룹을 하나 이상 지정하여이 작업을 수행할 수 있습니다. 그런 다음 다음 예제와 같이 CLI를 통해 Create 또는 UpdateService에서 VPC 커넥터를 참조할 수 있습니다.

```
            cat > vpc-connector.json <<EOF
{
"VpcConnectorName": "my-vpc-connector",
"Subnets": [
"subnet-a",
"subnet-b",
"subnet-c"
],
"SecurityGroups": [
"sg-1",
"sg-2"
]
}
EOF

aws apprunner create-vpc-connector \
--cli-input-json file:///vpc-connector.json

cat > service.json <<EOF

{
"ServiceName": "my-vpc-connected-service",
"SourceConfiguration": {
"ImageRepository": {
"ImageIdentifier": "<ecr-image-identifier> ",
"ImageConfiguration": {
"Port": "8000"
},
"ImageRepositoryType": "ECR"
}
},
"NetworkConfiguration": {
"EgressConfiguration": {
"EgressType": "VPC",
"VpcConnectorArn": "arn:aws:apprunner:..../my-vpc-connector"
}
}
```

```
}
EOF

aws apprunner create-service \
--cli-input-json file:///service.js
```

App Runner 서비스의 관찰성

AWS App Runner 는 여러 AWS 서비스와 통합되어 App Runner 서비스를 위한 광범위한 관찰성 도구 모음을 제공합니다. 이 장의 주제에서는 이러한 기능에 대해 설명합니다.

주제

App Runner 서비스 활동 추적

AWS App Runner 는 작업 목록을 사용하여 App Runner 서비스의 활동을 추적합니다. 작업은 서비스 생성, 구성 업데이트, 서비스 배포와 같은 API 작업에 대한 비동기 호출을 나타냅니다. 다음 섹션에서는 App Runner 콘솔 및 API를 사용하여 활동을 추적하는 방법을 보여줍니다.

App Runner 서비스 활동 추적

다음 방법 중 하나를 사용하여 App Runner 서비스 활동을 추적합니다.

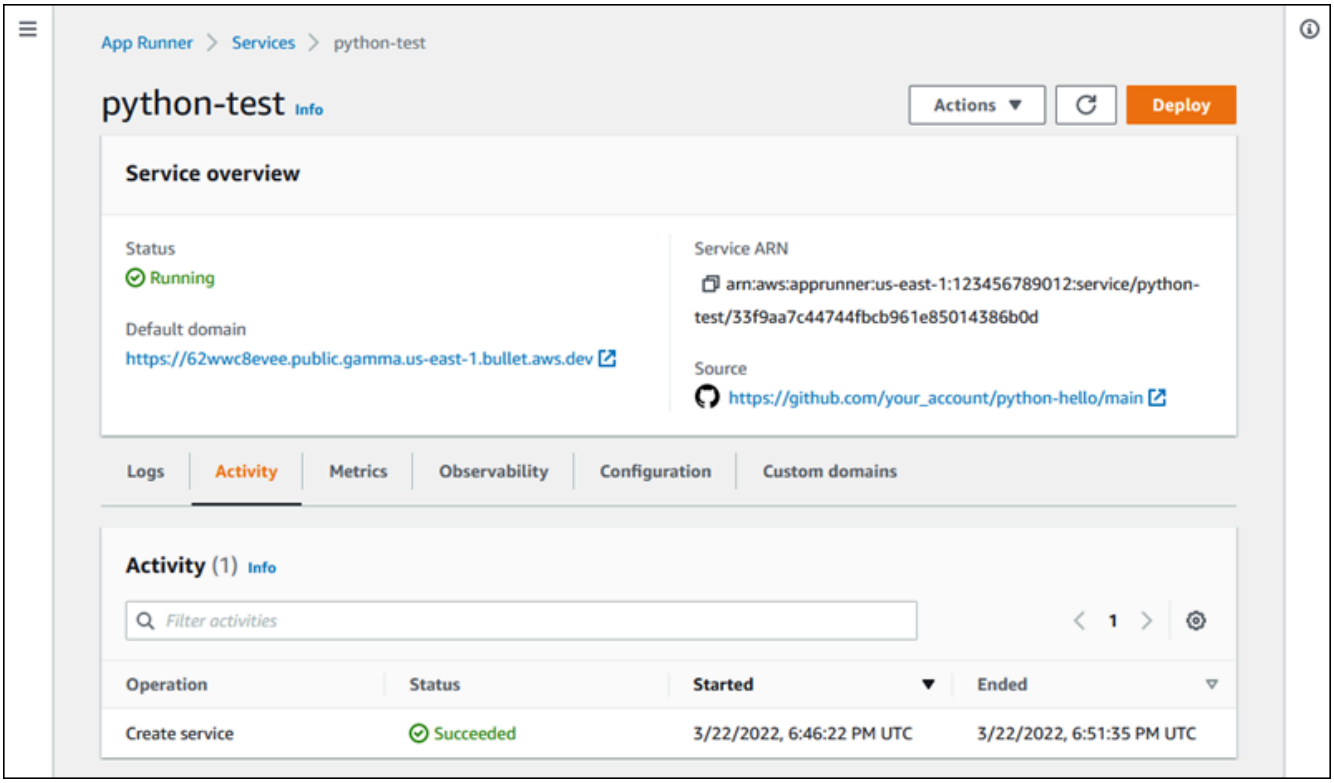
App Runner console

App Runner 콘솔은 App Runner 서비스 활동을 표시하고 작업을 탐색하는 더 많은 방법을 제공합니다.

서비스 활동을 보려면

1. App Runner 콘솔을 열고 리전 목록에서를 선택합니다 AWS 리전.
2. 탐색 창에서 서비스를 선택한 다음 App Runner 서비스를 선택합니다.

   콘솔에 서비스 개요와 함께 서비스 대시보드가 표시됩니다.



3. 아직 선택되지 않은 경우 서비스 대시보드 페이지에서 활동 탭을 선택합니다.

   콘솔에 작업 목록이 표시됩니다.

4. 특정 작업을 찾으려면 검색어를 입력하여 목록의 범위를 좁힙니다. 테이블에 나타나는 모든 값을 검색할 수 있습니다.

5. 나열된 작업을 선택하여 관련 로그를 보거나 다운로드합니다.

App Runner API or AWS CLI

App Runner 서비스의 Amazon 리소스 이름(ARN)을 고려하여 ListOperations 작업은이 서비스에서 발생한 작업 목록을 반환합니다. 각 목록 항목에는 작업 ID와 일부 추적 세부 정보가 포함됩니다.

CloudWatch Logs로 스트리밍된 App Runner 로그 보기

Amazon CloudWatch Logs를 사용하여 다양한 AWS 서비스의 리소스가 생성하는 로그 파일을 모니터링, 저장 및 액세스할 수 있습니다. 자세한 내용은 Amazon CloudWatch Logs 사용 설명서를 참조하세요.

AWS App Runner 는 애플리케이션 배포 및 활성 서비스의 출력을 수집하여 CloudWatch Logs로 스트리밍합니다. 다음 섹션에서는 App Runner 로그 스트림을 나열하고 App Runner 콘솔에서 보는 방법을 보여줍니다.

App Runner 로그 그룹 및 스트림

CloudWatch Logs는 로그 데이터를 로그 그룹에 추가로 구성하는 로그 스트림에 보관합니다. 로그 스트림은 특정 소스의 로그 이벤트 시퀀스입니다. 로그 그룹은 동일한 보존 기간, 모니터링 및 액세스 제어 설정을 공유하는 로그 스트림 그룹입니다.

App Runner는의 각 App Runner 서비스에 대해 각각 여러 로그 스트림이 있는 두 개의 CloudWatch Logs 로그 그룹을 정의합니다 AWS 계정.

서비스 로그

서비스 로그 그룹에는 App Runner가 App Runner 서비스를 관리하고 작업을 수행할 때 생성된 로깅 출력이 포함됩니다.

로그 그룹 이름	예제
`/aws/apprunner/` *`service-name`* `/` *`service-id`* `/service`	`/aws/apprunner/python-test/ac7ec8b51ff34746bcb6654e0bcb23da/service`

서비스 로그 그룹 내에서 App Runner는 이벤트 로그 스트림을 생성하여 App Runner 서비스의 수명 주기에서 활동을 캡처합니다. 예를 들어 애플리케이션을 시작하거나 일시 중지할 수 있습니다.

또한 App Runner는 서비스와 관련된 장기 실행 비동기 작업마다 로그 스트림을 생성합니다. 로그 스트림 이름에는 작업 유형과 특정 작업 ID가 반영됩니다.

배포는 작업의 한 유형입니다. 배포 로그에는 서비스를 생성하거나 애플리케이션의 새 버전을 배포할 때 App Runner가 수행하는 빌드 및 배포 단계의 로깅 출력이 포함됩니다. 배포 로그 스트림 이름은 로 시작하고 배포를 수행하는 작업의 ID로 `deployment/`끝납니다. 이 작업은 초기 애플리케이션 배포를 위한 CreateService 호출 또는 각 추가 배포를 위한 StartDeployment 호출입니다.

배포 로그 내에서 각 로그 메시지는 접두사로 시작합니다.

- `[AppRunner]` - App Runner가 배포 중에 생성하는 출력입니다.

- [Build] - 자체 빌드 스크립트의 출력입니다.

로그 스트림 이름	예제
events	해당 사항 없음(고정된 이름)
operation-type /*operation-id*	deployment/c2c8eeedea164f45 9cf78f12a8953390

애플리케이션 로그

애플리케이션 로그 그룹에는 실행 중인 애플리케이션 코드의 출력이 포함됩니다.

로그 그룹 이름	예제
/aws/apprunner/ *service-n ame* /*service-id* /application	/aws/apprunner/python-test/ ac7ec8b51ff34746bcb6654e0bcb23da/ application

애플리케이션 로그 그룹 내에서 App Runner는 애플리케이션을 실행하는 각 인스턴스(스케일링 단위)에 대한 로그 스트림을 생성합니다.

로그 스트림 이름	예제
instance/ *instance-id*	instance/1a80bc9134a84699b7 b3432ebeebb591

콘솔에서 App Runner 로그 보기

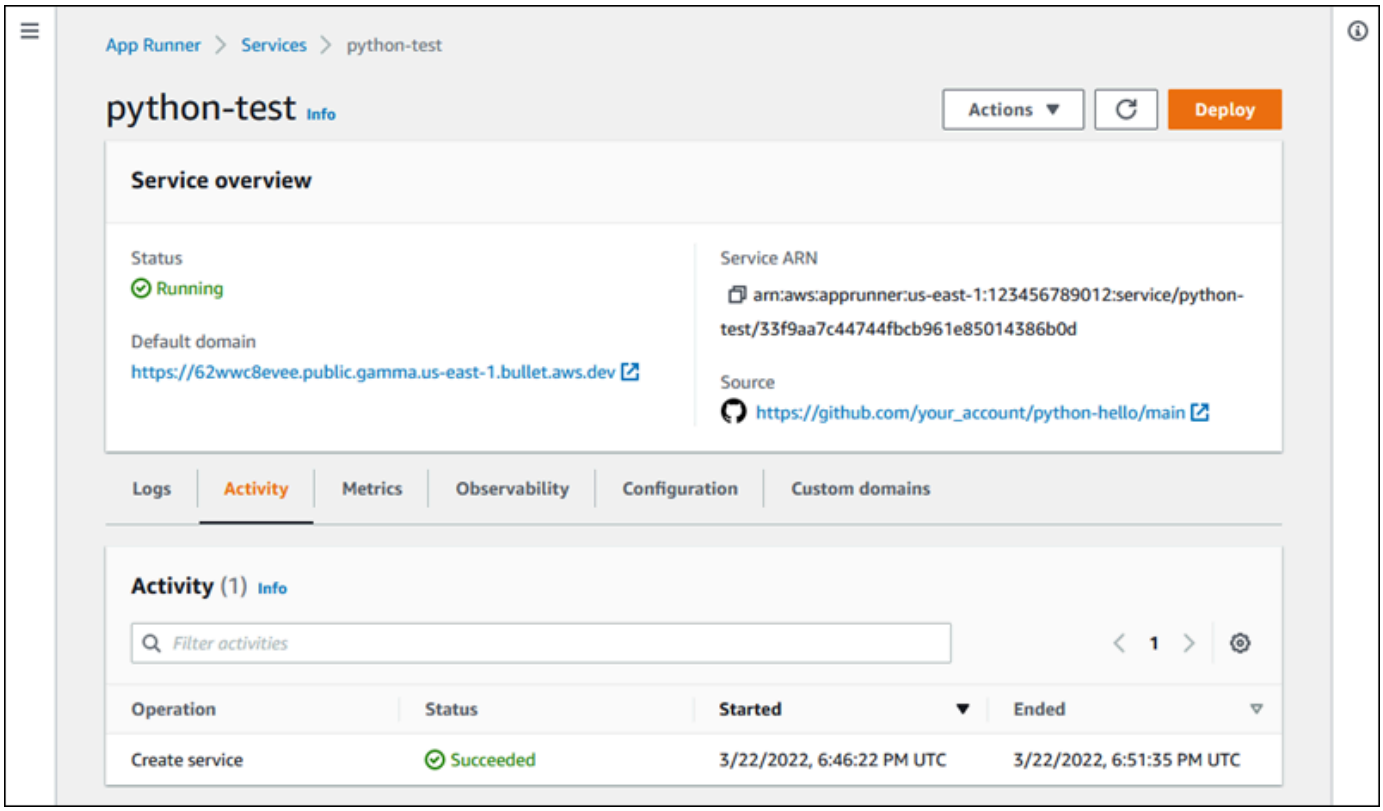
App Runner 콘솔에는 서비스에 대한 모든 로그의 요약이 표시되며 이를 보고 탐색하고 다운로드할 수 있습니다.

서비스에 대한 로그를 보려면

1. App Runner 콘솔을 열고 리전 목록에서를 선택합니다 AWS 리전.

2. 탐색 창에서 서비스를 선택한 다음 App Runner 서비스를 선택합니다.

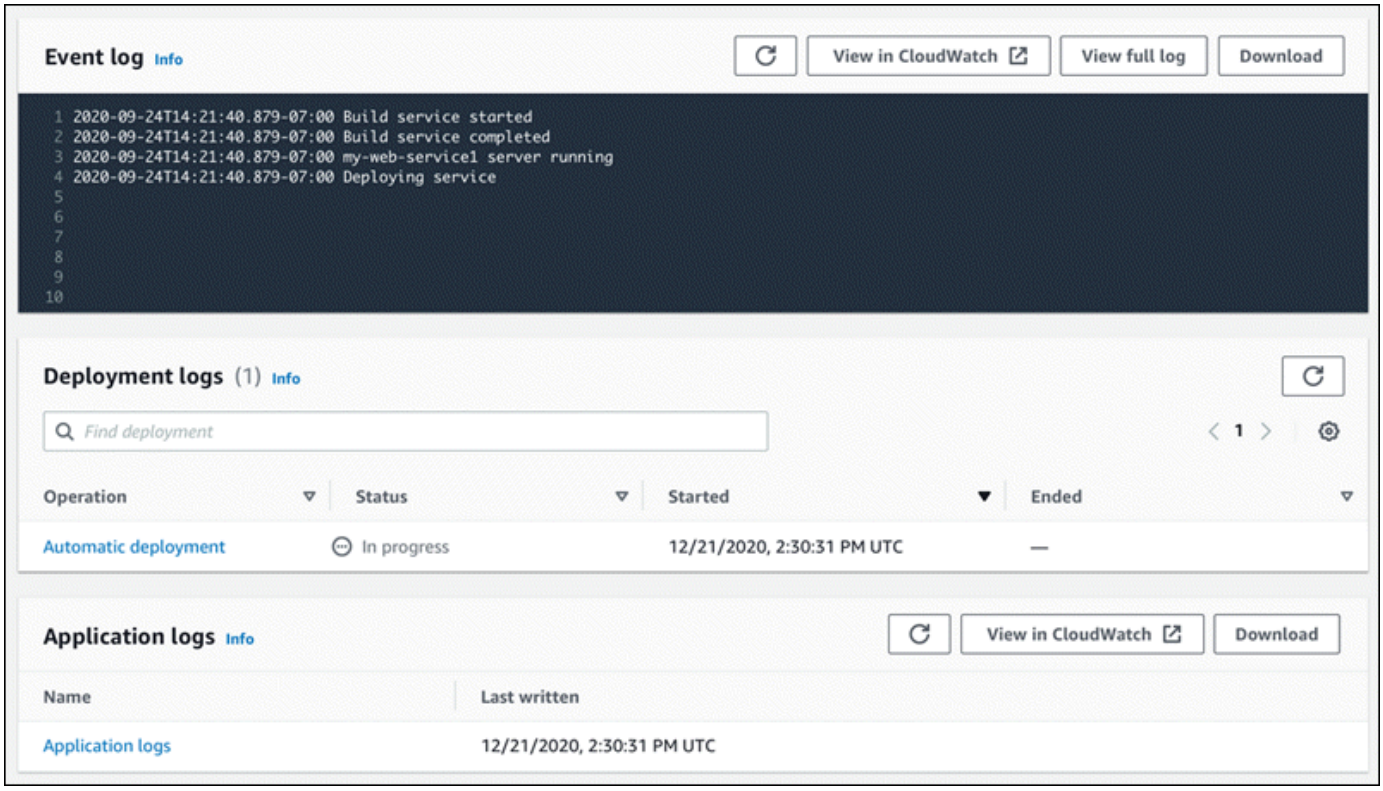
   콘솔에 서비스 개요와 함께 서비스 대시보드가 표시됩니다.



3. 서비스 대시보드 페이지에서 로그 탭을 선택합니다.

   콘솔에는 몇 가지 유형의 로그가 여러 섹션에 표시됩니다.

   - 이벤트 로그 - App Runner 서비스의 수명 주기 내 활동입니다. 콘솔에 최신 이벤트가 표시됩니다.
   - 배포 로그 - App Runner 서비스에 대한 소스 리포지토리 배포입니다. 콘솔에는 각 배포에 대해 별도의 로그 스트림이 표시됩니다.
   - 애플리케이션 로그 - App Runner 서비스에 배포된 웹 애플리케이션의 출력입니다. 콘솔은 실행 중인 모든 인스턴스의 출력을 단일 로그 스트림으로 결합합니다.



4. 특정 배포를 찾으려면 검색어를 입력하여 배포 로그 목록의 범위를 좁힙니다. 테이블에 나타나는 모든 값을 검색할 수 있습니다.
5. 로그의 콘텐츠를 보려면 전체 로그 보기(이벤트 로그) 또는 로그 스트림 이름(배포 및 애플리케이션 로그)을 선택합니다.
6. 다운로드를 선택하여 로그를 다운로드합니다. 배포 로그 스트림의 경우 먼저 로그 스트림을 선택합니다.
7. CloudWatch에서 보기를 선택하여 CloudWatch 콘솔을 열고 전체 기능을 사용하여 App Runner 서비스 로그를 탐색합니다. 배포 로그 스트림의 경우 먼저 로그 스트림을 선택합니다.

> **Note**
>
> CloudWatch 콘솔은 결합된 애플리케이션 로그 대신 특정 인스턴스의 애플리케이션 로그를 보려는 경우에 특히 유용합니다.

CloudWatch에 보고된 App Runner 서비스 지표 보기

Amazon CloudWatch는 Amazon Web Services(AWS) 리소스와 AWS 실행 중인 애플리케이션을 실시간으로 모니터링합니다. CloudWatch를 사용하여 리소스 및 애플리케이션에 대해 측정할 수 있는 변수인 지표를 수집하고 추적할 수 있습니다. 이를 사용하여 지표를 감시하는 경보를 생성할 수도 있습니다. 특정 임계값에 도달하면 CloudWatch는 알림을 보내거나 모니터링되는 리소스를 자동으로 변경합니다. 자세한 설명은 Amazon CloudWatch 사용 설명서를 참조하세요.

AWS App Runner 는 App Runner 서비스의 사용, 성능 및 가용성에 대한 가시성을 높이는 다양한 지표를 수집합니다. 일부 지표는 웹 서비스를 실행하는 개별 인스턴스를 추적하는 반면, 다른 지표는 전체 서비스 수준에 있습니다. 다음 섹션에서는 App Runner 지표를 나열하고 App Runner 콘솔에서 지표를 보는 방법을 보여줍니다.

App Runner 지표

App Runner는 서비스와 관련된 다음 지표를 수집하여 `AWS/AppRunner` 네임스페이스의 CloudWatch에 게시합니다.

> **Note**
>
> 2023년 8월 23일 이전에 CPU 사용률 및 메모리 사용률 지표는 현재 계산된 사용률 백분율 대신 vCPU 단위 및 사용된 메모리 메가바이트를 기반으로 했습니다. 애플리케이션이이 날짜 이전에 App Runner에서 실행되었고 App Runner 또는 CloudWatch 콘솔에서이 날짜의 지표를 보도록 다시 선택하면 두 단위로 지표가 표시되며 결과적으로 몇 가지 불규칙성이 표시됩니다.

> **Important**
>
> 2023년 8월 23일 이전에 CPU 사용률 및 메모리 사용률 지표 값을 기반으로 하는 CloudWatch 경보를 업데이트해야 합니다. vCPU 또는 메가바이트가 아닌 사용률을 기반으로 트리거하도록 경보를 업데이트합니다. 자세한 설명은 Amazon CloudWatch 사용 설명서를 참조하세요.

인스턴스 수준 지표는 각 인스턴스(스케일링 단위)에 대해 개별적으로 수집됩니다.

무엇이 측정되나요?	지표	설명
CPU utilization	`CPUUtilization`	서비스 구성에서 예약한 총 CPU 사용량 중 1분 동안의 평균 CPU 사용량 비율입니다.
Memory utilization	`MemoryUtilization`	서비스 구성에서 예약한 총 메모리 중 1분 동안 평균 메모리 사용량의 백분율입니다.

서비스 수준 지표는 전체 서비스에 대해 수집됩니다.

무엇이 측정되나요?	지표	설명
CPU utilization	`CPUUtilization`	서비스 구성에서 예약한 총 CPU 사용량 중 1분 동안 모든 인스턴스에서 집계된 CPU 사용량의 백분율입니다.
Memory utilization	`MemoryUtilization`	서비스 구성에서 예약한 총 메모리 중 1분 동안 모든 인스턴스에서 집계된 메모리 사용량의 백분율입니다.
Concurrency	`Concurrency`	서비스에서 처리 중인 동시 요청의 대략적인 수입니다.
HTTP request count	`Requests`	서비스가 수신한 HTTP 요청 수입니다.

HTTP status counts	`2xxStatusResponses`
	`4xxStatusResponses`
	`5xxStatusResponses`
		범주(2XX, 4XX, 5XX)별로 그룹화된 각 응답 상태를 반환한 HTTP 요청 수입니다.

무엇이 측정되나요?	지표	설명
HTTP request latency	`RequestLatency`	웹 서비스가 HTTP 요청을 처리하는 데 걸린 밀리초 단위의 시간입니다.
Instance counts	`ActiveInstances`	서비스에 대한 HTTP 요청을 처리하는 인스턴스 수입니다. **Note** 지표에 `ActiveInstances` `0`이 표시되면 서비스에 대한 요청이 없음을 의미합니다. 서비스의 인스턴스 수가 0임을 나타내지 않습니다.

콘솔에서 App Runner 지표 보기

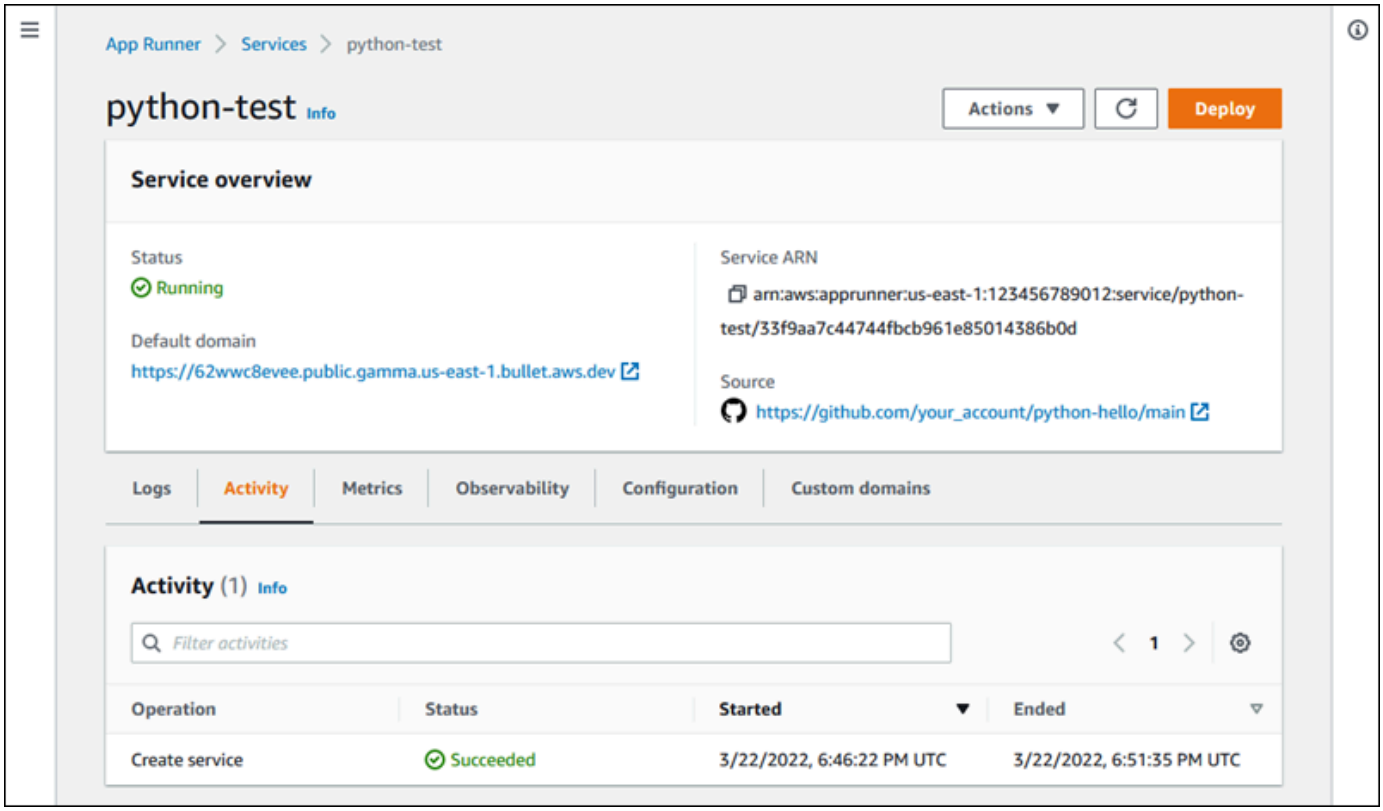
App Runner 콘솔은 App Runner가 서비스에 대해 수집하는 지표를 그래픽으로 표시하고 이를 탐색하는 더 많은 방법을 제공합니다.

> **Note**
> 현재 콘솔에는 서비스 지표만 표시됩니다. 인스턴스 지표를 보려면 CloudWatch 콘솔을 사용합니다.

서비스에 대한 로그를 보려면

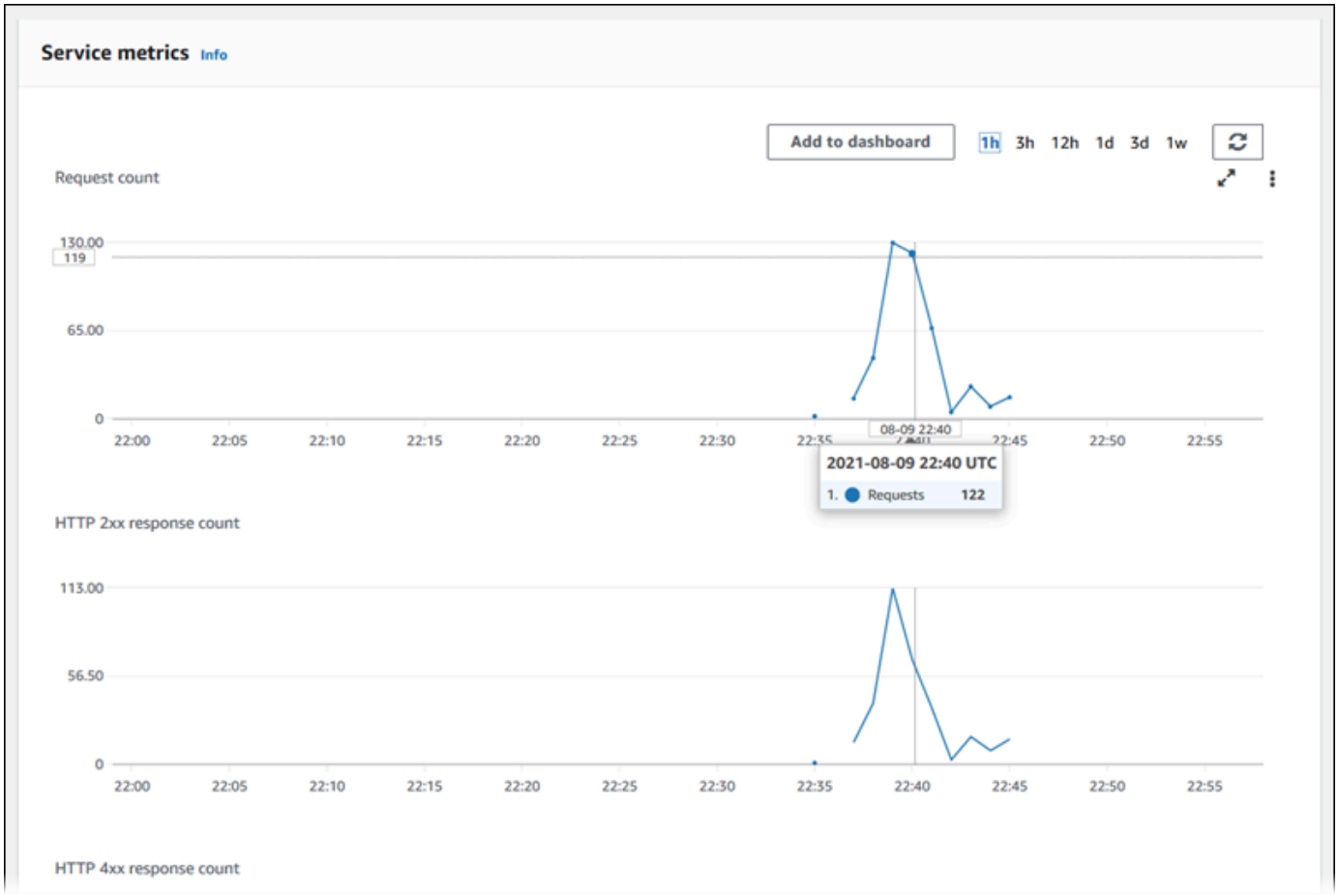
1. App Runner 콘솔을 열고 리전 목록에서를 선택합니다 AWS 리전.
2. 탐색 창에서 서비스를 선택한 다음 App Runner 서비스를 선택합니다.

   콘솔에 서비스 개요와 함께 서비스 대시보드가 표시됩니다.



3. 서비스 대시보드 페이지에서 지표 탭을 선택합니다.

   콘솔에는 지표 그래프 세트가 표시됩니다.



4. 기간(예: 12시간)을 선택하여 지표 그래프의 범위를 해당 기간의 최근 기간으로 지정합니다.
5. 그래프 섹션 중 하나 상단의 대시보드에 추가를 선택하거나 그래프의 메뉴를 사용하여 추가 조사를 위해 CloudWatch 콘솔의 대시보드에 관련 지표를 추가합니다.

EventBridge에서 App Runner 이벤트 처리

Amazon EventBridge를 사용하면 특정 패턴에 대해 AWS App Runner 서비스의 실시간 데이터 스트림을 모니터링하는 이벤트 기반 규칙을 설정할 수 있습니다. 규칙의 패턴이 일치하면 EventBridge는 Amazon ECS AWS Lambda및 Amazon SNS AWS Batch와 같은 대상에서 작업을 시작합니다. 예를 들어 서비스에 대한 배포가 실패할 때마다 Amazon SNS 주제에 신호를 보내 이메일 알림을 보내는 규칙을 설정할 수 있습니다. 또는 서비스 업데이트가 실패할 때마다 Slack 채널에 알리도록 Lambda 함수를 설정할 수 있습니다. EventBridge에 대한 자세한 내용은 Amazon EventBridge 사용 설명서를 참조하세요.

App Runner가 EventBridge로 다음 이벤트 유형 전송

- 서비스 상태 변경 - App Runner 서비스의 상태 변경입니다. 예를 들어 서비스 상태가 로 변경되었습니다`DELETE_FAILED`.
- 서비스 작업 상태 변경 - App Runner 서비스에서 긴 비동기 작업의 상태 변경입니다. 예를 들어 서비스가 생성되기 시작했거나, 서비스 업데이트가 성공적으로 완료되었거나, 서비스 배포가 오류와 함께 완료되었습니다.

App Runner 이벤트에 대한 EventBridge 규칙 생성

EventBridge 이벤트는 소스 AWS 서비스 및 세부 정보(이벤트) 유형과 같은 일부 표준 EventBridge 필드와 이벤트 세부 정보가 포함된 이벤트별 필드 세트를 정의하는 객체입니다. EventBridge 규칙을 생성하려면 EventBridge 콘솔을 사용하여 이벤트 패턴(추적해야 하는 이벤트)을 정의하고 대상 작업(일치에 대해 수행해야 하는 작업)을 지정합니다. 이벤트 패턴은 일치하는 이벤트와 유사합니다. 일치시킬 필드의 하위 집합을 지정하고 각 필드에 대해 가능한 값 목록을 지정합니다. 이 주제에서는 App Runner 이벤트 및 이벤트 패턴의 예를 제공합니다.

EventBridge 규칙 생성에 대한 자세한 내용은 Amazon EventBridge 사용 설명서의 서비스에 대한 AWS 규칙 생성을 참조하세요.

> **Note**
>
> 일부 서비스는 EventBridge에서 사전 정의된 패턴을 지원합니다. 이를 통해 이벤트 패턴이 생성되는 방법이 간소화됩니다. 양식에서 필드 값을 선택하면 EventBridge가 자동으로 패턴을 생성합니다. 현재 App Runner는 사전 정의된 패턴을 지원하지 않습니다. 패턴을 JSON 객체로 입력해야 합니다. 이 주제의 예제를 시작점으로 사용할 수 있습니다.

App Runner 이벤트 예제

다음은 App Runner가 EventBridge로 보내는 이벤트에 대한 몇 가지 예입니다.

- 서비스 상태 변경 이벤트입니다. 특히에서 `RUNNING` 상태로 변경된 서비스`OPERATION_IN_PROGRESS`입니다.

```
{
  "version": "0",
  "id": "6a7e8feb-b491-4cf7-a9f1-bf3703467718",
  "detail-type": "AppRunner Service Status Change",
  "source": "aws.apprunner",
```

```
  "account": "111122223333",
  "time": "2021-04-29T11:54:23Z",
  "region": "us-east-2",
  "resources": [
    "arn:aws:apprunner:us-east-2:123456789012:service/my-
app/8fe1e10304f84fd2b0df550fe98a71fa"
  ],
  "detail": {
    "previousServiceStatus": "OPERATION_IN_PROGRESS",
    "currentServiceStatus": "RUNNING",
    "serviceName": "my-app",
    "serviceId": "8fe1e10304f84fd2b0df550fe98a71fa",
    "message": "Service status is set to RUNNING.",
    "severity": "INFO"
  }
}
```

- 작업 상태 변경 이벤트입니다. 특히 작업이 성공적으로 UpdateService 완료되었습니다.

```
{
  "version": "0",
  "id": "6a7e8feb-b491-4cf7-a9f1-bf3703467718",
  "detail-type": "AppRunner Service Operation Status Change",
  "source": "aws.apprunner",
  "account": "111122223333",
  "time": "2021-04-29T18:43:48Z",
  "region": "us-east-2",
  "resources": [
    "arn:aws:apprunner:us-east-2:123456789012:service/my-
app/8fe1e10304f84fd2b0df550fe98a71fa"
  ],
  "detail": {
    "operationStatus": "UpdateServiceCompletedSuccessfully",
    "serviceName": "my-app",
    "serviceId": "8fe1e10304f84fd2b0df550fe98a71fa",
    "message": "Service update completed successfully. New application and
 configuration is deployed.",
    "severity": "INFO"
  }
}
```

App Runner 이벤트 패턴 예제

다음 예제에서는 EventBridge 규칙에서 하나 이상의 App Runner 이벤트를 일치시키는 데 사용할 수 있는 이벤트 패턴을 보여줍니다. 이벤트 패턴은 이벤트와 유사합니다. 일치시키려는 필드만 포함하고 각 필드에 스칼라 대신 목록을 제공합니다.

- 서비스가 더 이상 상태가 아닌 특정 계정의 서비스에 대한 모든 서비스 RUNNING 상태 변경 이벤트를 일치시킵니다.

```
{
  "detail-type": [ "AppRunner Service Status Change" ],
  "source": [ "aws.apprunner" ],
  "account": [ "111122223333" ],
  "detail": {
    "previousServiceStatus": [ "RUNNING" ]
  }
}
```

- 작업이 실패한 특정 계정의 서비스에 대한 모든 작업 상태 변경 이벤트를 일치시킵니다.

```
{
  "detail-type": [ "AppRunner Service Operation Status Change" ],
  "source": [ "aws.apprunner" ],
  "account": [ "111122223333" ],
  "detail": {
    "operationStatus": [
      "CreateServiceFailed",
      "DeleteServiceFailed",
      "UpdateServiceFailed",
      "DeploymentFailed",
      "PauseServiceFailed",
      "ResumeServiceFailed"
    ]
  }
}
```

App Runner 이벤트 참조

서비스 상태 변경

서비스 상태 변경 이벤트가 로 `detail-type` 설정되었습니다`AppRunner Service Status Change`. 여기에는 다음과 같은 세부 정보 필드와 값이 있습니다.

```
"serviceId": "your service ID",
"serviceName": "your service name",
"message": "Service status is set to CurrentStatus.",
"previousServiceStatus": "any valid service status",
"currentServiceStatus": "any valid service status",
"severity": "varies"
```

작업 상태 변경

작업 상태 변경 이벤트가 로 `detail-type` 설정되었습니다`AppRunner Service Operation Status Change`. 여기에는 다음과 같은 세부 정보 필드와 값이 있습니다.

```
"operationStatus": "see following table",
"serviceName": "your service name",
"serviceId": "your service ID",
"message": "see following table",
"severity": "varies"
```

다음 표에는 가능한 모든 상태 코드와 관련 메시지가 나열되어 있습니다.

Status	메시지
`CreateServiceStarted`	서비스 생성이 시작되었습니다.
`CreateServiceCompletedSuccessfully`	서비스 생성이 성공적으로 완료되었습니다.
`CreateServiceFailed`	서비스 생성에 실패했습니다. 자세한 내용은 서비스 로그를 참조하세요.
`DeleteServiceStarted`	서비스 삭제가 시작되었습니다.

Status	메시지
DeleteServiceCompletedSuccessfully	서비스 삭제가 성공적으로 완료되었습니다.
DeleteServiceFailed	서비스 삭제에 실패했습니다.
UpdateServiceStarted	
UpdateServiceCompletedSuccessfully	서비스 업데이트가 성공적으로 완료되었습니다. 새 애플리케이션 및 구성이 배포됩니다.
	서비스 업데이트가 성공적으로 완료되었습니다. 새 구성이 배포됩니다.
UpdateServiceFailed	서비스 업데이트에 실패했습니다. 자세한 내용은 서비스 로그를 참조하세요.
DeploymentStarted	배포가 시작되었습니다.
DeploymentCompletedSuccessfully	배포가 성공적으로 완료되었습니다.
DeploymentFailed	배포에 실패했습니다. 자세한 내용은 서비스 로그를 참조하세요.
PauseServiceStarted	서비스 일시 중지가 시작되었습니다.
PauseServiceCompletedSuccessfully	서비스 일시 중지가 성공적으로 완료되었습니다.
PauseServiceFailed	서비스 일시 중지에 실패했습니다.
ResumeServiceStarted	서비스 재개가 시작되었습니다.
ResumeServiceCompletedSuccessfully	서비스 재개가 성공적으로 완료되었습니다.
ResumeServiceFailed	서비스 재개에 실패했습니다.

를 사용하여 App Runner API 호출 로깅 AWS CloudTrail

App Runner는 App Runner에서 사용자 AWS CloudTrail, 역할 또는 서비스가 수행한 작업에 대한 레코드를 제공하는 AWS 서비스와 통합됩니다. CloudTrail은 App Runner에 대한 모든 API 호출을 이벤트로 캡처합니다. 캡처되는 호출에는 App Runner 콘솔의 호출과 App Runner API 작업에 대한 코드 호출이 포함됩니다. 추적을 생성하면 App Runner 이벤트를 포함한 CloudTrail 이벤트를 지속적으로 Amazon S3 버킷에 배포할 수 있습니다. 추적을 구성하지 않은 경우에도 이벤트 기록에서 CloudTrail 콘솔의 최신 이벤트를 볼 수 있습니다. CloudTrail에서 수집한 정보를 사용하여 App Runner에 수행된 요청, 요청이 수행된 IP 주소, 요청을 수행한 사람, 요청이 수행된 시간 및 추가 세부 정보를 확인할 수 있습니다.

CloudTrail에 대한 자세한 내용은 AWS CloudTrail 사용 설명서를 참조하세요.

CloudTrail의 App Runner 정보

CloudTrail은 계정을 생성할 AWS 계정 때에서 활성화됩니다. App Runner에서 활동이 발생하면 해당 활동이 이벤트 기록의 다른 AWS 서비스 이벤트와 함께 CloudTrail 이벤트에 기록됩니다. 에서 최근 이벤트를 보고 검색하고 다운로드할 수 있습니다 AWS 계정. 자세한 설명은 CloudTrail 이벤트 기록으로 이벤트 보기를 참조하세요.

App Runner에 대한 이벤트를 AWS 계정포함하여에 이벤트를 지속적으로 기록하려면 추적을 생성합니다. CloudTrail은 추적을 사용하여 Amazon S3 버킷으로 로그 파일을 전송할 수 있습니다. 콘솔에서 트레일을 생성하면 기본적으로 모든 AWS 리전에 트레일이 적용됩니다. 추적은 AWS 파티션의 모든 리전에서 이벤트를 로깅하고 지정한 Amazon S3 버킷으로 로그 파일을 전송합니다. 또한 CloudTrail 로그에서 수집된 이벤트 데이터를 추가로 분석하고 조치를 취하도록 다른 AWS 서비스를 구성할 수 있습니다. 자세한 내용은 다음 자료를 참조하세요.

- 추적 생성 개요
- CloudTrail 지원 서비스 및 통합
- CloudTrail에서 Amazon SNS 알림 구성
- 여러 리전으로부터 CloudTrail 로그 파일 받기 및 여러 계정으로부터 CloudTrail 로그 파일 받기

모든 App Runner 작업은 CloudTrail에서 로깅되며 AWS App Runner API 참조에 문서화됩니다. 예를 들어 `CreateService`, `DeleteConnection` 및 `StartDeployment` 작업을 직접적으로 호출하면 CloudTrail 로그 파일에 항목이 생성됩니다.

모든 이벤트 또는 로그 항목에는 요청을 생성했던 사용자에 관한 정보가 포함됩니다. ID 정보를 이용하면 다음을 쉽게 판단할 수 있습니다.

- 요청을 루트로 했는지 아니면 IAM 사용자 자격 증명 정보로 했는지 여부.
- 역할 또는 페더레이션 사용자의 임시 자격 증명을 사용하여 요청이 생성되었는지 여부.
- 요청이 다른 AWS 서비스에서 이루어졌는지 여부입니다.

자세한 설명은 CloudTrail userIdentity 요소를 참조하세요.

App Runner 로그 파일 항목 이해

트레일이란 지정한 S3 버킷에 이벤트를 로그 파일로 입력할 수 있게 하는 구성입니다. CloudTrail 로그 파일에는 하나 이상의 로그 항목이 포함될 수 있습니다. 이벤트는 모든 소스로부터의 단일 요청을 나타내며 요청 작업, 작업 날짜와 시간, 요청 파라미터에 관한 정보가 들어 있습니다. CloudTrail 로그 파일은 퍼블릭 API 직접 호출의 주문 스택 트레이스가 아니므로 특정 순서로 표시되지 않습니다.

다음 예제는 `CreateService` 작업을 보여주는 CloudTrail 로그 항목이 나타냅니다.

> **Note**
>
> 보안상의 이유로 일부 속성 값은 로그에서 수정되고 텍스트 로 대체됩니다`HIDDEN_DUE_TO_SECURITY_REASONS`. 이렇게 하면 비밀 정보가 의도치 않게 노출되는 것을 방지할 수 있습니다. 그러나 이러한 속성이 요청에서 전달되었거나 응답에서 반환되었음을 여전히 확인할 수 있습니다.

`CreateService` App Runner 작업에 대한 CloudTrail 로그 항목 예제

```
{
  "eventVersion": "1.08",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE",
    "arn": "arn:aws:iam::123456789012:user/aws-user",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "aws-user"
  },
  "eventTime": "2020-10-02T23:25:33Z",
  "eventSource": "apprunner.amazonaws.com",
  "eventName": "CreateService",
  "awsRegion": "us-east-2",
  "sourceIPAddress": "192.0.2.0",
```

```
  "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_6) AppleWebKit/537.36
 (KHTML, like Gecko) Chrome/77.0.3865.75 Safari/537.36",
  "requestParameters": {
    "serviceName": "python-test",
    "sourceConfiguration": {
      "codeRepository": {
        "repositoryUrl": "https://github.com/github-user/python-hello",
        "sourceCodeVersion": {
          "type": "BRANCH",
          "value": "main"
        },
        "codeConfiguration": {
          "configurationSource": "API",
          "codeConfigurationValues": {
            "runtime": "python3",
            "buildCommand": "HIDDEN_DUE_TO_SECURITY_REASONS",
            "startCommand": "HIDDEN_DUE_TO_SECURITY_REASONS",
            "port": "8080",
            "runtimeEnvironmentVariables": "HIDDEN_DUE_TO_SECURITY_REASONS"
          }
        }
      },
      "autoDeploymentsEnabled": true,
      "authenticationConfiguration": {
        "connectionArn": "arn:aws:apprunner:us-east-2:123456789012:connection/your-
connection/e7656250f67242d7819feade6800f59e"
      }
    },
    "healthCheckConfiguration": {
      "protocol": "HTTP"
    },
    "instanceConfiguration": {
      "cpu": "256",
      "memory": "1024"
    }
  },
  "responseElements": {
    "service": {
        "serviceName": "python-test",
        "serviceId": "dfa2b7cc7bcb4b6fa6c1f0f4efff988a",
        "serviceArn": "arn:aws:apprunner:us-east-2:123456789012:service/python-test/
dfa2b7cc7bcb4b6fa6c1f0f4efff988a",
        "serviceUrl": "generated domain",
        "createdAt": "2020-10-02T23:25:32.650Z",
```

```
        "updatedAt": "2020-10-02T23:25:32.650Z",
        "status": "OPERATION_IN_PROGRESS",
        "sourceConfiguration": {
            "codeRepository": {
                "repositoryUrl": "https://github.com/github-user/python-hello",
                "sourceCodeVersion": {
                    "type": "Branch",
                    "value": "main"
                },
                "sourceDirectory": "/",
                "codeConfiguration": {
                    "codeConfigurationValues": {
                        "configurationSource": "API",
                        "runtime": "python3",
                        "buildCommand": "HIDDEN_DUE_TO_SECURITY_REASONS",
                        "startCommand": "HIDDEN_DUE_TO_SECURITY_REASONS",
                        "port": "8080",
                        "runtimeEnvironmentVariables": "HIDDEN_DUE_TO_SECURITY_REASONS"
                    }
                }
            },
            "autoDeploymentsEnabled": true,
            "authenticationConfiguration": {
                "connectionArn": "arn:aws:apprunner:us-east-2:123456789012:connection/
your-connection/e7656250f67242d7819feade6800f59e"
            }
        },
        "healthCheckConfiguration": {
            "protocol": "HTTP",
            "path": "/",
            "interval": 5,
            "timeout": 2,
            "healthyThreshold": 3,
            "unhealthyThreshold": 5
        },
        "instanceConfiguration": {
            "cpu": "256",
            "memory": "1024"
        },
        "autoScalingConfigurationSummary": {
            "autoScalingConfigurationArn": "arn:aws:apprunner:us-
east-2:123456789012:autoscalingconfiguration/
DefaultConfiguration/1/00000000000000000000000000000001",
            "autoScalingConfigurationName": "DefaultConfiguration",
```

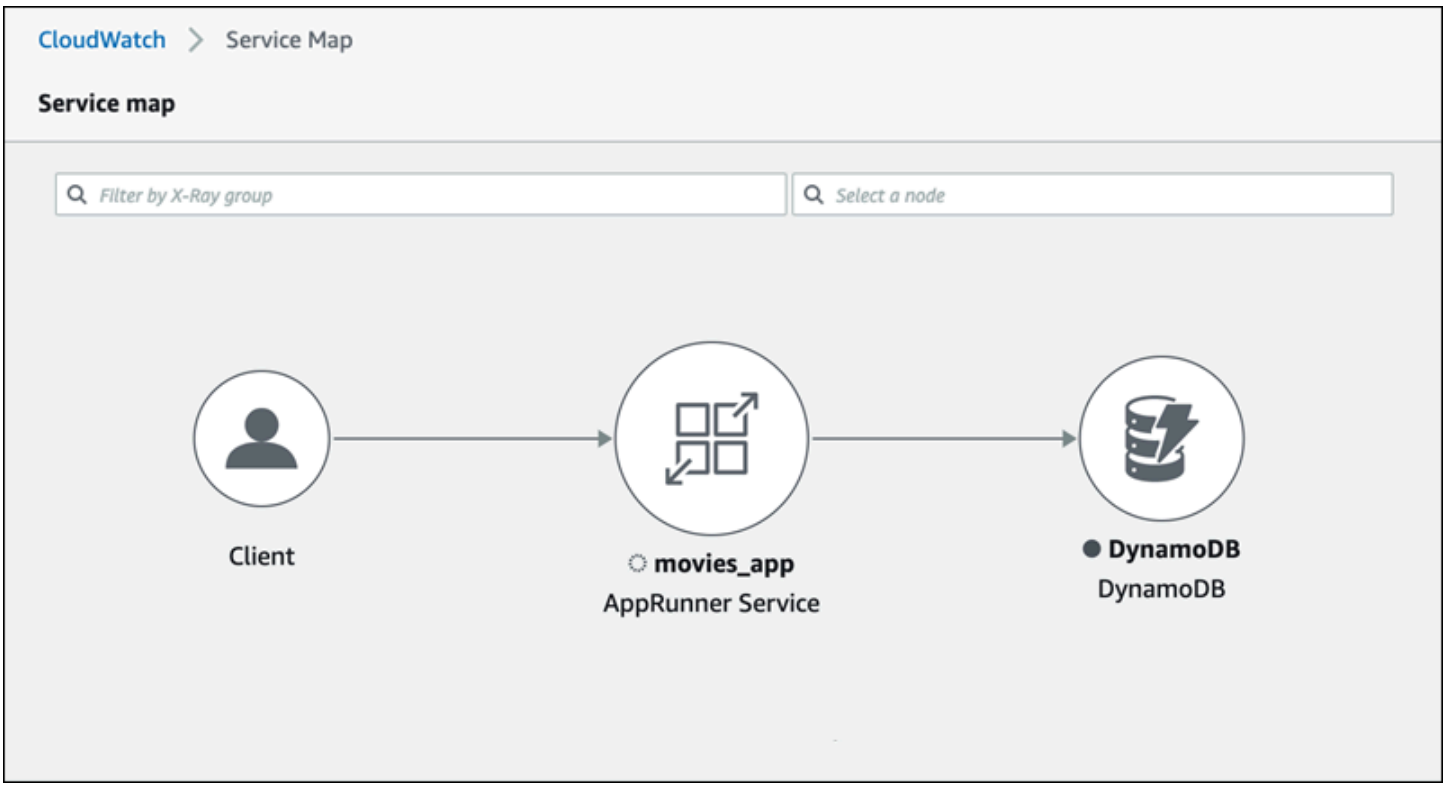
```
            "autoScalingConfigurationRevision": 1
        }
    }
},
  "requestID": "1a60af60-ecf5-4280-aa8f-64538319ba0a",
  "eventID": "e1a3f623-4d24-4390-a70b-bf08a0e24669",
  "readOnly": false,
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}
```


X-Ray로 App Runner 애플리케이션 추적

AWS X-Ray 는 애플리케이션이 처리하는 요청에 대한 데이터를 수집하고 해당 데이터를 보고, 필터링하고, 인사이트를 얻어 문제와 최적화 기회를 식별하는 데 사용할 수 있는 도구를 제공하는 서비스입니다. 애플리케이션에 대한 추적된 요청의 경우 요청 및 응답뿐만 아니라 애플리케이션이 다운스트림 AWS 리소스, 마이크로서비스, 데이터베이스 및 HTTP 웹 APIs.

X-Ray는 클라우드 애플리케이션을 지원하는 AWS 리소스의 추적 데이터를 사용하여 자세한 서비스 그래프를 생성합니다. 서비스 그래프는 프런트엔드 서비스가 요청을 처리하고 데이터를 유지하기 위해 호출하는 클라이언트, 프런트엔드 서비스 및 백엔드 서비스를 보여줍니다. 서비스 그래프를 사용하여 병목 현상, 지연 시간 스파이크 및 해결해야 할 기타 문제를 식별하여 애플리케이션의 성능을 개선합니다.

X-Ray에 대한 자세한 내용은 AWS X-Ray 개발자 안내서를 참조하세요.



추적을 위해 애플리케이션 계측

휴대용 원격 측정 사양인 OpenTelemetry를 사용하여 App Runner 서비스 애플리케이션을 추적하도록 계측합니다. 현재 App Runner는 AWS 서비스를 사용하여 원격 AWS 측정 정보를 수집하고 제공하는 OpenTelemetry 구현인 Distro for OpenTelemetryOpenTelemetry(ADOT)를 지원합니다. X-Ray는 추적 구성 요소를 구현합니다.

애플리케이션에서 사용하는 특정 ADOT SDK에 따라 ADOT는 자동 및 수동이라는 최대 두 가지 계측 접근 방식을 지원합니다. SDK를 사용한 계측에 대한 자세한 내용은 ADOT 설명서를 참조하고 탐색 창에서 SDK를 선택합니다.

런타임 설정

다음은 추적을 위해 App Runner 서비스 애플리케이션을 계측하기 위한 일반적인 런타임 설정 지침입니다.

런타임에 대한 추적을 설정하려면

1. AWS Distro for OpenTelemetry(ADOT)의 런타임에 제공된 지침에 따라 애플리케이션을 계측합니다.

2. 소스 코드 리포지토리를 사용하는 경우 `apprunner.yaml` 파일의 `build` 섹션에, 컨테이너 이미지를 사용하는 경우 Dockerfile에 필요한 `OTEL` 종속성을 설치합니다.
3. 소스 코드 리포지토리를 사용하는 경우 `apprunner.yaml` 파일에서, 컨테이너 이미지를 사용하는 경우 Dockerfile에서 환경 변수를 설정합니다.

   Example환경 변수

   > **Note**
   >
   > 다음 예제에서는 `apprunner.yaml` 파일에 추가할 중요한 환경 변수를 나열합니다. 컨테이너 이미지를 사용하는 경우 이러한 환경 변수를 Dockerfile에 추가합니다. 그러나 각 런타임에는 고유한 ID가 있을 수 있으며 다음 목록에 환경 변수를 더 추가해야 할 수 있습니다. 런타임별 지침 및 런타임에 맞게 애플리케이션을 설정하는 방법에 대한 예제에 대한 자세한 내용은 AWS Distro for OpenTelemetry를 참조하고 시작하기에서 런타임으로 이동합니다.

```
env:
    - name: OTEL_PROPAGATORS
      value: xray
    - name: OTEL_METRICS_EXPORTER
      value: none
    - name: OTEL_EXPORTER_OTLP_ENDPOINT
      value: http://localhost:4317
    - name: OTEL_RESOURCE_ATTRIBUTES
      value: 'service.name=example_app'
```

   > **Note**
   >
   > `OTEL_METRICS_EXPORTER=none` App Runner Otel 수집기는 지표 로깅을 허용하지 않으므로는 App Runner의 중요한 환경 변수입니다. 지표 추적만 허용합니다.

런타임 설정 예제

다음 예제에서는 ADOT Python SDK를 사용하여 애플리케이션을 자동 계측하는 방법을 보여줍니다. SDK는 한 줄의 Python 코드를 추가하지 않고 애플리케이션의 Python 프레임워크에서 사용하는 값을

설명하는 원격 측정 데이터로 스팬을 자동으로 생성합니다. 두 소스 파일에서 몇 줄만 추가하거나 수정해야 합니다.

먼저 다음 예제와 같이 일부 종속성을 추가합니다.

Example requirements.txt

```
opentelemetry-distro[otlp]>=0.24b0
opentelemetry-sdk-extension-aws~=2.0
opentelemetry-propagator-aws-xray~=1.0
```

그런 다음 애플리케이션을 계측합니다. 이를 수행하는 방법은 서비스 소스, 즉 소스 이미지 또는 소스 코드에 따라 다릅니다.

Source image

서비스 소스가 이미지인 경우 컨테이너 이미지 빌드 및 이미지에서 애플리케이션 실행을 제어하는 Dockerfile을 직접 계측할 수 있습니다. 다음 예제에서는 Python 애플리케이션에 대해 구성된 Dockerfile을 보여줍니다. 계측 추가는 굵게 강조 표시됩니다.

Example Dockerfile

```
FROM public.ecr.aws/amazonlinux/amazonlinux:latest
RUN yum install python3.7 -y && curl -O https://bootstrap.pypa.io/get-pip.py &&
 python3 get-pip.py && yum update -y
COPY . /app
WORKDIR /app
RUN pip3 install -r requirements.txt
RUN opentelemetry-bootstrap --action=install
ENV OTEL_PYTHON_DISABLED_INSTRUMENTATIONS=urllib3
ENV OTEL_METRICS_EXPORTER=none
ENV OTEL_RESOURCE_ATTRIBUTES='service.name=example_app'
CMD OTEL_PROPAGATORS=xray OTEL_PYTHON_ID_GENERATOR=xray opentelemetry-instrument
 python3 app.py
EXPOSE 8080
```

Source code repository

서비스 소스가 애플리케이션 소스가 포함된 리포지토리인 경우 App Runner 구성 파일 설정을 사용하여 이미지를 간접적으로 계측합니다. 이러한 설정은 App Runner가 생성하여 애플리케이션용

이미지를 빌드하는 데 사용하는 Dockerfile을 제어합니다. 다음 예제에서는 Python 애플리케이션에 대해 구성된 App Runner 구성 파일을 보여줍니다. 계측 추가는 굵게 강조 표시됩니다.

Example apprunner.yaml

```
version: 1.0
runtime: python3
build:
  commands:
    build:
      - pip install -r requirements.txt
      - opentelemetry-bootstrap --action=install
run:
  command: opentelemetry-instrument python app.py
  network:
    port: 8080
  env:
    - name: OTEL_PROPAGATORS
      value: xray
    - name: OTEL_METRICS_EXPORTER
      value: none
    - name: OTEL_PYTHON_ID_GENERATOR
      value: xray
    - name: OTEL_PYTHON_DISABLED_INSTRUMENTATIONS
      value: urllib3
    - name: OTEL_RESOURCE_ATTRIBUTES
      value: 'service.name=example_app'
```

App Runner 서비스 인스턴스 역할에 X-Ray 권한 추가

App Runner 서비스와 함께 X-Ray 추적을 사용하려면 서비스의 인스턴스에 X-Ray 서비스와 상호 작용할 수 있는 권한을 제공해야 합니다. 이렇게 하려면 인스턴스 역할을 서비스와 연결하고 관리형 정책을 X-Ray 권한과 추가합니다. App Runner 인스턴스 역할에 대한 자세한 내용은 섹션을 참조하세요the section called “인스턴스 역할”. 인스턴스 역할에 `AWSXRayDaemonWriteAccess` 관리형 정책을 추가하고 생성하는 동안 서비스에 할당합니다.

App Runner 서비스에 대한 X-Ray 추적 활성화

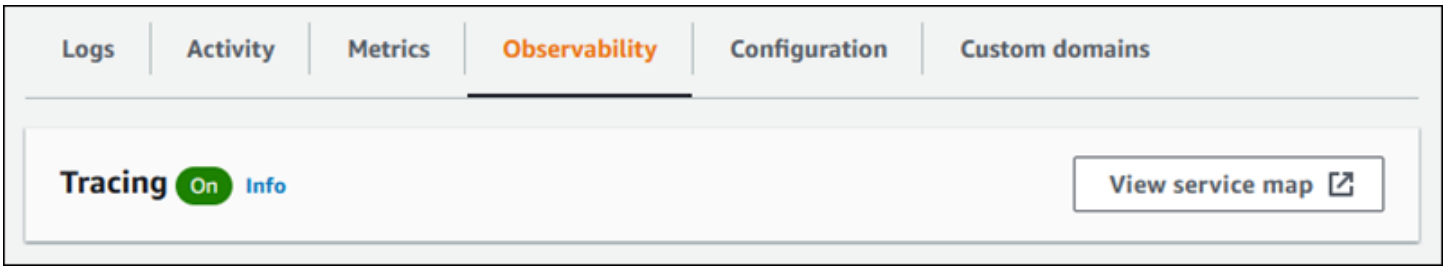
서비스를 생성하면 App Runner는 기본적으로 추적을 비활성화합니다. 관찰성 구성의 일환으로 서비스에 대해 X-Ray 추적을 활성화할 수 있습니다. 자세한 내용은 the section called “관찰성 관리” 단원을 참조하십시오.

App Runner API 또는를 사용하는 경우 ObservabilityConfiguration 리소스 객체 내의 AWS CLI TraceConfiguration 객체에는 추적 설정이 포함됩니다. TraceConfiguration ObservabilityConfiguration 추적을 비활성화 상태로 유지하려면 `TraceConfiguration` 객체를 지정하지 마십시오.

콘솔 및 API 사례 모두에서 이전 섹션에서 설명한 인스턴스 역할을 App Runner 서비스와 연결해야 합니다.

App Runner 서비스에 대한 X-Ray 추적 데이터 보기

App Runner 콘솔의 서비스 대시보드 페이지의 관찰성 탭에서 서비스 맵 보기를 선택하여 Amazon CloudWatch 콘솔로 이동합니다.



Amazon CloudWatch 콘솔을 사용하여 애플리케이션이 처리하는 요청에 대한 서비스 맵 및 추적을 볼 수 있습니다. 서비스 맵에는 요청 지연 시간 및 다른 애플리케이션 및 AWS 서비스와의 상호 작용과 같은 정보가 표시됩니다. 코드에 추가하는 사용자 지정 주석을 사용하면 추적을 쉽게 검색할 수 있습니다. 자세한 내용은 Amazon CloudWatch 사용 설명서의 ServiceLens를 사용하여 애플리케이션 상태 모니터링을 참조하세요.

AWS WAF 웹 ACL을 서비스와 연결

AWS WAF 는 App Runner 서비스를 보호하는 데 사용할 수 있는 웹 애플리케이션 방화벽입니다. AWS WAF 웹 액세스 제어 목록(웹 ACLs)을 사용하면 일반적인 웹 악용 및 원치 않는 봇으로부터 App Runner 서비스 엔드포인트를 보호할 수 있습니다.

웹 ACL을 사용하면 App Runner 서비스에 대한 모든 수신 웹 요청을 세밀하게 제어할 수 있습니다. 웹 ACL에서 웹 트래픽을 허용, 차단 또는 모니터링하는 규칙을 정의하여 승인되고 합법적인 요청만 웹 애플리케이션 및 APIs에 도달하도록 할 수 있습니다. 특정 비즈니스 및 보안 요구 사항에 따라 웹 ACL 규칙을 사용자 지정할 수 있습니다. 인프라 보안 및 네트워크 ACLs 적용 모범 사례에 대한 자세한 내용은 Amazon VPC 사용 설명서의 네트워크 트래픽 제어를 참조하세요.

> **Important**
>
> WAF 웹 ACLs과 연결된 App Runner 프라이빗 서비스에 대한 소스 IP 규칙은 IP 기반 규칙을 준수하지 않습니다. 이는 현재 요청 소스 IP 데이터를 WAF와 연결된 App Runner 프라이빗 서비스로 전달하는 것을 지원하지 않기 때문입니다. App Runner 애플리케이션에 소스 IP/CIDR 수신 트래픽 제어 규칙이 필요한 경우 WAF 웹 ACLs 대신 프라이빗 엔드포인트에 대한 보안 그룹 규칙을 사용해야 합니다.

수신 웹 요청 흐름

AWS WAF 웹 ACL이 App Runner 서비스와 연결되면 수신 웹 요청은 다음 프로세스를 거칩니다.

1. App Runner는 오리진 요청의 내용을에 전달합니다 AWS WAF.
2. AWS WAF 는 요청을 검사하고 해당 내용을 웹 ACL에서 지정한 규칙과 비교합니다.
3. 검사를 기반으로 App Runner에 `allow` 또는 `block` 응답을 AWS WAF 반환합니다.
   - `allow` 응답이 반환되면 App Runner는 요청을 애플리케이션에 전달합니다.
   - `block` 응답이 반환되면 App Runner는 요청이 웹 애플리케이션에 도달하지 못하도록 차단합니다. 의 `block` 응답을 애플리케이션 AWS WAF 으로 전달합니다.

> **Note**
>
> 기본적으로 App Runner는 응답이 반환되지 않는 경우 요청을 차단합니다 AWS WAF.

AWS WAF 웹 ACLs에 대한 자세한 내용은 AWS WAF 개발자 안내서의 웹 액세스 제어 목록(웹 ACLs)을 참조하세요.

> **Note**
>
> 표준 AWS WAF 요금을 지불합니다. App Runner 서비스에 AWS WAF 웹 ACLs 사용하는 데 따른 추가 비용은 발생하지 않습니다.
> 요금에 대한 자세한 내용은 AWS WAF 요금을 참조하세요.

WAF 웹 ACLs App Runner 서비스에 연결

다음은 AWS WAF 웹 ACL을 App Runner 서비스와 연결하는 상위 수준 프로세스입니다.

1. AWS WAF 콘솔에서 웹 ACL을 생성합니다. 자세한 내용은 AWS WAF 개발자 안내서의 웹 ACL 생성을 참조하세요.
2. 에 대한 AWS Identity and Access Management (IAM) 권한을 업데이트합니다 AWS WAF. 자세한 내용은 권한 단원을 참조하십시오.
3. 다음 방법 중 하나를 사용하여 웹 ACL을 App Runner 서비스와 연결합니다.
   - App Runner 콘솔: App Runner 서비스를 생성하거나 업데이트할 때 App Runner 콘솔을 사용하여 기존 웹 ACL을 연결합니다. 지침은 AWS WAF 웹 ACLs.
   - AWS WAF 콘솔: 기존 App Runner 서비스의 AWS WAF 콘솔을 사용하여 웹 ACL을 연결합니다. 자세한 내용은 AWS WAF 개발자 안내서의 웹 ACL과 AWS 리소스 연결 또는 연결 해제를 참조하십시오.
   - AWS CLI: AWS WAF 퍼블릭 APIs를 사용하여 웹 ACL을 연결합니다. AWS WAF 퍼블릭 APIs에 대한 자세한 내용은 AWS WAF API 참조 안내서의 AssociateWebACL을 참조하세요.

고려 사항

- WAF 웹 ACLs과 연결된 App Runner 프라이빗 서비스에 대한 소스 IP 규칙은 IP 기반 규칙을 준수하지 않습니다. 이는 현재 요청 소스 IP 데이터를 WAF와 연결된 App Runner 프라이빗 서비스로 전달하는 것을 지원하지 않기 때문입니다. App Runner 애플리케이션에 소스 IP/CIDR 수신 트래픽 제어 규칙이 필요한 경우 WAF 웹 ACLs 대신 프라이빗 엔드포인트에 대한 보안 그룹 규칙을 사용해야 합니다.

- App Runner 서비스는 하나의 웹 ACL에만 연결할 수 있습니다. 그러나 하나의 웹 ACL을 여러 App Runner 서비스 및 여러 AWS 리소스와 연결할 수 있습니다. 예를 들어 Amazon Cognito 사용자 풀과 Application Load Balancer 리소스가 있습니다.
- 웹 ACL을 생성하면 웹 ACL이 완전히 전파되고 App Runner에서 사용할 수 있을 때까지 약간의 시간이 걸립니다. 전파 시간은 몇 초에서 몇 분 사이일 수 있습니다. 웹 ACL이 완전히 전파되기 전에 연결을 시도`WAFUnavailableEntityException`하면가를 AWS WAF 반환합니다.

  웹 ACL이 완전히 전파되기 전에 브라우저를 새로 고치거나 App Runner 콘솔에서 다른 곳으로 이동하면 연결이 발생하지 않습니다. 그러나 App Runner 콘솔 내에서 탐색할 수 있습니다.
- AWS WAF 는 잘못된 상태인 App Runner 서비스에 대해 다음 AWS WAF APIs 중 하나를 호출할 때 `WAFNonexistentItemException` 오류를 반환합니다.
  - `AssociateWebACL`
  - `DisassociateWebACL`
  - `GetWebACLForResource`

  App Runner 서비스의 잘못된 상태는 다음과 같습니다.
  - `CREATE_FAILED`
  - `DELETE_FAILED`
  - `DELETED`
  - `OPERATION_IN_PROGRESS`

  > **Note**
  >
  > `OPERATION_IN_PROGRESS` 상태는 App Runner 서비스가 삭제되는 경우에만 유효하지 않습니다.
- 요청으로 인해가 검사할 수 있는 한도보다 큰 페이로드가 발생할 AWS WAF 수 있습니다. 가 App Runner의 과대 요청을 AWS WAF 처리하는 방법에 대한 자세한 내용은 AWS WAF 개발자 안내서의 과대 요청 구성 요소 처리를 참조하세요 AWS WAF .
- 적절한 규칙을 설정하지 않거나 트래픽 패턴이 변경되면 웹 ACL이 애플리케이션을 보호하는 데 효과적이지 않을 수 있습니다.

권한

에서 웹 ACL을 사용하려면 AWS WAF다음 IAM 권한을 AWS App Runner추가합니다.

- `apprunner:ListAssociatedServicesForWebAcl`
- `apprunner:DescribeWebAclForService`
- `apprunner:AssociateWebAcl`
- `apprunner:DisassociateWebAcl`

IAM 권한에 대한 자세한 내용은 IAM 사용 설명서의 IAM의 정책 및 권한을 참조하세요.

다음은에 대해 업데이트된 IAM 정책의 예입니다 AWS WAF. 이 IAM 정책에는 App Runner 서비스 작업에 필요한 권한이 포함되어 있습니다.

Example

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "wafv2:ListResourcesForWebACL",
            "wafv2:GetWebACLForResource",
            "wafv2:AssociateWebACL",
            "wafv2:DisassociateWebACL",
            "apprunner:ListAssociatedServicesForWebAcl",
            "apprunner:DescribeWebAclForService",
            "apprunner:AssociateWebAcl",
            "apprunner:DisassociateWebAcl"
         ],
         "Resource":"*"
      }
   ]
}
```

> **Note**
>
> IAM 권한을 부여해야 하지만 나열된 작업은 권한 전용이며 API 작업에 해당하지 않습니다.

AWS WAF 웹 ACLs 관리

다음 방법 중 하나를 사용하여 App Runner 서비스의 AWS WAF 웹 ACLs을 관리합니다.

- the section called “App Runner 콘솔”
- the section called “AWS CLI”

App Runner 콘솔

App Runner 콘솔에서 서비스를 생성하거나 기존 서비스를 업데이트할 때 AWS WAF 웹 ACL을 연결하거나 연결 해제할 수 있습니다.

> **Note**
>
> - App Runner 서비스는 하나의 웹 ACL에만 연결할 수 있습니다. 그러나 다른 AWS 리소스 외에도 웹 ACL 하나를 둘 이상의 App Runner 서비스와 연결할 수 있습니다.
> - 웹 ACL을 연결하기 전에에 대한 IAM 권한을 업데이트해야 합니다 AWS WAF. 자세한 내용은 Permissions 단원을 참조하십시오.

AWS WAF 웹 ACL 연결

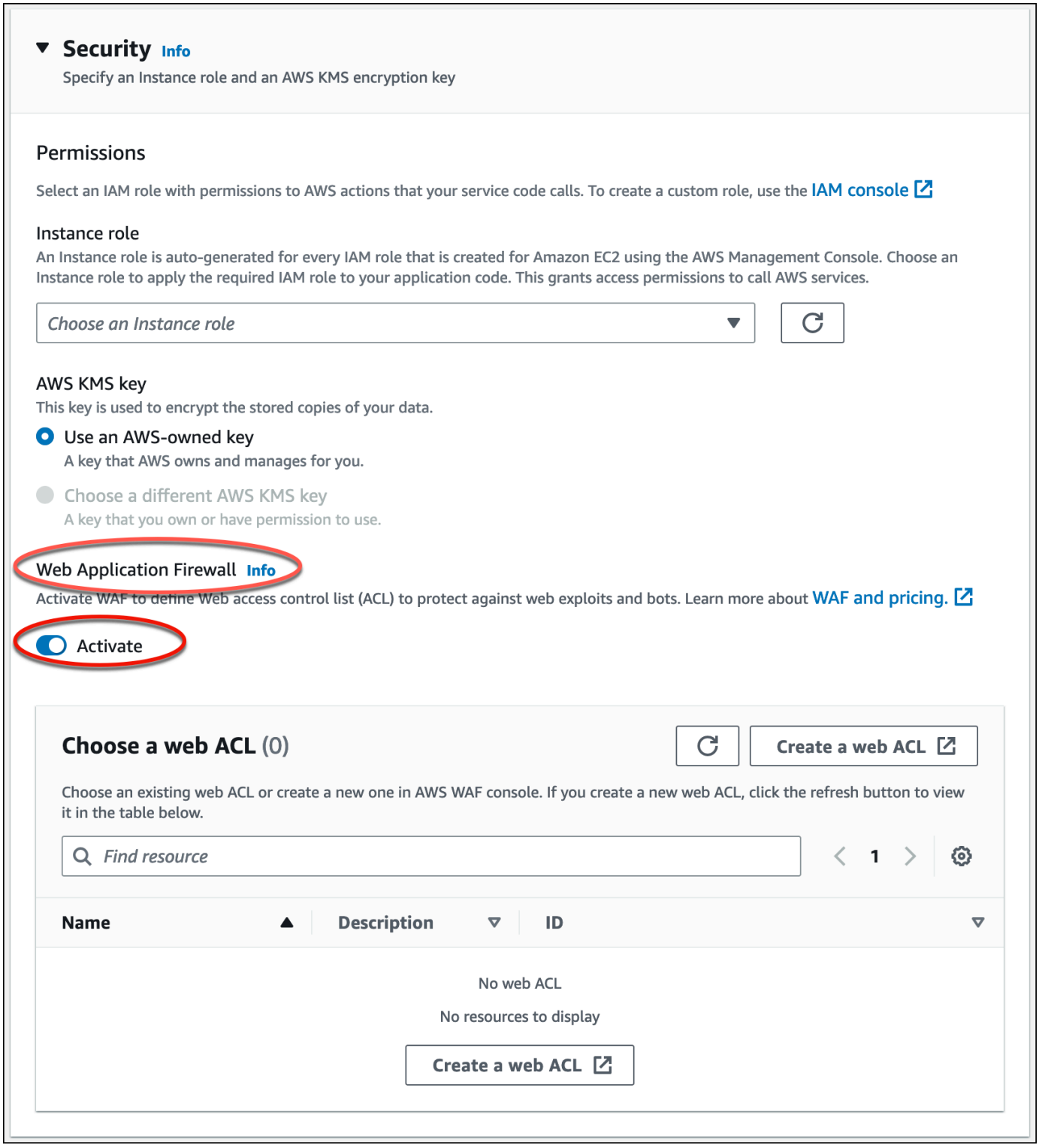
> **Important**
>
> WAF 웹 ACLs과 연결된 App Runner 프라이빗 서비스에 대한 소스 IP 규칙은 IP 기반 규칙을 준수하지 않습니다. 이는 현재 요청 소스 IP 데이터를 WAF와 연결된 App Runner 프라이빗 서비스로 전달하는 것을 지원하지 않기 때문입니다. App Runner 애플리케이션에 소스 IP/CIDR 수신 트래픽 제어 규칙이 필요한 경우 WAF 웹 ACLs 대신 프라이빗 엔드포인트에 대한 보안 그룹 규칙을 사용해야 합니다.

AWS WAF 웹 ACL을 연결하려면

1. App Runner 콘솔을 열고 리전 목록에서를 선택합니다 AWS 리전.
2. 서비스를 생성 또는 업데이트하는지 여부에 따라 다음 단계 중 하나를 수행합니다.
   - 새 서비스를 생성하는 경우 App Runner 서비스 생성을 선택하고 서비스 구성으로 이동합니다.

- 기존 서비스를 업데이트하는 경우 구성 탭을 선택한 다음 서비스 구성에서 편집을 선택합니다.

3. 보안 아래의 웹 애플리케이션 방화벽으로 이동합니다.
4. 옵션을 보려면 전환 활성화 버튼을 선택합니다.



5. 다음 단계 중 하나를 수행하세요.

- 기존 웹 ACL을 연결하려면: App Runner 서비스와 연결할 웹 ACL 선택 테이블에서 필요한 웹 ACL을 선택합니다.
- 새 웹 ACL을 생성하려면: AWS WAF 콘솔을 사용하여 새 웹 ACL을 생성하려면 웹 ACL 생성을 선택합니다. 자세한 내용은 AWS WAF 개발자 안내서의 웹 ACL 생성을 참조하세요.
  1. 새로 고침 버튼을 선택하여 웹 ACL 선택 테이블에서 새로 생성된 웹 ACL을 봅니다.
  2. 필요한 웹 ACL을 선택합니다.

6. 새 서비스를 생성하는 경우 다음을 선택하고 기존 서비스를 업데이트하는 경우 변경 사항 저장을 선택합니다. 선택한 웹 ACL은 App Runner 서비스와 연결됩니다.
7. 웹 ACL 연결을 확인하려면 서비스의 구성 탭을 선택하고 서비스 구성으로 이동합니다. 보안 아래의 웹 애플리케이션 방화벽으로 스크롤하여 서비스와 연결된 웹 ACL의 세부 정보를 확인합니다.

> **Note**
>
> 웹 ACL을 생성하면 웹 ACL이 완전히 전파되기 전에 약간의 시간이 경과하여 App Runner에서 사용할 수 있습니다. 전파 시간은 몇 초에서 몇 분 사이일 수 있습니다.는 웹 ACL이 완전히 전파되기 전에 연결을 시도할 `WAFUnavailableEntityException` 때를 AWS WAF 반환합니다.
> 웹 ACL이 완전히 전파되기 전에 브라우저를 새로 고치거나 App Runner 콘솔에서 벗어나면 연결이 실패합니다. 그러나 App Runner 콘솔 내에서 탐색할 수 있습니다.

AWS WAF 웹 ACL 연결 해제

App Runner 서비스를 업데이트하여 더 이상 필요하지 않은 AWS WAF 웹 ACI의 연결을 해제할 수 있습니다.

AWS WAF 웹 ACI의 연결을 해제하려면

1. App Runner 콘솔을 열고 리전 목록에서를 선택합니다 AWS 리전.
2. 업데이트하려는 서비스의 구성 탭으로 이동하여 구성 서비스에서 편집을 선택합니다.
3. 보안 아래의 웹 애플리케이션 방화벽으로 이동합니다.
4. 전환 활성화 버튼을 비활성화합니다. 삭제를 확인하는 메시지가 표시됩니다.
5. 확인을 선택합니다. 웹 ACL이 App Runner 서비스와 연결 해제되었습니다.

> **Note**
>
> - 서비스를 다른 웹 ACL과 연결하려면 웹 ACL 선택 테이블에서 웹 ACL을 선택합니다. App Runner는 현재 웹 ACL의 연결을 해제하고 선택한 웹 ACL과 연결하는 프로세스를 시작합니다.
> - 연결 해제된 웹 ACL을 사용하는 다른 App Runner 서비스 또는 리소스가 없는 경우 웹 ACL을 삭제하는 것이 좋습니다. 그렇지 않으면 비용이 계속 발생합니다. 요금에 대한 자세한 내용은 AWS WAF 요금을 참조하십시오. 웹 ACL을 삭제하는 방법에 대한 지침은 AWS WAF API 참조의 DeleteWebACL을 참조하세요.
> - 다른 활성 App Runner 서비스 또는 다른 리소스와 연결된 웹 ACL은 삭제할 수 없습니다.

AWS CLI

AWS WAF 퍼블릭 API를 사용하여 AWS WAF 웹 ACL을 연결하거나 연결 해제할 수 있습니다. APIs 웹 ACL을 연결하거나 연결 해제하려는 App Runner 서비스는 유효한 상태여야 합니다.

AWS WAF 는 잘못된 상태인 App Runner 서비스에 대해 다음 API 중 하나를 호출할 때 `WAFNonexistentItemException` 오류를 반환합니다. AWS WAF APIs

- `AssociateWebACL`
- `DisassociateWebACL`
- `GetWebACLForResource`

App Runner 서비스의 잘못된 상태는 다음과 같습니다.

- `CREATE_FAILED`
- `DELETE_FAILED`
- `DELETED`
- `OPERATION_IN_PROGRESS`

> **Note**
>
> `OPERATION_IN_PROGRESS` 상태는 App Runner 서비스가 삭제되는 경우에만 유효하지 않습니다.

AWS WAF 퍼블릭 APIs. AWS WAF

> **Note**
>
> 에 대한 IAM 권한을 업데이트합니다 AWS WAF. 자세한 내용은 Permissions 단원을 참조하십시오.

를 사용하여 AWS WAF 웹 ACL 연결 AWS CLI

> **Important**
>
> WAF 웹 ACLs과 연결된 App Runner 프라이빗 서비스에 대한 소스 IP 규칙은 IP 기반 규칙을 준수하지 않습니다. 이는 현재 요청 소스 IP 데이터를 WAF와 연결된 App Runner 프라이빗 서비스로 전달하는 것을 지원하지 않기 때문입니다. App Runner 애플리케이션에 소스 IP/CIDR 수신 트래픽 제어 규칙이 필요한 경우 WAF 웹 ACLs 대신 프라이빗 엔드포인트에 대한 보안 그룹 규칙을 사용해야 합니다.

AWS WAF 웹 ACL을 연결하려면

1. 에 대한 기본 규칙 작업 세트 `Allow` 또는 서비스에 대한 AWS WAF 웹 요청을 사용하여 서비스에 대한 `Block` 웹 ACL을 생성합니다. AWS WAF APIs에 대한 자세한 내용은 AWS WAF API 참조 안내서의 CreateWebACL을 참조하세요.

   Example웹 ACL 생성 - 요청

   ```
   aws wafv2
   create-web-acl
   --region <region>
   --name <web-acl-name>
   --scope REGIONAL
   ```

```
--default-action Allow={}
--visibility-config <file-name.json>
# This is the file containing the WAF web ACL rules.
```

2. `associate-web-acl` AWS WAF 퍼블릭 API를 사용하여 생성한 웹 ACL을 App Runner 서비스와 연결합니다. AWS WAF APIs에 대한 자세한 내용은 AWS WAF API 참조 안내서의 AssociateWebACL을 참조하세요.

> **Note**
>
> 웹 ACL을 생성하면 웹 ACL이 완전히 전파되기 전에 약간의 시간이 경과하여 App Runner에서 사용할 수 있습니다. 전파 시간은 몇 초에서 몇 분 사이일 수 있습니다.는 웹 ACL이 완전히 전파되기 전에 연결을 시도할 `WAFUnavailableEntityException` 때를 AWS WAF 반환합니다.
>
> 웹 ACL이 완전히 전파되기 전에 브라우저를 새로 고치거나 App Runner 콘솔에서 벗어나면 연결이 실패합니다. 그러나 App Runner 콘솔 내에서 탐색할 수 있습니다.

Example웹 ACL 연결 - 요청

```
aws wafv2 associate-web-acl
--resource-arn <apprunner_service_arn>
--web-acl-arn <web_acl_arn>
--region <region>
```

3. `get-web-acl-for-resource` AWS WAF 퍼블릭 API를 사용하여 웹 ACL이 App Runner 서비스와 연결되어 있는지 확인합니다. AWS WAF APIs에 대한 자세한 내용은 AWS WAF API 참조 안내서의 GetWebACLForResource를 참조하세요.

Example리소스에 대한 웹 ACL 확인 - 요청

```
aws wafv2 get-web-acl-for-resource
--resource-arn <apprunner_service_arn>
--region <region>
```

서비스와 연결된 웹 ACLs이 없는 경우 빈 응답을 받게 됩니다.

를 사용하여 AWS WAF 웹 ACL 삭제 AWS CLI

AWS WAF 웹 ACL이 App Runner 서비스와 연결된 경우 삭제할 수 없습니다.

AWS WAF 웹 ACL을 삭제하려면

1. `disassociate-web-acl` AWS WAF 퍼블릭 API를 사용하여 App Runner 서비스에서 웹 ACL의 연결을 해제합니다. AWS WAF APIs에 대한 자세한 내용은 AWS WAF API 참조 안내서의 DisassociateWebACL을 참조하세요.

   Example웹 ACL 연결 해제 - 요청

   ```
   aws wafv2 disassociate-web-acl
   --resource-arn <apprunner_service_arn>
   --region <region>
   ```

2. `get-web-acl-for-resource` AWS WAF 퍼블릭 API를 사용하여 웹 ACL이 App Runner 서비스에서 연결 해제되었는지 확인합니다.

   Example웹 ACL의 연결이 해제되었는지 확인 - 요청

   ```
   aws wafv2 get-web-acl-for-resource
   --resource-arn <apprunner_service_arn>
   --region <region>
   ```

   연결 해제된 웹 ACL은 App Runner 서비스에 대해 나열되지 않습니다. 서비스와 연결된 웹 ACLs이 없는 경우 빈 응답을 받게 됩니다.

3. `delete-web-acl` AWS WAF 퍼블릭 API를 사용하여 연결 해제된 웹 ACL을 삭제합니다. AWS WAF APIs에 대한 자세한 내용은 AWS WAF API 참조 안내서의 DeleteWebACL을 참조하세요.

   Example웹 ACL 삭제 - 요청

   ```
   aws wafv2 delete-web-acl
   --name <web_acl_name>
   --scope REGIONAL
   --id <web_acl_id>
   --lock-token <web_acl_lock_token>
   --region <region>
   ```

4. `list-web-acl` AWS WAF 퍼블릭 API를 사용하여 웹 ACL이 삭제되었는지 확인합니다. AWS WAF APIs 대한 자세한 내용은 AWS WAF API 참조 안내서의 ListWebACLs를 참조하세요.

Example웹 ACL이 삭제되었는지 확인 - 요청

```
aws wafv2 list-web-acls
--scope REGIONAL
--region <region>
```

삭제된 웹 ACL은 더 이상 나열되지 않습니다.

> **Note**
>
> 웹 ACL이 다른 활성 App Runner 서비스 또는 Amazon Cognito 사용자 풀과 같은 다른 리소스와 연결된 경우 웹 ACL을 삭제할 수 없습니다.

웹 ACL과 연결된 App Runner 서비스 나열

웹 ACL은 여러 App Runner 서비스 및 기타 리소스와 연결할 수 있습니다. `list-resources-for-web-acl` AWS WAF 퍼블릭 API를 사용하여 웹 ACL과 연결된 App Runner 서비스를 나열합니다. AWS WAF APIs 대한 자세한 내용은 AWS WAF API 참조 안내서의 ListResourcesForWebACL을 참조하세요.

Example웹 ACL과 연결된 App Runner 서비스 나열 - 요청

```
aws wafv2 list-resources-for-web-acl
--web-acl-arn <WEB_ACL_ARN>
--resource-type APP_RUNNER_SERVICE
--region <REGION>
```

Example웹 ACL과 연결된 App Runner 서비스 나열 - 응답

다음 예제에서는 웹 ACL과 연결된 App Runner 서비스가 없는 경우의 응답을 보여줍니다.

```
{
  "ResourceArns": []
}
```

Example웹 ACL과 연결된 App Runner 서비스 나열 - 응답

다음 예제에서는 웹 ACL과 연결된 App Runner 서비스가 있는 경우의 응답을 보여줍니다.

```
{
  "ResourceArns": [
    "arn:aws:apprunner:<region>:<aws_account_id>:service/<service_name>/<service_id>"
  ]
}
```

AWS WAF 웹 ACLs 테스트 및 로깅

웹 ACL에서 규칙 작업을 개수로 설정하면가 규칙과 일치하는 요청 수에 요청을 AWS WAF 추가합니다. App Runner 서비스를 사용하여 웹 ACL을 테스트하려면 규칙 작업을 개수로 설정하고 각 규칙과 일치하는 요청 볼륨을 고려합니다. 예를 들어, 정상적인 사용자 트래픽으로 판단되는 많은 수의 요청과 일치하는 `Block` 작업에 대한 규칙을 설정합니다. 이 경우 규칙을 재구성해야 할 수 있습니다. 자세한 내용은 AWS WAF 개발자 안내서의 AWS WAF 보호 기능 테스트 및 튜닝을 참조하세요.

요청 헤더를 Amazon CloudWatch Logs 로그 그룹, Amazon Simple Storage Service(Amazon S3) 버킷 또는 Amazon Data Firehose에 로깅 AWS WAF 하도록를 구성할 수도 있습니다. 자세한 내용은 AWS WAF 개발자 안내서의 웹 ACL 트래픽 로깅을 참조하세요.

App Runner 서비스와 연결된 웹 ACL과 관련된 로그에 액세스하려면 다음 로그 필드를 참조하세요.

- `httpSourceName`: 포함 `APPRUNNER`
- `httpSourceId`: 포함 `customeraccountid-apprunnerserviceid`

자세한 내용은 AWS WAF 개발자 안내서의 로그 예제를 참조하세요.

> **Important**
>
> WAF 웹 ACLs과 연결된 App Runner 프라이빗 서비스에 대한 소스 IP 규칙은 IP 기반 규칙을 준수하지 않습니다. 이는 현재 요청 소스 IP 데이터를 WAF와 연결된 App Runner 프라이빗 서비스로 전달하는 것을 지원하지 않기 때문입니다. App Runner 애플리케이션에 소스 IP/CIDR 수신 트래픽 제어 규칙이 필요한 경우 WAF 웹 ACLs 대신 프라이빗 엔드포인트에 대한 보안 그룹 규칙을 사용해야 합니다.

구성 파일을 사용하여 App Runner 서비스 옵션 설정

> **Note**
>
> 구성 파일은 소스 코드를 기반으로 하는 서비스에만 적용됩니다. 이미지 기반 서비스에서는 구성 파일을 사용할 수 없습니다.

소스 코드 리포지토리를 사용하여 AWS App Runner 서비스를 생성할 때는 서비스 구축 및 시작에 대한 정보가 AWS App Runner 필요합니다. App Runner 콘솔 또는 API를 사용하여 서비스를 생성할 때마다이 정보를 제공할 수 있습니다. 또는 구성 파일을 사용하여 서비스 옵션을 설정할 수 있습니다. 파일에서 지정하는 옵션은 소스 리포지토리의 일부가 되며, 이러한 옵션에 대한 모든 변경 사항은 소스 코드에 대한 변경 사항을 추적하는 방법과 유사하게 추적됩니다. App Runner 구성 파일을 사용하여 API가 지원하는 것보다 더 많은 옵션을 지정할 수 있습니다. API가 지원하는 기본 옵션만 필요한 경우 구성 파일을 제공할 필요가 없습니다.

App Runner 구성 파일은 애플리케이션 리포지토리의 소스 디렉터리`apprunner.yaml`에 이름이 지정된 YAML 파일입니다. 서비스에 대한 빌드 및 런타임 옵션을 제공합니다. 이 파일의 값은 App Runner에 서비스를 빌드 및 시작하고 네트워크 설정 및 환경 변수와 같은 런타임 컨텍스트를 제공하는 방법을 지시합니다.

App Runner 구성 파일에는 CPU 및 메모리와 같은 운영 설정이 포함되지 않습니다.

App Runner 구성 파일의 예는 섹션을 참조하세요the section called “예제”. 전체 참조 가이드는 섹션을 참조하세요the section called “레퍼런스”.

주제

- App Runner 구성 파일 예제
- App Runner 구성 파일 참조

App Runner 구성 파일 예제

> **Note**
>
> 구성 파일은 소스 코드를 기반으로 하는 서비스에만 적용됩니다. 이미지 기반 서비스에서는 구성 파일을 사용할 수 없습니다.

다음 예제에서는 AWS App Runner 구성 파일을 보여줍니다. 일부는 최소 수준이며 필수 설정만 포함합니다. 모든 구성 파일 섹션을 포함하여 다른 구성은 완료됩니다. App Runner 구성 파일의 개요는 섹션을 참조하세요App Runner 구성 파일.

구성 파일 예제

최소 구성 파일

App Runner는 최소한의 구성 파일을 사용하여 다음과 같은 가정을 합니다.

- 빌드 또는 실행 중에는 사용자 지정 환경 변수가 필요하지 않습니다.
- 최신 런타임 버전이 사용됩니다.
- 기본 포트 번호와 포트 환경 변수가 사용됩니다.

Example apprunner.yaml

```
version: 1.0
runtime: python3
build:
  commands:
    build:
      - pip install pipenv
      - pipenv install
run:
  command: python app.py
```

구성 파일 완료

이 예제에서는 관리형 런타임에서 `apprunner.yaml` 원래 형식의 모든 구성 키를 사용하는 방법을 보여줍니다.

Example apprunner.yaml

```
version: 1.0
runtime: python3
build:
  commands:
    pre-build:
      - wget -c https://s3.amazonaws.com/amzn-s3-demo-bucket/test-lib.tar.gz -O - | tar
 -xz
```

```
    build:
      - pip install pipenv
      - pipenv install
    post-build:
      - python manage.py test
  env:
    - name: DJANGO_SETTINGS_MODULE
      value: "django_apprunner.settings"
    - name: MY_VAR_EXAMPLE
      value: "example"
run:
  runtime-version: 3.7.7
  command: pipenv run gunicorn django_apprunner.wsgi --log-file -
  network:
    port: 8000
    env: MY_APP_PORT
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
  secrets:
    - name: my-secret
      value-from: "arn:aws:secretsmanager:us-
east-1:123456789012:secret:testingstackAppRunnerConstr-kJFXde2ULKbT-S7t8xR:username::"
    - name: my-parameter
      value-from: "arn:aws:ssm:us-east-1:123456789012:parameter/parameter-name"
    - name: my-parameter-only-name
      value-from: "parameter-name"
```

전체 구성 파일 - (개정된 빌드 사용)

이 예제에서는 관리형 런타임과 `apprunner.yaml` 함께의 모든 구성 키를 사용하는 방법을 보여줍니다.

`pre-run` 파라미터는 수정된 App Runner 빌드에서만 지원됩니다. 애플리케이션이 원래 App Runner 빌드에서 지원하는 런타임 버전을 사용하는 경우 구성 파일에이 파라미터를 삽입하지 마십시오. 자세한 내용은 관리형 런타임 버전 및 App Runner 빌드 단원을 참조하십시오.

> **Note**
>
> 이 예제는 Python 3.11용이므로 `pip3` 및 `python3` 명령을 사용합니다. 자세한 내용은 Python 플랫폼 주제특정 런타임 버전에 대한 콜아웃의 섹션을 참조하세요.

Example apprunner.yaml

```
version: 1.0
runtime: python311
build:
  commands:
    pre-build:
      - wget -c https://s3.amazonaws.com/amzn-s3-demo-bucket/test-lib.tar.gz -O - | tar
 -xz
    build:
      - pip3 install pipenv
      - pipenv install
    post-build:
      - python3 manage.py test
  env:
    - name: DJANGO_SETTINGS_MODULE
      value: "django_apprunner.settings"
    - name: MY_VAR_EXAMPLE
      value: "example"
run:
  runtime-version: 3.11
  pre-run:
    - pip3 install pipenv
    - pipenv install
    - python3 copy-global-files.py
  command: pipenv run gunicorn django_apprunner.wsgi --log-file -
  network:
    port: 8000
    env: MY_APP_PORT
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
  secrets:
    - name: my-secret
      value-from: "arn:aws:secretsmanager:us-
east-1:123456789012:secret:testingstackAppRunnerConstr-kJFXde2ULKbT-S7t8xR:username::"
    - name: my-parameter
      value-from: "arn:aws:ssm:us-east-1:123456789012:parameter/parameter-name"
    - name: my-parameter-only-name
      value-from: "parameter-name"
```

특정 관리형 런타임 구성 파일의 예는 아래의 특정 런타임 하위 주제를 참조하세요코드 기반 서비스.

App Runner 구성 파일 참조

> **Note**
>
> 구성 파일은 소스 코드를 기반으로 하는 서비스에만 적용됩니다. 이미지 기반 서비스에서는 구성 파일을 사용할 수 없습니다.

이 주제는 AWS App Runner 구성 파일의 구문 및 의미 체계에 대한 포괄적인 참조 가이드입니다. App Runner 구성 파일의 개요는 섹션을 참조하세요App Runner 구성 파일.

App Runner 구성 파일은 YAML 파일입니다. 이름을 로 `apprunner.yaml`지정하고 애플리케이션 리포지토리의 소스 디렉터리에 배치합니다.

구조 개요

App Runner 구성 파일은 YAML 파일입니다. 이름을 로 `apprunner.yaml`지정하고 애플리케이션 리포지토리의 소스 디렉터리에 배치합니다.

App Runner 구성 파일에는 다음과 같은 주요 부분이 포함되어 있습니다.

- 상단 섹션 - 최상위 키 포함
- 빌드 섹션 - 빌드 단계를 구성합니다.
- 실행 섹션 - 런타임 단계를 구성합니다.

상단 섹션

파일 상단의 키는 파일 및 서비스 런타임에 대한 일반적인 정보를 제공합니다. 다음 키를 사용할 수 있습니다.

- `version` - 필수 항목입니다. App Runner 구성 파일 버전입니다. 최신 버전을 사용하는 것이 가장 좋습니다.

  구문

```
version: version
```

Example

```
version: 1.0
```

- `runtime` - 필수 항목입니다. 애플리케이션이 사용하는 런타임의 이름입니다. App Runner가 제공하는 다양한 프로그래밍 플랫폼에 사용 가능한 런타임에 대한 자세한 내용은 섹션을 참조하세요코드 기반 서비스.

> **Note**
>
> 관리형 런타임의 이름 지정 규칙은 *<language-name><major-version>*입니다.

구문

```
runtime: runtime-name
```

Example

```
runtime: python3
```

빌드 섹션

빌드 섹션은 App Runner 서비스 배포의 빌드 단계를 구성합니다. 빌드 명령과 환경 변수를 지정할 수 있습니다. 빌드 명령이 필요합니다.

섹션은 `build:` 키로 시작하며 다음과 같은 하위 키가 있습니다.

- `commands` - 필수 항목입니다. 다양한 빌드 단계에서 App Runner가 실행하는 명령을 지정합니다. 다음 하위 키를 포함합니다.
  - `pre-build` - 선택 사항입니다. App Runner가 빌드 전에 실행하는 명령입니다. 예를 들어 npm 종속성 또는 테스트 라이브러리를 설치합니다.
  - `build` - 필수 항목입니다. App Runner가 애플리케이션을 빌드하기 위해 실행하는 명령입니다. 예를 들어를 사용합니다pipenv.
  - `post-build` - 선택 사항입니다. App Runner가 빌드 후 실행하는 명령입니다. 예를 들어 Maven을 사용하여 빌드 아티팩트를 JAR 또는 WAR 파일로 패키징하거나 테스트를 실행합니다.

구문

```
build:
  commands:
    pre-build:
      - command
      - …
    build:
      - command
      - …
    post-build:
      - command
      - …
```

Example

```
build:
  commands:
    pre-build:
      - yum install openssl
    build:
      - pip install -r requirements.txt
    post-build:
      - python manage.py test
```

- env - 선택 사항입니다. 빌드 단계의 사용자 지정 환경 변수를 지정합니다. 이름-값 스칼라 매핑으로 정의됩니다. 빌드 명령에서 이러한 변수를 이름으로 참조할 수 있습니다.

> **Note**
>
> 이 구성 파일의 서로 다른 두 위치에는 두 개의 고유한 env 항목이 있습니다. 한 세트는 빌드 섹션에 있고 다른 세트는 실행 섹션에 있습니다.
>
> - 빌드 섹션의 env 세트는 빌드 프로세스 중에 pre-build, buildpost-build, 및 pre-run 명령에서 참조할 수 있습니다.
>
> 중요 - pre-run 명령은 빌드 섹션에 정의된 환경 변수에만 액세스할 수 있더라도이 파일의 실행 섹션에 있습니다.
>
> - 실행 섹션의 env 세트는 런타임 환경의 run 명령에서 참조할 수 있습니다.

구문

```
build:
  env:
    - name: name1
      value: value1
    - name: name2
      value: value2
    - …
```

Example

```
build:
  env:
    - name: DJANGO_SETTINGS_MODULE
      value: "django_apprunner.settings"
    - name: MY_VAR_EXAMPLE
      value: "example"
```

실행 섹션

실행 섹션은 App Runner 애플리케이션 배포의 컨테이너 실행 단계를 구성합니다. 런타임 버전, 사전 실행 명령(개정된 형식만 해당), 시작 명령, 네트워크 포트 및 환경 변수를 지정할 수 있습니다.

섹션은 `run:` 키로 시작하며 다음과 같은 하위 키가 있습니다.

- `runtime-version` - 선택 사항입니다. App Runner 서비스에 대해 잠그려는 런타임 버전을 지정합니다.

  기본적으로 메이저 버전만 잠깁니다. App Runner는 모든 배포 또는 서비스 업데이트에서 런타임에 사용할 수 있는 최신 마이너 및 패치 버전을 사용합니다. 메이저 버전과 마이너 버전을 지정하면 둘 다 잠기고 App Runner는 패치 버전만 업데이트합니다. 메이저, 마이너 및 패치 버전을 지정하면 서비스가 특정 런타임 버전에서 잠기고 App Runner는 이를 업데이트하지 않습니다.

  구문

```
run:
  runtime-version: major[.minor[.patch]]
```

> **Note**
>
> 일부 플랫폼의 런타임에는 서로 다른 버전 구성 요소가 있습니다. 자세한 내용은 특정 플랫폼 주제를 참조하세요.

Example

```
runtime: python3
run:
  runtime-version: 3.7
```

- `pre-run` - 선택 사항입니다. 빌드 사용량만 수정했습니다. 빌드 이미지에서 실행 이미지로 애플리케이션을 복사한 후 App Runner가 실행하는 명령을 지정합니다. 여기에 명령을 입력하여 `/app` 디렉터리 외부에서 실행 이미지를 수정할 수 있습니다. 예를 들어 `/app` 디렉터리 외부에 있는 글로벌 종속성을 추가로 설치해야 하는 경우이 하위 섹션에 필요한 명령을 입력하여 설치합니다. App Runner 빌드 프로세스에 대한 자세한 내용은 섹션을 참조하세요관리형 런타임 버전 및 App Runner 빌드.

> **Note**
>
> - 중요 - `pre-run` 명령이 실행 섹션에 나열되더라도이 구성 파일의 빌드 섹션에 정의된 환경 변수만 참조할 수 있습니다. 이 실행 섹션에 정의된 환경 변수는 참조할 수 없습니다.
> - `pre-run` 파라미터는 수정된 App Runner 빌드에서만 지원됩니다. 애플리케이션이 원래 App Runner 빌드에서 지원하는 런타임 버전을 사용하는 경우 구성 파일에이 파라미터를 삽입하지 마십시오. 자세한 내용은 관리형 런타임 버전 및 App Runner 빌드 단원을 참조하십시오.

구문

```
run:
  pre-run:
      - command
      - …
```

- `command` - 필수 항목입니다. App Runner가 애플리케이션 빌드를 완료한 후 애플리케이션을 실행하는 데 사용하는 명령입니다.

구문

```
run:
  command: command
```

- network - 선택 사항입니다. 애플리케이션이 수신 대기하는 포트를 지정합니다. 여기에는 다음이 포함됩니다.
  - port - 선택 사항입니다. 지정된 경우 애플리케이션이 수신하는 포트 번호입니다. 기본값은 8080입니다.
  - env - 선택 사항입니다. 지정된 경우 App Runner는 기본 환경 변수인에서 동일한 포트 번호를 전달하는 것 외에도(대신)이 환경 변수의 컨테이너에 포트 번호를 전달합니다PORT. 즉,를 지정하면 envApp Runner는 두 환경 변수에 포트 번호를 전달합니다.

  구문

  ```
  run:
    network:
      port: port-number
      env: env-variable-name
  ```

  Example

  ```
  run:
    network:
      port: 8000
      env: MY_APP_PORT
  ```

- env - 선택 사항입니다. 실행 단계에 대한 사용자 지정 환경 변수의 정의입니다. 이름-값 스칼라 매핑으로 정의됩니다. 런타임 환경에서 이러한 변수를 이름으로 참조할 수 있습니다.

  > **Note**
  >
  > 이 구성 파일의 서로 다른 두 위치에는 두 개의 고유한 env 항목이 있습니다. 한 세트는 빌드 섹션에 있고 다른 세트는 실행 섹션에 있습니다.
  >
  > - 빌드 섹션의 env 세트는 빌드 프로세스 중에 pre-build, buildpost-build, 및 pre-run 명령에서 참조할 수 있습니다.

중요 - pre-run 명령은 빌드 섹션에 정의된 환경 변수에만 액세스할 수 있더라도이 파일의 실행 섹션에 있습니다.

- 실행 섹션의 env 세트는 런타임 환경의 run 명령에서 참조할 수 있습니다.

구문

```
run:
  env:
    - name: name1
      value: value1
    - name: name2
      value: value2
  secrets:
    - name: name1
      value-from: arn:aws:secretsmanager:region:aws_account_id:secret:secret-id
    - name: name2
      value-from: arn:aws:ssm:region:aws_account_id:parameter/parameter-name
    - …
```

Example

```
run:
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
  secrets:
    - name: my-secret
      value-from: "arn:aws:secretsmanager:us-
east-1:123456789012:secret:testingstackAppRunnerConstr-kJFXde2ULKbT-
S7t8xR:username::"
    - name: my-parameter
      value-from: "arn:aws:ssm:us-east-1:123456789012:parameter/parameter-name"
    - name: my-parameter-only-name
      value-from: "parameter-name"
```

App Runner API

AWS App Runner 애플리케이션 프로그래밍 인터페이스(API)는 App Runner 서비스에 요청하기 위한 RESTful API입니다. API를 사용하여에서 App Runner 리소스를 생성, 나열, 설명, 업데이트 및 삭제할 수 있습니다 AWS 계정.

애플리케이션 코드에서 직접 API를 호출하거나 AWS SDKs.

전체 API 참조 정보는 AWS App Runner API 참조를 참조하세요.

AWS 개발자 도구에 대한 자세한 내용은 빌드 기반 도구를 참조하세요 AWS.

주제

- AWS CLI 를 사용하여 App Runner 작업
- AWS CloudShell 를 사용하여 작업 AWS App Runner

AWS CLI 를 사용하여 App Runner 작업

명령줄 스크립트의 경우 AWS CLI를 사용하여 App Runner 서비스를 호출합니다. 전체 AWS CLI 참조 정보는 AWS CLI 명령 참조의 apprunner를 참조하세요.

AWS CloudShell 를 사용하면 개발 환경에서 AWS CLI 설치를 건너뛰고 AWS Management Console 대신에서 사용할 수 있습니다. 설치를 피하는 것 외에도 자격 증명을 구성할 필요가 없으며 리전을 지정할 필요가 없습니다. AWS Management Console 세션은이 컨텍스트를에 제공합니다 AWS CLI. CloudShell에 대한 자세한 내용과 사용 예제는 섹션을 참조하세요the section called “사용 AWS CloudShell”.

AWS CloudShell 를 사용하여 작업 AWS App Runner

AWS CloudShell 는 브라우저 기반의 사전 인증된 셸로,에서 직접 시작할 수 있습니다 AWS Management Console. 원하는 쉘(Bash, PowerShell 또는 Z 쉘 AWS App Runner)을 사용하여 AWS 서비스(포함)에 대해 AWS CLI 명령을 실행할 수 있습니다. 또한 명령줄 도구를 다운로드하거나 설치할 필요 없이 이 작업을 수행할 수 있습니다.

AWS CloudShell 에서를 시작 AWS Management Console하면 콘솔에 로그인하는 데 사용한 AWS 자격 증명이 새 쉘 세션에서 자동으로 사용할 수 있습니다. 이 AWS CloudShell 사용자 사전 인증을 사용

하면 AWS CLI 버전 2(셸의 컴퓨팅 환경에 사전 설치됨)를 사용하여 App Runner와 같은 AWS 서비스와 상호 작용할 때 자격 증명 구성을 건너뛸 수 있습니다.

주제

- 에 대한 IAM 권한 획득 AWS CloudShell
- 를 사용하여 App Runner와 상호 작용 AWS CloudShell
- 를 사용하여 App Runner 서비스 확인 AWS CloudShell

에 대한 IAM 권한 획득 AWS CloudShell

AWS Identity and Access Management관리자는에서 제공하는 액세스 관리 리소스를 사용하여 IAM 사용자에게 환경의 기능에 액세스 AWS CloudShell 하고 사용할 수 있는 권한을 부여할 수 있습니다.

관리자가 사용자에게 액세스 권한을 부여하는 가장 빠른 방법은 AWS 관리형 정책을 사용하는 것입니다. AWS 관리형 정책은 AWS에서 생성 및 관리하는 독립 실행형 정책입니다. CloudShell에 대한 다음 AWS 관리형 정책을 IAM 자격 증명에 연결할 수 있습니다.

- `AWSCloudShellFullAccess`: 모든 기능에 대한 전체 액세스 권한 AWS CloudShell 과 함께를 사용할 수 있는 권한을 부여합니다.

IAM 사용자가 수행할 수 있는 작업 범위를 제한하려면 `AWSCloudShellFullAccess` 관리형 정책을 템플릿으로 사용하는 사용자 지정 정책을 생성할 AWS CloudShell수 있습니다. CloudShell에서 사용자가 사용할 수 있는 작업을 제한하는 방법에 대한 자세한 내용은 AWS CloudShell 사용 설명서의 IAM 정책을 사용한 AWS CloudShell 액세스 및 사용 관리를 참조하세요.

> **Note**
>
> IAM 자격 증명에는 App Runner를 호출할 수 있는 권한을 부여하는 정책도 필요합니다. 자세한 내용은 the section called “App Runner 및 IAM” 단원을 참조하십시오.

를 사용하여 App Runner와 상호 작용 AWS CloudShell

AWS CloudShell 에서를 시작한 후 명령줄 인터페이스를 사용하여 App Runner와 즉시 상호 작용할 AWS Management Console수 있습니다.

다음 예제에서는 CloudShell의 AWS CLI 를 사용하여 App Runner 서비스 중 하나에 대한 정보를 검색합니다.

> **Note**
>
> AWS CLI 에서를 사용하는 AWS CloudShell경우 추가 리소스를 다운로드하거나 설치할 필요가 없습니다. 또한 셸 내에서 이미 인증되었기 때문에 직접 호출을 하기 전에 보안 인증을 구성하지 않아도 됩니다.

Example를 사용하여 App Runner 서비스 정보 검색 AWS CloudShell

1. 에서 탐색 모음에서 사용할 AWS Management Console수 있는 다음 옵션을 선택하여 CloudShell을 시작할 수 있습니다.

   - CloudShell 아이콘을 선택합니다.
   - **`cloudshell`** 검색 상자에 입력을 시작한 다음 검색 결과에 CloudShell 옵션이 표시되면 선택합니다.

2. 콘솔 세션의 AWS 리전에 AWS 있는 계정의 현재 App Runner 서비스를 모두 나열하려면 CloudShell 명령줄에 다음 명령을 입력합니다.

   ```
   $ aws apprunner list-services
   ```

   출력에는 서비스에 대한 요약 정보가 나열됩니다.

   ```
   {
     "ServiceSummaryList": [
       {
         "ServiceName": "my-app-1",
         "ServiceId": "8fe1e10304f84fd2b0df550fe98a71fa",
         "ServiceArn": "arn:aws:apprunner:us-east-2:123456789012:service/my-
   app-1/8fe1e10304f84fd2b0df550fe98a71fa",
         "ServiceUrl": "psbqam834h.us-east-1.awsapprunner.com",
         "CreatedAt": "2020-11-20T19:05:25Z",
         "UpdatedAt": "2020-11-23T12:41:37Z",
         "Status": "RUNNING"
       },
       {
         "ServiceName": "my-app-2",
         "ServiceId": "ab8f94cfe29a460fb8760afd2ee87555",
         "ServiceArn": "arn:aws:apprunner:us-east-2:123456789012:service/my-app-2/
   ab8f94cfe29a460fb8760afd2ee87555",
         "ServiceUrl": "e2m8rrrx33.us-east-1.awsapprunner.com",
   ```

```
      "CreatedAt": "2020-11-06T23:15:30Z",
      "UpdatedAt": "2020-11-23T13:21:22Z",
      "Status": "RUNNING"
    }
  ]
}
```

3. 특정 App Runner 서비스에 대한 자세한 설명을 보려면 이전 단계에서 검색된 ARNs 중 하나를 사용하여 CloudShell 명령줄에 다음 명령을 입력합니다.

```
$ aws apprunner describe-service --service-arn arn:aws:apprunner:us-
east-2:123456789012:service/my-app-1/8fe1e10304f84fd2b0df550fe98a71fa
```

출력에는 지정한 서비스에 대한 자세한 설명이 나열됩니다.

```
{
  "Service": {
    "ServiceName": "my-app-1",
    "ServiceId": "8fe1e10304f84fd2b0df550fe98a71fa",
    "ServiceArn": "arn:aws:apprunner:us-east-2:123456789012:service/my-
app-1/8fe1e10304f84fd2b0df550fe98a71fa",
    "ServiceUrl": "psbqam834h.us-east-1.awsapprunner.com",
    "CreatedAt": "2020-11-20T19:05:25Z",
    "UpdatedAt": "2020-11-23T12:41:37Z",
    "Status": "RUNNING",
    "SourceConfiguration": {
      "CodeRepository": {
        "RepositoryUrl": "https://github.com/my-account/python-hello",
        "SourceCodeVersion": {
          "Type": "BRANCH",
          "Value": "main"
        },
        "CodeConfiguration": {
          "CodeConfigurationValues": {
            "BuildCommand": "[pip install -r requirements.txt]",
            "Port": "8080",
            "Runtime": "PYTHON_3",
            "RuntimeEnvironmentVariables": [
              {
                "NAME": "Jane"
              }
            ],
            "StartCommand": "python server.py"
```

```
          },
          "ConfigurationSource": "API"
        }
      },
      "AutoDeploymentsEnabled": true,
      "AuthenticationConfiguration": {
        "ConnectionArn": "arn:aws:apprunner:us-east-2:123456789012:connection/my-
github-connection/e7656250f67242d7819feade6800f59e"
      }
    },
    "InstanceConfiguration": {
      "CPU": "1 vCPU",
      "Memory": "3 GB"
    },
    "HealthCheckConfiguration": {
      "Protocol": "TCP",
      "Path": "/",
      "Interval": 10,
      "Timeout": 5,
      "HealthyThreshold": 1,
      "UnhealthyThreshold": 5
    },
    "AutoScalingConfigurationSummary": {
      "AutoScalingConfigurationArn": "arn:aws:apprunner:us-
east-2:123456789012:autoscalingconfiguration/
DefaultConfiguration/1/00000000000000000000000000000001",
      "AutoScalingConfigurationName": "DefaultConfiguration",
      "AutoScalingConfigurationRevision": 1
    }
  }
}
```

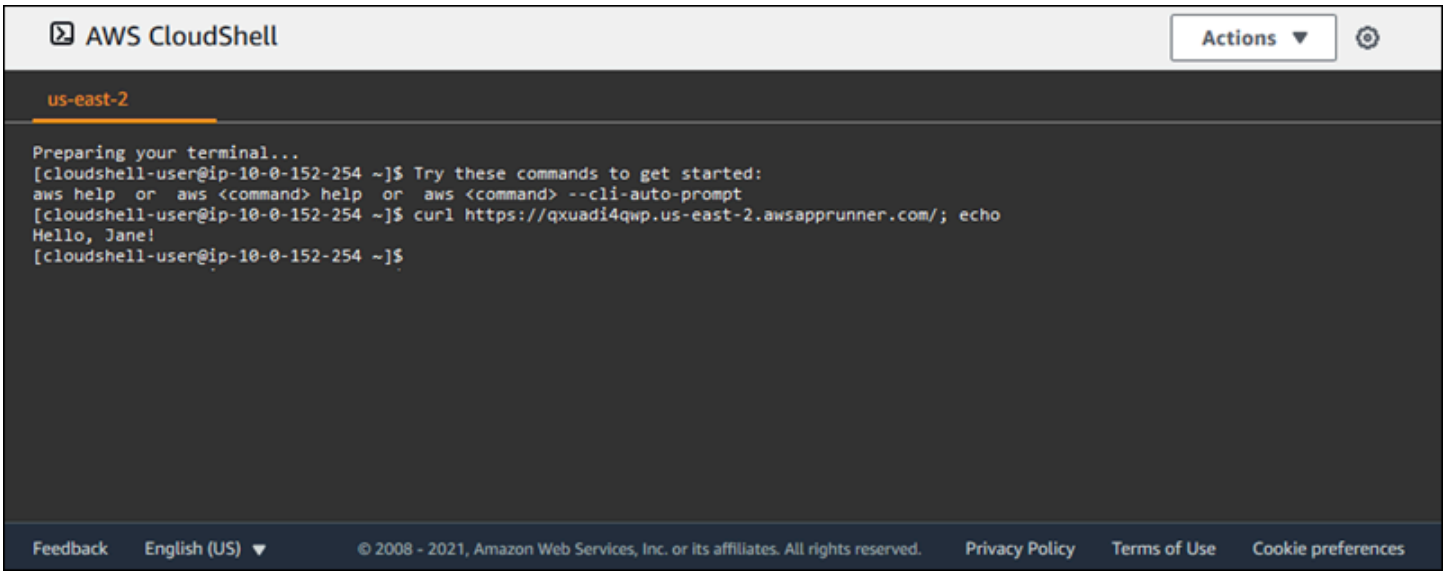

를 사용하여 App Runner 서비스 확인 AWS CloudShell

App Runner 서비스를 생성하면 App Runner는 서비스 웹 사이트의 기본 도메인을 생성하여 콘솔에 표시합니다(또는 API 호출 결과에 반환). CloudShell을 사용하여 웹 사이트를 호출하고 올바르게 작동하는지 확인할 수 있습니다.

예를 들어에 설명된 대로 App Runner 서비스를 생성한 후 CloudShell에서 다음 명령을 시작하기실행합니다.

```
$ curl https://qxuadi4qwp.us-east-2.awsapprunner.com/; echo
```

출력에는 예상 페이지 콘텐츠가 표시되어야 합니다.



문제 해결

이 장에서는 AWS App Runner 서비스를 사용하는 동안 발생할 수 있는 일반적인 오류 및 문제에 대한 문제 해결 단계를 제공합니다. 오류 메시지는 서비스 페이지의 콘솔, API 또는 로그 탭에 나타날 수 있습니다.

문제 해결 조언과 일반적인 지원 질문에 대한 답변은 지식 센터를 참조하세요.

주제

서비스가를 생성하지 못하는 경우

App Runner 서비스 생성 시도가 실패하면 서비스가 `CREATE_FAILED` 상태가 됩니다. 이 상태는 콘솔에서 생성 실패로 표시됩니다. 다음 중 하나 이상과 관련된 문제로 인해 서비스가 생성되지 않을 수 있습니다.

- 애플리케이션 코드
- 빌드 프로세스
- 구성
- 리소스 할당량
- 서비스 AWS 서비스 에서 사용하는 기본의 일시적인 문제

서비스 생성 실패 문제를 해결하려면 다음을 수행하는 것이 좋습니다.

1. 서비스 이벤트 및 로그를 읽고 서비스가 생성되지 않는 원인을 알아봅니다.
2. 코드 또는 구성을 필요한 대로 변경합니다.
3. 서비스 할당량에 도달한 경우 하나 이상의 서비스를 삭제합니다.
4. 다른 리소스 할당량에 도달한 경우 조정 가능한 경우 리소스 할당량을 늘릴 수 있습니다.

5. 위의 단계를 모두 완료한 후 서비스를 다시 빌드해 보십시오. 서비스를 다시 빌드하는 방법에 대한 자세한 내용은 섹션을 참조하세요the section called “실패한 서비스 재구축”.

> **Note**
>
> 문제를 일으킬 수 있는 조정 가능한 리소스 할당량 중 하나는 Fargate 온디맨드 vCPU 리소스입니다.
> vCPU 리소스 수는 App Runner가 서비스에 제공할 수 있는 인스턴스 수를 결정합니다. 서비스에 있는 Fargate 온디맨드 vCPU 리소스 수에 대한 조정 가능한 할당량 값입니다 AWS Fargate . 계정의 vCPU 할당량 설정을 보거나 할당량 증가를 요청하려면에서 Service Quotas 콘솔을 사용합니다 AWS Management Console. 자세한 내용은 Amazon Elastic Container Service 개발자 안내서의 AWS Fargate 서비스 할당량을 참조하세요.

> **Important**
>
> 실패한 서비스에 대한 최초 생성 시도 이후에는 추가 요금이 발생하지 않습니다. 실패한 서비스를 사용할 수 없더라도 여전히 서비스 할당량에 포함됩니다. App Runner는 실패한 서비스를 자동으로 삭제하지 않으므로 실패 분석을 마쳤을 때 삭제해야 합니다.

사용자 지정 도메인 이름

이 섹션에서는 사용자 지정 도메인에 연결하는 동안 발생할 수 있는 다양한 오류를 해결하고 해결하는 방법을 다룹니다.

> **Note**
>
> App Runner 애플리케이션의 보안을 강화하기 위해 *.awsapprunner.com 도메인은 퍼블릭 접미사 목록(PSL)에 등록됩니다. 보안을 강화하기 위해 App Runner 애플리케이션의 기본 도메인 이름에 민감한 쿠키를 설정해야 하는 경우 `__Host-` 접두사가 있는 쿠키를 사용하는 것이 좋습니다. 이렇게 쿠키를 설정하면 교차 사이트 요청 위조 시도(CSRF)로부터 도메인을 보호하는 데 도움이 됩니다. 자세한 내용은 Mozilla 개발자 네트워크의 Set-Cookie 페이지를 참조하세요.

사용자 지정 도메인에 대한 생성 실패 오류 가져오기

- 이 오류가 CAA 레코드 문제로 인한 것인지 확인합니다. DNS 트리에 CAA 레코드가 없는 경우 메시지를 수신`fail open`하고 인증서를 AWS Certificate Manager 발급하여 사용자 지정 도메인을 확인합니다. 이렇게 하면 App Runner가 사용자 지정 도메인을 수락할 수 있습니다. DNS 레코드에서 CAA 인증을 사용하는 경우 하나 이상의 도메인 CAA 레코드에이 포함되어 있는지 확인합니다`amazon.com`. 그렇지 않으면 ACM이 인증서를 발급하지 못합니다. 따라서 App Runner의 사용자 지정 도메인이 생성되지 않습니다.

  다음 예제에서는 DNS 조회 도구 DiG를 사용하여 필수 항목이 누락된 CAA 레코드를 표시합니다. 이 예제에서는를 사용자 지정 도메인`example.com`으로 사용합니다. 예제에서 다음 명령을 실행하여 CAA 레코드를 확인합니다.

```
...
;; QUESTION SECTION:
;example.com.            IN  CAA

;; ANSWER SECTION:
example.com.         7200    IN  CAA 0 iodef "mailto:hostmaster@example.com"
example.com.         7200    IN  CAA 0 issue "letsencrypt.org"
...note absence of "amazon.com" in any of the above CAA records...
```

- 도메인 레코드를 수정하고 하나 이상의 CAA 레코드에이 포함되어 있는지 확인합니다`amazon.com`.
- 사용자 지정 도메인을 App Runner와 연결하려고 다시 시도합니다.

CAA 오류를 해결하는 방법에 대한 지침은 다음을 참조하세요.

- 인증 기관 인증(CAA) 문제
- ACM 인증서 발급 또는 갱신에 대한 CAA 오류를 해결하려면 어떻게 해야 합니까?

사용자 지정 도메인에 대한 DNS 인증서 검증 보류 중 오류 가져오기

- 사용자 지정 도메인 설정에서 중요한 단계를 건너뛰었는지 확인합니다. 또한 DiG와 같은 DNS 조회 도구를 사용하여 DNS 레코드를 잘못 구성했는지 확인합니다. 특히 다음과 같은 실수가 있는지 확인합니다.
  - 누락된 단계.

  - DNS 레코드의 큰따옴표와 같이 지원되지 않는 문자입니다.
- 실수를 수정합니다.
- 사용자 지정 도메인을 App Runner와 연결하려고 다시 시도합니다.

CAA 검증 오류를 해결하는 방법에 대한 지침은 다음을 참조하세요.

- DNS 검증
- the section called “사용자 지정 도메인 이름”

기본 문제 해결 명령

- 서비스를 찾을 수 있는지 확인합니다.

```
aws apprunner list-services
```

- 서비스를 설명하고 상태를 확인합니다.

```
aws apprunner describe-service --service-arn
```

- 사용자 지정 도메인의 상태를 확인합니다.

```
aws apprunner describe-custom-domains --service-arn
```

- 진행 중인 모든 작업을 나열합니다.

```
aws apprunner list-operations --service-arn
```

사용자 지정 도메인 인증서 갱신

서비스에 사용자 지정 도메인을 추가하면 App Runner는 DNS 서버에 추가하는 CNAME 레코드 세트를 제공합니다. 이러한 CNAME 레코드에는 인증서 레코드가 포함됩니다. App Runner는 AWS Certificate Manager (ACM)을 사용하여 도메인을 확인합니다. App Runner는 이러한 DNS 레코드를 검증하여이 도메인의 지속적인 소유권을 보장합니다. DNS 영역에서 CNAME 레코드를 제거하면 App Runner가 더 이상 DNS 레코드를 검증할 수 없으며 사용자 지정 도메인 인증서가 자동으로 갱신되지 않습니다.

이 섹션에서는 다음과 같은 사용자 지정 도메인 인증서 갱신 문제를 해결하는 방법을 다룹니다.

- the section called “CNAME이 DNS 서버에서 제거됩니다.”.
- the section called “인증서가 만료되었습니다.”.

CNAME이 DNS 서버에서 제거됩니다.

- DescribeCustomDomains API를 사용하거나 App Runner 콘솔의 사용자 지정 도메인 설정에서 CNAME 레코드를 검색합니다. 저장된 CNAMEs 대한 자세한 내용은 CertificateValidationRecords.
- DNS 서버에 인증서 검증 CNAME 레코드를 추가합니다. 그런 다음 App Runner는 사용자가 도메인을 소유하고 있는지 확인할 수 있습니다. CNAME 레코드를 추가한 후 DNS 레코드가 전파되는 데 최대 30분이 걸릴 수 있습니다. App Runner와 ACM이 인증서 갱신 프로세스를 재시도하는 데 몇 시간이 걸릴 수도 있습니다. CNAME 레코드를 추가하는 방법에 대한 지침은 섹션을 참조하세요the section called “사용자 지정 도메인 관리”.

인증서가 만료되었습니다.

- App Runner 콘솔 또는 API를 사용하여 App Runner 서비스의 사용자 지정 도메인을 연결 해제(링크 해제)한 다음 연결(링크)합니다. App Runner는 새 인증서 검증 CNAME 레코드를 생성합니다.
- DNS 서버에 새 인증서 검증 CNAME 레코드를 추가합니다.

  사용자 지정 도메인의 연결을 해제(링크 해제) 및 연결(링크)하는 방법에 대한 지침은 섹션을 참조하세요the section called “사용자 지정 도메인 관리”.

인증서가 성공적으로 갱신되었는지 확인하려면 어떻게 해야 합니까?

인증서 레코드의 상태를 확인하여 인증서가 성공적으로 갱신되었는지 확인할 수 있습니다. curl과 같은 도구를 사용하여 인증서의 상태를 확인할 수 있습니다.

인증서 갱신에 대한 자세한 내용은 다음 링크를 참조하세요.

- ACM 인증서가 갱신 부적격으로 표시된 이유는 무엇인가요?
- ACM 인증서에 대한 관리형 갱신
- DNS 검증

HTTP/HTTPS 요청 라우팅 오류

이 섹션에서는 HTTP/HTTPS 트래픽을 App Runner 서비스 엔드포인트로 라우팅할 때 발생할 수 있는 오류를 해결하고 해결하는 방법을 다룹니다.

404 App Runner 서비스 엔드포인트로 HTTP/HTTPS 트래픽을 전송할 때 오류를 찾을 수 없음

- App RunnerHost Header가 호스트 헤더 정보를 사용하여 요청을 라우팅하므로가 HTTP 요청의 서비스 URL을 가리키고 있는지 확인합니다. 및 웹 브라우저cURL와 같은 대부분의 클라이언트는 자동으로 호스트 헤더가 서비스 URL을 가리키도록 합니다. 클라이언트가 서비스 URL을 로 설정하지 않으면 404 Not Found 오류가 Host Header발생합니다.

  Example잘못된 호스트 헤더

  ```
  $ ~ curl -I -H "host: foobar.com" https://testservice.awsapprunner.com/
  HTTP/1.1 404 Not Found
  transfer-encoding: chunked
  ```

  Example올바른 호스트 헤더

  ```
  $ ~ curl -I -H "host: testservice.awsapprunner.com" https://
  testservice.awsapprunner.com/
  HTTP/1.1 200 OK
  content-length: 11772
  content-type: text/html; charset=utf-8
  ```

- 클라이언트가 퍼블릭 또는 프라이빗 서비스로 라우팅하는 요청에 대해 서버 이름 표시기(SNI)를 올바르게 설정하고 있는지 확인합니다. TLS 종료 및 요청 라우팅의 경우 App Runner는 HTTPS 연결에서 SNI 세트를 사용합니다.

서비스가 Amazon RDS 또는 다운스트림 서비스에 연결되지 않는 경우

Amazon RDS 데이터베이스 또는 기타 다운스트림 애플리케이션 또는 서비스에 연결하지 못하면 서비스에 네트워크 구성 문제가 있을 수 있습니다. 이 주제에서는 네트워크 구성에 문제가 있는지 확인하는 몇 가지 단계와 이를 수정하는 옵션을 안내합니다. App Runner의 아웃바운드 트래픽 구성에 대한 자세한 내용은 섹션을 참조하세요발신 트래픽에 대한 VPC 액세스 활성화.

> **Note**
>
> VPC 커넥터 구성을 보려면 App Runner 콘솔 왼쪽 탐색 창에서 네트워크 구성을 선택합니다. 그런 다음 발신 트래픽 탭을 선택합니다. VPC 커넥터를 선택합니다. 다음 페이지에는 VPC 커넥터에 대한 세부 정보가 표시됩니다. 이 페이지에서 VPC를 사용하는 서브넷, 보안 그룹 및 App Runner 서비스를 보고 자세히 알아볼 수 있습니다.

애플리케이션이 다른 다운스트림 서비스에 연결할 수 없는 원인을 좁히려면

1. VPC 커넥터에 사용되는 서브넷이 프라이빗 서브넷인지 확인합니다. 커넥터가 퍼블릭 서브넷으로 구성된 경우 각 서브넷의 기본 Hyperplane ENIs(탄력적 네트워크 인터페이스)에 퍼블릭 IP 공간이 없기 때문에 서비스에 오류가 발생합니다.

   VPC 커넥터가 퍼블릭 서브넷을 사용하는 경우 다음 옵션을 사용하여이 구성을 수정할 수 있습니다.

   a. 새 프라이빗 서브넷을 생성하고 VPC 커넥터의 퍼블릭 서브넷 대신 사용합니다. 자세한 내용은 Amazon VPC 사용 설명서의 VPC 서브넷을 참조하세요.
   b. NAT 게이트웨이를 통해 기존 퍼블릭 서브넷을 라우팅합니다. 자세한 내용은 Amazon VPC 사용 설명서의 NAT 게이트웨이를 참조하세요.
2. VPC 커넥터에 대한 보안 그룹 수신 및 송신 규칙이 올바른지 확인합니다. App Runner 콘솔 왼쪽 탐색 창에서 네트워크 구성 > 송신 트래픽을 선택합니다. 목록에서 VPC 커넥터를 선택합니다. 다음 페이지에는 검사할 보안 그룹이 나열됩니다.
3. 연결하려는 RDS 인스턴스 또는 기타 다운스트림 서비스에 대해 보안 그룹 인바운드 및 아웃바운드 규칙이 올바른지 확인합니다. 자세한 내용은 App Runner 애플리케이션이 연결하려는 다운스트림 서비스에 대한 서비스 가이드를 참조하세요.
4. App Runner 구성 외부에 다른 유형의 네트워크 설정 문제가 없는지 확인하려면 App Runner 외부의 RDS 또는 다운스트림 서비스에 연결해 보세요.

a. 동일한 VPC의 Amazon EC2 인스턴스에서 RDS 인스턴스 또는 서비스에 연결해 보세요.

b. 서비스 VPC 엔드포인트에 연결하려는 경우 동일한 VPC의 EC2 인스턴스에서 동일한 엔드포인트에 액세스하여 연결을 확인합니다.

5. 4단계의 연결 테스트 중 하나가 실패하면 App Runner 구성 외부에서 AWS계정의 다른 리소스와 관련된 문제가 있을 가능성이 높습니다. 다른 네트워크 구성과 관련된 문제를 추가로 격리하고 해결하려면 AWS Support에 문의하십시오.

6. 4단계의 지침을 수행하여 RDS 인스턴스 또는 다운스트림 서비스에 성공적으로 연결한 경우이 단계의 지침을 진행합니다. Hyperplane ENI 흐름 로그를 활성화하고 검사하여 트래픽이 ENI로 들어오는지 확인합니다.

> **Note**
>
> 이러한 단계를 완료하고 필요한 ENI 흐름 로그 정보를 얻으려면 App Runner 서비스가 성공적으로 시작된 후 RDS 또는 다운스트림 서비스에 대한 연결 시도를 수행해야 합니다. 애플리케이션이 실행 중 상태일 때 RDS 또는 다운스트림 서비스에 대한 연결 작업을 수행해야 합니다. 그렇지 않으면 App Runner의 롤백 워크플로의 일부로 ENIs를 정리할 수 있습니다. 이 접근 방식을 사용하면 추가 조사를 위해 ENIs를 계속 사용할 수 있습니다.

a. AWS 콘솔에서 EC2 콘솔을 시작합니다.

b. 왼쪽 탐색 창의 네트워크 및 보안 그룹화에서 네트워크 인터페이스를 선택합니다.

c. 인터페이스 유형 및 설명 열로 스크롤하여 VPC 커넥터와 연결된 서브넷에서 ENIs를 찾습니다. 여기에는 다음과 같은 이름 지정 패턴이 있습니다.

- 인터페이스 유형: fargate
- 설명: 로 시작AWSAppRunner ENI(예: AWSAppRunner ENI - abcde123-abcd-1234-1234-abcde1233456)

d. 행 시작 부분에 있는 확인란을 사용하여 해당하는 ENIs 선택합니다.

e. 작업 메뉴에서 흐름 로그 생성을 선택합니다.

f. 프롬프트에 정보를 입력하고 페이지 하단에서 흐름 블로그 생성을 선택합니다.

g. 생성된 흐름 로그를 검사합니다.

- 연결을 테스트할 때 트래픽이 ENI로 들어오는 경우이 문제는 ENI 설정과 관련이 없습니다. App Runner 서비스 외에 AWS 계정의 다른 리소스에 네트워크 구성 문제가 있을 수 있습니다. 추가 지원이 필요한 경우 Support에 문의 AWS 하세요.

- 연결을 테스트할 때 트래픽이 ENI에 들어가지 않은 경우 AWS Support에 문의하여 Fargate 서비스에 알려진 문제가 있는지 확인하는 것이 좋습니다.

h. 네트워크 Reachability Analyzer 도구를 사용합니다. 이 도구는 가상 네트워크 경로의 소스에 연결할 수 없을 때 차단 구성 요소를 식별하여 네트워크 구성 오류를 확인하는 데 도움이 됩니다. 자세한 내용은 Amazon VPC Reachability Analyzer 안내서의 Reachability Analyzer란 무엇입니까?를 참조하세요.

App Runner ENI를 소스로 입력하고 RDS ENI를 대상으로 입력합니다.

7. 문제를 더 좁힐 수 없거나 이전 단계를 완료한 후에도 RDS 또는 다운스트림 서비스에 연결할 수 없는 경우 AWS Support에 문의하여 추가 지원을 받는 것이 좋습니다.

인스턴스를 시작하거나 확장할 IP 주소가 충분하지 않은 경우

> **Note**
>
> 퍼블릭 서비스의 경우 App Runner는 VPCs에 탄력적 네트워크 인터페이스(ENI)를 생성하지 않으므로 퍼블릭 서비스는이 변경의 영향을 받지 않습니다.

이 가이드는 발신 트래픽에 대한 VPC 액세스가 활성화된 App Runner 서비스에서 발생할 수 있는 IP 소진 오류를 해결하는 데 도움이 됩니다.

App Runner는 VPC 커넥터와 연결된 서브넷에서 인스턴스를 시작합니다. App Runner는 인스턴스가 시작되는 서브넷에서 인스턴스당 1개의 ENI를 생성합니다. 각 ENI는 해당 서브넷에서 프라이빗 IP를 사용합니다. 서브넷에는 해당 서브넷과 연결된 CIDR 블록에 따라 사용 가능한 IPs 수가 고정되어 있습니다. App Runner가 ENI를 생성하기에 충분한 IPs 있는 서브넷(들)을 찾을 수 없는 경우 App Runner 서비스에 대한 새 인스턴스를 시작하지 못합니다. 이로 인해 서비스 확장에 문제가 발생할 수 있습니다. 이 경우 App Runner가 사용 가능한 IPs가 있는 서브넷을 찾을 수 없음을 나타내는 App Runner 이벤트 로그가 표시됩니다. 아래 지침에 따라 서비스를 업데이트하여 이러한 오류를 해결할 수 있습니다.

사용 가능한 IPs 더 많도록 서비스를 업데이트하는 방법

서브넷에서 사용 가능한 IP 주소 수는 해당 서브넷과 연결된 CIDR 블록을 기반으로 합니다. 서브넷과 연결된 CIDR 블록은 생성 후 업데이트할 수 없습니다. App Runner VPC 커넥터는 생성된 후에는 업데이트할 수 없습니다. 발신 트래픽에 대한 VPC 액세스가 활성화된 App Runner 서비스에 더 많은 IPs를 제공하려면:

1. 더 큰 CIDR 블록을 사용하여 새 서브넷(들)을 생성합니다.
2. 새 서브넷(들)을 사용하여 새 VPC 커넥터를 생성합니다.
3. 새 VPC 커넥터를 사용하도록 App Runner 서비스를 업데이트합니다.

서비스에 필요한 IPs 계산

더 큰 CIDR 블록으로 새 서브넷(들)을 생성하기 전에 App Runner 서비스 전체에서 필요한 IPs 수를 결정합니다. 다음과 같이 커넥터에 필요한 IPs 계산하는 것이 좋습니다.

1. 발신 트래픽에 대한 VPC 액세스가 활성화된 각 서비스에 대해 Auto Scaling 구성의 최대 크기(최대 인스턴스)를 기록해 둡니다.
2. 모든 서비스에서 값을 합산합니다.
3. 블루-그린 배포 중에 시작된 새 인스턴스를 고려하려면이 합계를 두 배로 늘립니다.

예제

동일한 VPC 커넥터를 사용하여 두 서비스 A와 B를 고려합니다.

1. 서비스 A의 최대 크기는 25로 구성됩니다.
2. 서비스 B의 최대 크기는 15로 구성되어 있습니다.

필수 IPs 2 × (25 + 15) = 80

서브넷에 사용 가능한 IPs가 80개 이상 결합되어 있는지 확인합니다.

새 서브넷(들) 생성

1. 이 공식을 사용하여 IPv4에 필요한 CIDR 블록 크기를 결정합니다(AWS에서 5IPs를 예약함: 서브넷 크기 조정).

```
Number of available IP addresses = 2^(32 - prefix length) - 5
```

```
Example :
For 192.168.1.0/24:
Prefix length is 24
Number of available IP addresses = 2^(32 - 24) - 5 = 2^8-5 = 251 IP addresses
```

```
For 10.0.0.0/16:
Prefix length is 16
Number of available IP addresses = 2^(32 - 16) - 5 = 2^16-5 = 65,531 IP addresses

Quick reference:
/24 = 251 IP addresses
/16 = 65,531 IP addresses
```

2. AWS EC2 CLI를 사용하여 새 서브넷을 생성합니다.

```
aws ec2 create-subnet --vpc-id <my-vpc-id> --cidr-block <cidr-block>
```

예제(IP가 4,096IPs 서브넷 생성):

```
aws ec2 create-subnet --vpc-id my-vpc-id --cidr-block 10.0.0.0/20
```

3. 새 VPC 커넥터를 생성합니다. 참조: VPC 액세스 관리
4. 이 새 VPC 커넥터를 사용하도록 VPC로 송신 트래픽이 활성화된 상태로 서비스를 업데이트합니다. 서비스가 업데이트되면 App Runner가 새 서브넷 사용을 시작합니다.

> **Note**
>
> 또한 VPCs는 CIDR 블록을 통해 서브넷에 할당할 수 있는 사용 가능한 IPs 수로 제한됩니다. 더 큰 CIDR 블록이 있는 서브넷을 생성할 수 없는 경우 새 서브넷(들)을 생성하기 전에 보조 CIDR 블록으로 VPC를 업데이트해야 할 수 있습니다.

VPC에 보조 CIDR 블록 연결

보조 CIDR 블록을이 VPC에 연결합니다.

```
aws ec2 associate-vpc-cidr-block --vpc-id <my-vpc-id> --cidr-block <cidr-block>
```

예제 :

```
aws ec2 associate-vpc-cidr-block --vpc-id my-vpc-id --cidr-block 10.1.0.0/16
```

Verification(확인)

서비스를 업데이트한 후 다음을 사용하여 수정 사항을 확인할 수 있습니다.

1. 이벤트 로그 모니터링: App Runner 서비스 이벤트 로그를 모니터링하여 새 IP 또는 ENI 비가용성 오류가 표시되지 않는지 확인합니다.
2. 서비스 조정 확인:
    1. Auto Scaling 구성에서 최소 인스턴스 수를 변경하여 서비스를 완전히 확장합니다.
    2. 모든 새 인스턴스가 IP 관련 오류 없이 시작되었는지 확인
    3. 여러 조정 이벤트를 모니터링하여 일관된 성능 보장
3. 콘솔 배너: AWS 관리 콘솔을 사용하는 경우 App Runner가 더 이상 IP 부족IPs 확인합니다.
4. VPC 및 서브넷 IP 사용률:
    1. VPC 대시보드 또는 CLI 명령을 사용하여 새 서브넷의 IP 주소 사용률을 확인합니다.
    2. 서비스가 스케일 업된 후에도 여전히 사용 가능한 IPs의 정상 마진이 있는지 확인

일반적인 함정

App Runner 서비스의 IP 소진 문제를 해결할 때는 다음과 같은 잠재적 문제에 유의하세요.

1. 부적절한 IP 주소 계획: 향후 IP 요구 사항을 과소 평가하면 반복적인 소진 문제가 발생할 수 있습니다. 잠재적 서비스 증가 및 최대 사용량 시나리오를 고려하여 철저한 용량 계획을 수행합니다.
2. VPC 전체 IP 사용량 검토: 동일한 VPC 내의 다른 AWS 서비스도 IP 주소를 사용합니다. VPC 및 서브넷 구성을 계획할 때 모든 서비스의 IP 요구 사항을 고려합니다.
3. 서비스 업데이트 무시: 새 서브넷 또는 VPC 커넥터를 생성한 후 새 구성을 사용하도록 App Runner 서비스를 업데이트해야 합니다. 이렇게 하지 않으면 소진된 IP 범위가 계속 사용됩니다.
4. CIDR 블록 오버랩의 오해: VPC에 보조 CIDR 블록을 추가할 때 기존 블록과 겹치지 않도록 합니다. CIDR 블록이 겹치면 라우팅 충돌과 IP 주소 모호성이 발생할 수 있습니다.
5. VPC 제한 초과: VPC는 최대 5개의 CIDR 블록(기본 블록 1개와 보조 블록 4개)을 가질 수 있습니다. 이러한 제약 내에서 IP 주소 공간 확장을 계획합니다.
6. 서브넷 AZ 배포 무시: 새 서브넷을 생성할 때 고가용성 및 내결함성을 위해 여러 가용 영역에 분산되어 있는지 확인합니다.
7. ENI 제한 검토: 인스턴스에 연결할 수 있는 ENIs 수에는 제한이 있습니다. AWS 계정 제한이 계획된 네트워크 인터페이스 사용량과 일치하는지 확인합니다.

이러한 위험을 인식하면 VPC 리소스를 보다 효과적으로 관리하고 App Runner 서비스의 IP 소진 문제를 방지할 수 있습니다.

추가 리소스

1. AWS VPC 설명서
2. CIDR 블록 이해
3. App Runner VPC 커넥터

용어집

1. ENI:Elastic Network Interface, AWS의 가상 네트워크 인터페이스.
2. CIDR:Classless Inter-Domain Routing, IP 주소를 할당하는 방법입니다.
3. VPC 커넥터: App Runner가 VPC에 연결할 수 있도록 하는 리소스입니다.

App Runner의 보안

의 클라우드 보안 AWS 이 최우선 순위입니다. AWS 고객은 보안에 가장 민감한 조직의 요구 사항을 충족하도록 구축된 데이터 센터 및 네트워크 아키텍처의 이점을 누릴 수 있습니다.

보안은 AWS 와 사용자 간의 공동 책임입니다. 공동 책임 모델은 이 사항을 클라우드의 보안 및 클라우드 내 보안으로 설명합니다.

- 클라우드 보안 - AWS 는에서 AWS 서비스를 실행하는 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 프로그램 AWS 제공 범위 내 서비스규정 준수 프로그램 제공 범위 내 서비스를 AWS App Runner참조하세요.
- 클라우드의 보안 - 사용자의 책임은 사용하는 AWS 서비스에 따라 결정됩니다. 또한 귀하는 귀사의 데이터 민감도, 귀사의 요구 사항, 관련 법률 및 규정을 비롯한 기타 요소에 대해서도 책임이 있습니다.

이 설명서는 App Runner를 사용할 때 공동 책임 모델을 적용하는 방법을 이해하는 데 도움이 됩니다. 다음 주제에서는 보안 및 규정 준수 목적에 맞게 App Runner를 구성하는 방법을 보여줍니다. 또한 App Runner 리소스를 모니터링하고 보호하는 데 도움이 되는 다른 AWS 서비스를 사용하는 방법을 알아봅니다.

주제

- App Runner의 데이터 보호
- App Runner의 자격 증명 및 액세스 관리
- App Runner에서 로깅 및 모니터링
- App Runner에 대한 규정 준수 검증
- App Runner의 복원력
- 의 인프라 보안 AWS App Runner
- VPC 엔드포인트와 함께 App Runner 사용
- App Runner의 구성 및 취약성 분석
- App Runner의 보안 모범 사례

App Runner의 데이터 보호

AWS 공동 책임 모델의 데이터 보호에 적용됩니다 AWS App Runner. 이 모델에 설명된 대로 AWS 는 모든를 실행하는 글로벌 인프라를 보호할 책임이 있습니다 AWS 클라우드. 사용자는 이 인프라에 호스팅되는 콘텐츠에 대한 통제 권한을 유지할 책임이 있습니다. 사용하는 AWS 서비스 의 보안 구성과 관리 태스크에 대한 책임도 사용자에게 있습니다. 데이터 프라이버시에 관한 자세한 내용은 데이터 프라이버시 FAQ를 참조하세요. 유럽의 데이터 보호에 대한 자세한 내용은 AWS 보안 블로그의 AWS 공동 책임 모델 및 GDPR 블로그 게시물을 참조하세요.

데이터 보호를 위해 자격 증명을 보호하고 AWS 계정 AWS IAM Identity Center 또는 AWS Identity and Access Management (IAM)를 사용하여 개별 사용자를 설정하는 것이 좋습니다. 이렇게 하면 개별 사용자에게 자신의 직무를 충실히 이행하는 데 필요한 권한만 부여됩니다. 또한 다음과 같은 방법으로 데이터를 보호하는 것이 좋습니다.

- 각 계정에 다중 인증(MFA)을 사용합니다.
- SSL/TLS를 사용하여 AWS 리소스와 통신합니다. TLS 1.2는 필수이며 TLS 1.3을 권장합니다.
- 를 사용하여 API 및 사용자 활동 로깅을 설정합니다 AWS CloudTrail. CloudTrail 추적을 사용하여 AWS 활동을 캡처하는 방법에 대한 자세한 내용은 AWS CloudTrail 사용 설명서의 CloudTrail 추적 작업을 참조하세요.
- 내부의 모든 기본 보안 제어와 함께 AWS 암호화 솔루션을 사용합니다 AWS 서비스.
- Amazon S3에 저장된 민감한 데이터를 검색하고 보호하는 데 도움이 되는 Amazon Macie와 같은 고급 관리형 보안 서비스를 사용합니다.
- 명령줄 인터페이스 또는 API를 AWS 통해에 액세스할 때 FIPS 140-3 검증 암호화 모듈이 필요한 경우 FIPS 엔드포인트를 사용합니다. 사용 가능한 FIPS 엔드포인트에 대한 자세한 내용은 연방 정보 처리 표준(FIPS) 140-3을 참조하세요.

고객의 이메일 주소와 같은 기밀 정보나 중요한 정보는 태그나 이름 필드와 같은 자유 형식 텍스트 필드에 입력하지 않는 것이 좋습니다. 여기에는 App Runner 또는 기타 AWS 서비스 에서 콘솔 AWS CLI, API 또는 AWS SDKs를 사용하여 작업하는 경우가 포함됩니다. 이름에 사용되는 태그 또는 자유 형식 텍스트 필드에 입력하는 모든 데이터는 청구 또는 진단 로그에 사용될 수 있습니다. 외부 서버에 URL을 제공할 때 해당 서버에 대한 요청을 검증하기 위해 자격 증명을 URL에 포함해서는 안 됩니다.

다른 App Runner 보안 주제는 섹션을 참조하세요보안.

주제

- 암호화를 사용하여 데이터 보호

- 인터네트워크 트래픽 개인 정보 보호

암호화를 사용하여 데이터 보호

AWS App Runner 는 지정한 리포지토리에서 애플리케이션 소스(소스 이미지 또는 소스 코드)를 읽고 서비스에 배포하기 위해 저장합니다. 자세한 내용은 아키텍처 및 개념 단원을 참조하십시오.

데이터 보호는 전송 중(App Runner와 주고받는 동안) 및 저장 중(AWS 데이터 센터에 저장되는 동안) 데이터를 보호하는 것을 말합니다.

데이터 보호에 대한 자세한 내용은 the section called “데이터 보호” 섹션을 참조하세요.

다른 App Runner 보안 주제는 섹션을 참조하세요보안.

전송 중 암호화

전송 중 데이터 보호는 전송 계층 보안(TLS)을 사용하여 연결을 암호화하거나 클라이언트 측 암호화(객체가 전송되기 전에 암호화됨)를 사용하는 두 가지 방법으로 수행할 수 있습니다. 두 방법 모두 애플리케이션 데이터를 보호하는 데 유효합니다 연결을 보호하려면 애플리케이션, 개발자 및 관리자, 최종 사용자가 객체를 보내거나 받을 때마다 TLS를 사용하여 암호화합니다. App Runner는 TLS를 통해 트래픽을 수신하도록 애플리케이션을 설정합니다.

클라이언트 측 암호화는 배포를 위해 App Runner에 제공하는 소스 이미지 또는 코드를 보호하는 유효한 방법이 아닙니다. App Runner는 애플리케이션 소스에 대한 액세스 권한이 필요하므로 암호화할 수 없습니다. 따라서 개발 또는 배포 환경과 App Runner 간의 연결을 보호해야 합니다.

저장 시 암호화 및 키 관리

App Runner는 애플리케이션의 저장 데이터를 보호하기 위해 애플리케이션 소스 이미지 또는 소스 번들의 저장된 모든 복사본을 암호화합니다. App Runner 서비스를 생성할 때를 제공할 수 있습니다 AWS KMS key. 제공하면 App Runner는 제공된 키를 사용하여 소스를 암호화합니다. 제공하지 않으면 App Runner는 AWS 관리형 키 대신를 사용합니다.

App Runner 서비스 생성 파라미터에 대한 자세한 내용은 CreateService를 참조하세요. AWS Key Management Service (AWS KMS)에 대한 자세한 내용은 AWS Key Management Service 개발자 안내서를 참조하세요.

인터네트워크 트래픽 개인 정보 보호

App Runner는 Amazon Virtual Private Cloud(VPC)를 사용하여 App Runner 애플리케이션의 리소스 간에 경계를 생성하고 리소스, 온프레미스 네트워크 및 인터넷 간의 트래픽을 제어합니다. Amazon

VPC 보안에 대한 자세한 내용은 Amazon VPC 사용 설명서의 Amazon VPC의 Internetwork 트래픽 개인 정보 보호를 참조하세요.

App Runner 애플리케이션을 사용자 지정 Amazon VPC와 연결하는 방법에 대한 자세한 내용은 섹션을 참조하세요the section called “발신 트래픽”.

VPC 엔드포인트를 사용하여 App Runner에 대한 요청을 보호하는 방법에 대한 자세한 내용은 섹션을 참조하세요the section called “VPC 엔드포인트”.

데이터 보호에 대한 자세한 내용은 the section called “데이터 보호” 섹션을 참조하세요.

다른 App Runner 보안 주제는 섹션을 참조하세요보안.

App Runner의 자격 증명 및 액세스 관리

AWS Identity and Access Management (IAM)는 관리자가 AWS 리소스에 대한 액세스를 안전하게 제어하는 데 도움이 AWS 서비스 되는 입니다. IAM 관리자는 App Runner 리소스를 사용할 수 있는 인증(로그인) 및 권한(권한 있음)을 받을 수 있는 사용자를 제어합니다. IAM은 추가 비용 없이 사용할 수 AWS 서비스 있는 입니다.

다른 App Runner 보안 주제는 섹션을 참조하세요보안.

주제

- 대상
- ID를 통한 인증
- 정책을 사용하여 액세스 관리
- App Runner가 IAM에서 작동하는 방식
- App Runner 자격 증명 기반 정책 예제
- App Runner에 서비스 연결 역할 사용
- AWS 에 대한 관리형 정책 AWS App Runner
- App Runner 자격 증명 및 액세스 문제 해결

대상

AWS Identity and Access Management (IAM)를 사용하는 방법은 역할에 따라 다릅니다.

- 서비스 사용자 - 기능에 액세스할 수 없는 경우 관리자에게 권한 요청(참조App Runner 자격 증명 및 액세스 문제 해결)

- 서비스 관리자 - 사용자 액세스 결정 및 권한 요청 제출(App Runner가 IAM에서 작동하는 방식 참조)
- IAM 관리자 - 액세스를 관리하기 위한 정책 작성(App Runner 자격 증명 기반 정책 예제 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 사용자가 필요한 작업 목록은 IAM 사용자 설명서의 루트 사용자 자격 증명이 필요한 작업을 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.

액세스 제어 목록(ACL)

액세스 제어 목록(ACL)은 어떤 위탁자(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.

Amazon S3 AWS WAF및 Amazon VPC는 ACLs. ACL에 관한 자세한 내용은 Amazon Simple Storage Service 개발자 가이드의 액세스 제어 목록(ACL) 개요를 참조하세요.

기타 정책 타입

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.

App Runner가 IAM에서 작동하는 방식

IAM을 사용하여 액세스를 관리하기 전에 App Runner에서 사용할 수 있는 IAM 기능을 이해해야 AWS App Runner합니다. App Runner 및 기타 AWS 서비스에서 IAM을 사용하는 방법을 전체적으로 알아보려면 IAM 사용 설명서의 AWS IAM으로 작업하는 서비스를 참조하세요.

다른 App Runner 보안 주제는 섹션을 참조하세요보안.

주제

- App Runner 자격 증명 기반 정책
- App Runner 리소스 기반 정책
- App Runner 태그 기반 권한 부여
- App Runner 사용자 권한
- App Runner IAM 역할

App Runner 자격 증명 기반 정책

IAM ID 기반 정책을 사용하면 허용되거나 거부되는 작업과 리소스뿐 아니라 작업이 허용되거나 거부되는 조건을 지정할 수 있습니다. App Runner는 특정 작업, 리소스 및 조건 키를 지원합니다. JSON 정책에서 사용하는 모든 요소에 대해 알고 싶다면 IAM 사용 설명서의 IAM JSON 정책 요소 참조를 참조하세요.

작업

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.

App Runner의 정책 작업은 작업 앞에 접두사를 사용합니다`apprunner:`. 예를 들어 누군가에게 Amazon EC2 `RunInstances` API 작업을 통해 Amazon EC2 인스턴스를 실행할 권한을 부여하려면 해당 정책에 `ec2:RunInstances` 작업을 포함하세요. 정책 문에는 `Action` 또는 `NotAction` 요소가 포함되어야 합니다. App Runner는이 서비스로 수행할 수 있는 작업을 설명하는 고유한 작업 세트를 정의합니다.

명령문 하나에 여러 태스크를 지정하려면 다음과 같이 쉼표로 구분합니다.

```
"Action": [
   "apprunner:CreateService",
   "apprunner:CreateConnection"
]
```

와일드카드(*)를 사용하여 여러 작업을 지정할 수 있습니다. 예를 들어, `Describe`라는 단어로 시작하는 모든 작업을 지정하려면 다음 작업을 포함합니다.

```
"Action": "apprunner:Describe*"
```

App Runner 작업 목록을 보려면 서비스 승인 참조의에서 정의한 작업을 AWS App Runner 참조하세요.

리소스

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": "*"
```

App Runner 리소스의 ARN 구조는 다음과 같습니다.

```
arn:aws:apprunner:region:account-id:resource-type/resource-name[/resource-id]
```

ARN 형식에 대한 자세한 내용은의 Amazon 리소스 이름(ARNs) 및 AWS 서비스 네임스페이스를 참조하세요AWS 일반 참조.

예를 들어 문에서 `my-service` 서비스를 지정하려면 다음 ARN을 사용합니다.

```
"Resource": "arn:aws:apprunner:us-east-1:123456789012:service/my-service"
```

특정 계정에 속하는 모든 서비스를 지정하려면 와일드카드(*)를 사용합니다.

```
"Resource": "arn:aws:apprunner:us-east-1:123456789012:service/*"
```

리소스를 생성하기 위한 작업과 같은 일부 App Runner 작업은 특정 리소스에서 수행할 수 없습니다. 이러한 경우, 와일드카드(*)를 사용해야 합니다.

```
"Resource": "*"
```

App Runner 리소스 유형 및 해당 ARNs 목록을 보려면 서비스 승인 참조의에서 정의한 리소스를 AWS App Runner 참조하세요. 각 리소스의 ARN을 지정할 수 있는 작업을 알아보려면 AWS App Runner가 정의한 작업을 참조하십시오.

조건 키

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.

App Runner는 일부 전역 조건 키 사용을 지원합니다. 모든 AWS 전역 조건 키를 보려면 IAM 사용 설명서의 AWS 전역 조건 컨텍스트 키를 참조하세요.

App Runner는 서비스별 조건 키 세트를 정의합니다. 또한 App Runner는 조건 키를 사용하여 구현되는 태그 기반 액세스 제어를 지원합니다. 자세한 내용은 the section called “App Runner 태그 기반 권한 부여”을 참조하세요.

App Runner 조건 키 목록을 보려면 서비스 승인 참조의에 대한 조건 키를 AWS App Runner 참조하세요. 조건 키를 사용할 수 있는 작업과 리소스를 알아보려면 에서 정의한 작업을 AWS App Runner 참조하세요.

예제

App Runner 자격 증명 기반 정책의 예를 보려면 섹션을 참조하세요App Runner 자격 증명 기반 정책 예제.

App Runner 리소스 기반 정책

App Runner는 리소스 기반 정책을 지원하지 않습니다.

App Runner 태그 기반 권한 부여

App Runner 리소스에 태그를 연결하거나 App Runner에 대한 요청에서 태그를 전달할 수 있습니다. 태그에 근거하여 액세스를 제어하려면 `apprunner:ResourceTag/`*`key-name`*, `aws:RequestTag/`*`key-name`*또는 `aws:TagKeys`조건 키를 사용하여 정책의 조건 요소에 태그 정보를 제공합니다. App Runner 리소스 태그 지정에 대한 자세한 내용은 섹션을 참조하세요the section called “구성”.

리소스의 태그를 기반으로 리소스에 대한 액세스를 제한하는 자격 증명 기반 정책의 예시는 태그를 기반으로 App Runner 서비스에 대한 액세스 제어에서 확인할 수 있습니다.

App Runner 사용자 권한

App Runner를 사용하려면 IAM 사용자에게 App Runner 작업에 대한 권한이 필요합니다. 사용자에게 권한을 부여하는 일반적인 방법은 IAM 사용자 또는 그룹에 정책을 연결하는 것입니다. 사용자 권한 관리에 대한 자세한 내용은 IAM 사용 설명서의 IAM 사용자의 권한 변경을 참조하세요.

App Runner는 사용자에게 연결할 수 있는 두 가지 관리형 정책을 제공합니다.

- `AWSAppRunnerReadOnlyAccess` - App Runner 리소스에 대한 세부 정보를 나열하고 볼 수 있는 권한을 부여합니다.
- `AWSAppRunnerFullAccess`- 모든 App Runner 작업에 대한 권한을 부여합니다.

사용자 권한을 보다 세밀하게 제어하려면 사용자 지정 정책을 생성하여 사용자에게 연결할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 IAM 정책 생성을 참조하세요.

사용자 정책의 예는 섹션을 참조하세요the section called “사용자 정책”.

App Runner IAM 역할

IAM 역할은 특정 권한이 AWS 계정 있는 내의 엔터티입니다.

서비스 연결 역할

서비스 연결 역할을 사용하면 AWS 서비스가 다른 서비스의 리소스에 액세스하여 사용자를 대신하여 작업을 완료할 수 있습니다. 서비스 연결 역할은 IAM 계정에 나타나고 서비스가 소유합니다. IAM 관리자는 서비스 연결 역할의 권한을 볼 수 있지만 편집할 수 없습니다.

App Runner는 서비스 연결 역할을 지원합니다. App Runner 서비스 연결 역할 생성 또는 관리에 대한 자세한 내용은 섹션을 참조하세요the section called “서비스 연결 역할 사용”.

서비스 역할

이 기능을 사용하면 서비스가 사용자를 대신하여 서비스 역할을 수임할 수 있습니다. 이 역할을 사용하면 서비스가 다른 서비스의 리소스에 액세스해 사용자를 대신해 작업을 완료할 수 있습니다. 서비스 역할은 IAM 계정에 나타나고, 해당 계정이 소유합니다. 즉, IAM 사용자는이 역할에 대한 권한을 변경할 수 있습니다. 그러나 권한을 변경하면 서비스의 기능이 손상될 수 있습니다.

App Runner는 몇 가지 서비스 역할을 지원합니다.

액세스 역할

액세스 역할은 App Runner가 계정의 Amazon Elastic Container Registry(Amazon ECR)에 있는 이미지에 액세스하는 데 사용하는 역할입니다. Amazon ECR의 이미지에 액세스하는 데 필요하며 Amazon ECR Public에서는 필요하지 않습니다.

Amazon ECR의 이미지를 기반으로 서비스를 생성하기 전에 IAM을 사용하여 서비스 역할을 생성합니다. 서비스 역할`AWSAppRunnerServicePolicyForECRAccess`에서 관리형 정책을 사용합니다. 그런 다음 SourceConfiguration 파라미터의 AuthenticationConfiguration 멤버에서 CreateService API를 호출하거나 App Runner 콘솔을 사용하여 서비스를 생성할 때이 역할을 App Runner에 전달할 수 있습니다.

> **Note**
>
> 액세스 역할에 대한 사용자 지정 정책을 직접 생성하는 경우 `ecr:GetAuthorizationToken` 작업에 `"Resource": "*"`를 지정해야 합니다. 토큰은 액세스 권한이 있는 모든 Amazon ECR 레지스트리에 액세스하는 데 사용할 수 있습니다.

액세스 역할을 생성할 때 App Runner 서비스 보안 주체를 신뢰할 수 있는 엔터티`build.apprunner.amazonaws.com`로 선언하는 신뢰 정책을 추가해야 합니다.

액세스 역할에 대한 신뢰 정책

JSON

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

App Runner 콘솔을 사용하여 서비스를 생성하는 경우 콘솔은 자동으로 액세스 역할을 생성하고 새 서비스에 대해 선택할 수 있습니다. 콘솔에는 계정의 다른 역할도 나열되며 원하는 경우 다른 역할을 선택할 수 있습니다.

인스턴스 역할

인스턴스 역할은 App Runner가 서비스의 컴퓨팅 인스턴스에 필요한 AWS 서비스 작업에 대한 권한을 제공하는 데 사용하는 선택적 역할입니다. 애플리케이션 코드가 AWS 작업(APIs. 인스턴스 역할에 필요한 권한을 포함하거나 자체 사용자 지정 정책을 생성하여 인스턴스 역할에 사용합니다. 코드에서 사용하는 호출을 예상할 수 있는 방법은 없습니다. 따라서이 목적을 위한 관리형 정책은 제공하지 않습니다.

App Runner 서비스를 생성하기 전에 IAM을 사용하여 필요한 사용자 지정 또는 임베디드 정책으로 서비스 역할을 생성합니다. 그런 다음 InstanceConfiguration 파라미터의 `InstanceRoleArn` 멤버에서 CreateService API를 호출하거나 App Runner 콘솔을 사용하여 서비스를 생성할 때이 역할을 App Runner에 인스턴스 역할로 전달할 수 있습니다.

인스턴스 역할을 생성할 때 App Runner 서비스 보안 주체를 신뢰할 수 있는 엔터티`tasks.apprunner.amazonaws.com`로 선언하는 신뢰 정책을 추가해야 합니다.

인스턴스 역할에 대한 신뢰 정책

JSON

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

App Runner 콘솔을 사용하여 서비스를 생성하는 경우 콘솔은 계정의 역할을 나열하고이 목적으로 생성한 역할을 선택할 수 있습니다.

서비스 생성에 대한 자세한 내용은 섹션을 참조하세요the section called “만들기”.

App Runner 자격 증명 기반 정책 예제

기본적으로 IAM 사용자 및 역할에는 AWS App Runner 리소스를 생성하거나 수정할 수 있는 권한이 없습니다. 또한 AWS Management Console AWS CLI, 또는 AWS API를 사용하여 작업을 수행할 수 없습니다. IAM 관리자는 지정된 리소스에서 특정 API 작업을 수행할 수 있는 권한을 사용자와 역할에게 부여하는 IAM 정책을 생성해야 합니다. 그런 다음 관리자는 해당 권한이 필요한 IAM 사용자 또는 그룹에 이러한 정책을 연결해야 합니다.

이러한 예제 JSON 정책 문서를 사용하여 IAM ID 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서의 JSON 탭에서 정책 생성을 참조하세요.

다른 App Runner 보안 주제는 섹션을 참조하세요보안.

주제

- 정책 모범 사례
- 사용자 정책
- 태그를 기반으로 App Runner 서비스에 대한 액세스 제어

정책 모범 사례

자격 증명 기반 정책에 따라 계정에서 사용자가 App Runner 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 정

책 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.

사용자 정책

App Runner 콘솔에 액세스하려면 IAM 사용자에게 최소 권한 집합이 있어야 합니다. 이러한 권한은에서 App Runner 리소스에 대한 세부 정보를 나열하고 볼 수 있도록 허용해야 합니다 AWS 계정. 최소 필수 권한보다 더 제한적인 자격 증명 기반 정책을 생성하는 경우 콘솔은 해당 정책이 있는 사용자에 대해 의도한 대로 작동하지 않습니다.

App Runner는 사용자에게 연결할 수 있는 두 가지 관리형 정책을 제공합니다.

- `AWSAppRunnerReadOnlyAccess` - App Runner 리소스에 대한 세부 정보를 나열하고 볼 수 있는 권한을 부여합니다.
- `AWSAppRunnerFullAccess` - 모든 App Runner 작업에 대한 권한을 부여합니다.

사용자가 App Runner 콘솔을 사용할 수 있도록 하려면 최소한 `AWSAppRunnerReadOnlyAccess` 관리형 정책을 사용자에게 연결합니다. 대신 `AWSAppRunnerFullAccess` 관리형 정책을 연결하거나 특정 추가 권한을 추가하여 사용자가 리소스를 생성, 수정 및 삭제할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 사용자에게 권한 추가를 참조하십시오.

AWS CLI 또는 AWS API만 호출하는 사용자에게는 최소 콘솔 권한을 허용할 필요가 없습니다. 대신 사용자가 수행하도록 허용하려는 API 작업과 일치하는 작업에만 액세스하도록 허용합니다.

다음 예제에서는 사용자 지정 사용자 정책을 보여줍니다. 이를 사용자 지정 사용자 정책을 정의하기 위한 시작점으로 사용할 수 있습니다. 예제를 복사하거나 작업을 제거하고, 리소스 범위를 좁히고, 조건을 추가합니다.

예: 콘솔 및 연결 관리 사용자 정책

이 예제 정책은 콘솔 액세스를 활성화하고 연결 생성 및 관리를 허용합니다. App Runner 서비스 생성 및 관리는 허용되지 않습니다. 소스 코드 자산에 대한 App Runner 서비스 액세스를 관리하는 역할을 가진 사용자에게 연결할 수 있습니다.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "apprunner:List*",
        "apprunner:Describe*",
        "apprunner:CreateConnection",
        "apprunner:DeleteConnection"
      ],
      "Resource": "*"
    }
  ]
}
```

예: 조건 키를 사용하는 사용자 정책

이 섹션의 예제는 일부 리소스 속성 또는 작업 파라미터에 의존하는 조건부 권한을 보여줍니다.

이 예제 정책은 App Runner 서비스를 생성할 수 있지만 라는 연결을 사용하여 거부합니다`prod`.

JSON

```
{ "Version":"2012-10-17",
"Statement":
   [ { "Sid": "AllowCreateAppRunnerServiceWithNonProdConnections",
       "Effect": "Allow",
       "Action": "apprunner:CreateService",
       "Resource": "*",
       "Condition":
```

```
            { "ArnNotLike":
               {"apprunner:ConnectionArn":"arn:aws:apprunner:*:*:connection/prod/
*"}
            }
       }
     ]
   }
```

이 예제 정책은 라는 Auto Scaling 구성으로preprod만 라는 App Runner 서비스를 업데이트할 수 있도록 합니다preprod.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowUpdatePreProdAppRunnerServiceWithPreProdASConfig",
            "Effect": "Allow",
            "Action": "apprunner:UpdateService",
            "Resource": "arn:aws:apprunner:*:*:service/preprod/*",
            "Condition": {
                "ArnLike": {
                    "apprunner:AutoScalingConfigurationArn":
 "arn:aws:apprunner:us-east-1:*:autoscalingconfiguration/preprod/*"
                }
            }
        }
    ]
}
```

태그를 기반으로 App Runner 서비스에 대한 액세스 제어

자격 증명 기반 정책의 조건을 사용하여 태그를 기반으로 App Runner 리소스에 대한 액세스를 제어할 수 있습니다. 이 예제에서는 App Runner 서비스 삭제를 허용하는 정책을 생성하는 방법을 보여줍니다. 하지만 Owner 서비스 태그가 해당 사용자의 사용자 이름 값을 가지고 있는 경우에만 권한이 부여됩니다. 이 정책은 콘솔에서 이 태스크를 완료하는 데 필요한 권한도 부여합니다.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "ListServicesInConsole",
      "Effect": "Allow",
      "Action": "apprunner:ListServices",
      "Resource": "*"
    },
    {
      "Sid": "DeleteServiceIfOwner",
      "Effect": "Allow",
      "Action": "apprunner:DeleteService",
      "Resource": "arn:aws:apprunner:us-east-1:*:service/*",
      "Condition": {
        "StringEquals": {"aws:ResourceTag/Owner": "${aws:username}"}
      }
    }
  ]
}
```

이 정책을 계정의 IAM 사용자에게 연결할 수 있습니다. 라는 사용자가 App Runner 서비스를 삭제하려고 `richard-roe` 하면 서비스에 `Owner=richard-roe` 또는 태그를 지정해야 합니다`owner=richard-roe`. 그렇지 않으면 액세스가 거부됩니다. 조건 키 이름은 대소문자를 구분하지 않기 때문에 조건 태그 키 `Owner`는 `Owner` 및 `owner` 모두와 일치합니다. 자세한 정보는 IAM 사용 설명서의 IAM JSON 정책 요소: 조건을 참조하세요.

App Runner에 서비스 연결 역할 사용

AWS App Runner 는 AWS Identity and Access Management (IAM) 서비스 연결 역할을 사용합니다. 서비스 연결 역할은 App Runner에 직접 연결된 고유한 유형의 IAM 역할입니다. 서비스 연결 역할은 App Runner에서 사전 정의하며 서비스가 사용자를 대신하여 다른 AWS 서비스를 호출하는 데 필요한 모든 권한을 포함합니다.

주제

- 관리를 위한 역할 사용
- 네트워킹에 역할 사용

관리를 위한 역할 사용

AWS App Runner 는 AWS Identity and Access Management (IAM) 서비스 연결 역할을 사용합니다. 서비스 연결 역할은 App Runner에 직접 연결된 고유한 유형의 IAM 역할입니다. 서비스 연결 역할은 App Runner에서 사전 정의하며 서비스가 사용자를 대신하여 다른 AWS 서비스를 호출하는 데 필요한 모든 권한을 포함합니다.

서비스 연결 역할을 사용하면 필요한 권한을 수동으로 추가할 필요가 없으므로 App Runner를 더 쉽게 설정할 수 있습니다. App Runner는 서비스 연결 역할의 권한을 정의하며, 달리 정의되지 않은 한 App Runner만 해당 역할을 수임할 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 리소스에 대한 액세스 권한을 실수로 제거할 수 없기 때문에 App Runner 리소스가 보호됩니다.

서비스 연결 역할을 지원하는 기타 서비스에 대한 자세한 내용은 IAM으로 작업하는AWS 서비스를 참조해 서비스 연결 역할 열이 예(Yes)인 서비스를 찾으세요. 해당 서비스에 대한 서비스 연결 역할 설명서를 보려면 예(Yes) 링크를 선택합니다.

App Runner에 대한 서비스 연결 역할 권한

App Runner는 AWSServiceRoleForAppRunner라는 서비스 연결 역할을 사용합니다.

역할을 통해 App Runner는 다음 작업을 수행할 수 있습니다.

- Amazon CloudWatch Logs에 로그 그룹에 로그를 푸시합니다.
- Amazon CloudWatch Events 규칙을 생성하여 Amazon Elastic Container Registry(Amazon ECR) 이미지 푸시를 구독합니다.
- 추적 정보를 로 전송합니다 AWS X-Ray.

AWSServiceRoleForAppRunner 서비스 연결 역할은 역할을 수임하기 위해 다음 서비스를 신뢰합니다.

- `apprunner.amazonaws.com`

AWSServiceRoleForAppRunner 서비스 연결 역할의 권한 정책에는 App Runner가 사용자를 대신하여 작업을 완료하는 데 필요한 모든 권한이 포함되어 있습니다.

- 관리형 정책 `AppRunnerServiceRolePolicy`
- X-Ray 추적 정책 - 다음 정책 콘텐츠를 참조하세요.

X-Ray 추적 정책

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "xray:PutTraceSegments",
        "xray:PutTelemetryRecords",
        "xray:GetSamplingRules",
        "xray:GetSamplingTargets",
        "xray:GetSamplingStatisticSummaries"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

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 권한 섹션을 참조하세요.

App Runner에 대한 서비스 연결 역할 생성

서비스 연결 역할은 수동으로 생성할 필요가 없습니다. AWS Management Console AWS CLI, 또는 AWS API에서 App Runner 서비스를 생성하면 App Runner가 서비스 연결 역할을 생성합니다.

이 서비스 연결 역할을 삭제했다가 다시 생성해야 하는 경우 동일한 프로세스를 사용하여 계정에서 역할을 다시 생성할 수 있습니다. App Runner 서비스를 생성하면 App Runner가 서비스 연결 역할을 다시 생성합니다.

App Runner에 대한 서비스 연결 역할 편집

App Runner는 AWSServiceRoleForAppRunner 서비스 연결 역할을 편집하도록 허용하지 않습니다. 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.

App Runner에 대한 서비스 연결 역할 삭제

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. 단, 서비스 연결 역할을 정리해야 수동으로 삭제할 수 있습니다.

서비스 연결 역할을 정리

IAM을 사용하여 서비스 연결 역할을 삭제하기 전에 먼저 역할에서 사용되는 리소스를 삭제해야 합니다.

App Runner에서 이는 계정의 모든 App Runner 서비스를 삭제하는 것을 의미합니다. App Runner 서비스 삭제에 대한 자세한 내용은 섹션을 참조하세요the section called “삭제”.

> **Note**
>
> 리소스를 삭제하려고 할 때 App Runner 서비스가 역할을 사용하는 경우 삭제에 실패할 수 있습니다. 이 문제가 발생하면 몇 분 기다렸다가 작업을 다시 시도하세요.

수동으로 서비스 연결 역할 삭제

IAM 콘솔 AWS CLI, 또는 AWS API를 사용하여 AWSServiceRoleForAppRunner 서비스 연결 역할을 삭제합니다. 자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 삭제를 참조하세요.

App Runner 서비스 연결 역할에 지원되는 리전

App Runner는 서비스를 사용할 수 있는 모든 리전에서 서비스 연결 역할 사용을 지원합니다. 자세한 내용은AWS 일반 참조에서 AWS App Runner 엔드포인트 및 할당량을 참조하세요.

네트워킹에 역할 사용

AWS App Runner 는 AWS Identity and Access Management (IAM) 서비스 연결 역할을 사용합니다. 서비스 연결 역할은 App Runner에 직접 연결된 고유한 유형의 IAM 역할입니다. 서비스 연결 역할은 App Runner에서 사전 정의하며 서비스가 사용자를 대신하여 다른 AWS 서비스를 호출하는 데 필요한 모든 권한을 포함합니다.

서비스 연결 역할을 사용하면 필요한 권한을 수동으로 추가할 필요가 없으므로 App Runner를 더 쉽게 설정할 수 있습니다. App Runner는 서비스 연결 역할의 권한을 정의하며, 달리 정의되지 않은 한 App Runner만 해당 역할을 수임할 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 리소스에 대한 액세스 권한을 실수로 제거할 수 없기 때문에 App Runner 리소스가 보호됩니다.

서비스 연결 역할을 지원하는 기타 서비스에 대한 자세한 내용은 IAM으로 작업하는AWS 서비스를 참조해 서비스 연결 역할 열이 예(Yes)인 서비스를 찾으세요. 해당 서비스에 대한 서비스 연결 역할 설명서를 보려면 예(Yes) 링크를 선택합니다.

App Runner에 대한 서비스 연결 역할 권한

App Runner는 AWSServiceRoleForAppRunnerNetworking이라는 서비스 연결 역할을 사용합니다.

역할을 통해 App Runner는 다음 작업을 수행할 수 있습니다.

- VPC를 App Runner 서비스에 연결하고 네트워크 인터페이스를 관리합니다.

AWSServiceRoleForAppRunnerNetworking 서비스 연결 역할은 역할을 수임하기 위해 다음 서비스를 신뢰합니다.

- `networking.apprunner.amazonaws.com`

라는 역할 권한 정책에는 App Runner가 사용자를 대신하여 작업을 완료하는 데 필요한 모든 권한이 `AppRunnerNetworkingServiceRolePolicy` 포함되어 있습니다.

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 권한 섹션을 참조하세요.

App Runner에 대한 서비스 연결 역할 생성

서비스 연결 역할은 수동으로 생성할 필요가 없습니다. AWS Management Console AWS CLI, 또는 AWS API에서 VPC 커넥터를 생성하면 App Runner가 서비스 연결 역할을 생성합니다.

이 서비스 연결 역할을 삭제했다가 다시 생성해야 하는 경우 동일한 프로세스를 사용하여 계정에서 역할을 다시 생성할 수 있습니다. VPC 커넥터를 생성하면 App Runner가 서비스 연결 역할을 다시 생성합니다.

App Runner에 대한 서비스 연결 역할 편집

App Runner는 AWSServiceRoleForAppRunnerNetworking 서비스 연결 역할을 편집하도록 허용하지 않습니다. 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.

App Runner에 대한 서비스 연결 역할 삭제

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. 단, 서비스 연결 역할을 정리해야 수동으로 삭제할 수 있습니다.

서비스 연결 역할 정리

IAM을 사용하여 서비스 연결 역할을 삭제하기 전에 먼저 역할에서 사용되는 리소스를 삭제해야 합니다.

App Runner에서 이는 계정의 모든 App Runner 서비스에서 VPC 커넥터를 연결 해제하고 VPC 커넥터를 삭제하는 것을 의미합니다. 자세한 내용은 the section called “발신 트래픽” 단원을 참조하십시오.

> **Note**
>
> 리소스를 삭제하려고 할 때 App Runner 서비스가 역할을 사용하는 경우 삭제에 실패할 수 있습니다. 이 문제가 발생하면 몇 분 기다렸다가 작업을 다시 시도하세요.

수동으로 서비스 연결 역할 삭제

IAM 콘솔 AWS CLI, 또는 AWS API를 사용하여 AWSServiceRoleForAppRunnerNetworking 서비스 연결 역할을 삭제합니다. 자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 삭제를 참조하세요.

App Runner 서비스 연결 역할에 지원되는 리전

App Runner는 서비스를 사용할 수 있는 모든 리전에서 서비스 연결 역할 사용을 지원합니다. 자세한 내용은AWS 일반 참조에서 AWS App Runner 엔드포인트 및 할당량을 참조하세요.

AWS 에 대한 관리형 정책 AWS App Runner

AWS 관리형 정책은에서 생성하고 관리하는 독립 실행형 정책입니다 AWS. AWS 관리형 정책은 사용자, 그룹 및 역할에 권한 할당을 시작할 수 있도록 많은 일반적인 사용 사례에 대한 권한을 제공하도록 설계되었습니다.

AWS 관리형 정책은 모든 AWS 고객이 사용할 수 있으므로 특정 사용 사례에 대해 최소 권한 권한을 부여하지 않을 수 있습니다. 사용 사례에 고유한 고객 관리형 정책을 정의하여 권한을 줄이는 것이 좋습니다.

AWS 관리형 정책에 정의된 권한은 변경할 수 없습니다. 가 AWS 관리형 정책에 정의된 권한을 AWS 업데이트하면 정책이 연결된 모든 보안 주체 자격 증명(사용자, 그룹 및 역할)에 영향을 줍니다. AWS 는 새 AWS 서비스 가 시작되거나 기존 서비스에 새 API 작업을 사용할 수 있게 되면 AWS 관리형 정책을 업데이트할 가능성이 높습니다.

자세한 내용은 IAM 사용 설명서의 AWS 관리형 정책을 참조하세요.

AWS 관리형 정책에 대한 App Runner 업데이트

이 서비스가 이러한 변경 사항을 추적하기 시작한 이후 App Runner의 AWS 관리형 정책 업데이트에 대한 세부 정보를 봅니다. 이 페이지의 변경 사항에 대한 자동 알림을 받으려면 App Runner 문서 기록 페이지에서 RSS 피드를 구독하세요.

변경 사항	설명	날짜
AWSAppRunnerReadOnlyAccess – 새 정책	App Runner는 사용자가 App Runner 리소스에 대한 세부 정보를 나열하고 볼 수 있도록 허용하는 새 정책을 추가했습니다.	2022년 2월 24일
AWSAppRunnerFullAccess - 기존 정책에 대한 업데이트	App Runner는 `AWSServiceRoleForAppRunnerNetworking` 서비스 연결 역할을 생성할 수 있도록 `iam:CreateServiceLinkedRole` 작업에 대한 리소스 목록을 업데이트했습니다.	2022년 2월 8일
AppRunnerNetworkingServiceRolePolicy – 새 정책	App Runner는 App Runner가 Amazon Virtual Private Cloud를 호출하여 App Runner 서비스에 VPC를 연결하고 App Runner 서비스를 대신하여 네트워크 인터페이스를 관리할 수 있도록 허용하는 새 정책을 추가했습니다. 정책은 `AWSServiceRoleForAppRunnerNetworking` 서비스 연결 역할에 사용됩니다.	2022년 2월 8일

변경 사항	설명	날짜
AWSAppRunnerFullAccess – 새 정책	App Runner는 사용자가 모든 App Runner 작업을 수행할 수 있도록 허용하는 새 정책을 추가했습니다.	2022년 1월 10일
AppRunnerServiceRolePolicy – 새 정책	App Runner는 App Runner가 App Runner 서비스를 대신하여 Amazon CloudWatch Logs 및 Amazon CloudWatch Events를 호출할 수 있도록 허용하는 새 정책을 추가했습니다. 정책은 `AWSServiceRoleForAppRunner` 서비스 연결 역할에 사용됩니다.	2021년 3월 1일
AWSAppRunnerServicePolicyForECRAccess – 새 정책	App Runner는 App Runner가 계정의 Amazon Elastic Container Registry(Amazon ECR) 이미지에 액세스할 수 있도록 허용하는 새 정책을 추가했습니다.	2021년 3월 1일
App Runner에서 변경 사항 추적 시작	App Runner는 AWS 관리형 정책에 대한 변경 사항 추적을 시작했습니다.	2021년 3월 1일

App Runner 자격 증명 및 액세스 문제 해결

다음 정보를 사용하여 및 IAM으로 작업할 때 발생할 수 있는 일반적인 문제를 진단 AWS App Runner 하고 수정할 수 있습니다.

다른 App Runner 보안 주제는 섹션을 참조하세요보안.

주제

- App Runner에서 작업을 수행할 권한이 없음
- 내 외부의 사람이 내 App Runner 리소스 AWS 계정 에 액세스하도록 허용하고 싶습니다.

App Runner에서 작업을 수행할 권한이 없음

에서 작업을 수행할 권한이 없다는 AWS Management Console 메시지가 표시되면 관리자에게 문의하여 지원을 받으세요. 관리자는 AWS 로그인 자격 증명을 제공한 사람입니다.

다음 예제 오류는 라는 IAM 사용자가 콘솔을 사용하여 App Runner 서비스에 대한 세부 정보를 보marymajor려고 하지만 apprunner:DescribeService 권한이 없는 경우에 발생합니다.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 apprunner:DescribeService on resource: my-example-service
```

이 경우 Mary는 apprunner:DescribeService 작업을 사용하여 *my-example-service* 리소스에 액세스할 수 있도록 정책을 업데이트하도록 관리자에게 요청합니다.

내 외부의 사람이 내 App Runner 리소스 AWS 계정 에 액세스하도록 허용하고 싶습니다.

다른 계정의 사용자 또는 조직 외부의 사람이 리소스에 액세스할 때 사용할 수 있는 역할을 생성할 수 있습니다. 역할을 수임할 신뢰할 수 있는 사람을 지정할 수 있습니다. 리소스 기반 정책 또는 액세스 제어 목록(ACL)을 지원하는 서비스의 경우, 이러한 정책을 사용하여 다른 사람에게 리소스에 대한 액세스 권한을 부여할 수 있습니다.

자세한 내용은 다음을 참조하세요.

- App Runner가 이러한 기능을 지원하는지 여부를 알아보려면 섹션을 참조하세요App Runner가 IAM에서 작동하는 방식.
- 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.

App Runner에서 로깅 및 모니터링

모니터링은 AWS App Runner 서비스의 안정성, 가용성 및 성능을 유지하는 데 중요한 부분입니다. AWS 솔루션의 모든 부분에서 모니터링 데이터를 수집하면 장애가 발생할 경우 더 쉽게 디버깅할 수 있습니다. App Runner는 App Runner 서비스를 모니터링하고 잠재적 인시던트에 대응하기 위한 여러 AWS 도구와 통합됩니다.

Amazon CloudWatch 경보

Amazon CloudWatch 경보를 사용하면 지정한 기간 동안 서비스 지표를 볼 수 있습니다. 지표가 지정된 기간 수 동안 지정된 임계값을 초과하면 알림을 받게 됩니다.

App Runner는 서비스 전체와 웹 서비스를 실행하는 인스턴스(스케일링 단위)에 대한 다양한 지표를 수집합니다. 자세한 내용은 지표(CloudWatch) 단원을 참조하십시오.

애플리케이션 로그

App Runner는 애플리케이션 코드의 출력을 수집하여 Amazon CloudWatch Logs로 스트리밍합니다. 이 출력의 내용은 사용자에게 달려 있습니다. 예를 들어 웹 서비스에 대한 요청의 세부 레코드를 포함할 수 있습니다. 이러한 로그 레코드는 보안 및 액세스 감사에 유용할 수 있습니다. 자세한 내용은 로그(CloudWatch Logs) 단원을 참조하십시오.

AWS CloudTrail 작업 로그

App Runner는 App Runner에서 사용자 AWS CloudTrail, 역할 또는 서비스가 수행한 작업에 대한 레코드를 제공하는 AWS 서비스와 통합됩니다. CloudTrail은 App Runner에 대한 모든 API 호출을 이벤트로 캡처합니다. CloudTrail 콘솔에서 최신 이벤트를 볼 수 있으며, 추적을 생성하여 CloudTrail 이벤트를 Amazon Simple Storage Service(Amazon S3) 버킷에 지속적으로 전달할 수 있습니다. 자세한 내용은 API 작업(CloudTrail) 단원을 참조하십시오.

App Runner에 대한 규정 준수 검증

타사 감사자는 여러 규정 준수 프로그램의 AWS App Runner 일환으로의 보안 및 AWS 규정 준수를 평가합니다. 여기에는 SOC, PCI, FedRAMP, HIPAA 등이 포함됩니다.

AWS 서비스 가 특정 규정 준수 프로그램의 범위 내에 있는지 알아보려면 AWS 서비스 규정 준수 프로그램 범위 내를 참조하고 관심 있는 규정 준수 프로그램을 선택합니다. 일반 정보는 AWS 규정 준수 프로그램.

를 사용하여 타사 감사 보고서를 다운로드할 수 있습니다 AWS Artifact. 자세한 내용은 Downloading Reports inDownloading AWS Artifact을 참조하세요.

사용 시 규정 준수 책임은 데이터의 민감도, 회사의 규정 준수 목표 및 관련 법률과 규정에 따라 AWS 서비스 결정됩니다. 사용 시 규정 준수 책임에 대한 자세한 내용은 AWS 보안 설명서를 AWS 서비스참조하세요.

다른 App Runner 보안 주제는 섹션을 참조하세요보안.

App Runner의 복원력

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

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

AWS App Runner 는 사용자를 대신하여 AWS 글로벌 인프라 사용을 관리하고 자동화합니다. App Runner를 사용하면에서 AWS 제공하는 가용성 및 내결함성 메커니즘의 이점을 누릴 수 있습니다.

다른 App Runner 보안 주제는 섹션을 참조하세요보안.

의 인프라 보안 AWS App Runner

관리형 서비스인는 Amazon Web Services: 보안 프로세스 개요 백서에 설명된 AWS 글로벌 네트워크 보안 절차로 AWS App Runner 보호됩니다.

AWS 게시된 API 호출을 사용하여 네트워크를 통해 App Runner를 관리하고 운영합니다. App Runner APIs를 호출하는 클라이언트는 TLS(전송 계층 보안) 1.2 이상을 지원해야 합니다. 클라이언트는 Ephemeral Diffie-Hellman(DHE) 또는 Elliptic Curve Ephemeral Diffie-Hellman(ECDHE)과 같은 Perfect Forward Secrecy(PFS)가 포함된 암호 제품군도 지원해야 합니다. Java 7 이상의 최신 시스템은 대부분 이러한 모드를 지원합니다. 이러한 요구 사항은 App Runner 애플리케이션의 엔드포인트에는 적용되지 않습니다.

또한 요청은 액세스 키 ID 및 IAM 위탁자와 관련된 시크릿 액세스 키를 사용하여 서명해야 합니다. 또는 AWS Security Token Service(AWS STS)를 사용하여 임시 보안 자격 증명을 생성하여 요청에 서명할 수 있습니다.

다른 App Runner 보안 주제는 섹션을 참조하세요보안.

VPC 엔드포인트와 함께 App Runner 사용

AWS 애플리케이션은 Amazon Virtual Private Cloud(VPC)의 VPC에서 AWS 서비스 실행되는 다른와 AWS App Runner 서비스를 통합할 수 있습니다. 애플리케이션의 일부는 VPC 내에서 App Runner에 요청할 수 있습니다. 예를 들어 AWS CodePipeline 를 사용하여 App Runner 서비스에 지속적으로 배포할 수 있습니다. 애플리케이션의 보안을 개선하는 한 가지 방법은 VPC 엔드포인트를 통해 이러한 App Runner 요청(및 다른에 대한 요청 AWS 서비스)을 보내는 것입니다.

VPC 엔드포인트를 사용하면 로 구동되는 지원되는 AWS 서비스 및 VPC 엔드포인트 서비스에 VPC를 비공개로 연결할 수 있습니다 AWS PrivateLink. 인터넷 게이트웨이, NAT 디바이스, VPN 연결 또는 Direct Connect 연결이 필요하지 않습니다.

VPC의 리소스는 퍼블릭 IP 주소를 사용하여 App Runner 리소스와 상호 작용하지 않습니다. VPC와 App Runner 간의 트래픽은 Amazon 네트워크를 벗어나지 않습니다. VPC 엔드포인트에 대한 자세한 내용은 AWS PrivateLink 가이드의 VPC 엔드포인트를 참조하세요.

> **Note**
>
> 기본적으로 App Runner 서비스의 웹 애플리케이션은 App Runner가 제공하고 구성하는 VPC에서 실행됩니다. 이 VPC는 퍼블릭입니다. 인터넷에 연결되어 있음을 의미합니다. 선택적으로 애플리케이션을 사용자 지정 VPC와 연결할 수 있습니다. 자세한 내용은 the section called “발신 트래픽” 단원을 참조하십시오.
> 서비스가 VPC에 연결되어 있더라도 APIs 포함하여 AWS 인터넷에 액세스하도록 서비스를 구성할 수 있습니다. VPC 아웃바운드 트래픽에 대한 퍼블릭 인터넷 액세스를 활성화하는 방법에 대한 지침은 섹션을 참조하세요the section called “서브넷 선택 시 고려 사항”.
> App Runner는 애플리케이션에 대한 VPC 엔드포인트 생성을 지원하지 않습니다.

App Runner용 VPC 엔드포인트 설정

VPC에서 App Runner 서비스에 대한 인터페이스 VPC 엔드포인트를 생성하려면 AWS PrivateLink 가이드의 인터페이스 엔드포인트 생성 절차를 따릅니다. 서비스 이름에서 `com.amazonaws.region.apprunner`를 선택합니다.

VPC 네트워크 개인 정보 보호 고려 사항

> **Important**
>
> App Runner에 VPC 엔드포인트를 사용한다고 해서 VPC의 모든 트래픽이 인터넷에서 벗어나는 것은 아닙니다. VPC는 퍼블릭일 수 있습니다. 또한 솔루션의 일부 부분에서는 VPC 엔드포인트를 사용하여 AWS API를 호출하지 않을 수 있습니다. 예를 들어 AWS 서비스 는 퍼블릭 엔드포인트를 사용하여 다른 서비스를 호출할 수 있습니다. VPC의 솔루션에 트래픽 개인 정보가 필요한 경우이 섹션을 참조하세요.

VPC에서 네트워크 트래픽의 프라이버시를 보장하려면 다음을 고려하세요.

- DNS 이름 활성화 - 애플리케이션의 일부는 여전히 `apprunner.region.amazonaws.com` 퍼블릭 엔드포인트를 사용하여 인터넷을 통해 App Runner에 요청을 보낼 수 있습니다. VPC가 인터넷 액세스로 구성된 경우 이러한 요청은 아무런 표시 없이 성공합니다. 엔드포인트를 생성할 때 DNS 이름 활성화가 활성화되어 있는지 확인하여 이를 방지할 수 있습니다. 기본적으로 true로 설정됩니다. 그러면 퍼블릭 서비스 엔드포인트를 인터페이스 VPC 엔드포인트에 매핑하는 DNS 항목이 VPC에 추가됩니다.
- 추가 서비스를 위한 VPC 엔드포인트 구성 - 솔루션이 다른에 요청을 보낼 수 있습니다 AWS 서비스. 예를 들어 AWS CodePipeline 는 요청을 보낼 수 있습니다 AWS CodeBuild. 이러한 서비스에 대한 VPC 엔드포인트를 구성하고 이러한 엔드포인트에서 DNS 이름을 활성화합니다.
- 프라이빗 VPC 구성 - 가능하면(솔루션에 인터넷 액세스가 전혀 필요하지 않은 경우) VPC를 프라이빗으로 설정합니다. 즉, 인터넷에 연결되어 있지 않습니다. 이렇게 하면 누락된 VPC 엔드포인트에서 오류가 표시되므로 누락된 엔드포인트를 추가할 수 있습니다.

엔드포인트 정책을 사용하여 VPC 엔드포인트로 액세스 제어

VPC 엔드포인트 정책은 App Runner에서 지원됩니다. 기본적으로 인터페이스 엔드포인트를 통해 App Runner에 대한 전체 액세스가 허용됩니다. VPC 엔드포인트 정책을 사용하여 App Runner 엔드포인트에 액세스할 수 있는 AWS 보안 주체를 제어할 수 있습니다. 또는 보안 그룹을 엔드포인트 네트워크 인터페이스와 연결하여 인터페이스 엔드포인트를 통해 App Runner에 대한 트래픽을 제어할 수 있습니다.

인터페이스 엔드포인트와 통합

App Runner는 App Runner에 대한 프라이빗 연결을 AWS PrivateLink제공하고 인터넷에 대한 트래픽 노출을 제거하는를 지원합니다. 애플리케이션이를 사용하여 App Runner에 요청을 보내도록 하려면 인터페이스 엔드포인트라고 하는 VPC 엔드포인트 유형을 AWS PrivateLink구성합니다. 자세한 내용은 AWS PrivateLink 설명서의 인터페이스 VPC 엔드포인트(AWS PrivateLink)를 참조하십시오.

App Runner의 구성 및 취약성 분석

AWS 및 고객은 높은 수준의 소프트웨어 구성 요소 보안 및 규정 준수를 달성할 책임이 있습니다. 자세한 내용은 AWS 공동 책임 모델을 참조하세요.

패치 컨테이너 이미지

컨테이너 이미지 패치는 공유 보안 모델에서 고객의 책임입니다. 이미지 소유자는 컨테이너 이미지를 업데이트하고 정기적으로 패치를 적용할 책임이 있습니다. 컨테이너 이미지에 업데이트를 확인하고

적용하기 위한 일상적인 일정을 설정하는 것이 좋습니다. 이미지에서 취약성을 스캔하는 방법에 대한 자세한 내용은 AWS App Runner 설명서를 참조하세요.

다른 App Runner 보안 주제는 섹션을 참조하세요보안.

App Runner의 보안 모범 사례

AWS App Runner 는 자체 보안 정책을 개발하고 구현할 때 고려해야 할 몇 가지 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 주십시오.

다른 App Runner 보안 주제는 섹션을 참조하세요보안.

예방 보안 모범 사례

예방적 보안 통제는 사고가 발생하기 전에 사고를 예방하려고 시도합니다.

최소 권한 액세스 구현

App Runner는 IAM 사용자 및 액세스 역할에 대한 AWS Identity and Access Management (IAM) 관리형 정책을 제공합니다. 이러한 관리형 정책은 App Runner 서비스의 올바른 작동에 필요할 수 있는 모든 권한을 지정합니다.

애플리케이션에 우리의 관리형 정책의 모든 권한이 필요한 것은 아닙니다. 사용자 지정하고 사용자 및 App Runner 서비스가 작업을 수행하는 데 필요한 권한만 부여할 수 있습니다. 이는 다른 사용자 역할이 다른 권한 요구를 가질 수 있는 사용자 정책과 특히 관련이 있습니다. 최소 권한 액세스를 구현하는 것이 오류 또는 악의적인 의도로 인해 발생할 수 있는 보안 위험과 영향을 최소화할 수 있는 근본적인 방법입니다.

이미지를 스캔하여 취약성 확인

Amazon ECR의 APIs 사용하여 컨테이너 이미지의 소프트웨어 취약성을 식별할 수 있습니다. 자세한 내용은 Amazon ECR 설명서를 참조하십시오.

탐지 보안 모범 사례

탐지 보안 통제는 보안 위반이 발생한 후 이를 식별합니다. 잠재적인 보안 위협이나 사고를 탐지하는 데 도움이 됩니다.

모니터링 구현

모니터링은 App Runner 솔루션의 안정성, 보안, 가용성 및 성능을 유지하는 데 중요한 부분입니다.는 서비스를 모니터링하는 데 도움이 되는 여러 도구와 AWS 서비스를 AWS 제공합니다.

다음은 모니터링 대상 항목의 예입니다:

- App Runner에 대한 Amazon CloudWatch 지표 - 주요 App Runner 지표 및 애플리케이션의 사용자 지정 지표에 대한 경보를 설정합니다. 자세한 내용은 지표(CloudWatch)을 참조하세요.
- AWS CloudTrail 항목 - `PauseService` 또는와 같이 가용성에 영향을 미칠 수 있는 작업을 추적합니다`DeleteConnection`. 자세한 내용은 API 작업(CloudTrail) 섹션을 참조하십시오.

AWS 용어집

최신 AWS 용어는 AWS 용어집 참조의 AWS 용어집을 참조하세요.