



INFINITE

AWS DevOps 환경에서의 CI/CD Security Best..? Practice

Team. INFINITE

멘토 | 권현준 멘토님

팀원 | 김수민 김수민 남지우 천예슬

Cotents

DevOps 정의

- DevOps 개념 및 목적 정의
- DevOps Lifecycle

CICD Security Best Practice

- CICD Security Best Practice
- 아키텍처

CICD 무중단 배포

- CICD BlueGreen 배포 구현
- Golden AMI

SAST

- 도구 선정 비교 분석
- SonarQube VS. CodeQL

SCA

- 도구 선정 비교 분석
- Synk VS. Dependency-Track

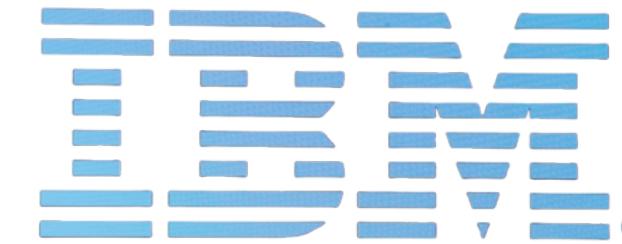
DAST

- 도구 선정 비교 분석
- OWASP ZAP VS. Arachni

| DevOps 정의

: DevOps 개념 및 목적 정의

◆ DevOps 개념 및 목적 정의



Google Cloud

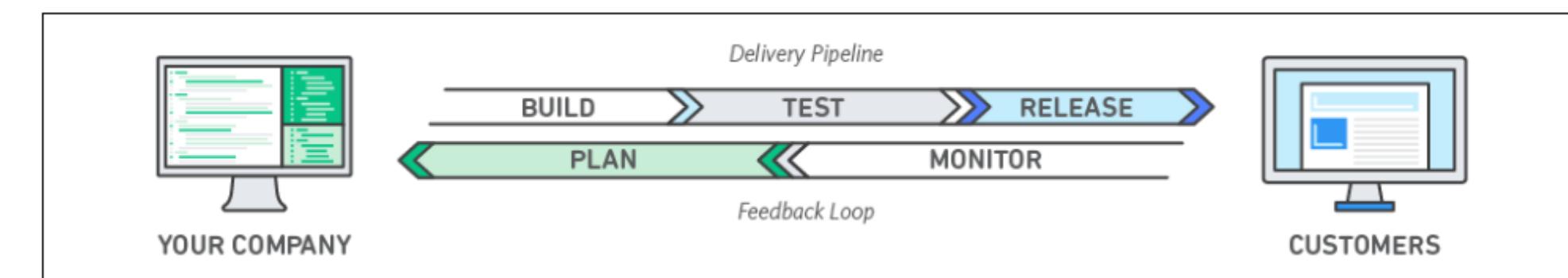
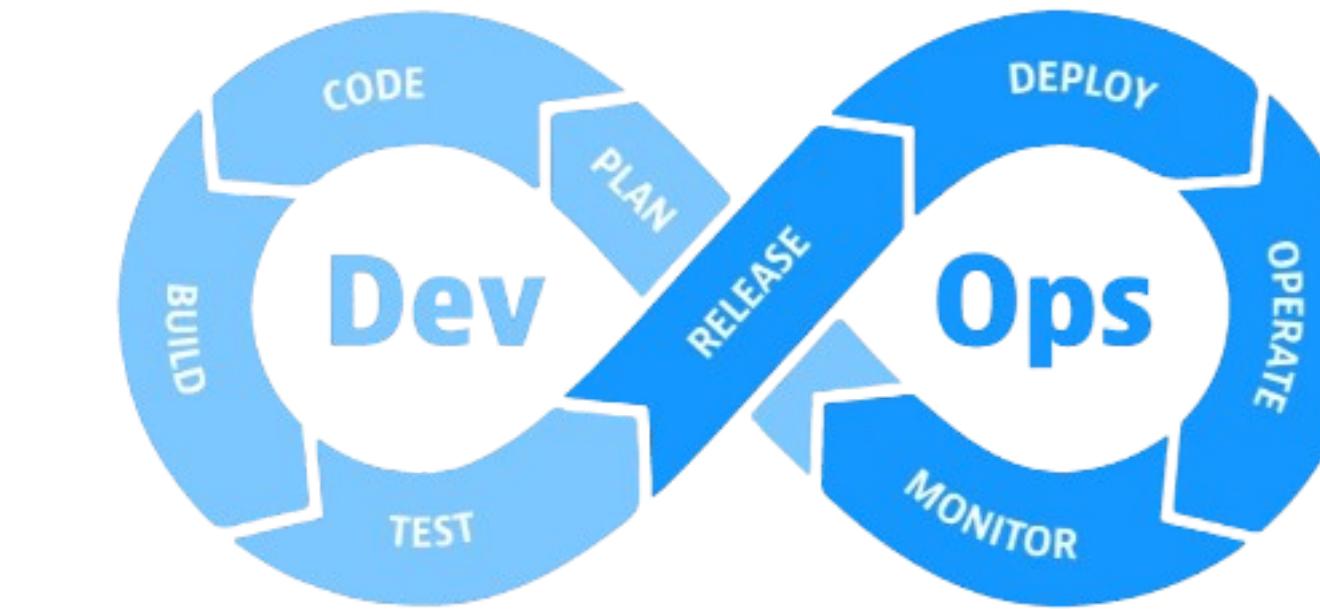


- 정의: DevOps는 소프트웨어 개발과 운영을 통합하는 것
- 목적: 빠르고 안정적인 소프트웨어 서비스를 지속적으로 제공하기 위함

| DevOps 정의

: DevOps Lifecycle

◆ DevOps Lifecycle



| DevOps 정의

: DevOps Lifecycle

◆ DevOps Lifecycle

1. Plan

서비스 설계 단계에서 개발자와 Co-work 하여 안전한 서비스 구조를 설계하기 위한 단계

2. Build

소스 코드 작성 및 커밋 후, 사용한 언어에 맞는 컴파일러 또는 인터프리터로 빌드하여 빌드 아티팩트를 생성하는 단계

3. Test

빌드 아티팩트의 안정성과 기능성 검증을 통한 최종 배포 준비 단계

4. Deploy

실제 운영 환경에 배포하는 단계

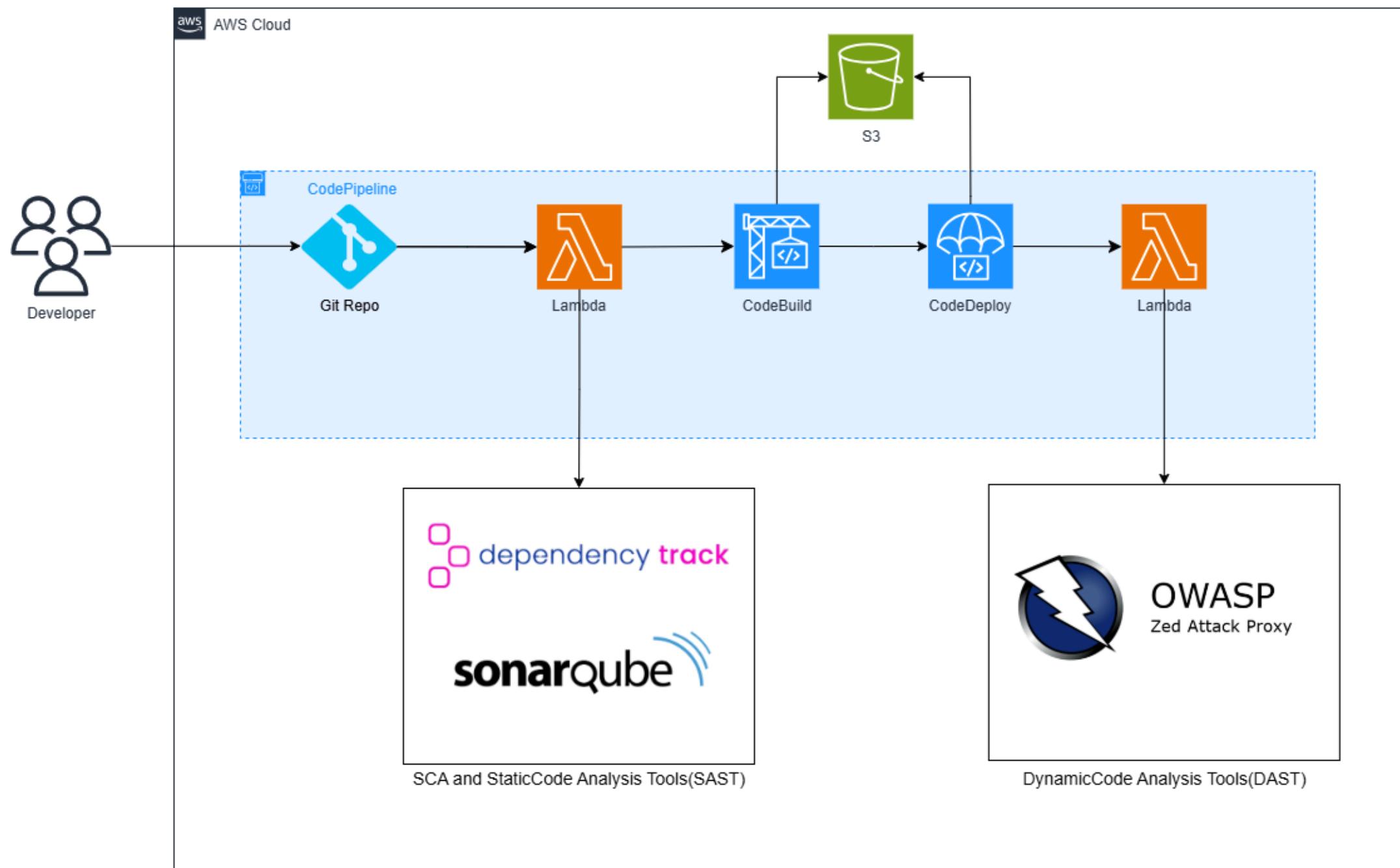
5. Monitor

운영 중인 서비스의 상태와 성능을 실시간으로 모니터링하고, 발생한 오류를 파악하여 서비스의 지속적인 개선을 도모하는 단계

| CICD Security Best..? Practice

: CICD Security Best..? Practice 아키텍처

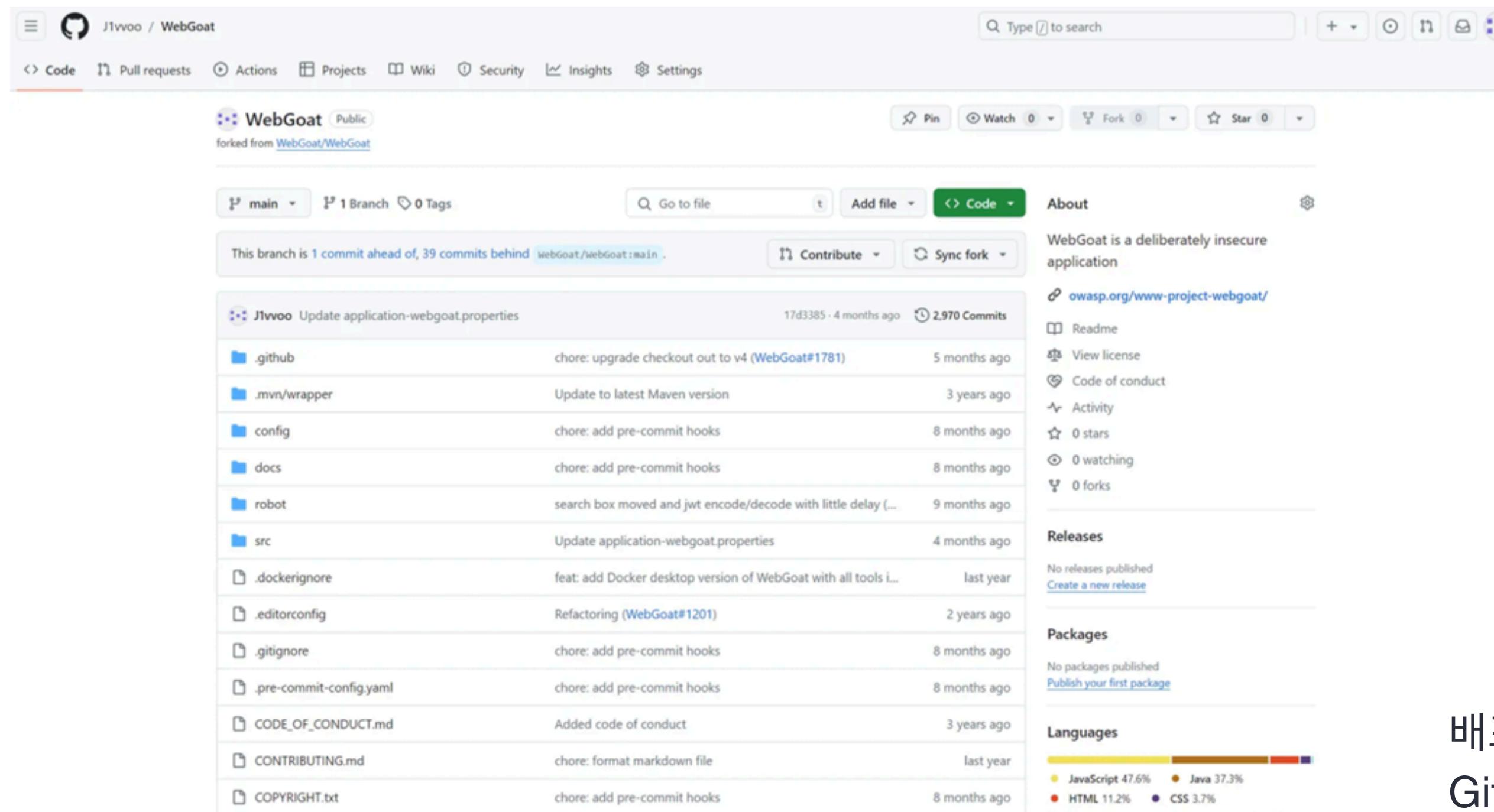
◆ CICD Security Best..? Practice 아키텍처



| CICD 무중단 배포

: CICD BlueGreen 배포 구현

◆ GitHub Repository 생성



The screenshot shows a GitHub repository page for 'WebGoat'. The repository is public and was forked from 'WebGoat/WebGoat'. The commit history on the 'main' branch shows 2,970 commits, with the most recent being 4 months ago. The repository details on the right indicate it's a deliberately insecure application, with links to 'owasp.org/www-project-webgoat/' and other repository metrics like 0 stars, 0 forks, and 0 watching. The repository uses multiple languages, with JavaScript being the most prominent at 47.6%.

Commit	Message	Time
J1vvo Update application-webgoat.properties	17d3385 · 4 months ago	2,970 Commits
.github	chore: upgrade checkout out to v4 (WebGoat#1781)	5 months ago
.mvn/wrapper	Update to latest Maven version	3 years ago
config	chore: add pre-commit hooks	8 months ago
docs	chore: add pre-commit hooks	8 months ago
robot	search box moved and jwt encode/decode with little delay (...)	9 months ago
src	Update application-webgoat.properties	4 months ago
.dockerignore	feat: add Docker desktop version of WebGoat with all tools i...	last year
.editorconfig	Refactoring (WebGoat#1201)	2 years ago
.gitignore	chore: add pre-commit hooks	8 months ago
.pre-commit-config.yaml	chore: add pre-commit hooks	8 months ago
CODE_OF_CONDUCT.md	Added code of conduct	3 years ago
CONTRIBUTING.md	chore: format markdown file	last year
COPYRIGHT.txt	chore: add pre-commit hooks	8 months ago

배포할 서비스(WebGoat)의
GitHub에서 Fork 진행

| CICD 무중단 배포

: CICD BlueGreen 배포 구현

◆ 구현에 필요한 파일 정의

buildspec.yml

```
version: 0.2

phases:
  install:
    runtime-versions:
      java: corretto17
  pre_build:
    commands:
      - echo "Pre-build phase starting..."
  build:
    commands:
      - echo "Build started on `date`"
      - ./mvnw clean install -D skipTests
  post_build:
    commands:
      - echo "Build completed on `date`"

artifacts:
  files:
    - target/webgoat-2023.9-SNAPSHOT.jar
    - appspec.yml
    - start.sh
```

appspec.yml

```
version: 0.0
os: linux
files:
  - source: /
    destination: /home/ec2-user/build/
permissions:
  - object: /
    pattern: "*"
    owner: ec2-user
    group: ec2-user
hooks:
  AfterInstall:
    - location: start.sh
      timeout: 60
      runas: ec2-user
```

start.sh

```
#!/bin/bash

JARFILE=$(ls /home/ec2-user/build/target/*.jar)
echo "filename : $JARFILE" >> startsh.log

CURRENT_PID=$(pgrep -f $JARFILE)
echo "currentpid : $CURRENT_PID" >> startsh.log
kill -9 $CURRENT_PID 2>/dev/null
sleep 5

nohup java -jar $JARFILE > /dev/null 2> /dev/null < /dev/null &
```

CICD 무중단 배포

: CICD BlueGreen 배포 구현

◆ 빌드 진행 테스트

codebuild-jw:661d740d-518b-48f5-a92f-a96224cf633a

빌드 중지 빌드 재시도

빌드 상태			
상태	시작한 사용자	빌드 ARN	해결된 소스 버전
성공함	codepipeline/CodePipeline-jw2	arn:aws:codebuild:ap-northeast-2:381491915575:build/codebuild-jw:661d740d-518b-48f5-a92f-a96224cf633a	b06306abc89bf76016fa4ee52482b92e6c57b9de
시작 시간	종료 시간	빌드 번호	
10월 14, 2024 11:25 오후 (UTC+9:00)	10월 14, 2024 11:27 오후 (UTC+9:00)	100	

빌드 로그 단계 세부 정보 보고서 환경 변수 빌드 세부 정보 리소스 사용률

이름	상태	컨텍스트	기간	시작 시간	종료 시간
SUBMITTED	성공함	-	<1 sec	10월 14, 2024 11:25 오후 (UTC+9:00)	10월 14, 2024 11:25 오후 (UTC+9:00)
QUEUED	성공함	-	<1 sec	10월 14, 2024 11:25 오후 (UTC+9:00)	10월 14, 2024 11:25 오후 (UTC+9:00)
PROVISIONING	성공함	-	5 secs	10월 14, 2024 11:25 오후 (UTC+9:00)	10월 14, 2024 11:25 오후 (UTC+9:00)
DOWNLOAD_SOURCE	성공함	-	3 secs	10월 14, 2024 11:25 오후 (UTC+9:00)	10월 14, 2024 11:25 오후 (UTC+9:00)
INSTALL	성공함	-	<1 sec	10월 14, 2024 11:25 오후 (UTC+9:00)	10월 14, 2024 11:25 오후 (UTC+9:00)
PRE_BUILD	성공함	-	<1 sec	10월 14, 2024 11:25 오후 (UTC+9:00)	10월 14, 2024 11:25 오후 (UTC+9:00)
BUILD	성공함	-	90 secs	10월 14, 2024 11:25 오후 (UTC+9:00)	10월 14, 2024 11:27 오후 (UTC+9:00)
POST_BUILD	성공함	-	<1 sec	10월 14, 2024 11:27 오후 (UTC+9:00)	10월 14, 2024 11:27 오후 (UTC+9:00)

정상적인 빌드 확인

| CICD 무중단 배포

: CICD BlueGreen 배포 구현

◆ 빌드 아티팩트 확인

Amazon S3 > 버킷 > build-artifacts-jw > codebuild-jw

codebuild-jw 정보

S3 URI 복사 다운로드 열기 객체 작업

속성 권한 버전

객체 개요

이름	압축 크기	원본 크기	파일 종류
target	167	272	Yaml 원본 파일
appspec.yml	194	289	SH 원본 파일
start.sh			

소유자: 9c878004c9ba17348f8bc952a2a657779df40ba984db412e557d56d321ebf73c

S3 URI: s3://build-artifacts-jw/codebuild-jw

AWS 리전: 아시아 태평양(서울) ap-northeast-2

마지막 수정: 2024. 9. 25. am 4:16:22 AM KST

크기: 103.3MB

유형: ZIP

키: codebuild-jw

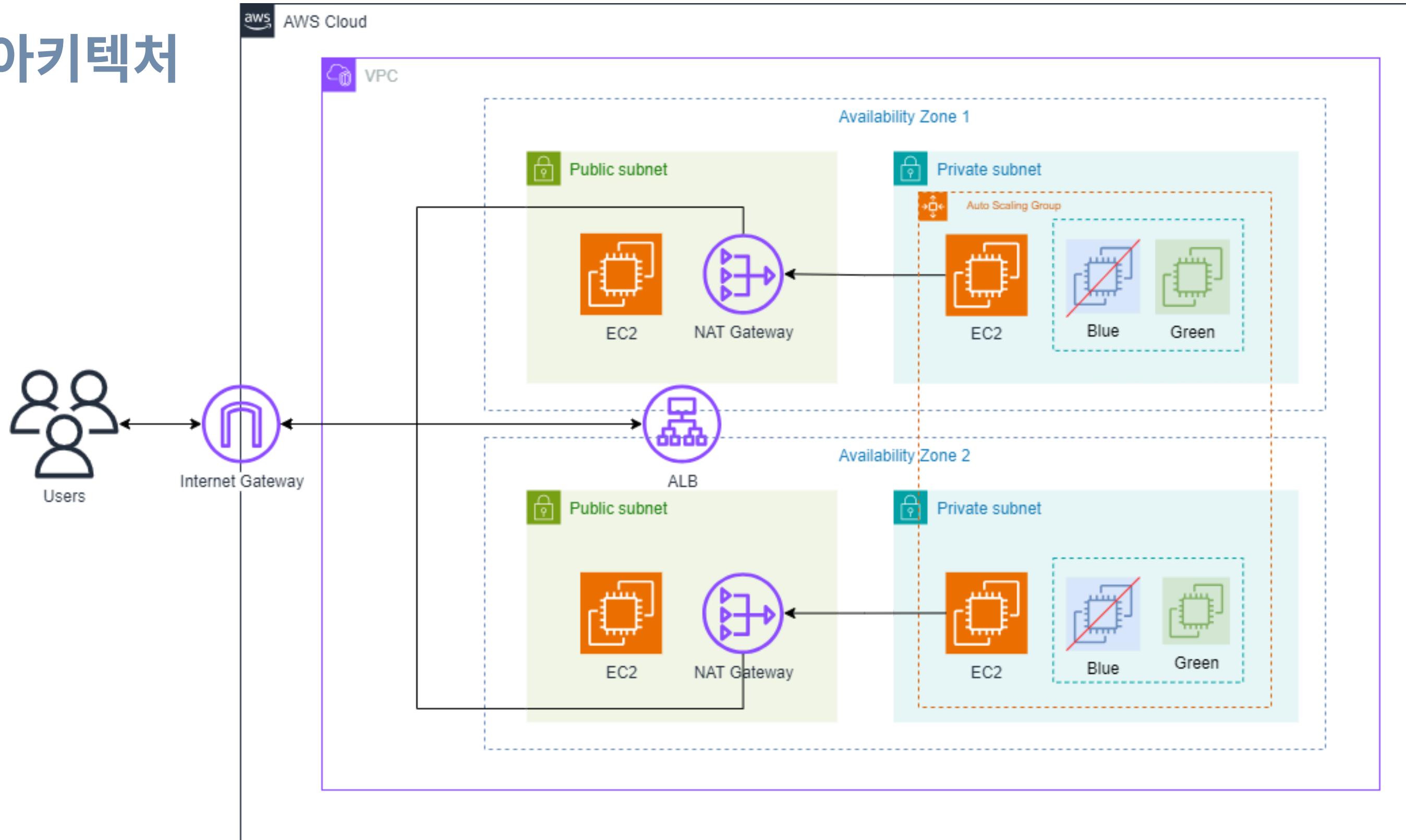
내부 구조:

- target
- appspec.yml
- start.sh

| CICD 무중단 배포

: CICD BlueGreen 배포 구현

◆ VPC 아키텍처



| CICD 무중단 배포

: CICD BlueGreen 배포 구현

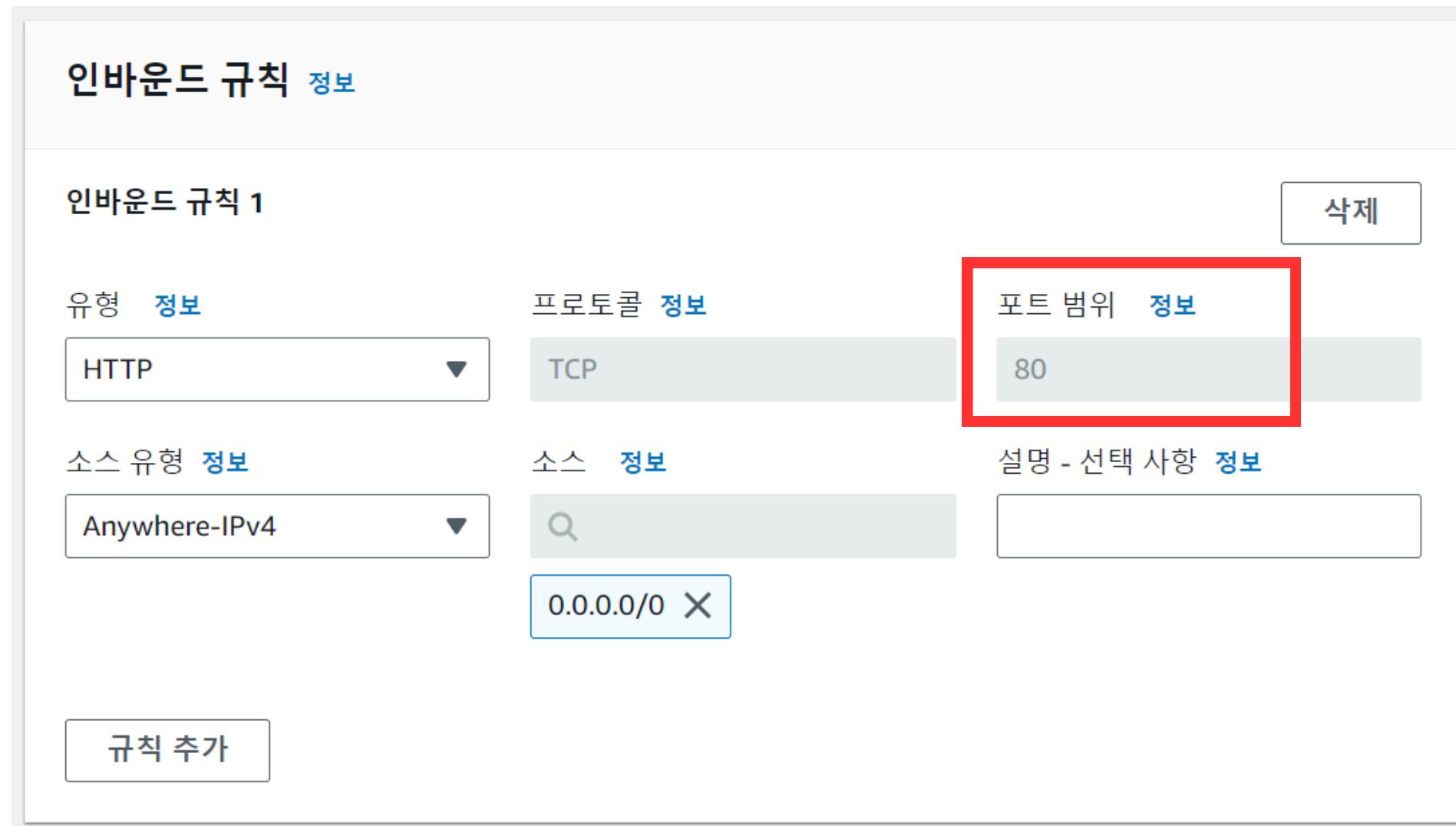
◆ VPC 리소스 맵



| CICD 무중단 배포

: CICD BlueGreen 배포 구현

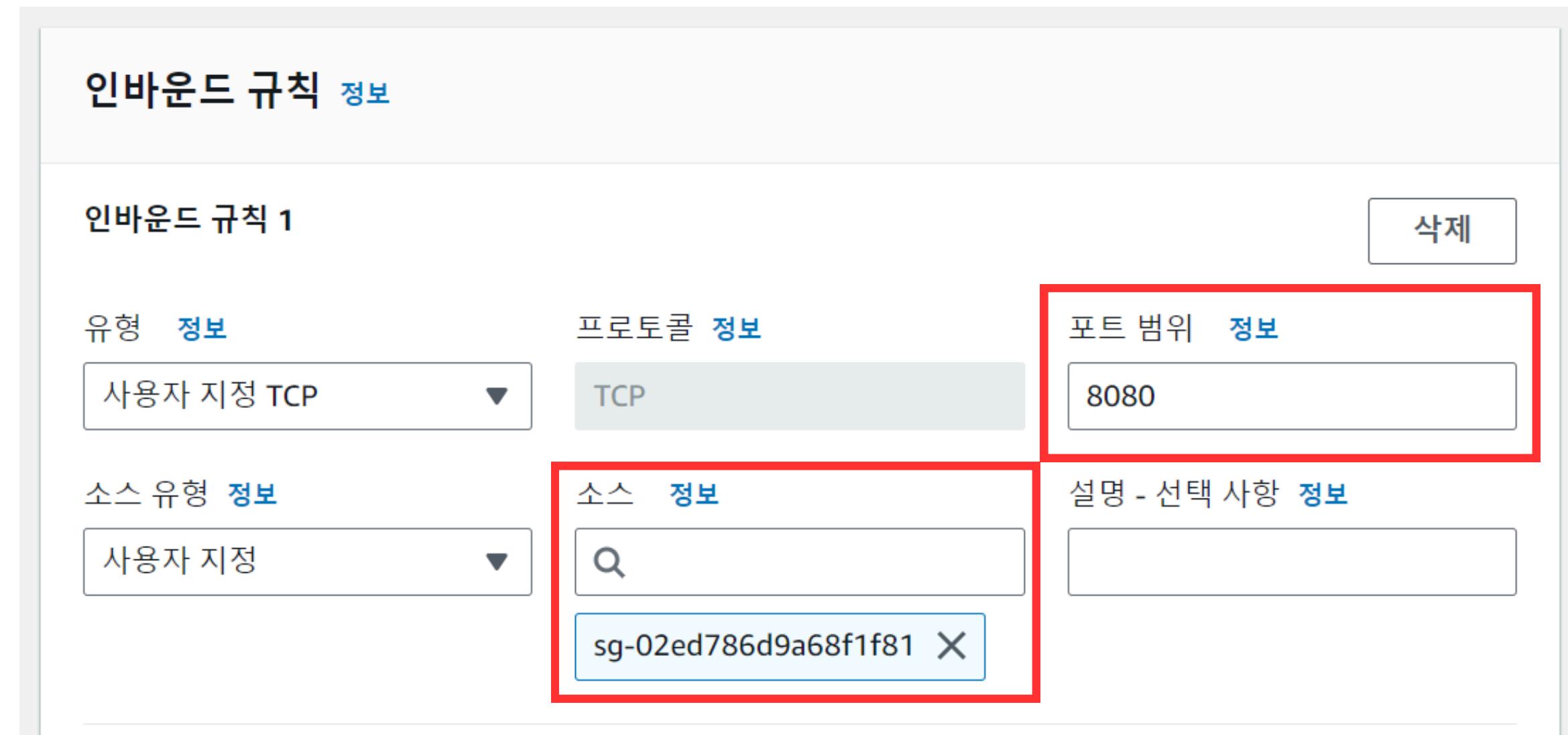
◆ 보안 그룹 설정 Security Group_LB



| CICD 무중단 배포

: CICD BlueGreen 배포 구현

◆ 보안 그룹 설정 Security Group_APP



CICD 무중단 배포

: CICD BlueGreen 배포 구현

IAM 역할 생성

BlueGreenEC2Role

[IAM](#) > [역할](#) > [BlueGreenEC2Role](#)

BlueGreenEC2Role 정보

Allows EC2 instances to call AWS services on your behalf.

요약		
생성 날짜 October 08, 2024, 03:01 (UTC+09:00)	ARN <input type="checkbox"/> arn:aws:iam::381491915575:role/BlueGreenEC2Role	인스턴스 프로파일 ARN <input type="checkbox"/> arn:aws:iam::381491915575:instance-profile/BlueGreenEC2Role
마지막 활동 -	최대 세션 지속 시간 1시간	

권한 | 신뢰 관계 | 태그 | Last Accessed | 세션 취소

권한 정책 (1) 정보

최대 10개의 관리형 정책을 연결할 수 있습니다.

필터링 기준 유형
검색 <input type="text"/> 모든 유형

정책 이름 정책 이름 AmazonEC2RoleforAWSCodeDeploy

유형 AWS 관리형

연결된 엔터티 6

BlueGreenCodeDeployServiceRole

[IAM](#) > [역할](#) > [BlueGreenCodeDeployServiceRole_jw](#)

BlueGreenCodeDeployServiceRole_jw 정보

Allows CodeDeploy to call AWS services such as Auto Scaling on your behalf.

요약		
생성 날짜 September 06, 2024, 15:59 (UTC+09:00)	ARN <input type="checkbox"/> arn:aws:iam::381491915575:role/BlueGreenCodeDeployServiceRole_jw	최대 세션 지속 시간 1시간
마지막 활동 <input checked="" type="checkbox"/> 6일 전		

권한 | 신뢰 관계 | 태그 | Last Accessed | 세션 취소

권한 정책 (2) 정보

최대 10개의 관리형 정책을 연결할 수 있습니다.

필터링 기준 유형

검색	모든 유형
<input type="checkbox"/> 정책 이름 <input type="checkbox"/> 정책 이름 <input type="checkbox"/> AWSCodeDeployRole	AWS 관리형
<input type="checkbox"/> <input checked="" type="checkbox"/> BlueGreenPolicy_jw	고객 인라인

| CICD 무중단 배포

: CICD BlueGreen 배포 구현

◆ Auto Scaling Group 생성

네트워크 Info

대부분의 애플리케이션에서는 여러 가용 영역을 사용할 수 있으며 EC2 Auto Scaling이 여러 영역 간에 인스턴스를 균일하게 분산할 수 있습니다. 기본 VPC와 기본 서브넷은 빠르게 시작하는 데 적합합니다.

VPC
Auto Scaling 그룹의 가상 네트워크를 정의하는 VPC를 선택합니다.

vpc-06c90836fbda77c4d (BlueGreen-jw)
10.0.0.0/16

VPC 생성

가용 영역 및 서브넷
선택한 VPC에서 Auto Scaling 그룹이 사용할 수 있는 가용 영역과 서브넷을 정의합니다.

가용 영역 및 서브넷 선택

ap-northeast-2a | subnet-
0ddce741b93403224
(PrivateSubnet1_jw)
10.0.2.0/24

ap-northeast-2c | subnet-
0b98c7e84f46eac0e (PrivateSubnet2_jw)
10.0.3.0/24

서브넷 생성

애플리케이션의 가용성과 성능 유지를 위한 Auto Scaling Group 설정

그룹 크기 Info
오토 스케일링의 초기 크기를 설정합니다. 그룹을 생성한 후 수동으로 또는 Auto Scaling을 사용하여 필요에 맞게 그룹 크기를 변경할 수 있습니다.

원하는 용량 유형
원하는 용량 값의 측정 단위를 선택합니다. vCPU 및 메모리(GiB)는 일련의 인스턴스 속성으로 구성된 혼합 인스턴스 그룹에 대해서만 지원됩니다.

단위(인스턴스 개수)

원하는 용량
그룹 크기를 지정하세요.

1

크기 조정 Info
수요 변화에 따라 오토 스케일링의 크기를 수동 또는 자동으로 조정할 수 있습니다.

크기 조정 한도
원하는 용량을 늘리거나 줄일 수 있는 양의 한도를 설정합니다.

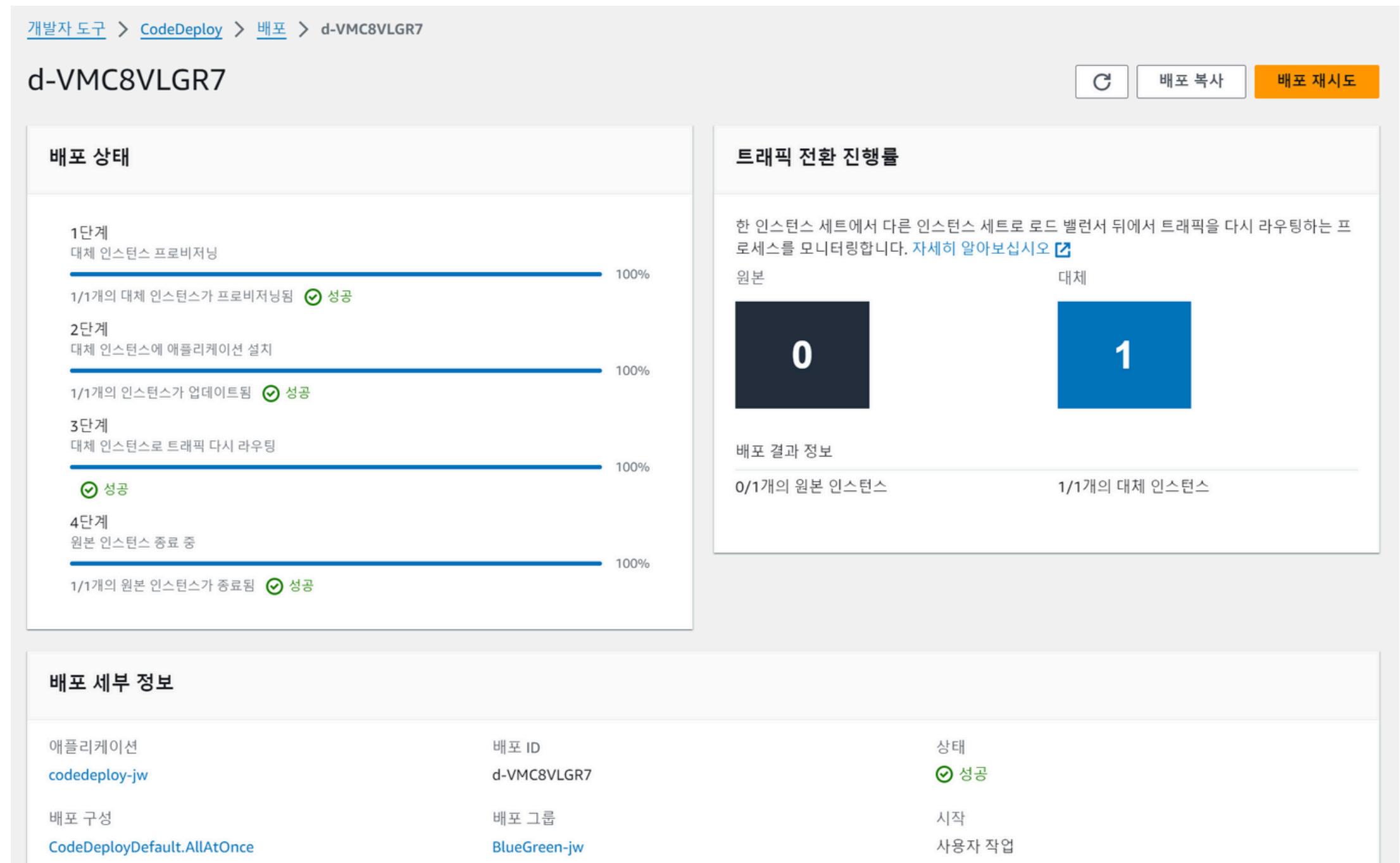
원하는 최소 용량 1
원하는 최대 용량 1

원하는 용량보다 작거나 같음
원하는 용량보다 크거나 같음

| CICD 무중단 배포

: CICD BlueGreen 배포 구현

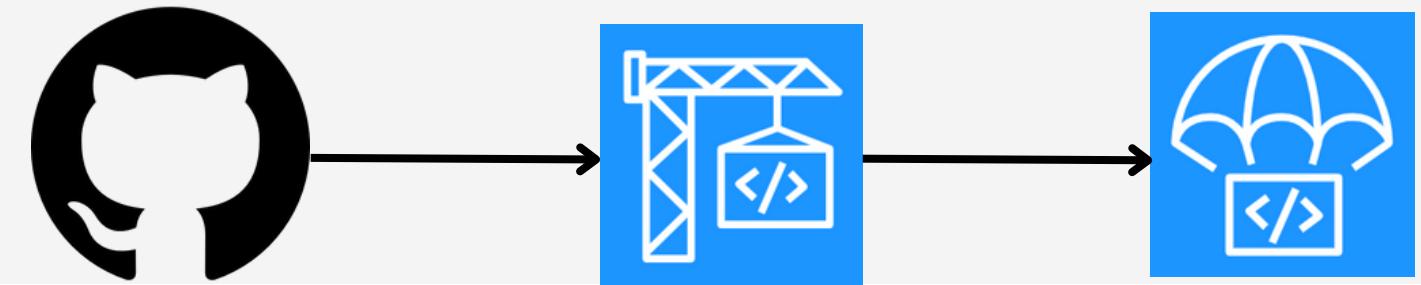
◆ CodeDeploy 배포 테스트



CodeDeploy를 통한
블루그린 방식의 배포 완료

CICD 무중단 배포

: CICD BlueGreen 배포 구현



◆ CodePipeline 실행

WebGoat / buildspec.yml in main

Cancel changes Commit changes...

Edit Preview Code 55% faster with GitHub Copilot

Spaces 4 No wrap

```
version: 0.2
phases:
  install:
    runtime-versions:
      java: corretto17
  pre_build:
    commands:
      - echo "Pre-build phase starting...."
  build:
    commands:
      - echo "Build started on `date`"
      - ./mvnw clean install -D skipTests
  post_build:
    commands:
      - echo "Build completed on `date`"
artifacts:
  files:
    - target/webgoat-2023.9-SNAPSHOT.jar
    - appspec.yml
    - start.sh
```

코드 변경 후 커밋 즉시,
파이프라인 자동 실행

파이프라인 정보

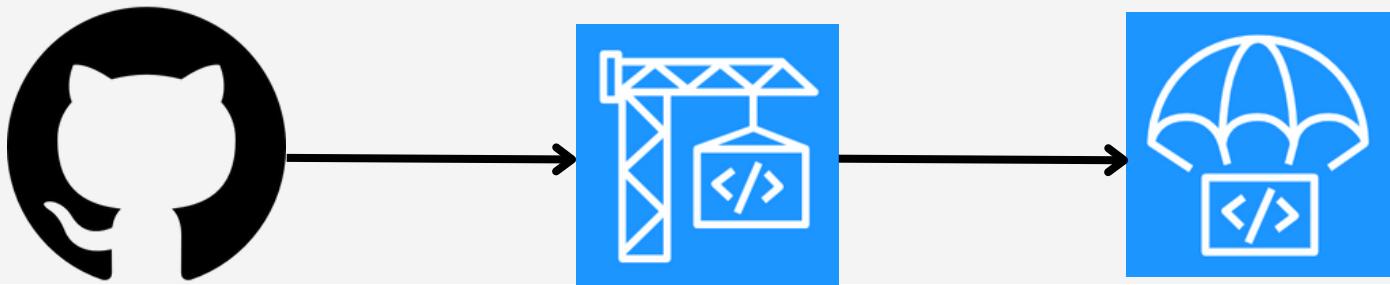
C 알림 기록 보기 변경 사항 릴리스 파이프라인 삭제 파이프라인 생성

< 1 > 설정

이름	최근 실행 상태	최신 소스 개정	시작된 최근 실행	가장 최근 실행
CodePipeline-jw (유형: V2 실행 모드: QUEUED)	진행 중	Source - c5b3e1ac: Update buildspec.yml	방금	세부 정보 보기

CICD 무중단 배포

: CICD BlueGreen 배포 구현



◆ CodePipeline 실행

개발자 도구 > CodePipeline > 파이프라인 > CodePipeline-jw

CodePipeline-jw

파이프라인 유형: V2 실행 모드: QUEUED

Source 성공
파이프라인 실행 ID: [6d71672a-2594-414b-8eda-47825e0a1376](#)

GitHub(버전 2)
성공 - 13분 전
[f68f8133](#)

세부 정보 보기

전환 비활성화

Build 성공
파이프라인 실행 ID: [6d71672a-2594-414b-8eda-47825e0a1376](#)

Build
AWS CodeBuild
성공 - 11분 전
[f68f8133](#) Source: Update start.sh

세부 정보 보기

전환 비활성화

Deploy 성공
파이프라인 실행 ID: [6d71672a-2594-414b-8eda-47825e0a1376](#)

Deploy
AWS CodeDeploy
성공 - 빨금
[f68f8133](#) Source: Update start.sh

세부 정보 보기

전환 비활성화

돌백 시작

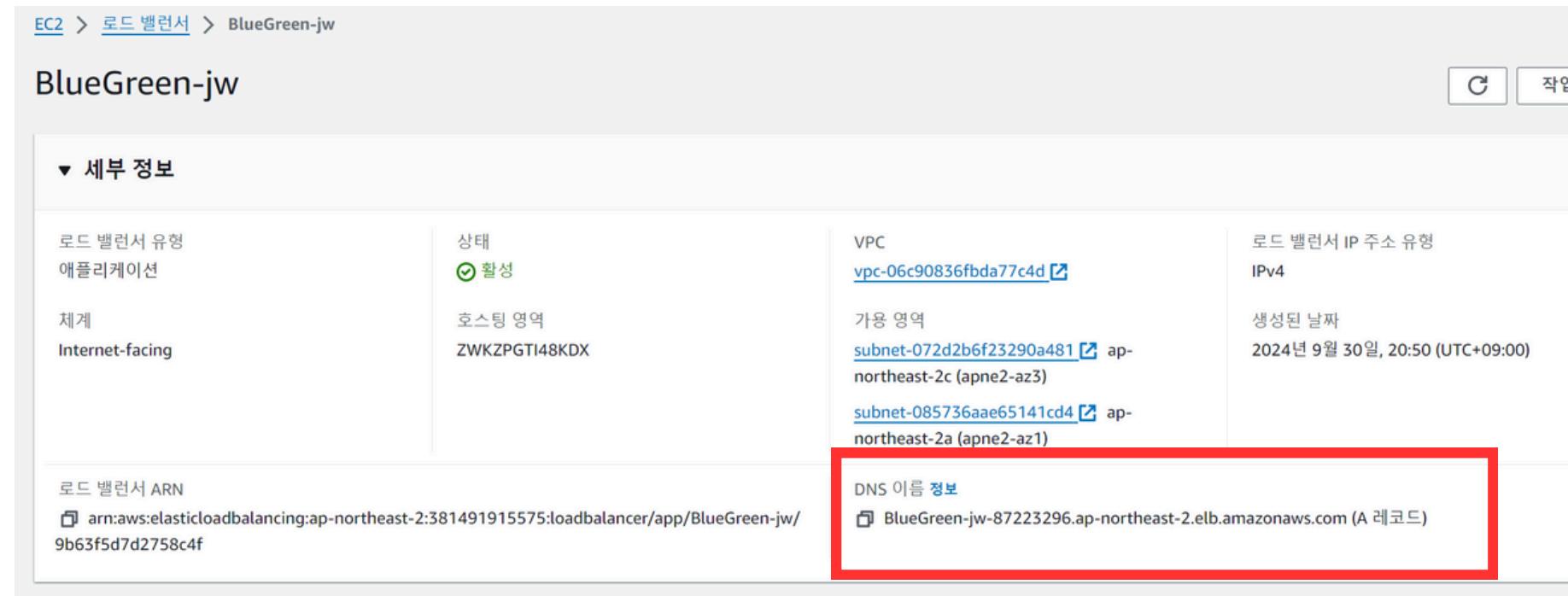
돌백 시작

돌백 시작

| CICD 무중단 배포

: CICD BlueGreen 배포 구현

◆ 배포 서비스(WebGoat) 접속



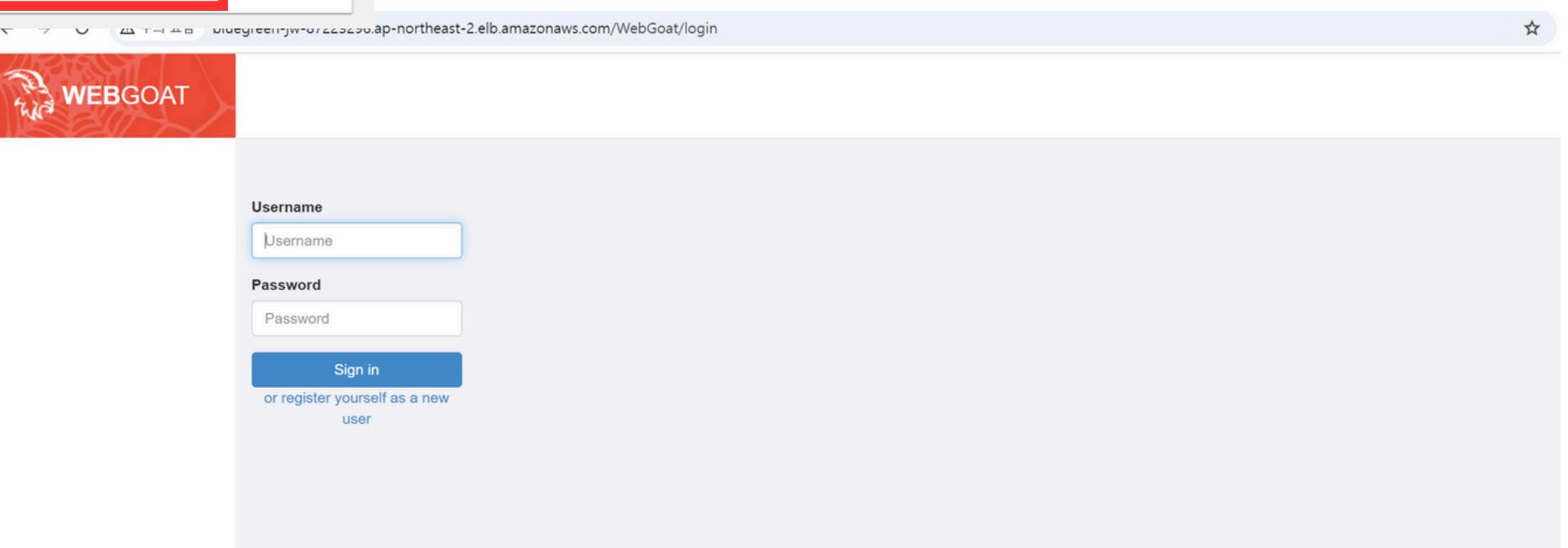
EC2 > 로드밸런서 > BlueGreen-jw

BlueGreen-jw

▼ 세부 정보

로드밸런서 유형 애플리케이션	상태 활성	VPC vpc-06c90836fbda77c4d	로드밸런서 IP 주소 유형 IPv4
체계 Internet-facing	호스팅 영역 ZWKZPGTI48KDX	가용 영역 subnet-072d2b6f23290a481 ap-northeast-2c (apne2-az3) subnet-085736aae65141cd4 ap-northeast-2a (apne2-az1)	생성된 날짜 2024년 9월 30일, 20:50 (UTC+09:00)
로드밸런서 ARN arn:aws:elasticloadbalancing:ap-northeast-2:381491915575:loadbalancer/app/BlueGreen-jw/9b63f5d7d2758c4f	DNS 이름 정보 BlueGreen-jw-87223296.ap-northeast-2.elb.amazonaws.com (A 레코드)		

로드밸런서 DNS 주소 복사 후
웹 브라우저로 서비스 접속 확인



| CICD 무중단 배포

: Golden Image를 위한 Image Builder VS Packer & Ansible

◆ **Golden Image** 정의

“ 표준화된 시스템 구성을 포함하는 기준 이미지 파일 ”

운영체제, 애플리케이션, 보안 패치, 유틸리티 등 **특정 시스템 환경을 사전에 설정하고, 이를 다른 시스템에 동일하게 배포할 수 있도록 저장한 템플릿**

- 스크립팅 및 자동화를 통해 클라우드 환경에서 **배포 시간 단축 가능**
- **신속한 패치 관리 및 업데이트 가능** & 자동화
- **사전에 보안 패치와 필수 보안 설정이 적용된 상태로** 관리되며, 이를 통해 배포된 모든 시스템이 동일한 보안 기준을 준수하도록 보장

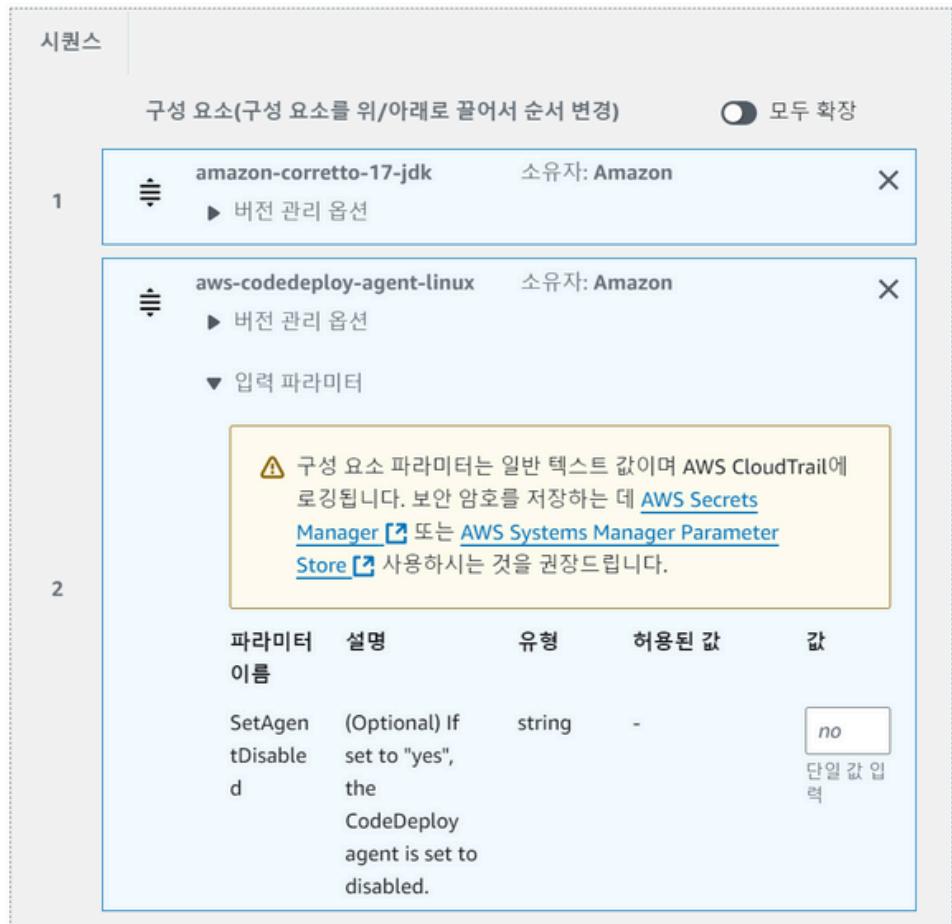
| AWS Golden Image: <https://docs.aws.amazon.com/managedservices/latest/appguide/ex-immute-gold-ami.html>

CICD 무중단 배포

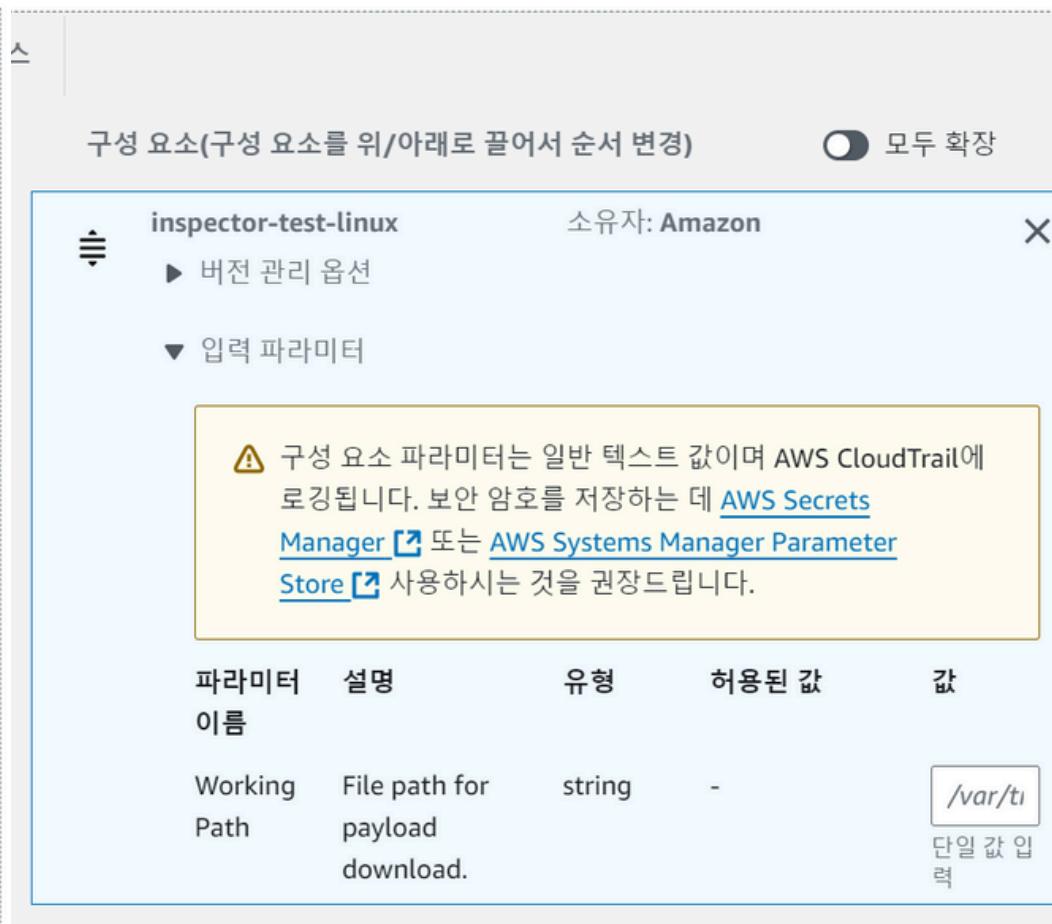
: Golden Image를 위한 Image Builder VS Packer & Ansible

◆ Image Builder

빌드 컴포넌트 추가



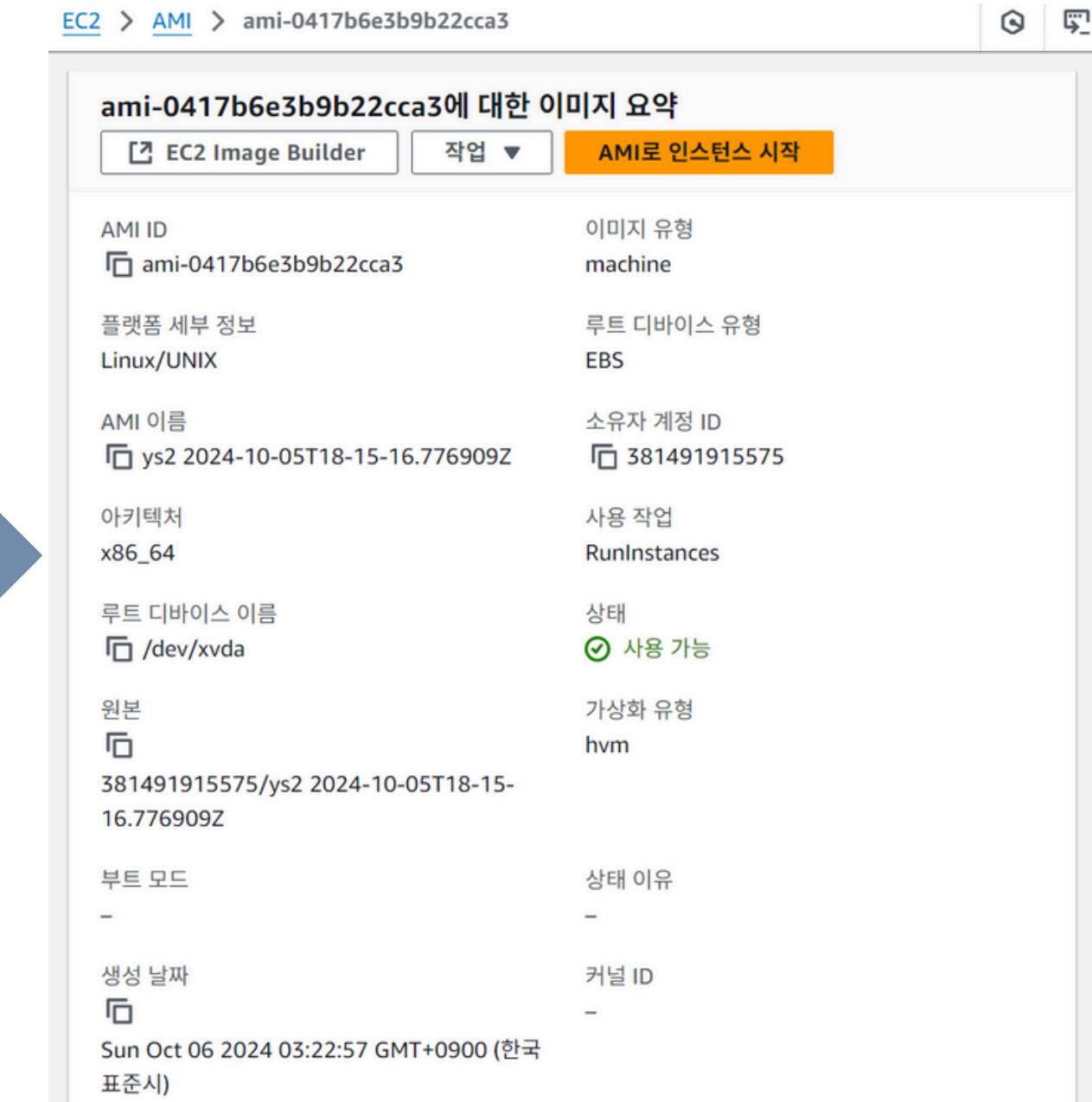
테스트 컴포넌트 추가



- amazon-corretto-17-jdk
- aws-codedeploy-agent-linux

- inspector-test-linux

AMI 생성



| CICD 무중단 배포

: Golden Image를 위한 Image Builder VS Packer & Ansible

◆ **Packer & Ansible**

(AWS CLI)

1. **packer & ansible 설치**
2. Packer.hcl 생성
3. Ansible 디렉토리 구조로 작업 정의

| CICD 무중단 배포

: Golden Image를 위한 Image Builder VS Packer & Ansible

◆ Packer & Ansible

(AWS CLI)

1. packer & ansible 설치

2. **Packer.hcl** 생성

3. Ansible 디렉토리 구조로 작업 정의

```
packer {  
    required_plugins {  
        amazon = {  
            source  = "github.com/hashicorp/amazon"  
            version = "~> 1"  
        }  
        ansible = {  
            source  = "github.com/hashicorp/ansible"  
            version = "~> 1"  
        }  
    }  
  
    source "amazon-ebs" "amazon-linux" {  
        ami_name          = "packer-zs"  
        instance_type     = "t3.medium"  
        region            = "ap-northeast-2"  
        vpc_id             = "vpc-06c90836fbda77c4d"  
        subnet_id          = "subnet-085736aae65141cd4"  
        ssh_username       = "ec2-user"  
        security_group_ids = ["sg-010282d370f05e952"]  
        ami_users          = ["381491915575"]  
        associate_public_ip_address = true  
    }  
}
```

```
source_ami_filter {  
    filters = {  
        name          = "amzn2-ami-hvm-*-x86_64-gp2"  
        root-device-type = "ebs"  
        virtualization-type = "hvm"  
    }  
    most_recent = true  
    owners      = ["137112412989"] # Amazon 공식 계정  
}  
}  
  
build {  
    name      = "packerandansible-zs"  
    sources  = [  
        "source.amazon-ebs.amazon-linux"  
    ]  
  
    provisioner "ansible" {  
        playbook_file = "./playbook.yml"  
    }  
}
```

| CICD 무중단 배포

: Golden Image를 위한 Image Builder VS Packer & Ansible

◆ Packer & Ansible

(AWS CLI)

1. packer & ansible 설치

2. Packer.hcl 생성

3. Ansible 디렉토리 구조로 작업 정의

```
packer_tutorial/
├── packer_zs.pkr.hcl    # Packer 설정 파일 (이미지 생성 및 빌드를 위한 설정)
├── playbook.yml          # Ansible Playbook (역할을 지정하여 서버 설정을 자동화)
└── roles/
    ├── code_agent/
    │   ├── tasks/
    │   │   └── main.yml    # CodeDeploy 에이전트 설치 및 설정 작업 정의
    │   ├── java17/
    │   │   ├── tasks/
    │   │   │   └── main.yml  # Java 17 설치 관련 역할
    │   │   └── main.yml      # Java 17 설치 및 확인 작업 정의
    └── common/
        ├── tasks/
        └── main.yml          # 공통 설정 관련 역할 (모든 인스턴스에 적용될 설정)
```

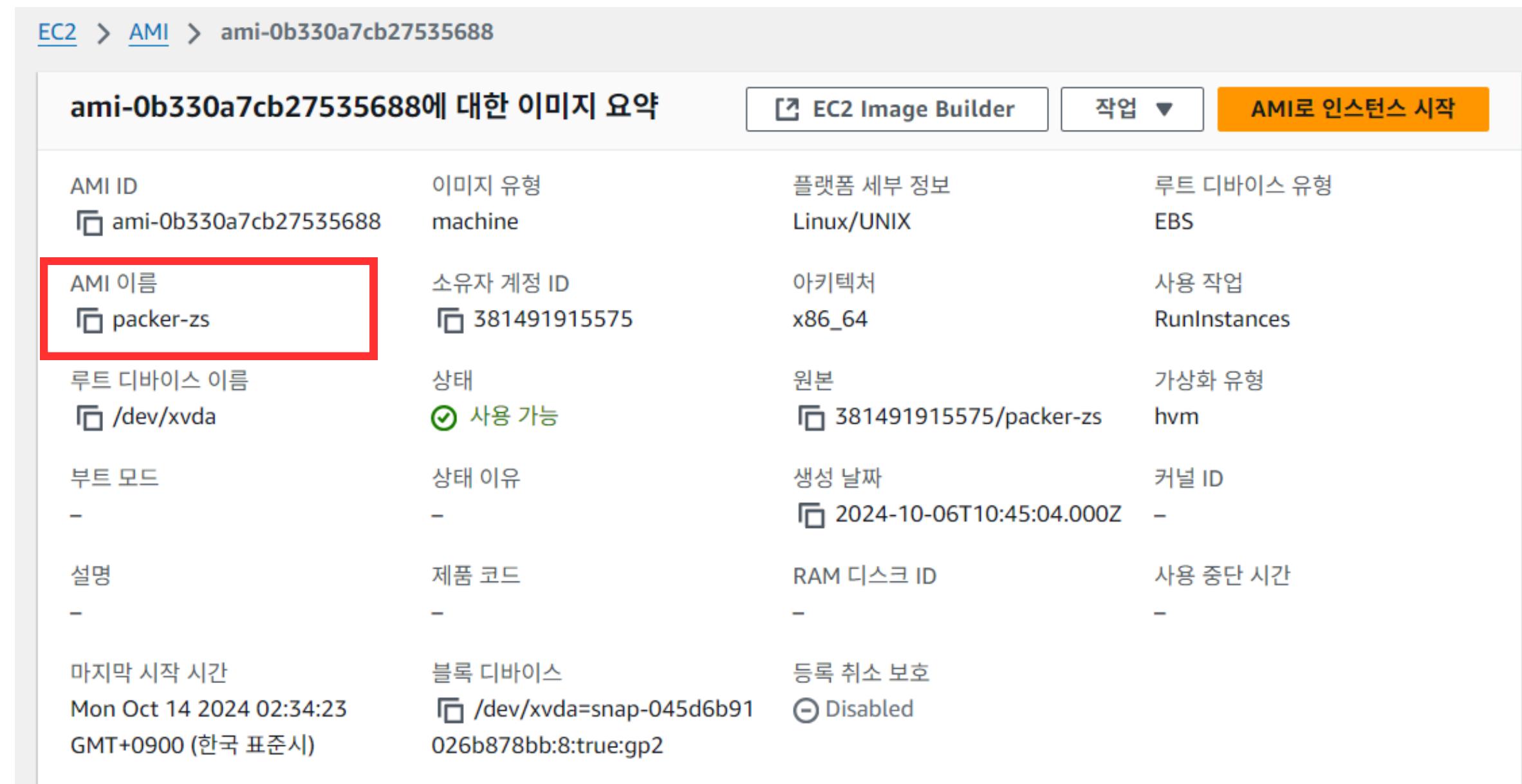
```
...
- name: Provision Java 17 and Code Agent
  hosts: all
  become: yes
  roles:
    - code_agent
    - java17
```

CICD 무중단 배포

: Golden Image를 위한 Image Builder VS Packer & Ansible

◆ Packer & Ansible

AMI 생성



EC2 > AMI > ami-0b330a7cb27535688

ami-0b330a7cb27535688에 대한 이미지 요약

AMI ID	이미지 유형	플랫폼 세부 정보	루트 디바이스 유형
ami-0b330a7cb27535688	machine	Linux/UNIX	EBS
AMI 이름	소유자 계정 ID	아키텍처	사용 작업
packer-zs	381491915575	x86_64	RunInstances
루트 디바이스 이름	상태	원본	가상화 유형
/dev/xvda	사용 가능	381491915575/packer-zs	hvm
부트 모드	상태 이유	생성 날짜	커널 ID
-	-	2024-10-06T10:45:04.000Z	-
설명	제품 코드	RAM 디스크 ID	사용 중단 시간
-	-	-	-
마지막 시작 시간	블록 디바이스	등록 취소 보호	
Mon Oct 14 2024 02:34:23 GMT+0900 (한국 표준시)	/dev/xvda=snap-045d6b91 026b878bb:8:true:gp2	Disabled	

| CICD 무중단 배포

: Golden Image를 위한 Image Builder VS Packer & Ansible



VS



HashiCorp
Packer



| CICD 무중단 배포

: Golden Image를 위한 Image Builder VS Packer & Ansible



VS



- Image Builder에 비해 비교적 빠른 AMI 생성
- 커스터마이징 유연성 우수

| SAST 분석

: SAST 분석의 개요

◆ SAST 분석이란?

SAST (Static Application Security Tools)

정적 코드 분석으로 소스 코드의 취약점 검사 진행

- 개발 초기에 취약점을 발견
- SQL injection, XSS, 오버플로우 등과 같은 코드 수준의 보안 취약점 찾아냄
- 도구에 따른 다양한 프로그래밍 언어 지원

◆ SAST 분석 도구 종류



sonarQube

Checkmarx

VERACODE

| CodeQL vs. SonarQube

: CodeQL과 SonarQube의 정성적인 비교 분석 지표

	CodeQL	SonarQube
언어 지원 및 호환성	C/C++, C#, Go, Java, Kotlin, JavaScript, Python, Ruby, TypeScript, Swift ⇒ SonarQube에 비해 제한된 언어 지원	C, C++, C#, Go, Java, JavaScript, Python, Ruby, PHP, HTML, CSS, XML 등
실시간 분석	제공하지 않음	제공 (상용 버전)
다른 도구와의 통합	GitHub Actions 통합에 최적화, Jenkins, Azure DevOps도 지원	거의 모든 CI/CD 툴과 통합 가능 (Jenkins, GitLab, Azure DevOps 등)
주요 목적	보안 취약점 분석에 중점	코드 품질 및 보안 취약점
라이선스 비용	무료	-무료 버전 -유료 버전의 기능으로 Pull Request에 대해 정적 분석 코멘트를 남겨주는 Pull Request Decoration

CodeQL vs. SonarQube

: CodeQL과 SonarQube의 정량적 비교분석을 위한 테스트 구현

◆ CodeQL

codeql_setup.sh(환경구축코드)

```
GNU nano 6.2                               codeql_setup.sh
echo -e "\033[32m[+] codeQL Install & Setting\033[0m $@"
# Create directories if not exist
mkdir -p /home/target-repo/
mkdir -p /home/codeql

cd /home/codeql

# Install required packages
echo -e "\033[32m[+] Installing required packages (git, wget, unzip)\033[0m"
sudo apt update && sudo apt install git wget unzip -y

# Clone codeql repository if it doesn't exist
if [ ! -d "/home/codeql/codeql-repo" ]; then
  echo -e "\033[32m[+] Cloning CodeQL repository\033[0m"
  git clone https://github.com/github/codeql /home/codeql/codeql-repo
else
  echo -e "\033[33m[+] CodeQL repository already exists, skipping clone\033[0m"
fi

echo -e "\033[32m[+] CodeQL repository download complete\033[0m"

# Download and install CodeQL CLI if not already installed
if [ ! -d "/home/codeql/codeql-cli" ]; then
  echo -e "\033[32m[+] Downloading and installing CodeQL CLI\033[0m"
  wget https://github.com/github/codeql-cli-binaries/releases/download/v2.19.1/codeql
  unzip codeql-linux64.zip
  mv ./codeql ./codeql-cli
  rm -rf /home/codeql/codeql-linux64.zip
else
  echo -e "\033[33m[+] CodeQL CLI is already installed, skipping download\033[0m"
fi
```

codeql_analysis.sh(실행코드)

```
GNU nano 6.2                               codeql_analysis.sh
#!/bin/bash

# Function to display and clone Git repository
clone_repository() {
  echo -e "\033[32m[+] Git clone\033[0m"
  read -p "Enter git clone address: " repository_url

  datetime=$(date +"%m%d-%H%M")
  directory_name="${datetime}_$(basename "$repository_url")-repo"
  target_directory="/home/codeql/target-repo/$directory_name"
  mkdir -p "$target_directory"

  echo -e "\033[32m[+] Cloning into: $target_directory\033[0m"
  git clone --depth=1 "$repository_url" "$target_directory"
  if [ $? -ne 0 ]; then
    echo "Git 클론에 실패했습니다. URL을 확인하세요."
    exit 1
  fi
}

# Function to create CodeQL database and analyze
create_and_analyze_db() {
  echo -e "\033[32m[+] Create database\033[0m"

  # RAM 옵션을 추가하여 메모리 할당 증가 (8GB 할당)
  /home/codeql/codeql-cli/codeql database create --language="$language" --source-root="$target_directory"
  if [ $? -ne 0 ]; then
    echo "CodeQL 데이터베이스 생성에 실패했습니다."
    exit 1
  fi

  # 패키지 쿼리 분석 수행 (쿼리 경로를 명시적으로 추가)
  /home/codeql/codeql-cli/codeql database analyze "$target_directory/codeql-db" --force
  if [ $? -ne 0 ]; then
    echo "CodeQL 분석에 실패했습니다."
    exit 1
  fi

  echo -e "\033[32m[+] CodeQL analysis complete\033[0m"
}

# Main script execution
clone_repository

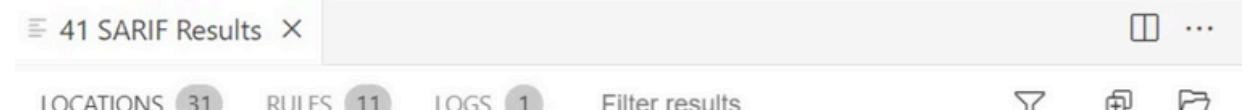
clear
echo -e "\033[32m[+] codeQL Language Selection\033[0m"
```

| CodeQL vs. SonarQube

: CodeQL과 SonarQube의 정량적 비교분석을 위한 테스트 구현

◆ CodeQL

SARIF 파일



The screenshot shows a SARIF Results interface with the following details:

- 41 SARIF Results
- LOCATIONS 31
- RULES 11
- LOGS 1
- Filter results

Line ↓ Message

- > JWTRefreshEndpoint.java src/main/java/org/owasp/webgoat/lessons/jwt 3
- > JWTVotesEndpoint.java src/main/java/org/owasp/webgoat/lessons/jwt 3
- > ProfileZipSlip.java src/main/java/org/owasp/webgoat/lessons/pathtraversal 2
- > ProfileUploadBase.java src/main/java/org/owasp/webgoat/lessons/pathtraversal 2
- > SqlInjectionLesson6a.java src/main/java/org/owasp/webgoat/lessons/sqlinjection/advanced 1
- > SqlInjectionLesson8.java src/main/java/org/owasp/webgoat/lessons/sqlinjection/introduction 1
- > TokenTest.java src/test/java/org/owasp/webgoat/lessons/jwt 2
- > CrossSiteScriptingLesson5a.java src/main/java/org/owasp/webgoat/lessons/xss 2
- > HashingAssignment.java src/main/java/org/owasp/webgoat/lessons/cryptography 1
- > SSRFTask2.java src/main/java/org/owasp/webgoat/lessons/ssrf 1
- > CommentsCache.java src/main/java/org/owasp/webgoat/lessons/xxe 1
- > UserService.java src/main/java/org/owasp/webgoat/container/users 1
- > Assignment5.java src/main/java/org/owasp/webgoat/lessons/challenges/challenge5 1
- > JWTHeaderKIDEndpoint.java src/main/java/org/owasp/webgoat/lessons/jwt/claimmisuse 1
- > SqlInjectionChallenge.java src/main/java/org/owasp/webgoat/lessons/sqlinjection/advanced 1
- > SqlInjectionLesson10.java src/main/java/org/owasp/webgoat/lessons/sqlinjection/introduction 1
- > SqlInjectionLesson2.java src/main/java/org/owasp/webgoat/lessons/sqlinjection/introduction 1
- > SqlInjectionLesson3.java src/main/java/org/owasp/webgoat/lessons/sqlinjection/introduction 1
- > SqlInjectionLesson4.java src/main/java/org/owasp/webgoat/lessons/sqlinjection/introduction 1
- > SqlInjectionLesson5.java src/main/java/org/owasp/webgoat/lessons/sqlinjection/introduction 1
- > SqlInjectionLesson5a.java src/main/java/org/owasp/webgoat/lessons/sqlinjection/introduction 1
- > SqlInjectionLesson5b.java src/main/java/org/owasp/webgoat/lessons/sqlinjection/introduction 1
- > SqlInjectionLesson9.java src/main/java/org/owasp/webgoat/lessons/sqlinjection/introduction 1
- > Servers.java src/main/java/org/owasp/webgoat/lessons/sqlinjection/mitigation 1
- > WebSecurityConfig.java src/main/java/org/owasp/webgoat/container 1
- > WebSecurityConfig.java src/main/java/org/owasp/webgoat/webwolf 1
- > JWTLessonIntegrationTest.java src/it/java/org/owasp/webgoat 1
- > InsecureDeserializationTask.java src/main/java/org/owasp/webgoat/lessons/deserialization 1

CodeQL vs. SonarQube

: CodeQL과 SonarQube의 정량적 비교분석을 위한 테스트 구현

SonarQube

sonarqube_setup.sh(환경구축코드)

```
mkdir /home/codevuln/sonarqube/
mkdir /home/codevuln/target-repo/

echo -e "\033[32m[+] SonarQube Install & Setting\033[0m $@"
cd /home/codevuln/sonarqube
# SonarQube 및 SonarScanner가 설치되어 있는지 확인
if ! [ -d "/home/codevuln/sonarqube/sonarqube" ]; then
    # SonarQube 설치
    cd /home/codevuln/sonarqube
    sudo wget https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-10.4.1.88267.zip
    sudo unzip sonarqube-10.4.1.88267.zip
    sudo mv /home/codevuln/sonarqube/sonarqube-10.4.1.88267 /home/codevuln/sonarqube/sonarqube
    sudo chmod 777 /home/codevuln/sonarqube/sonarqube
    sudo rm -r sonarqube-10.4.1.88267.zip

    # 'sonar' 사용자 추가 및 권한 설정
    sudo adduser --system --no-create-home --group sonar
    sudo chown -R sonar:sonar /home/codevuln/sonarqube/sonarqube
else
    echo "SonarQube is already installed."
fi

# SonarQube 시작
sudo -u sonar /home/codevuln/sonarqube/sonarqube/bin/linux-x86-64/sonar.sh start
sudo -u sonar /home/codevuln/sonarqube/sonarqube/bin/linux-x86-64/sonar.sh status
sleep 3

if ! [ -d "/home/codevuln/sonarqube/sonarscanner" ]; then
    # SonarScanner 설치
    cd /home/codevuln/sonarqube
    sudo wget https://binaries.sonarsource.com/Distribution/sonar-scanner-cli/sonar-scanner-cli-5.0.1.3006-linux.zip
    sudo unzip sonar-scanner-cli-5.0.1.3006-linux.zip
    sudo mv /home/codevuln/sonarqube/sonar-scanner-5.0.1.3006-linux /home/codevuln/sonarqube/sonarscanner
    sudo chmod 777 /home/codevuln/sonarqube/sonarscanner
    sudo rm -r sonar-scanner-5.0.1.3006-linux.zip
else
    echo "SonarScanner is already installed."
fi
echo -e "\033[32m[+] SonarQube Install & Setting complete\033[0m $@"

python3 <<END
from sonarqube import SonarQubeClient
```

sonarqube_analysis.sh(실행코드)

```
#!/bin/bash

directory_name=$1
clone_directory_name=$2

# 토큰 파일에서 토큰 값을 읽어옴
token=$(tail -n 1 token.txt)
rm -r ./token.txt

# SonarScanner 실행
/home/codevuln/sonarqube/sonarscanner/bin/sonar-scanner -X \
-Dsonar.projectKey="$directory_name" \
-Dsonar.sources="/home/codevuln/target-repo/$directory_name/$clone_directory_name" \
-Dsonar.java.binaries=/home/codevuln/sonarqube/sonarscanner/jre/bin \
-Dsonar.host.url="http://localhost:9000" \
-Dsonar.login="$token"

echo "SonarScanner has finished scanning."
echo "Waiting for SonarQube to process the results..."
sleep 180 # 3분 대기

export directory_name
echo "Checking project key: $directory_name"

# 특정 프로젝트의 이슈들 검색 및 CSV 및 JSON 파일 경로 설정
python3 <<END
import time
import csv
import json
import os
from datetime import datetime
from sonarqube import SonarQubeClient

directory_name = os.getenv('directory_name')
print("Start checking SonarQube issues...")
print("Project key:", directory_name)

url = "http://localhost:9000"
username = "admin"
password = "admin"

sonar = SonarQubeClient(sonarqube_url=url, username=username, password=password)
```

| CodeQL vs. SonarQube

: CodeQL과 SonarQube의 정량적 비교분석을 위한 테스트 구현

◆ SonarQube

CSV 파일

	A	B	C	D	E	F	G	H	I	J	K
10	SonarQub	2024-10-08	23:50:42 java:S1192	CRITICAL	Define a c WebGoat:target-repo/WebGoat/WebGoat-repo/src/it/java/org/owasp/webgoat/CSRFIntegrationTest.java			119	20	119	32
11	SonarQub	2024-10-08	23:50:42 java:S1192	CRITICAL	Define a c WebGoat:target-repo/WebGoat/WebGoat-repo/src/it/java/org/owasp/webgoat/CSRFIntegrationTest.java			139	20	139	29
12	SonarQub	2024-10-08	23:50:42 java:S1192	CRITICAL	Define a c WebGoat:target-repo/WebGoat/WebGoat-repo/src/it/java/org/owasp/webgoat/CSRFIntegrationTest.java			139	42	139	59
13	SonarQub	2024-10-08	23:50:42 java:S1192	CRITICAL	Define a c WebGoat:target-repo/WebGoat/WebGoat-repo/src/it/java/org/owasp/webgoat/CSRFIntegrationTest.java			212	15	212	25
14	SonarQub	2024-10-08	23:50:42 java:S1192	CRITICAL	Define a c WebGoat:target-repo/WebGoat/WebGoat-repo/src/it/java/org/owasp/webgoat/ChallengeIntegrationTest.java			24	20	24	32
15	SonarQub	2024-10-08	23:50:42 java:S1192	CRITICAL	Define a c WebGoat:target-repo/WebGoat/WebGoat-repo/src/it/java/org/owasp/webgoat/ChallengeIntegrationTest.java			53	24	53	40
16	SonarQub	2024-10-08	23:50:42 java:S1192	CRITICAL	Define a c WebGoat:target-repo/WebGoat/WebGoat-repo/src/it/java/org/owasp/webgoat/CryptoIntegrationTest.java			54	20	54	32
17	SonarQub	2024-10-08	23:50:42 java:S1192	CRITICAL	Define a c WebGoat:target-repo/WebGoat/WebGoat-repo/src/it/java/org/owasp/webgoat/GeneralLessonIntegrationTest.java			44	20	44	32
18	SonarQub	2024-10-08	23:50:42 java:S1192	CRITICAL	Define a c WebGoat:target-repo/WebGoat/WebGoat-repo/src/it/java/org/owasp/webgoat/IDORIntegrationTest.java			52	20	52	32
19	SonarQub	2024-10-08	23:50:42 java:S1192	CRITICAL	Define a c WebGoat:target-repo/WebGoat/WebGoat-repo/src/it/java/org/owasp/webgoat/IntegrationTest.java			59	23	59	33
20	SonarQub	2024-10-08	23:50:42 java:S1192	CRITICAL	Define a c WebGoat:target-repo/WebGoat/WebGoat-repo/src/it/java/org/owasp/webgoat/IntegrationTest.java			60	23	60	33
21	SonarQub	2024-10-08	23:50:42 java:S1192	CRITICAL	Define a c WebGoat:target-repo/WebGoat/WebGoat-repo/src/it/java/org/owasp/webgoat/IntegrationTest.java			61	22	61	29
22	SonarQub	2024-10-08	23:50:42 java:S1192	CRITICAL	Define a c WebGoat:target-repo/WebGoat/WebGoat-repo/src/it/java/org/owasp/webgoat/IntegrationTest.java			63	20	63	32
23	SonarQub	2024-10-08	23:50:42 java:S1192	CRITICAL	Define a c WebGoat:target-repo/WebGoat/WebGoat-repo/src/it/java/org/owasp/webgoat/IntegrationTest.java			106	20	106	36

| CodeQL vs. SonarQube

: CodeQL과 SonarQube의 정량적인 비교 분석 지표

	CodeQL	SonarQube
Severity (심각도) 분류	0~10 점수로 구분	Critical, Major, Minor, Info 총 4단계로 구분
분석 소요 시간	약 5분	약 1분 40초
총 탐지된 취약점 개수	약 50개	약 100개
탐지된 주요 취약점	<ul style="list-style-type: none">- XPath Injection- SSRF- SQL Injection- Zip Slip	<ul style="list-style-type: none">-XPath Injection- SSRF-SQL Injection-Zip Slip-상수명 규칙 미준수-중복된 리터럴 사용

| CodeQL vs. SonarQube

: CodeQL과 SonarQube의 정량적인 비교 분석 지표



VS

sonarQube

| CodeQL vs. SonarQube

: CodeQL과 SonarQube의 정량적인 비교 분석 지표



VS

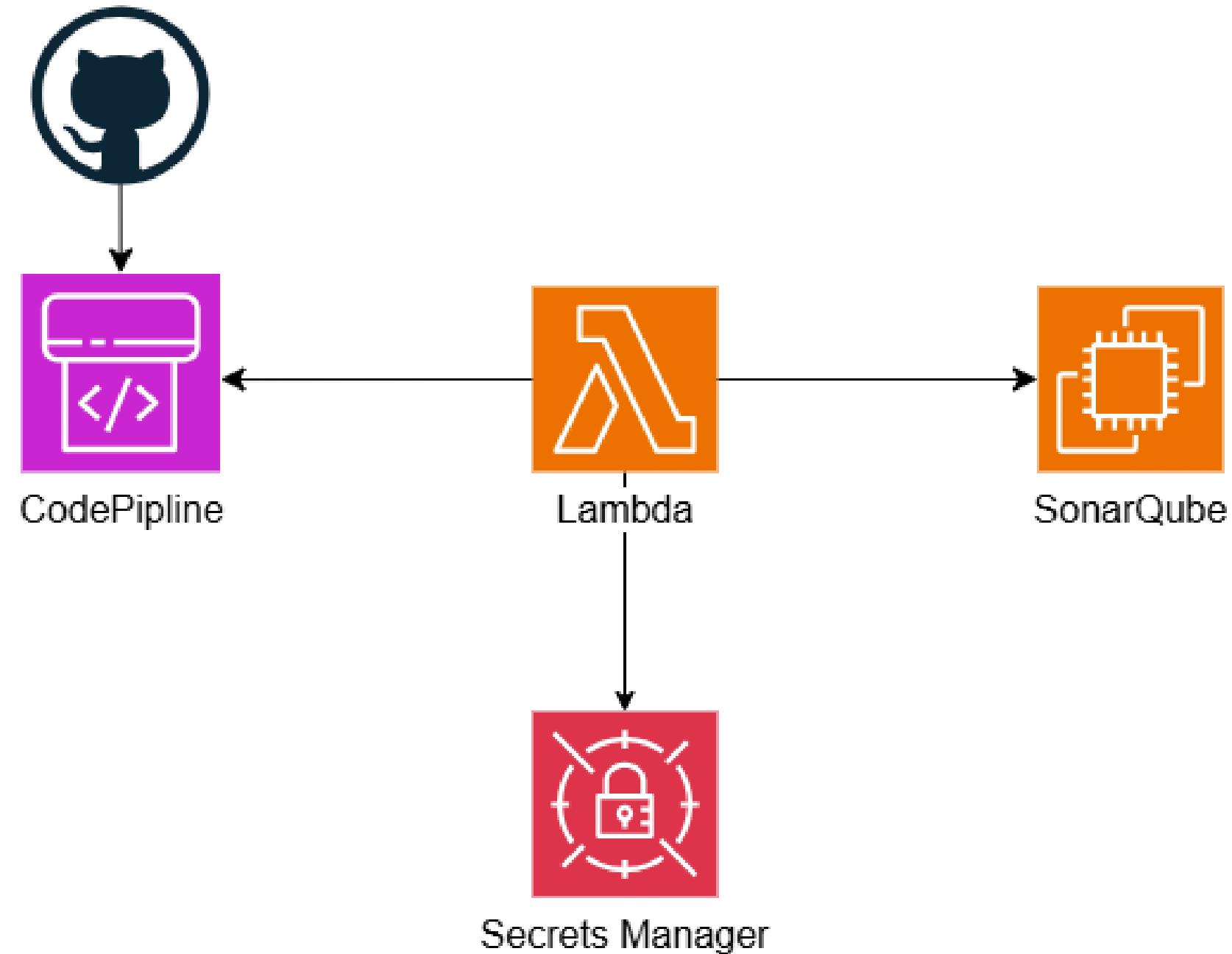


- 다양한 언어 지원
- 실시간 분석 지원
- 보안 취약점 뿐만 아니라 코드 품질 모두 분석
- 다양한 CI/CD 도구 통합 가능성

| SAST CodePipeline 통합

: 최종 선정한 SonarQube를 CodePipeline에 통합한 자동화 구현

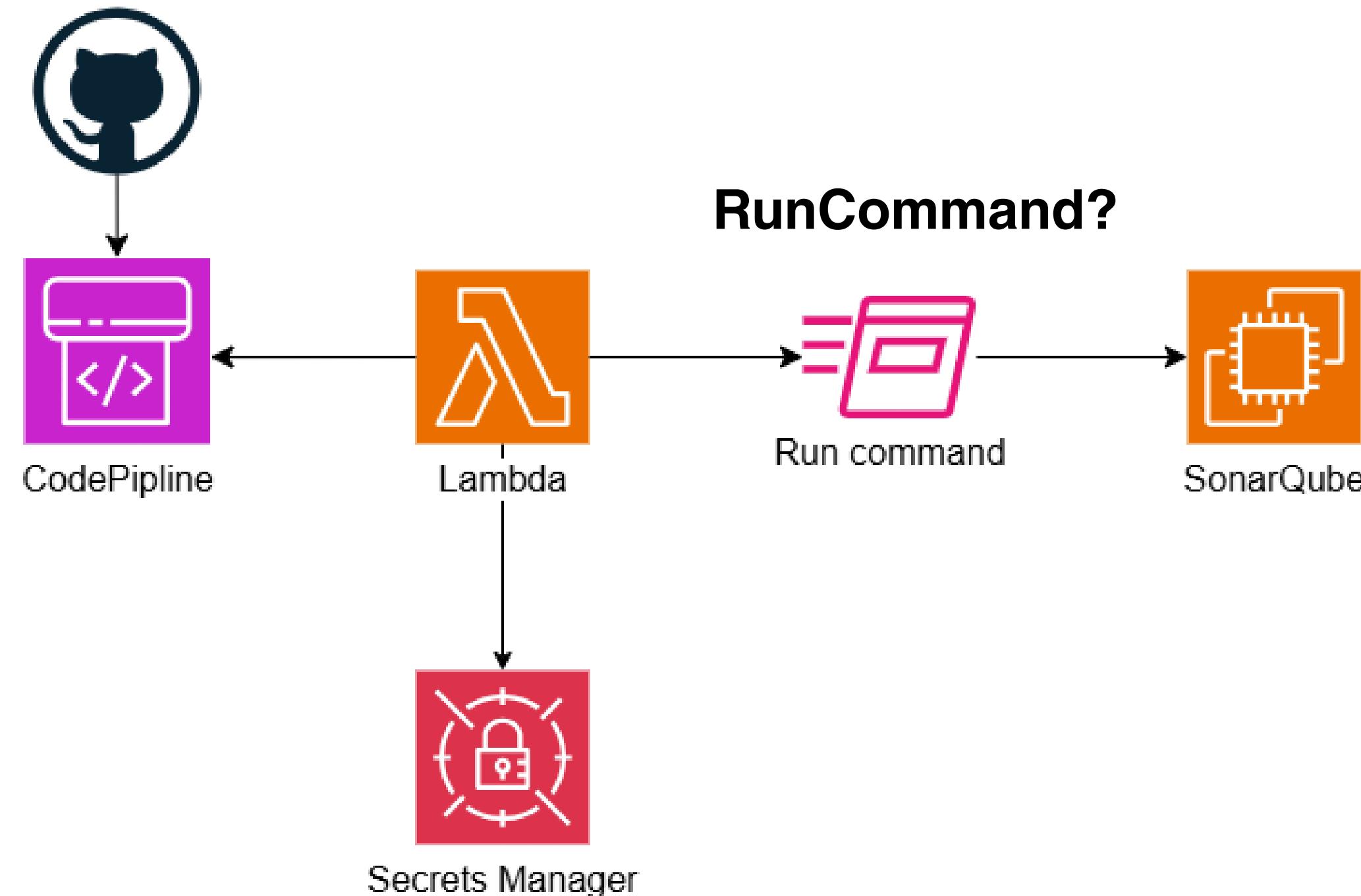
◆ SonarQube



| SAST CodePipeline 통합

: 최종 선정한 SonarQube를 CodePipeline에 통합한 자동화 구현

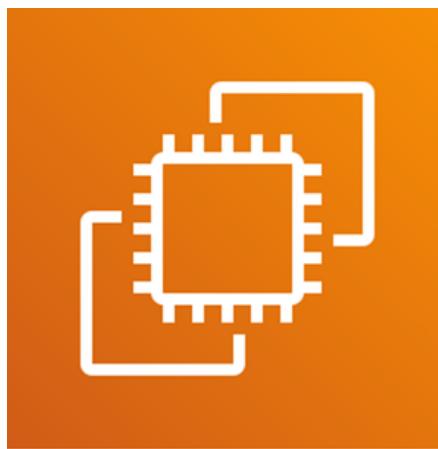
◆ SonarQube



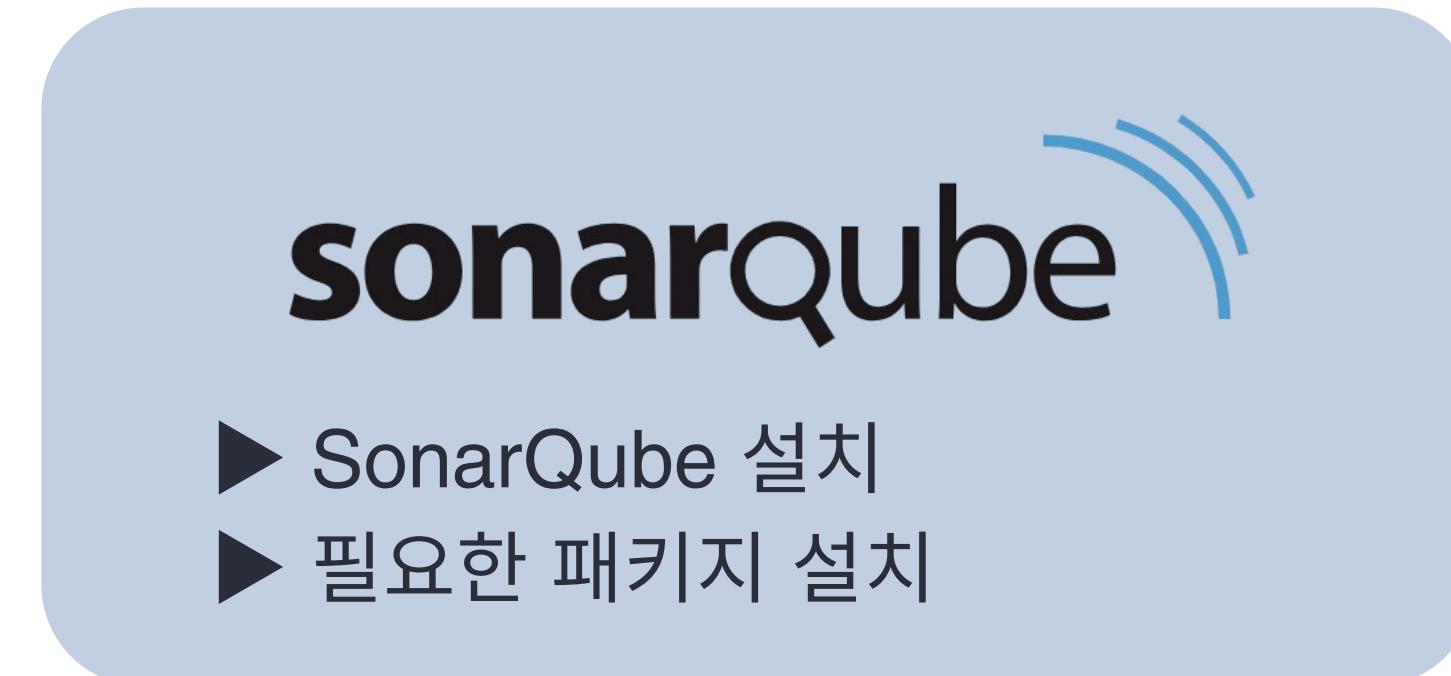
| SAST CodePipeline 통합

: 최종 선정한 SonarQube를 CodePipeline에 통합한 자동화 구현

◆ SonarQube



Amazon EC2

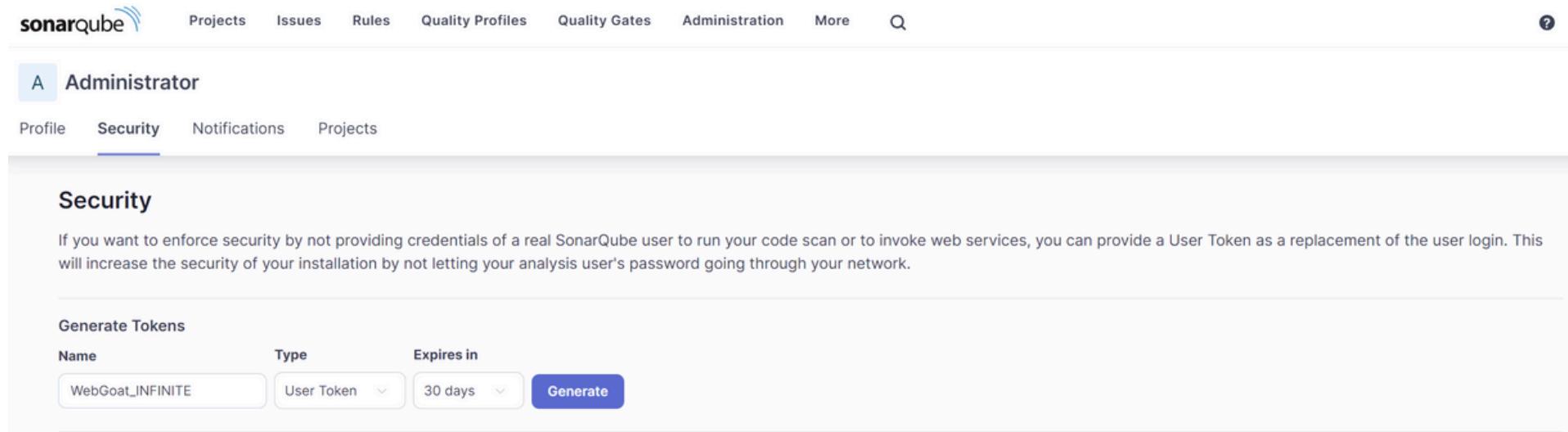


SAST CodePipeline 통합

: 최종 선정한 SonarQube를 CodePipeline에 통합한 자동화 구현

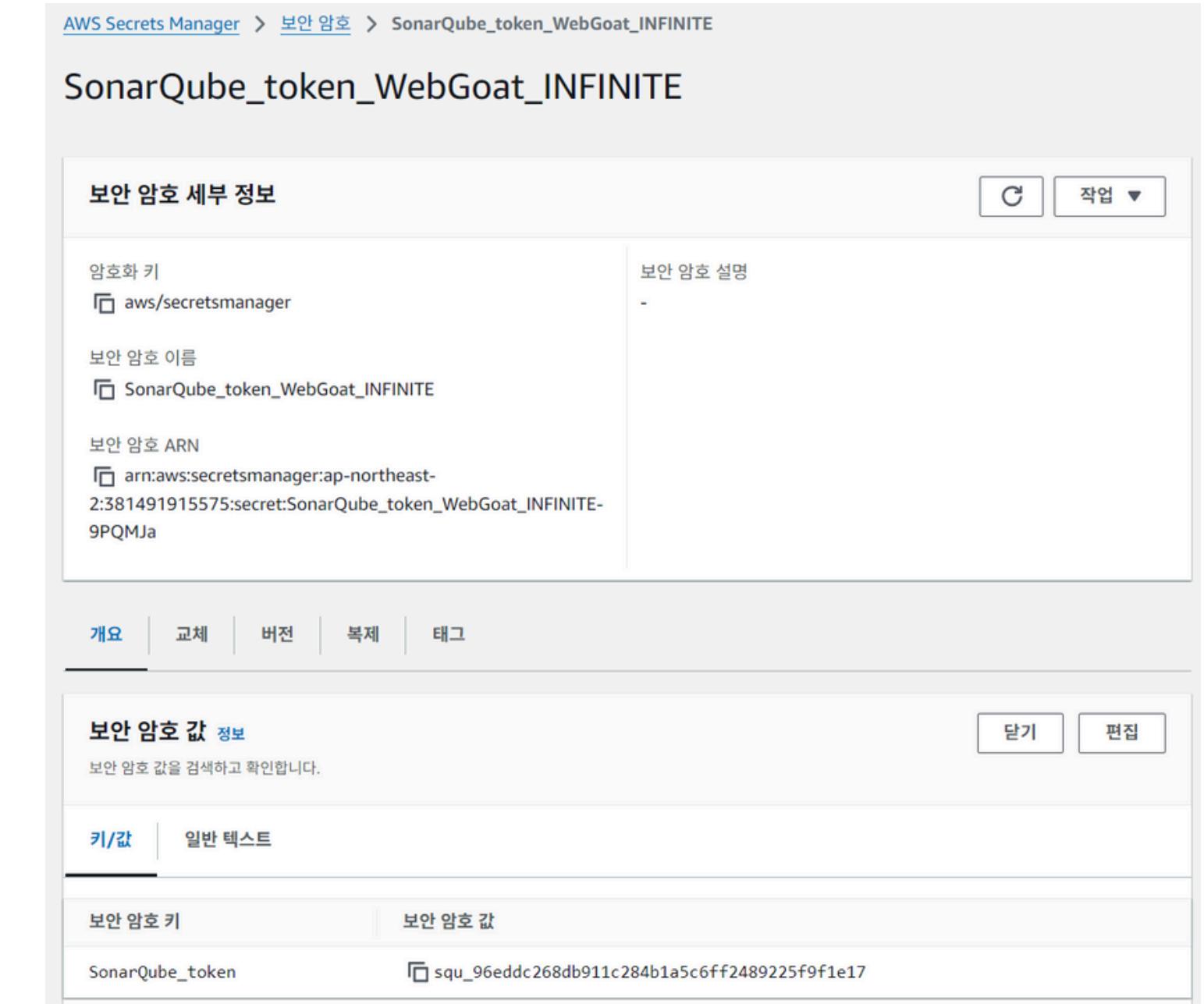
SonarQube

SonarQube 서버에 접속하여 User Token Type의 토큰 생성



The screenshot shows the SonarQube Administration interface. In the top navigation bar, 'Administration' is selected. Under 'Profile', 'Security' is selected. In the 'Security' section, there is a sub-section titled 'Generate Tokens'. A table is present with one row: 'Name' (WebGoat_INFINITE), 'Type' (User Token), and 'Expires in' (30 days). A 'Generate' button is at the bottom of the table.

Token을 Secrets Manager에 저장



The screenshot shows the AWS Secrets Manager console. The URL is 'AWS Secrets Manager > 보안 암호 > SonarQube_token_WebGoat_INFINITE'. The secret name is 'SonarQube_token_WebGoat_INFINITE'. The '보안 암호 세부 정보' (Security Secret Details) section shows the following fields: '암호화 키' (aws/secretsmanager), '보안 암호 이름' (SonarQube_token_WebGoat_INFINITE), and '보안 암호 ARN' (arn:aws:secretsmanager:ap-northeast-2:381491915575:secret:SonarQube_token_WebGoat_INFINITE-9PQMJa). At the bottom, there are tabs for '개요' (Overview), '교체' (Replace), '버전' (Versions), '복제' (Copy), and '태그' (Tags). The '보안 암호 값' (Security Secret Value) section shows the secret value: 'SonarQube_token' and 'squ_96eddc268db911c284b1a5c6ff2489225f9f1e17'.

| SAST CodePipeline 통합

: 최종 선정한 SonarQube를 CodePipeline에 통합한 자동화 구현

◆ SonarQube



Layer 추가

- requests Layer 추가 - zip 다운로드 방식
- sonar-scanner Layer 추가 - zip 다운로드 방식
- git Layer 추가 - arn 방식

계층 정보				
병합 주문	이름	계층 버전	호환 런타임	호환 아키텍처
1	requests-layer-jw	1	python3.12	x86_64
2	sonar-scanner-ys	2	python3.12	x86_64
3	git-lambda2	8	-	-

| SAST CodePipeline 통합

: 최종 선정한 SonarQube를 CodePipeline에 통합한 자동화 구현

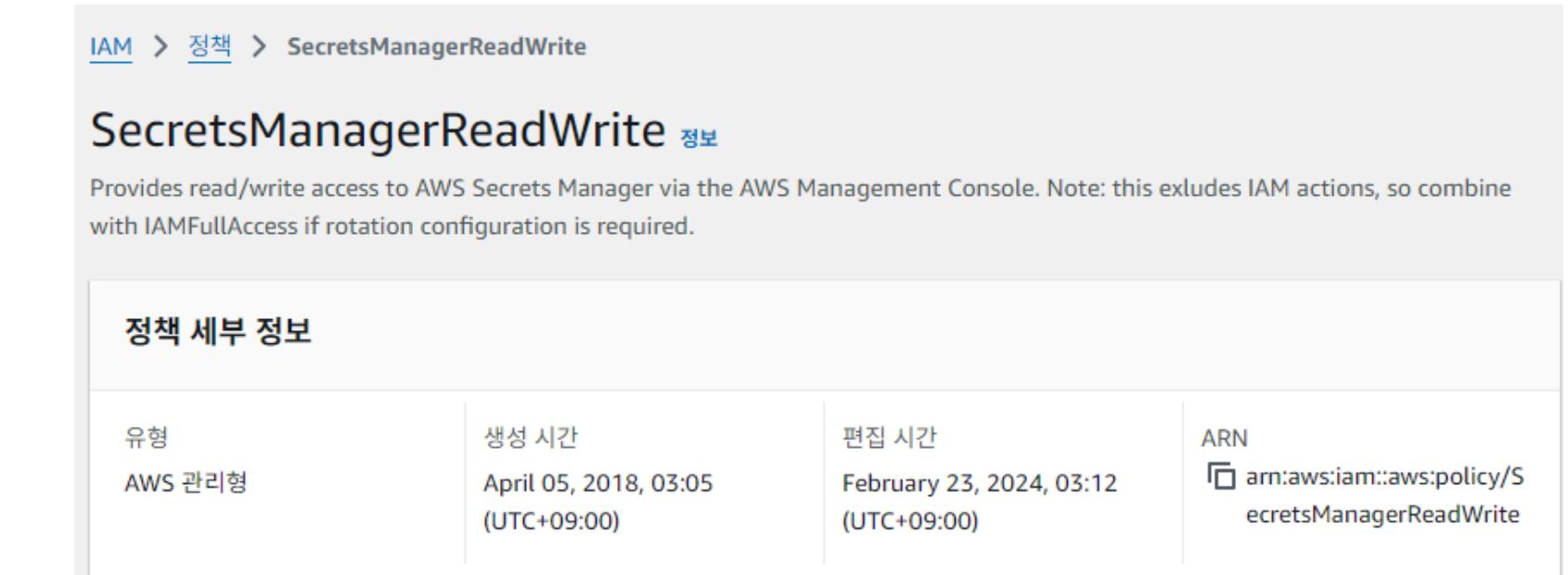
◆ SonarQube

IAM 역할 생성

AWS CodePipeline에서 작업 실패 또는 성공 결과를 기록

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": [  
        "codepipeline:PutJobFailureResult",  
        "codepipeline:PutJobSuccessResult"  
      ],  
      "Resource": "*"  
    }  
  ]  
}
```

AWS Secrets Manager에 대한 읽기 및 쓰기 권한



| SAST CodePipeline 통합

: 최종 선정한 SonarQube를 CodePipeline에 통합한 자동화 구현

◆ SonarQube



Lambda

Step 1.

Secrets Manager에서 SonarQube 토큰 가져오기

```
def get_sonar_token():
    try:
        # Secrets Manager에서 SonarQube 토큰 가져오기
        secret_name = "SonarQube_token_WebGoat_INFINITE"
        response = secrets_client.get_secret_value(SecretId=secret_name)
        secret = json.loads(response['SecretString'])
        token = secret.get('SonarQube_token')
        if token is None:
            raise ValueError("'SonarQube_token' key not found in secret.")
        return token
    except Exception as e:
        print(f"Error retrieving token from Secrets Manager: {str(e)}")
        return None
```

| SAST CodePipeline 통합

: 최종 선정한 SonarQube를 CodePipeline에 통합한 자동화 구현

◆ SonarQube

Step 2.



Lambda

SonarQube 프로젝트가 존재하는지 확인하고, 없으면 새로 생성

```
# 프로젝트 존재 여부 확인
existing_projects_response = requests.get(f"{url}/api/projects/search", headers=headers)
if existing_projects_response.status_code == 200:
    existing_projects = existing_projects_response.json().get('components', [])
    if any(project['key'] == project_name for project in existing_projects):
        print(f"Project '{project_name}' already exists.")
        return project_name
    else:
        # 프로젝트가 없으면 새로 생성
        print(f"Project '{project_name}' does not exist. Creating new project...")
        data = {
            "name": project_name,
            "project": project_name,
            "visibility": "private"
        }
        create_response = requests.post(f"{url}/api/projects/create", headers=headers, data=data)
```

| SAST CodePipeline 통합

: 최종 선정한 SonarQube를 CodePipeline에 통합한 자동화 구현

◆ SonarQube



Lambda

Step 3.

GitHub 리포지토리 클론

```
# GitHub에서 WebGoat 리포지토리 클론
git_url = "https://github.com/J1vvoo/WebGoat.git"
subprocess.run(["git", "clone", git_url, webgoat_directory], check=True)
```

| SAST CodePipeline 통합

: 최종 선정한 SonarQube를 CodePipeline에 통합한 자동화 구현

◆ SonarQube



Lambda

Step 4.

SonarScanner로 SonarQube 분석 실행

서브프로세스 모듈

```
# SonarScanner 실행
subprocess.run(command, env=env, check=True)
print("SonarScanner has finished scanning.")
```

| SAST CodePipeline 통합

: 최종 선정한 SonarQube를 CodePipeline에 통합한 자동화 구현

◆ SonarQube



Step 5.

CodePipeline 성공 또는 실패 결과 전송

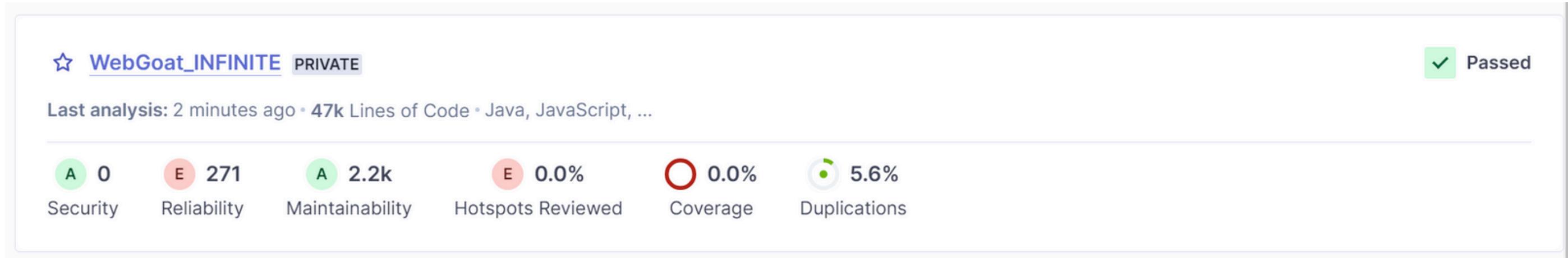
```
def lambda_handler(event, context):
    # CodePipeline 실패 결과 전송
    if 'CodePipeline.job' in event:
        codepipeline = boto3.client('codepipeline')
        codepipeline.put_job_failure_result(
            jobId=event['CodePipeline.job']['id'],
            failureDetails={'message': str(e), 'type': 'JobFailed'}
        )
```

| SAST CodePipeline 통합

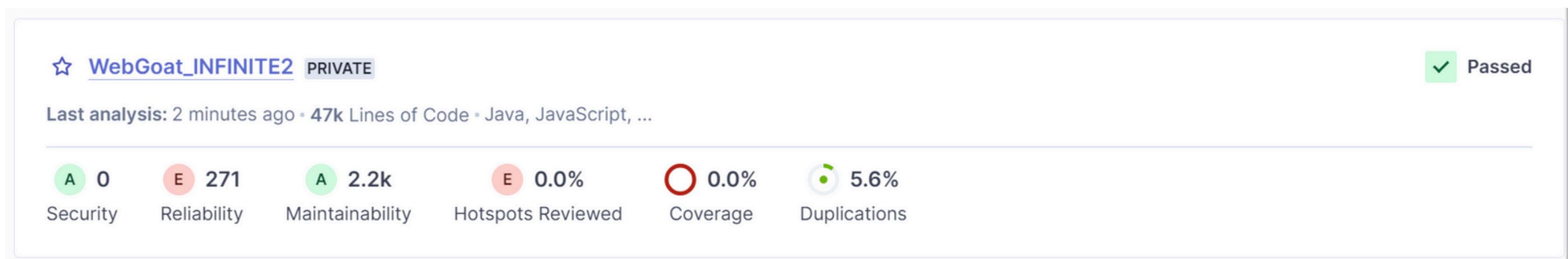
: 최종 선정한 SonarQube를 CodePipeline에 통합한 자동화 구현

◆ SonarQube

▶ 기존 프로젝트에 분석을 진행할 경우, 생성 없이 동일한 프로젝트에서 분석 진행



▶ 새로운 프로젝트에 분석을 진행할 경우, 프로젝트 생성 후 토큰은 기존 것으로 사용하여 분석 진행

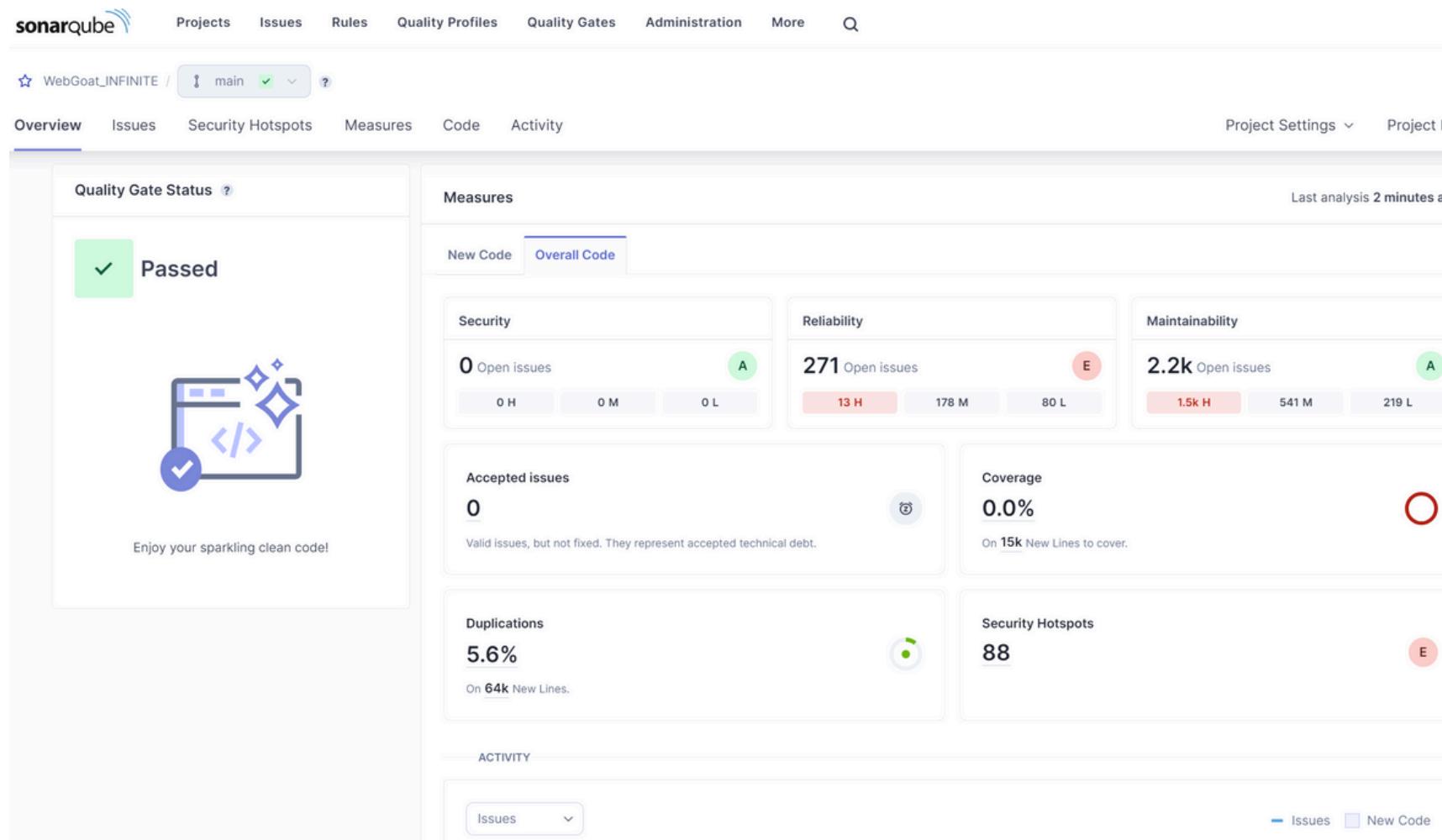


SAST CodePipeline 통합

: 최종 선정한 SonarQube를 CodePipeline에 통합한 자동화 구현

SonarQube

► SonarQube에서 분석 결과 확인 가능



sonarqube

Projects Issues Rules Quality Profiles Quality Gates Administration More

Overview Issues Security Hotspots Measures Code Activity

Quality Gate Status ?

Passed

Measures

New Code Overall Code

Security 0 Open issues A

Reliability 271 Open issues E

Maintainability 2.2k Open issues A

Accepted issues 0

Coverage 0.0%

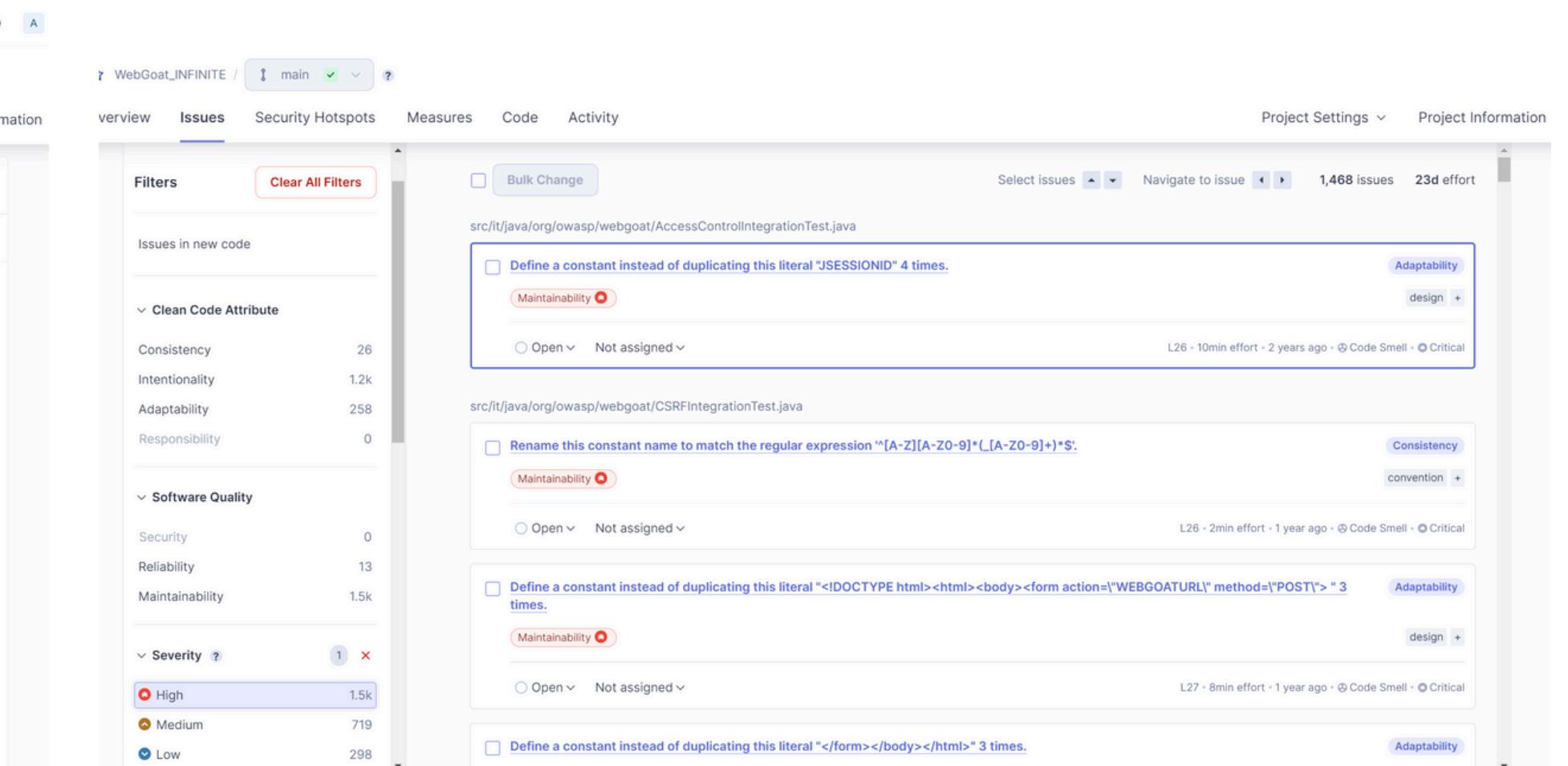
Duplications 5.6%

Security Hotspots 88

Last analysis 2 minutes ago

ACTIVITY

Issues New Code



sonarqube

Projects Issues Rules Quality Profiles Quality Gates Administration More

Overview Issues Security Hotspots Measures Code Activity

Issues in new code

Filters Clear All Filters

Bulk Change

Select issues Navigate to issue 1,468 issues 23d effort

src/it/java/org/owasp/webgoat/AccessControllIntegrationTest.java

Define a constant instead of duplicating this literal "JSESSIONID" 4 times. Maintainability

src/it/java/org/owasp/webgoat/CSRFIntegrationTest.java

Rename this constant name to match the regular expression ^[A-Z][A-Z0-9]*_[A-Z0-9]+\$. Consistency

src/it/java/org/owasp/webgoat/CSRFIntegrationTest.java

Define a constant instead of duplicating this literal <!DOCTYPE html><html><body><form action="WEBGOATURL" method="POST"> " 3 times. Maintainability

src/it/java/org/owasp/webgoat/CSRFIntegrationTest.java

Define a constant instead of duplicating this literal </form></body></html> " 3 times. Maintainability

Project Settings Project Information

| SCA 분석

: SCA 분석의 개요

◆ SCA 분석이란?

SCA (Software Composition Analysis)는 오픈 소스 구성 요소를 관리하기 위한 애플리케이션 보안 방법론

- 관련된 모든 구성 요소, 지원 라이브러리, 직간접적 종속성 검색
- 소프트웨어 라이센스, 더 이상 사용되지 않는 종속성, 취약점 및 잠재적인 악용도 감지 가능
- SBOM을 생성하여 프로젝트 소프트웨어 자산 전체 인벤토리를 제공하는 프로세스

◆ SCA 분석 도구 종류



Snyk vs. Dependency-Track

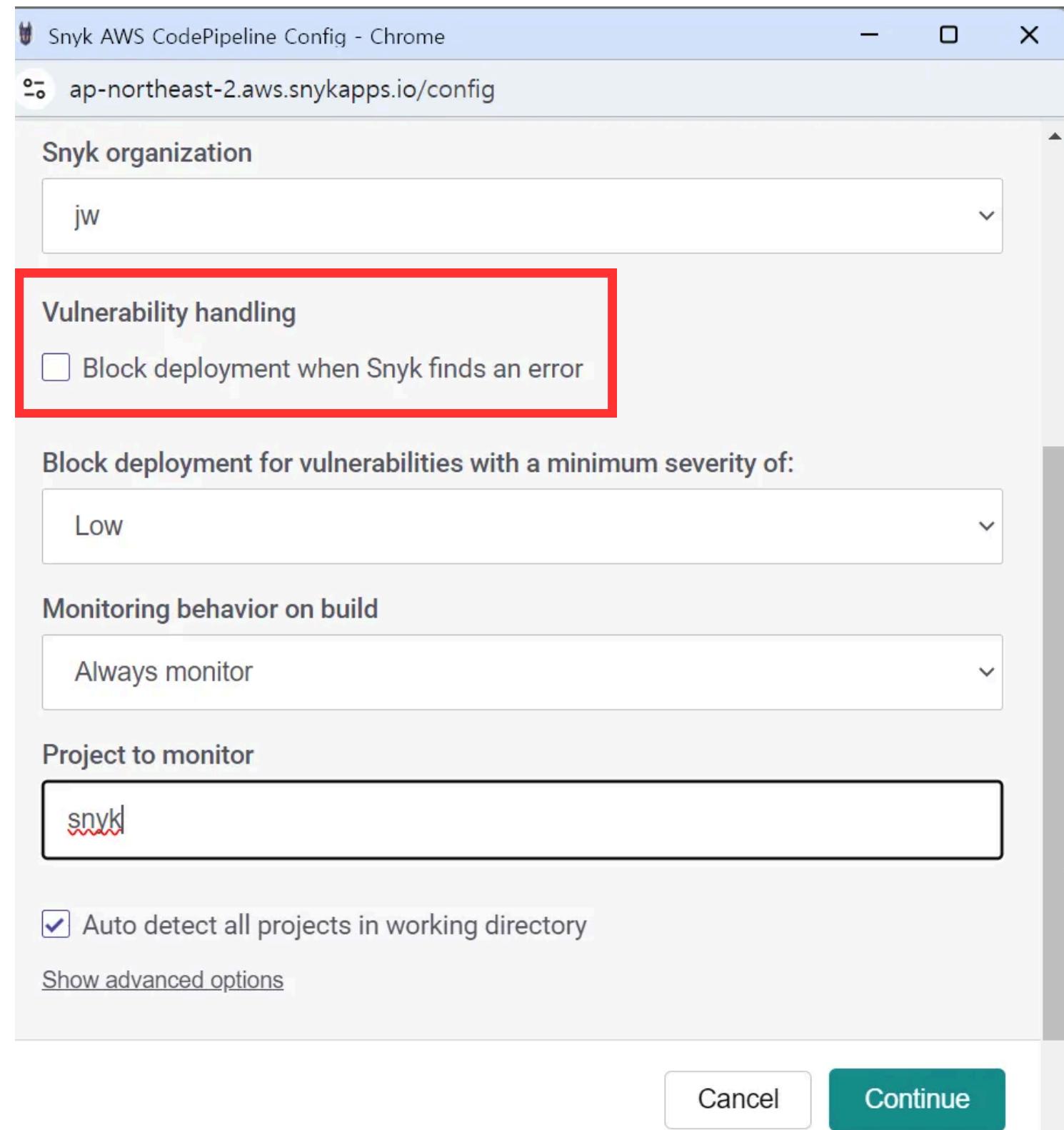
: Snyk과 Dependency-Track의 정성적인 비교 분석 지표

	Snyk	Dependency-Track
주요 초점 대상	<ul style="list-style-type: none">새로운 취약점에 대한 대응이 빠르고 지속적인 모니터링 제공개발자 중심, 코드베이스에서의 취약점 탐지 및 자동 수정	<ul style="list-style-type: none">보안 오케스트레이션, SBOM 기반 취약점 관리 및 추적해당 구성 요소가 의존하는 하위 라이브러리까지 취약점 탐지
취약점 탐지 범위	CVE 및 Github나 NPM 등과 같은 오픈소스 라이브러리 기반 취약점을 통해 취약점 정보를 제공하고 패키지 관리 수행	SBOM 구성요소를 비롯하여 NVD, OSS Index, GitHub Security Advisories 등 다양한 취약점 피드에서 정보를 가져옴.
지원 언어 및 패키지 관리 시스템	JavaScript, Java, .NET, Python, Golang, Swift, Objective-C, Scala, Ruby, PHP 및 Bazel 등 다양한 언어를 지원	SBOM 표준 형식 중 CycloneDX 형식으로 작성된 SBOM 파일을 대상으로 분석 수행
다른 도구와의 통합	GitHub, GitLab, Heroku, AWS Lambda, Bitbucket, Azure DevOps 등 다양한 CI/CD 도구 및 플랫폼 과 원활한 통합 지원	OPENAPI, Jenkins와 같은 API 통합 지원 , slack, microsoft teams 등 알림을 제공해주는 다양한 플랫폼과의 통합도 지원함
성능 및 효율성	빠른 분석 속도와 효율적인 자동화 제공, 실시간 모니터링 및 알림 기능을 제공하여 보안 취약점이 발견되면 즉시 알림	새로운 SBOM을 프로젝트에 제출할 때마다, Dependency-Track은 자동으로 취약점 스캔을 실행.
라이선스 비용	무료 플랜과 유료플랜(팀, 기업)으로 나누어지며 Snyk 무료 버전은 한달에 200번 만 테스트 가능	무료 오픈소스

| Snyk vs. Dependency-Track

: Snyk과 Dependency-Track의 정량적 비교분석을 위한 테스트 구현

◆ Snyk



| Snyk vs. Dependency-Track

: Snyk과 Dependency-Track의 정량적 비교분석을 위한 테스트 구현

◆ Snyk



작업 편집

작업 이름
작업의 이름을 선택합니다.
Snyk

작업 공급자
Snyk

입력 아티팩트
이 작업의 입력 아티팩트를 선택합니다. [자세히 알아보십시오.](#)

SourceArtifact

100자 이하

Snyk에 연결
재연결

✓ 공급자를 사용해 작업을 성공적으로 구성했습니다.

Source

파이프라인 실행 ID: [26a875cf-5729-418a-a836-f98da4c6ed3d](#)

Source

GitHub(버전 2) [\[\]](#)

성공 - 16 [\[\]](#) 전

[c5b3e1ac](#) [\[\]](#)

세부 정보 보기

전환 비활성화

SCA

파이프라인 실행 ID: [26a875cf-5729-418a-a836-f98da4c6ed3d](#)

Snyk

[Snyk](#) [\[\]](#)

성공 - 13 [\[\]](#) 전

세부 정보 보기

[c5b3e1ac](#) [\[\]](#) Source: Update buildspec.yml

전환 비활성화

Build

파이프라인 실행 ID: [26a875cf-5729-418a-a836-f98da4c6ed3d](#)

Rollback 시작

Snyk vs. Dependency-Track

: Snyk과 Dependency-Track의 정량적 비교분석을 위한 테스트 구현

Snyk

Targets 2

Search targets

Snyk

1 C 37 H 24 M 1 L

J1vwoo/WebGoat

1 SEVERITY

Critical: 1

High: 37

Medium: 24

Low: 1

63 of 63 issues

Sort by highest priority score

Ready to import another project

Secure your entire stack

Add projects

PRIORITY SCORE

Scored between 0 - 1000

854

com.thoughtworks.xstream:xstream

VULNERABILITY

- Remote Code Execution (RCE)

SCORE 854

Introduced through com.thoughtworks.xstream:xstream@1.4.5

Fixed in com.thoughtworks.xstream:xstream@1.4.18

Exploit maturity MATURE

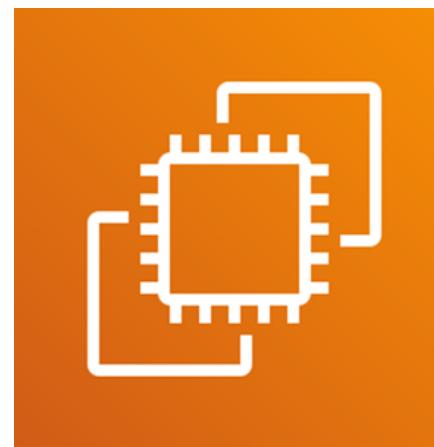
Show more detail

Learn about this type of vulnerability

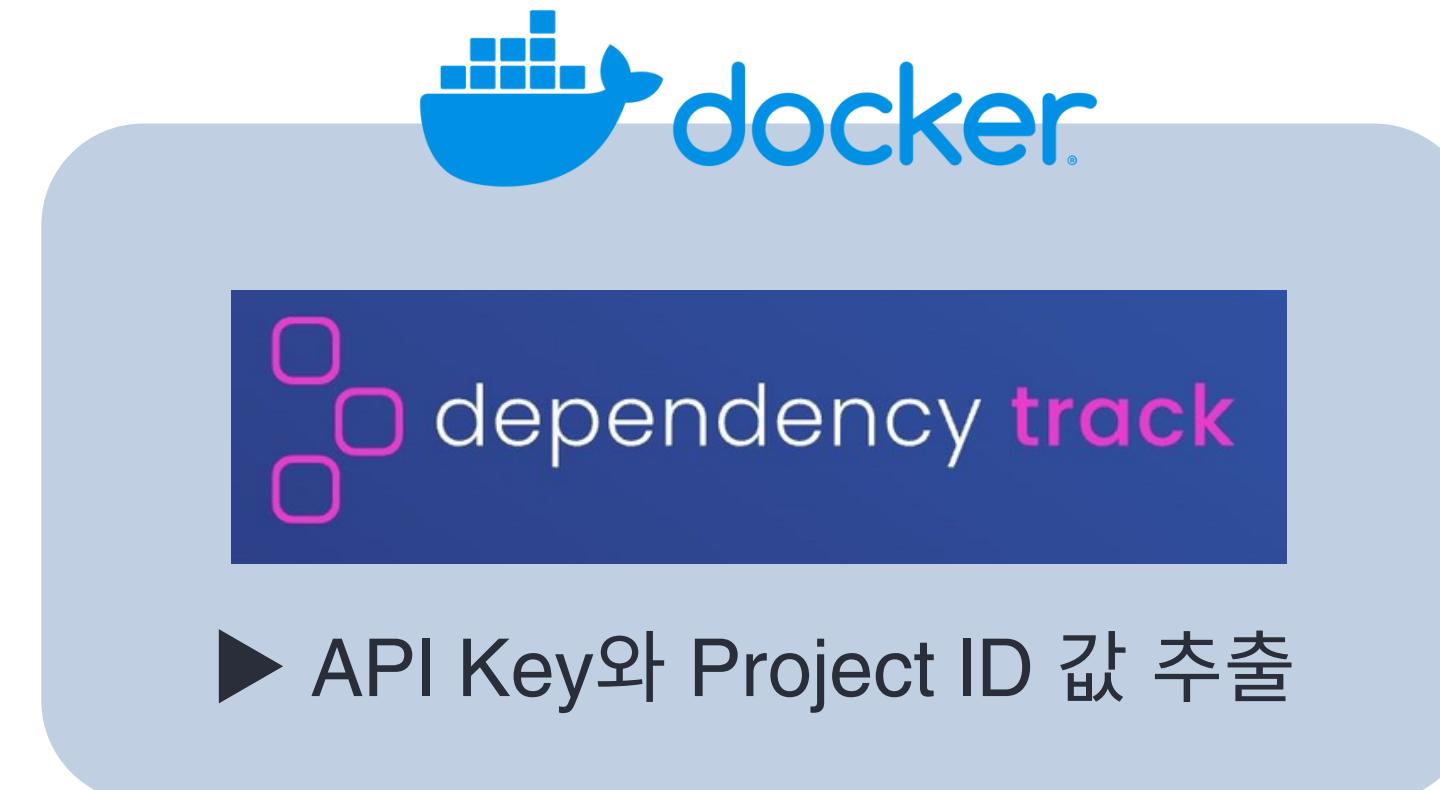
| Snyk vs. Dependency-Track

: Snyk과 Dependency-Track의 정량적 비교분석을 위한 테스트 구현

◆ Dependency-Track



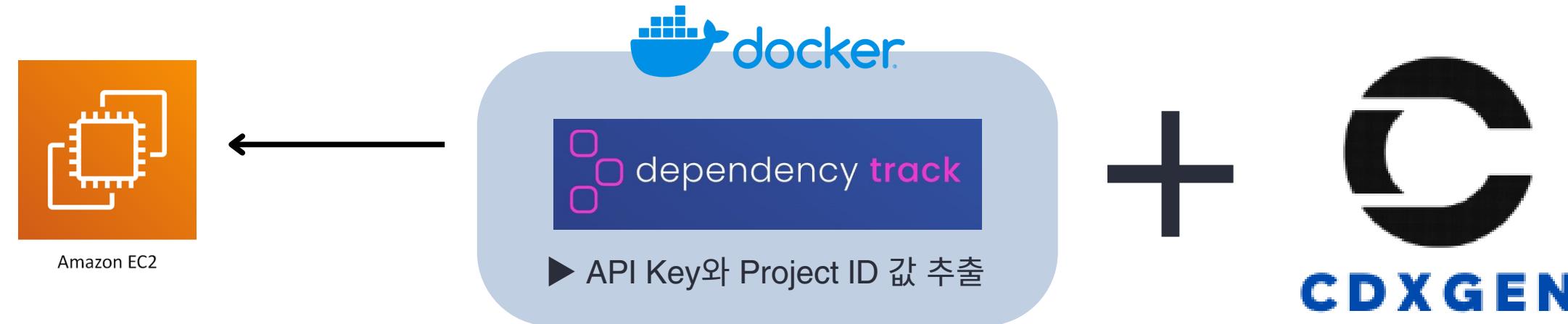
Amazon EC2



| Snyk vs. Dependency-Track

: Snyk과 Dependency-Track의 정량적 비교분석을 위한 테스트 구현

◆ Dependency-Track



Step 1.

```
docker run --name cdxgen --network="host" -d --rm -t ghcr.io/cyclonedx/cdxgen -r /app --server --server-host 0.0.0.0 --server-url <--Dependency-Track 동작 주소--> --api-key "<--Dependency-Track에서 사용하고자 하는 그룹의 API Key-->" --project-id "<--Dependency-Track에 업로드하고자 하는 프로젝트의 ID-->"
```

| Snyk vs. Dependency-Track

: Snyk과 Dependency-Track의 정량적 비교분석을 위한 테스트 구현

◆ Dependency-Track



Step 1.

```
docker run --name cdxgen --network="host" -d  
host 0.0.0.0 --server-url <--Dependency-Track  
고자 하는 그룹의 API Key--> --project-id "<--De
```

▶ 호스트 네트워크 사용 이유

Docker는 컨테이너 내부의 네트워크와 호스트 네트워크를 별도로 관리하므로, 포트 포워딩을 설정해야 외부에서 컨테이너로 접근할 수 있다. 따라서 **컨테이너 내부의 네트워크 설정을 따로 지정하지 않고도, 외부에서 컨테이너에 접근하기 위해** 사용한다.

--server --server-
ack에서 사용하
젝트의 ID-->"

| Snyk vs. Dependency-Track

: Snyk과 Dependency-Track의 정량적 비교분석을 위한 테스트 구현

◆ Dependency-Track



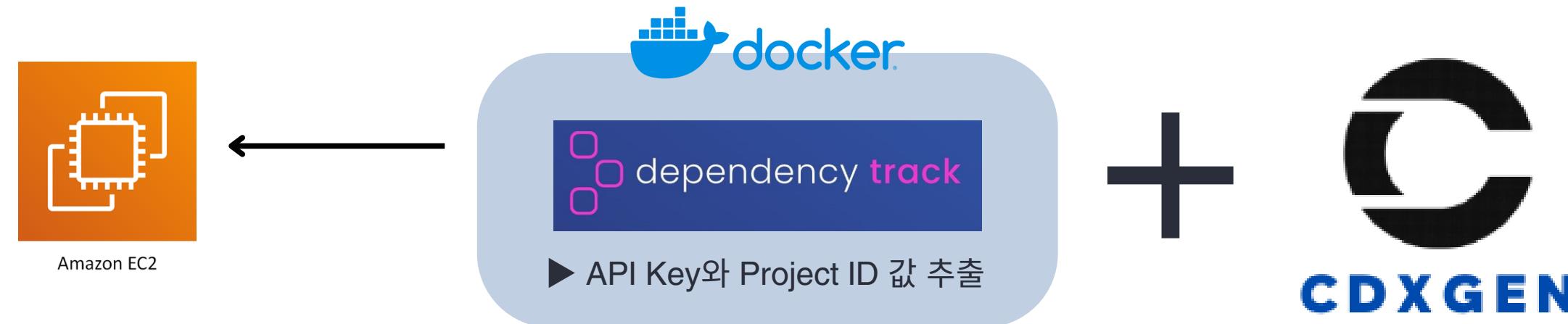
Step 1.

```
docker run --name cdxgen --network="host" -d --rm -t ghcr.io/cyclonedx/cdxgen -r /app --server --server-host 0.0.0.0 --server-url <--Dependency-Track 동작 주소--> --api-key "<--Dependency-Track에서 사용하고자 하는 그룹의 API Key-->" --project-id "<--Dependency-Track에 업로드하고자 하는 프로젝트의 ID-->"
```

| Snyk vs. Dependency-Track

: Snyk과 Dependency-Track의 정량적 비교분석을 위한 테스트 구현

◆ Dependency-Track



Step 1.

```
docker run --name cdxgen --network="host" -d --rm -t ghcr.io/cyclonedx/cdxgen -r /app --server --server-host 0.0.0.0 --server-url <--Dependency-Track 동작 주소--> --api-key "<--Dependency-Track에서 사용하고자 하는 그룹의 API Key-->" --project-id "<--Dependency-Track에 업로드하고자 하는 프로젝트의 ID-->"
```

| Snyk vs. Dependency-Track

: Snyk과 Dependency-Track의 정량적 비교분석을 위한 테스트 구현

◆ Dependency-Track



Step 1.

```
docker run --name cdxgen --network="host" -d --rm -t ghcr.io/cyclonedx/cdxgen -r /app --server --server-host 0.0.0.0 --server-url <--Dependency-Track 동작 주소--> --api-key "<--Dependency-Track에서 사용하고자 하는 그룹의 API Key-->" --project-id "<--Dependency-Track에 업로드하고자 하는 프로젝트의 ID-->"
```

| Snyk vs. Dependency-Track

: Snyk과 Dependency-Track의 정량적 비교분석을 위한 테스트 구현

◆ Dependency-Track



Step 2.

```
curl "http://localhost:9090/sbom?
```

```
url=https://github.com/J1vvoo/WebGoat.git&multiProject=true&type=java"
```

| Snyk vs. Dependency-Track

: Snyk과 Dependency-Track의 정량적 비교분석을 위한 테스트 구현

◆ Dependency-Track



Step 2.

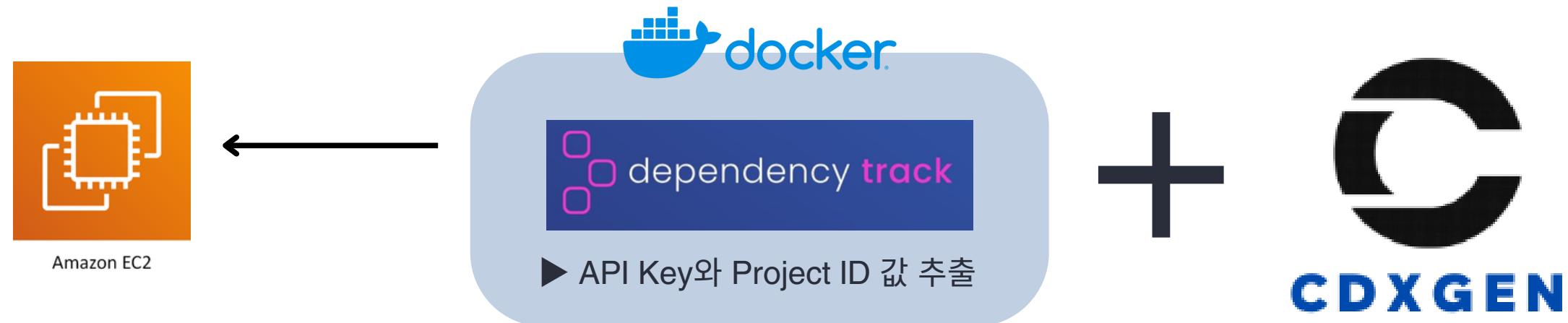
```
curl 'http://localhost:9090/sbom?
```

```
url=https://github.com/J1vvoo/WebGoat.git&multiProject=true&type=java"
```

| Snyk vs. Dependency-Track

: Snyk과 Dependency-Track의 정량적 비교분석을 위한 테스트 구현

◆ Dependency-Track



Step 2.

```
curl "http://localhost:9090/sbom?
```

```
url=https://github.com/J1vvoo/WebGoat.git&multiProject=true&type=java"
```

Snyk vs. Dependency-Track

: Snyk과 Dependency-Track의 정량적 비교분석을 위한 테스트 구현

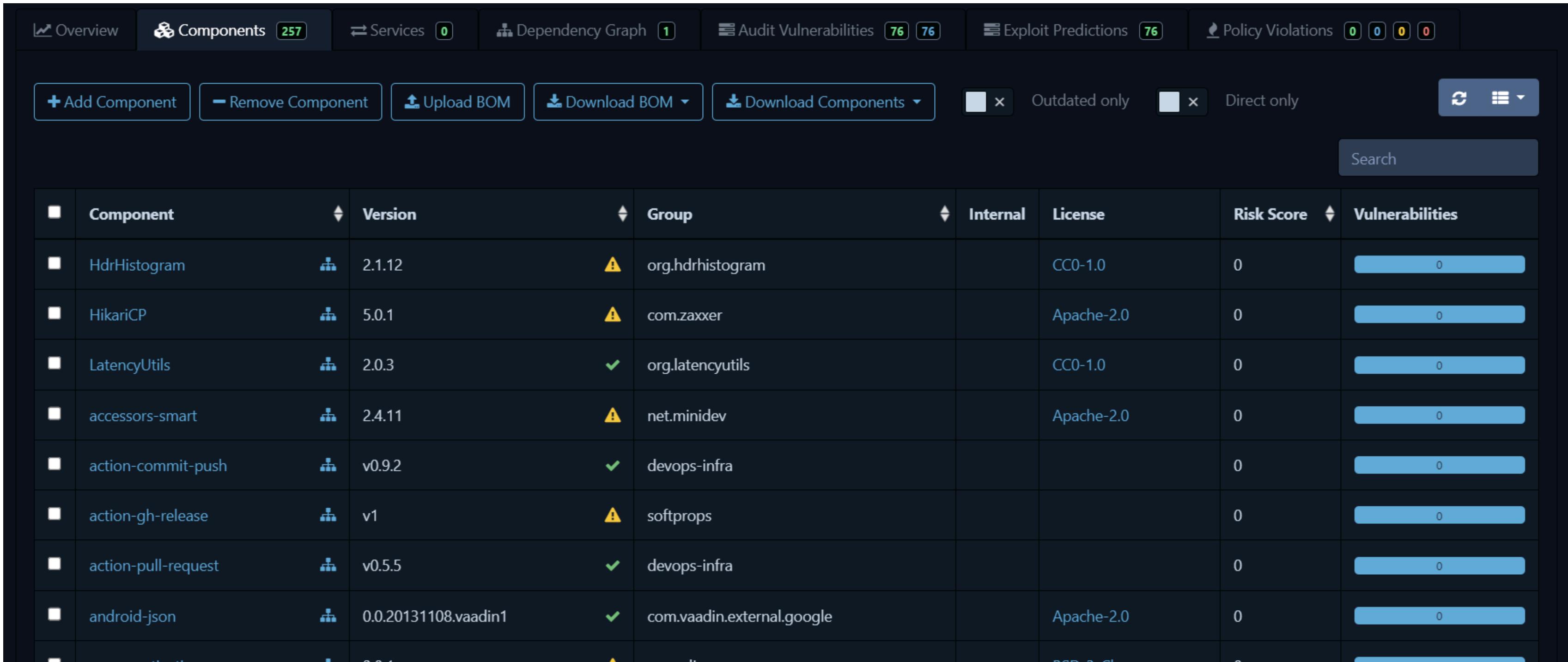
◆ Dependency-Track



Snyk vs. Dependency-Track

: Snyk과 Dependency-Track의 정량적 비교분석을 위한 테스트 구현

◆ Dependency-Track



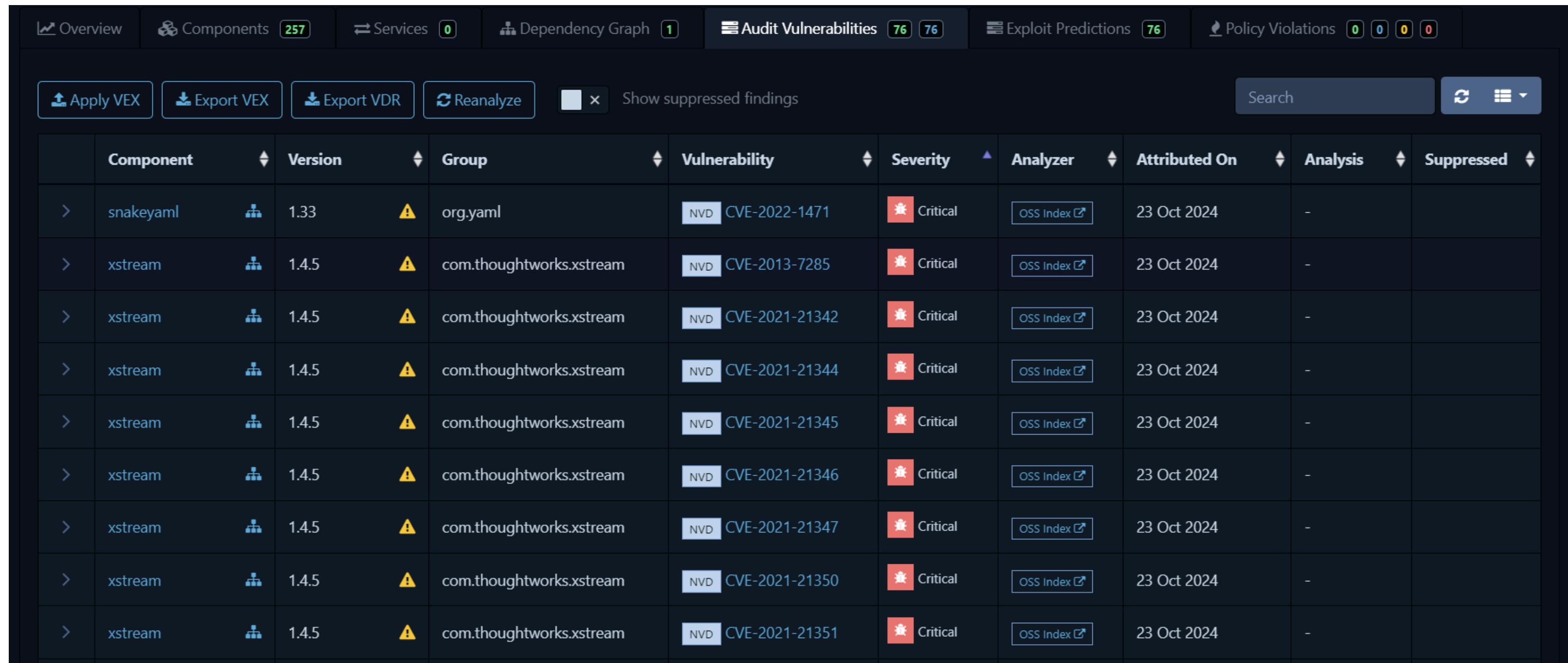
The screenshot shows the Dependency-Track web application interface. The top navigation bar includes links for Overview, Components (257), Services (0), Dependency Graph (1), Audit Vulnerabilities (76), Exploit Predictions (76), and Policy Violations (0). Below the navigation is a toolbar with buttons for Add Component, Remove Component, Upload BOM, Download BOM, Download Components, and filters for Outdated only and Direct only. A search bar is also present. The main content area is a table listing project components with the following columns: Component, Version, Group, Internal, License, Risk Score, and Vulnerabilities. The table lists several components, including HdrHistogram, HikariCP, LatencyUtils, accessors-smart, action-commit-push, action-gh-release, action-pull-request, android-json, and another entry for HdrHistogram. Most components have a risk score of 0 and 0 vulnerabilities, except for HdrHistogram which has a risk score of 0 and 0 vulnerabilities.

Component	Version	Group	Internal	License	Risk Score	Vulnerabilities
HdrHistogram	2.1.12	org.hdrhistogram		CC0-1.0	0	0
HikariCP	5.0.1	com.zaxxer		Apache-2.0	0	0
LatencyUtils	2.0.3	org.latencyutils		CC0-1.0	0	0
accessors-smart	2.4.11	net.minidev		Apache-2.0	0	0
action-commit-push	v0.9.2	devops-infra			0	0
action-gh-release	v1	softprops			0	0
action-pull-request	v0.5.5	devops-infra			0	0
android-json	0.0.20131108.vaadin1	com.vaadin.external.google		Apache-2.0	0	0

Snyk vs. Dependency-Track

: Snyk과 Dependency-Track의 정량적 비교분석을 위한 테스트 구현

◆ Dependency-Track



The screenshot shows the Dependency-Track interface with the following navigation bar:

- Overview
- Components (257)
- Services (0)
- Dependency Graph (1)
- Audit Vulnerabilities (76) (76)
- Exploit Predictions (76)
- Policy Violations (0 0 0 0)

Below the navigation bar is a toolbar with the following buttons:

- Apply VEX
- Export VEX
- Export VDR
- Reanalyze
- Show suppressed findings
- Search
- Refresh
- More

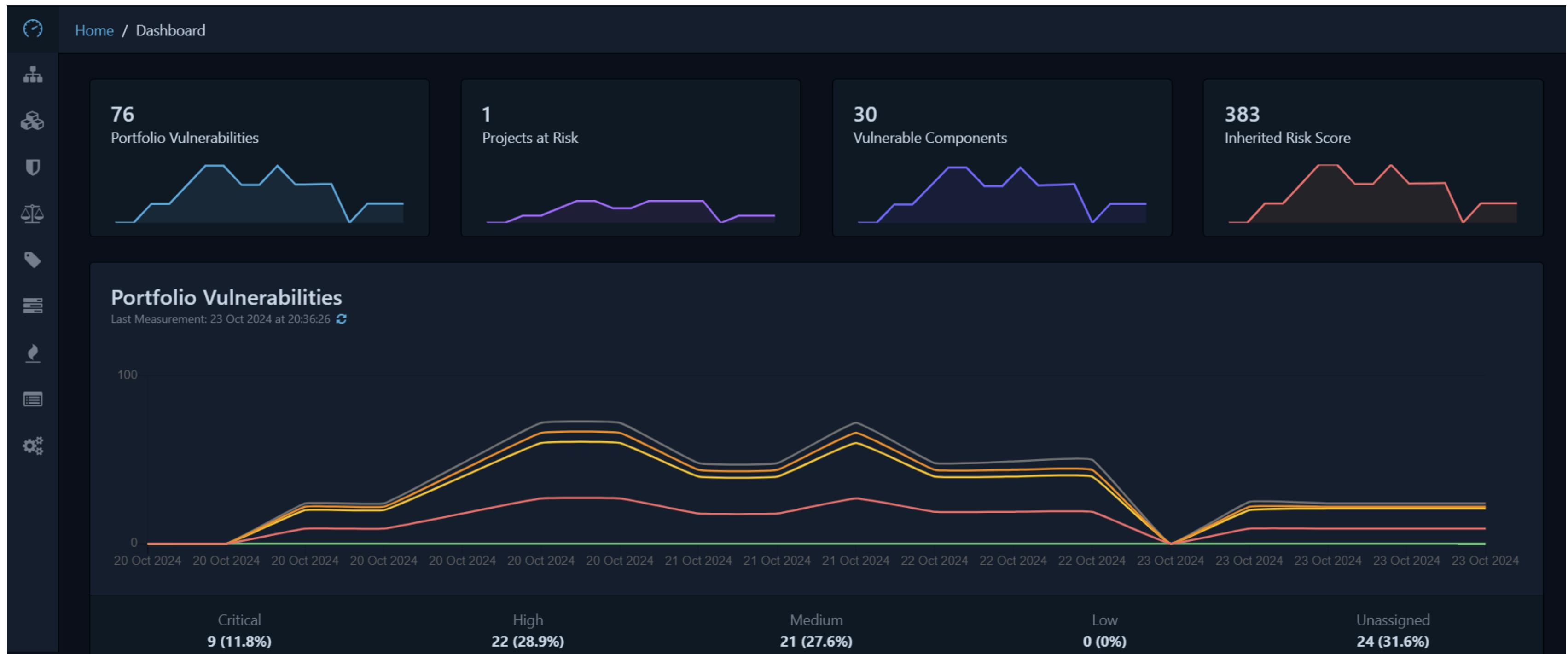
The main content area is a table listing vulnerabilities:

	Component	Version	Group	Vulnerability	Severity	Analyzer	Attributed On	Analysis	Suppressed
>	snakeyaml	1.33	⚠️ org.yaml	NVD CVE-2022-1471	Critical	OSS Index	23 Oct 2024	-	
>	xstream	1.4.5	⚠️ com.thoughtworks.xstream	NVD CVE-2013-7285	Critical	OSS Index	23 Oct 2024	-	
>	xstream	1.4.5	⚠️ com.thoughtworks.xstream	NVD CVE-2021-21342	Critical	OSS Index	23 Oct 2024	-	
>	xstream	1.4.5	⚠️ com.thoughtworks.xstream	NVD CVE-2021-21344	Critical	OSS Index	23 Oct 2024	-	
>	xstream	1.4.5	⚠️ com.thoughtworks.xstream	NVD CVE-2021-21345	Critical	OSS Index	23 Oct 2024	-	
>	xstream	1.4.5	⚠️ com.thoughtworks.xstream	NVD CVE-2021-21346	Critical	OSS Index	23 Oct 2024	-	
>	xstream	1.4.5	⚠️ com.thoughtworks.xstream	NVD CVE-2021-21347	Critical	OSS Index	23 Oct 2024	-	
>	xstream	1.4.5	⚠️ com.thoughtworks.xstream	NVD CVE-2021-21350	Critical	OSS Index	23 Oct 2024	-	
>	xstream	1.4.5	⚠️ com.thoughtworks.xstream	NVD CVE-2021-21351	Critical	OSS Index	23 Oct 2024	-	

Snyk vs. Dependency-Track

: Snyk과 Dependency-Track의 정량적 비교분석을 위한 테스트 구현

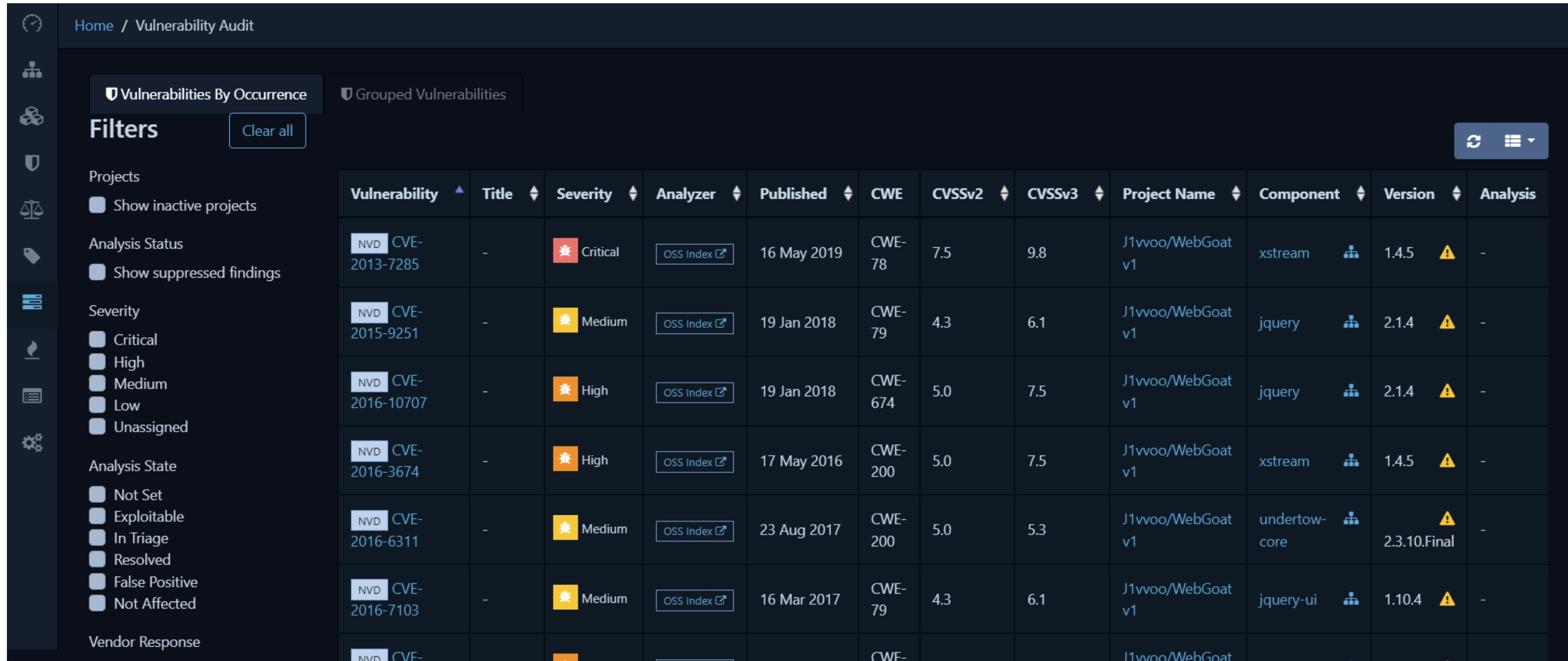
◆ Dependency-Track



Snyk vs. Dependency-Track

: Snyk과 Dependency-Track의 정량적 비교분석을 위한 테스트 구현

◆ Dependency-Track



The screenshot shows the Dependency-Track interface for a 'Vulnerability Audit'. The left sidebar contains various filters for 'Projects', 'Analysis Status', 'Severity', 'Analysis State', and 'Vendor Response'. The main area displays a table of vulnerabilities with the following columns: Vulnerability, Title, Severity, Analyzer, Published, CWE, CVSSv2, CVSSv3, Project Name, Component, Version, and Analysis. The table lists several entries, each with a 'NVD' badge and a unique CVE identifier. The first entry is CVE-2013-7285, marked as 'Critical' with a 'Medium' CVSS score of 7.5. The second entry is CVE-2015-9251, marked as 'Medium' with a 'Medium' CVSS score of 4.3. The third entry is CVE-2016-10707, marked as 'High' with a 'High' CVSS score of 7.5. The fourth entry is CVE-2016-3674, marked as 'High' with a 'High' CVSS score of 7.5. The fifth entry is CVE-2016-6311, marked as 'Medium' with a 'Medium' CVSS score of 5.3. The sixth entry is CVE-2016-7103, marked as 'Medium' with a 'Medium' CVSS score of 6.1.

Vulnerability	Title	Severity	Analyzer	Published	CWE	CVSSv2	CVSSv3	Project Name	Component	Version	Analysis
NVD CVE-2013-7285	-	Critical	OSS Index	16 May 2019	CWE-78	7.5	9.8	J1vwoo/WebGoat v1	xstream	1.4.5	⚠
NVD CVE-2015-9251	-	Medium	OSS Index	19 Jan 2018	CWE-79	4.3	6.1	J1vwoo/WebGoat v1	jquery	2.1.4	⚠
NVD CVE-2016-10707	-	High	OSS Index	19 Jan 2018	CWE-674	5.0	7.5	J1vwoo/WebGoat v1	jquery	2.1.4	⚠
NVD CVE-2016-3674	-	High	OSS Index	17 May 2016	CWE-200	5.0	7.5	J1vwoo/WebGoat v1	xstream	1.4.5	⚠
NVD CVE-2016-6311	-	Medium	OSS Index	23 Aug 2017	CWE-200	5.0	5.3	J1vwoo/WebGoat v1	undertow-core	2.3.10.Final	⚠
NVD CVE-2016-7103	-	Medium	OSS Index	16 Mar 2017	CWE-79	4.3	6.1	J1vwoo/WebGoat v1	jquery-ui	1.10.4	⚠

| Snyk vs. Dependency-Track

: Snyk과 Dependency-Track의 정량적인 비교 분석 지표

	Snyk	Dependency-Track
Severity (심각도) 분류	Critical, High, Medium, Low 총 4단계로 구분	Critical, High, Medium, Low, Unassigned 총 5단계로 구분
분석 소요 시간	약 2분 30초	31초
총 탐지된 취약점 개수	63개	76개
제공 기능	<ul style="list-style-type: none">CVSS 점수와 심각도를 기반으로 리포트를 생성하고, 각 취약점이 어떤 구성 요소에서 발생했는지 추적 가능취약점이 탐지된 후, 이를 자동으로 수정할 수 있는 PR (Pull Request) 기능 제공 → 유료 버전만 제공취약점에 대해 우선순위 점수, 수정 가능성, 익스플로잇 성숙도, 상태 (취약점 해결 여부) 등을 제공	<ul style="list-style-type: none">SBOM으로 추출된 Component들의 버전, 그룹, 라이선스, 의존성 그래프 등 다양한 정보를 확인할 수 있음탐지된 취약점의 심각도와 상세 분석 결과에 대한 설명을 함께 제공Dependency-Track에 업로드 된 모든 SBOM 요소들에 대한 전체 대시보드, 취약점 목록 등을 제공

| Snyk vs. Dependency-Track

: Snyk과 Dependency-Track의 정량적인 비교 분석 지표

▶ 각 도구에서만 단독으로 탐지되는 취약점 (CVE 취약점 기준)

◆ Snyk (총 22개)

1. com.thoughtworks.xstream:xstream와 관련된 취약점 CVE-2021-39144 외 17개
2. io.undertow:undertow-core와 관련된 취약점 CVE-2024-27316 외 1개
3. CVE-2023-6481 → ch.qos.logback:logback-classic
4. CVE-2023-6481 → ch.qos.logback:logback-core
5. CVE-2024-22259 → org.springframework:spring-web

◆ Dependency-Track (총 33개)

1. CVE-2016-10707 외 18개 → jquery, jquery-ui에서 발생하는 취약점
2. CVE-2024-22233 → spring-core
3. CVE-2023-24998 → commons-fileupload
4. CVE-2016-6311 → undertow-core
5. CVE-2024-5971 → undertow-core
6. CVE-2023-51074 → json-path
7. CVE-2023-41329 → wiremock
8. CVE-2024-31033 → jjwt (io.jsonwebtoken)
9. CVE-2024-6484 → bootstrap (org.webjars)
10. CVE-2024-31573 → xmlunit-core (org.xmlunit)
11. CVE-2024-8184 외 4개 → jetty (org.eclipse.jetty)

| Snyk vs. Dependency-Track

: Snyk과 Dependency-Track의 정량적인 비교 분석 지표

▶ 각 도구에서만 단독으로 탐지되는 취약점 (CVE 취약점 기준)

◆ Snyk

- 특정 라이브러리와 서브 컴포넌트에 대해 다수의 취약점이 탐지되며, 라이브러리별로 깊이 있는 분석이 가능
- 자주 사용되는 오픈소스 라이브러리와 긴밀하게 연동되어, 개발자 환경에서 발생할 수 있는 보안 문제를 실시간으로 감지하고 대응 가능
- 자동화된 취약점 수정과 실시간 경고 기능이 강력하며, 개발자 중심의 보안 관리 도구로 설계됨
- GitHub, npm, Maven Central 등 대형 오픈소스 저장소와의 통합을 통해 패키지에 대한 취약점을 빠르게 감지함

◆ Dependency-Track

- SBOM을 기반으로 전체 종속성 트리를 분석하여, 직접 사용하는 라이브러리뿐만 아니라 하위 의존성에서 발생하는 취약점까지 탐지 가능
 - 모든 구성 요소를 분석하므로 누락되거나 잘 알려지지 않은 라이브러리도 추적할 수 있음
- 주로 대규모 커뮤니티 및 오픈소스 라이브러리에서 발생하는 취약점을 탐지하는 Snyk에 비해 다양한 소스에서 취약점 수집 (ex. Jetty 서버)
- 서버 측 라이브러리나 간접적으로 포함된 구성 요소에서 발생하는 취약점은 더 잘 탐지되는 경향이 있음

| Snyk vs. Dependency-Track

: Snyk과 Dependency-Track의 정량적인 비교 분석 지표



VS



| Snyk vs. Dependency-Track

: Snyk과 Dependency-Track의 정량적인 비교 분석 지표



VS

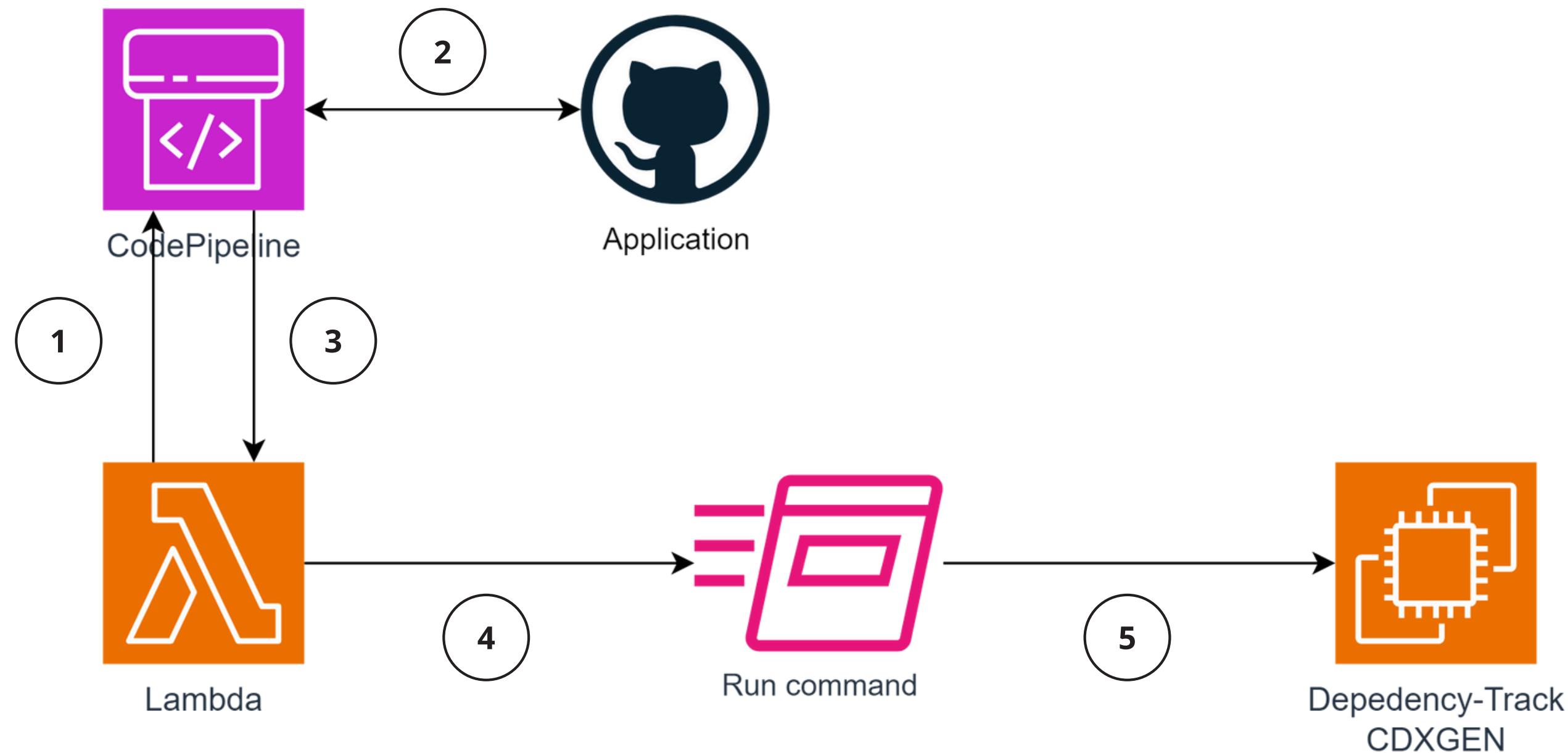


- SBOM 기반 관리로 소프트웨어 구성 요소를 체계적으로 관리
- 종속성 트리 전체 분석으로 **하위 의존성에서 발생하는 보안 취약점까지 포괄적으로 분석** 가능
- 다양한 취약점 데이터 소스와 통합되어, **최신 보안 취약점 데이터를 빠르게 반영**하고 관리 가능

| SCA CodePipeline 통합

: 최종 선정한 Dependency-Track을 CodePipeline에 통합한 자동화 구현

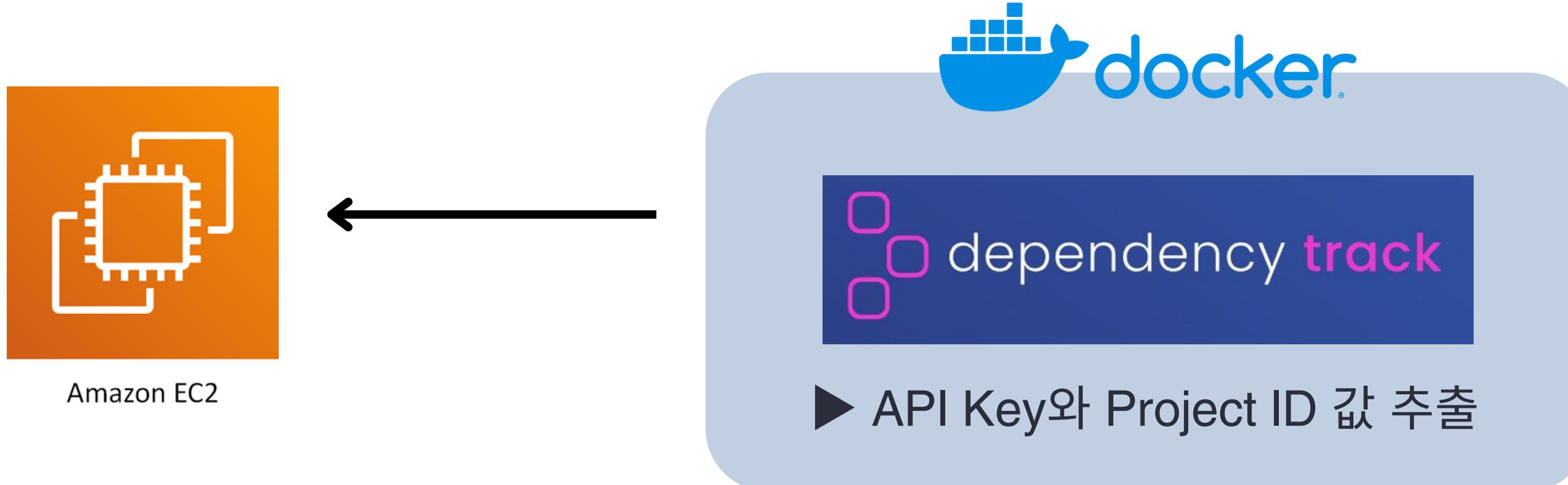
◆ Dependency-Track



| SCA CodePipeline 통합

: 최종 선정한 Dependency-Track을 CodePipeline에 통합한 자동화 구현

◆ Dependency-Track

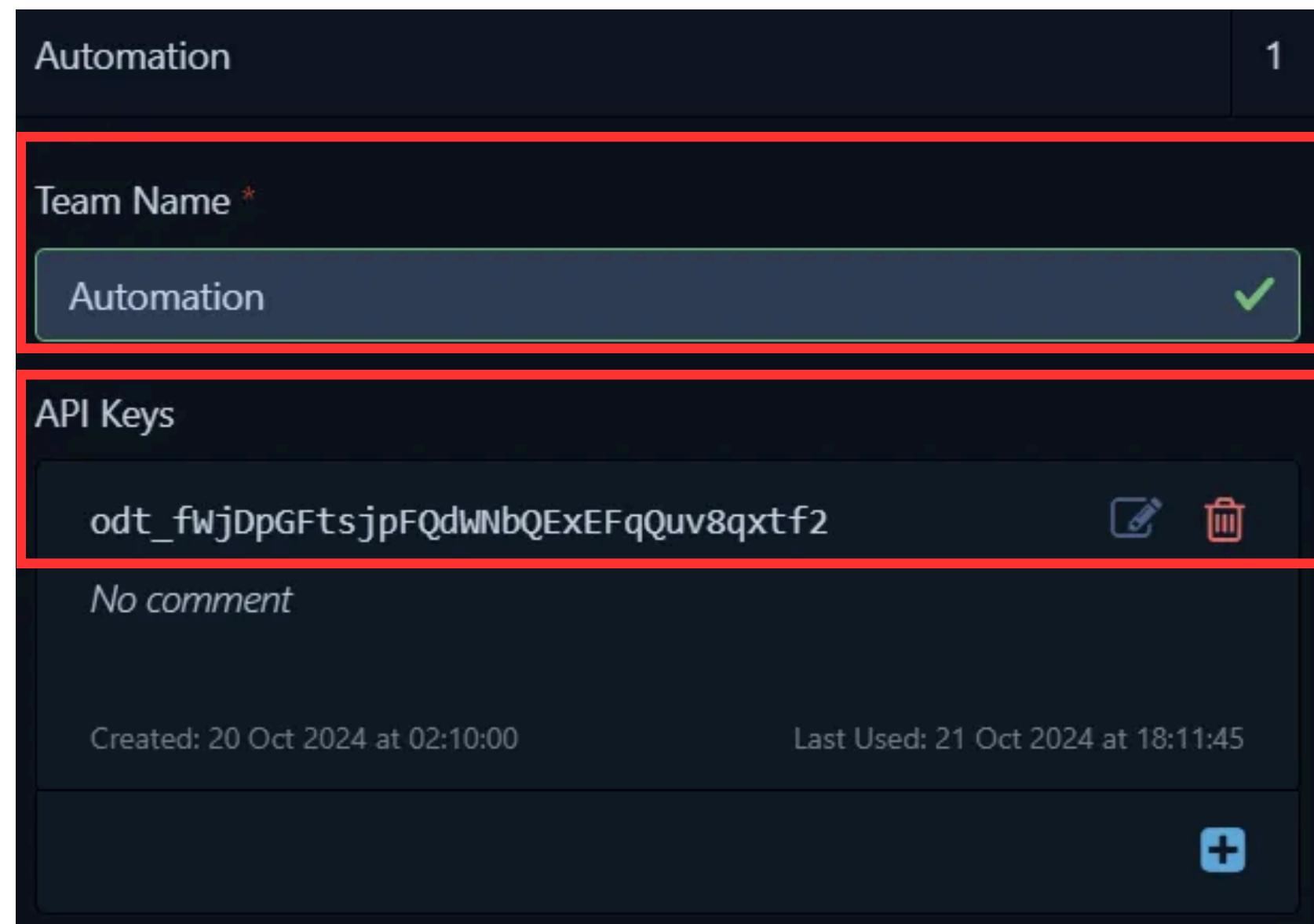


| SCA CodePipeline 통합

: 최종 선정한 Dependency-Track을 CodePipeline에 통합한 자동화 구현

◆ Dependency-Track

▶ API Key



'Automation'이라는 이름을 가진 Team 생성 후 API Key를 발급받음

| SCA CodePipeline 통합

: 최종 선정한 Dependency-Track을 CodePipeline에 통합한 자동화 구현

◆ Dependency-Track

▶ API Key

→ SBOM 업로드, 프로젝트 생성, 취약점 확인 등을 위한 최소한의 권한만 부여

Permissions	
BOM_UPLOAD	刪除
PORTFOLIO_MANAGEMENT	刪除
PROJECT_CREATION_UPLOAD	刪除
VIEW_PORTFOLIO	刪除
VIEW_VULNERABILITY	刪除
VULNERABILITY_ANALYSIS	刪除
VULNERABILITY_MANAGEMENT	刪除

1. BOM_UPLOAD: CycloneDX SBOM을 업로드할 수 있는 기능을 제공
2. PORTFOLIO_MANAGEMENT: 포트폴리오에서 데이터 생성, 수정 및 삭제를 허용
3. PROJECT_CREATION_UPLOAD: BOM 또는 스캔 업로드 시 선택적으로 프로젝트를 생성할 수 있는 기능을 제공 (존재하지 않는 경우)
4. VIEW_PORTFOLIO: 프로젝트, 구성 요소 및 라이선스 포트폴리오를 볼 수 있는 기능을 제공
5. VIEW_VULNERABILITY: 프로젝트에 영향을 미치는 취약점을 볼 수 있는 기능을 제공
6. VULNERABILITY_ANALYSIS: 취약점에 대한 분석 결정을 내릴 수 있는 기능을 제공

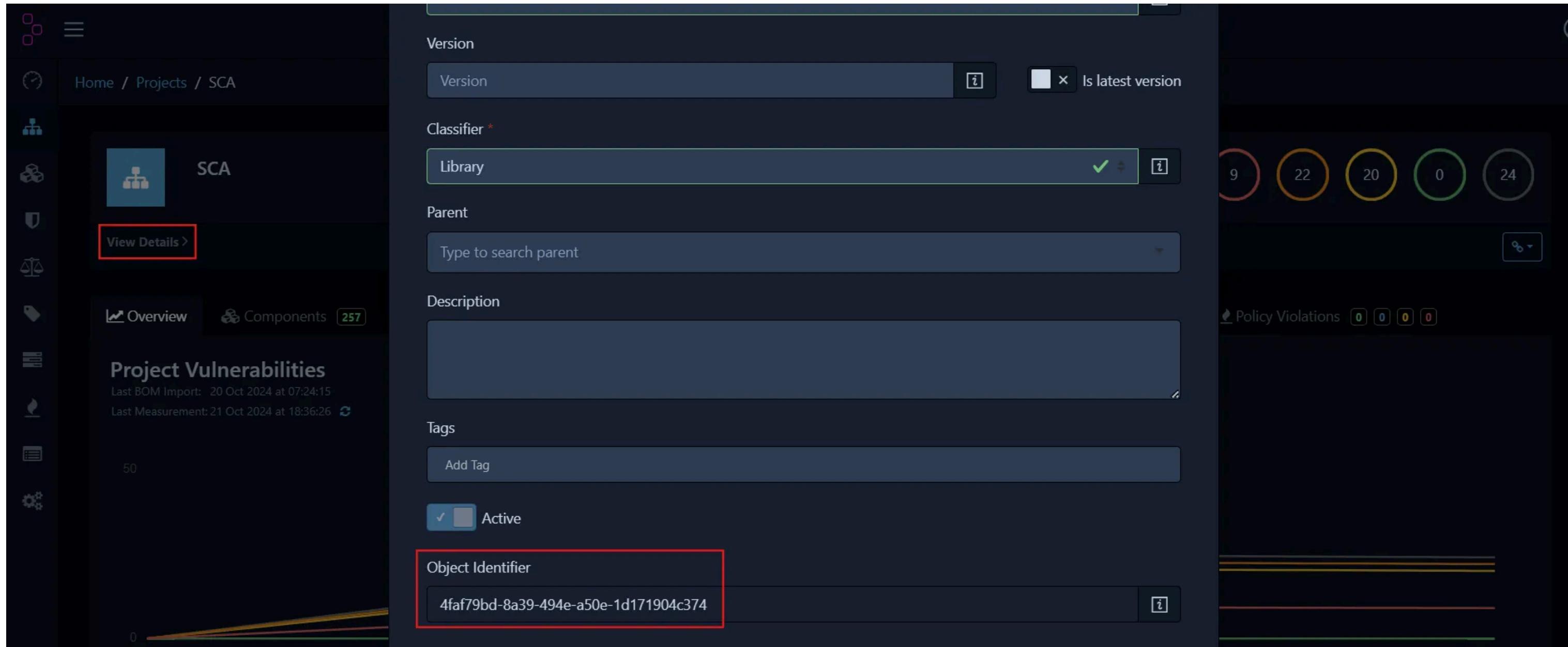
| SCA CodePipeline 통합

: 최종 선정한 Dependency-Track을 CodePipeline에 통합한 자동화 구현

◆ Dependency-Track

▶ Project uuid

→ 사용하고자 하는 프로젝트의 ‘View details’ 탭을 통해 확인 가능



| SCA CodePipeline 통합

: 최종 선정한 Dependency-Track을 CodePipeline에 통합한 자동화 구현

◆ Dependency-Track



```
1  {
2    "schemaVersion": "2.2",
3    "description": "SCA cdxgen to dependency-track",
4    "parameters": {
5      "dependencytracktoken": {
6        "type": "String",
7        "description": "dependencytrack authentication token"
8      },
9      "projectid": {
10        "type": "String",
11        "description": "dependencytrack project_id"
12      },
13      "repo": {
14        "type": "String",
15        "description": "source repo"
16      }
17    }
}
```

→ Run command 기본 설정과 사용할 파라미터들에 대한 정의

| SCA CodePipeline 통합

: 최종 선정한 Dependency-Track을 CodePipeline에 통합한 자동화 구현

◆ Dependency-Track



```
18  "mainSteps": [
19  {
20    "action": "aws:runShellScript",
21    "name": "run_cdxgen_server",
22    "inputs": {
23      "runCommand": [
24        "#!/bin/bash",
25
26        "URL=\"http://localhost:8081\"",
27        "echo $URL",
28
29        "apikey=\"{{dependencytracktoken}}\"",
30        "projectid=\"{{projectid}}\"",
31        "echo $projectid",
32
33        "repo=\"https://github.com/{{repo}}.git\"",
34        "echo $repo",
35
36        "if [ -z \"$apikey\" ]; then echo \"api key is missing\"; exit 1; fi",
37        "docker stop cdxgen 2>/dev/null",
38
39        "docker run --name cdxgen --network=\"host\" -d --rm -t ghcr.io/cyclonedx/cdxgen -r /app --server
40        --server-host 0.0.0.0 --server-url $URL --api-key $apikey --project-id $projectid",
41
42        "sleep 5",
43
44        "curl http://localhost:9090/sbom?url=$repo&multiProject=true&type=java",
45
46        "echo \"cdxgen complete.\""
47    ]
}
```

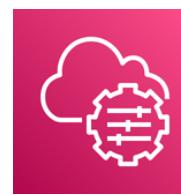
→ SBOM을 업로드할 서버 URL 설정

→ AWS Systems Manager 파라미터로 읽어온 값을
api key, project id로 설정

| SCA CodePipeline 통합

: 최종 선정한 Dependency-Track을 CodePipeline에 통합한 자동화 구현

◆ Dependency-Track



```
18 "mainSteps": [
19 {
20   "action": "aws:runShellScript",
21   "name": "run_cdxgen_server",
22   "inputs": {
23     "runCommand": [
24       "#!/bin/bash",
25
26       "URL=\"http://localhost:8081\"",
27       "echo $URL",
28
29       "apikey=\"{{dependencytracktoken}}\"",
30       "projectid=\"{{projectid}}\"",
31       "echo $projectid",
32
33       "repo=\"https://github.com/{{repo}}.git\"",
34       "echo $repo",
35
36       "if [ -z \"$apikey\" ]; then echo \"api key is missing\"; exit 1; fi",
37       "docker stop cdxgen 2>/dev/null",
38
39       "docker run --name cdxgen --network=\"host\" -d --rm -t ghcr.io/cyclonedx/cdxgen -r /app --server
40       --server-host 0.0.0.0 --server-url $URL --api-key $apikey --project-id $projectid",
41
42       "sleep 5",
43
44       "curl http://localhost:9090/sbom?url=$repo&multiProject=true&type=java",
45
46       "echo \"cdxgen complete.\""
47   ]
```

→ API 키 확인 & 실행중인 cdxgen 컨테이너가 있을 경우 중지

→ Cdxgen 컨테이너 실행

→ SBOM 생성 요청

| SCA CodePipeline 통합

: 최종 선정한 Dependency-Track을 CodePipeline에 통합한 자동화 구현

◆ Dependency-Track



```
1  import json
2  import urllib.request
3  import urllib.parse
4  import boto3
5  import traceback
6  import os
7
8  # Lambda 함수 로딩시 'Loading function' 메시지 출력
9  print('Loading function')
10
11 # AWS CodePipeline 클라이언트 객체 생성
12 code_pipeline = boto3.client('codepipeline')
13
14 # AWS SSM (System Manager) 클라이언트 객체 생성
15 ssm = boto3.client('ssm')
16
17 # 환경 변수에서 API 키와 URL 정보 불러오기
18 dependency_track_apikey = os.getenv("dependencytrack_apikey")
19 dependency_track = os.getenv("dependencytrack")
```

환경 변수 (2)

아래의 환경 변수는 기본 Lambda 서비스 키를 이용해 저장 중 암호화됩니다.

환경 변수 찾기

키	값
dependencytrack	http://43.203.179.103:8081
dependencytrack_apikey	odt_fWjDpGFtsjpFQdWNbQExEFqQuv8qxtf2



| SCA CodePipeline 통합

: 최종 선정한 Dependency-Track을 CodePipeline에 통합한 자동화 구현

◆ Dependency-Track



```
34  # 프로젝트 UUID를 가져오는 함수
35  def get_projects(repo):
36      # Dependency-Track 프로젝트 조회 API 호출을 위한 URL 생성
37      url = f"{dependency_track}/api/v1/project/lookup?name={repo}&version=v1"
38      print(url)
39
40      # API 요청 생성 (HTTP GET 요청)
41      req = urllib.request.Request(url)
42      req.add_header("X-Api-Key", dependency_track_apikey)
43
44  try:
45      # API 호출하여 프로젝트 정보 조회
46      with urllib.request.urlopen(req) as res:
47          res = urllib.request.urlopen(req)
48          data = res.read() # 응답 데이터 읽기
49          data = data.decode('utf8').replace("'", "") # 데이터를 JSON 형식으로 변환
50          data = json.loads(data)
51          return data['uuid'] # 프로젝트 UUID 반환
52  except urllib.error.HTTPError as e:
53      # 프로젝트가 없을 경우 에러 메시지 출력 후 새 프로젝트 생성
54      print(e)
55      print('need to create project')
56      return create_project(repo)
```

→ Repository 명으로 된 프로젝트가
Dependency-Track에 있는지 확인

→ 해당 프로젝트가 없을 경우
create_project() 함수 호출하여
새 프로젝트 생성

| SCA CodePipeline 통합

: 최종 선정한 Dependency-Track을 CodePipeline에 통합한 자동화 구현

◆ Dependency-Track



```
59  # 프로젝트를 생성하는 함수
60  def create_project(repo):
61      # 프로젝트 생성 API 호출을 위한 URL 및 데이터 설정
62      url = f"{dependency_track}/api/v1/project"
63      data = {
64          "name": repo,
65          "version": "v1",
66          "parent": {
67              "uuid": "839f8255-f4c9-4f42-ae43-552c63dc6674" # 상위 프로젝트 UUID 설정
68          },
69          "classifier": "APPLICATION", # 프로젝트 유형 설정
70          "tags": [], # 태그 설정 (없음)
71          "active": True, # 활성화 상태
72          "isLatest": True # 최신 버전으로 설정
73      }
74
75      # JSON 데이터를 바이트로 인코딩 후 요청 생성 (HTTP PUT 요청)
76      data = json.dumps(data).encode('utf-8')
77      req = urllib.request.Request(url, data=data, method='PUT')
78      req.add_header("Content-Type", 'application/json')
79      req.add_header("X-API-Key", dependency_track_apikey)
80
81      # API 호출하여 프로젝트 생성 후 UUID 반환
82      with urllib.request.urlopen(req) as res:
83          res = res.read().decode('utf-8').replace("'", "")
84          res = json.loads(res)
85          return res['uuid']
```

→ 새 프로젝트 생성

→ 생성된 프로젝트의 uuid 반환

| SCA CodePipeline 통합

: 최종 선정한 Dependency-Track을 CodePipeline에 통합한 자동화 구현

◆ Dependency-Track



```
87  # SSM을 통해 특정 인스턴스에서 SCA 분석 수행
88  def analyze(projectid, repo):
89      res = ssm.send_command(
90          DocumentName="sca-cdxgen-bs", # SSM 명령 문서 이름
91          Targets=[ # 대상 인스턴스 설정
92              {
93                  "Key": "InstanceIds",
94                  "Values": [
95                      "i-0fdec2be7f9745181" # 분석을 수행할 EC2 인스턴스 ID
96                  ]
97              }
98          ],
99          TimeoutSeconds=600, # 명령 실행 제한 시간 (초)
100         MaxConcurrency='50', # 최대 동시 실행 수
101         MaxErrors='0', # 허용되는 최대 오류 수
102         DocumentVersion='4', # 명령 문서 버전
103         Parameters={ # 명령에 전달할 파라미터
104             "dependencytracktoken": [
105                 dependency_track_apikey
106             ],
107             "projectid": [
108                 projectid
109             ],
110             "repo": [
111                 repo
112             ]
113         }
114     )
115
116     return res
```

→ sca 분석을 위해 작성한 run command 문서 이름 정의

→ sca 분석을 수행할 대상 인스턴스 설정

→ run command 명령에 전달할 파라미터 설정

| SCA CodePipeline 통합

: 최종 선정한 Dependency-Track을 CodePipeline에 통합한 자동화 구현

◆ Dependency-Track



```
120  def lambda_handler(event, context):  
121      try:  
122          # 이벤트에서 Job ID를 추출  
123          print(event)  
124          job_id = event['CodePipeline.job']['id']  
125  
126          # CodePipeline에서 해당 Job ID로 세부 정보 조회  
127          res = code_pipeline.get_job_details(  
128              jobId=job_id  
129          )  
130  
131          print(res)  
132  
133          # 파이프라인 이름을 추출  
134          pipeline = res['jobDetails']['data']['pipelineContext']['pipelineName']  
135          print(res['jobDetails']['data']['pipelineContext']['pipelineName'])  
136  
137          # 파이프라인 세부 정보 조회  
138          res = code_pipeline.get_pipeline(  
139              name=pipeline  
140          )  
141          print(res)  
142  
143          # 저장소와 브랜치 정보 추출  
144          repo = res['pipeline']['stages'][0]['actions'][0]['configuration']['FullRepositoryId']  
145          branch = res['pipeline']['stages'][0]['actions'][0]['configuration']['BranchName']
```

→ 이벤트의 Job ID를 통해 CodePipeline의 세부 정보 조회

→ CodePipeline의 정보 추출

| SCA CodePipeline 통합

: 최종 선정한 Dependency-Track을 CodePipeline에 통합한 자동화 구현

◆ Dependency-Track



```
147     print(repo)
148     print(branch)
149
150     # 프로젝트 UUID 조회 또는 생성
151     project_uuid = get_projects(repo)
152     print(f"project uuid : {project_uuid}")
153
154     # SCA 분석 실행
155     analyze(project_uuid, repo)
156
157     except Exception as e:
158         # 예외 처리: 에러 메시지 출력 및 CodePipeline 작업 실패 처리
159         print('Function failed due to exception.')
160         print(e)
161         traceback.print_exc()
162         put_job_failure(job_id, 'Function exception: ' + str(e))
163
164     # 함수 완료 메시지 출력
165     print('Function complete.')
166     put_job_success(job_id, 'Function completed successfully.')
167     return "Complete."
```

→ get_projects() 함수를 통해 프로젝트의 정보를 조회하고 해당 프로젝트의 uuid 값 반환

→ analyze() 함수를 통해 SCA 분석 수행

→ 에러 처리

→ Lambda 동작 완료 후 파이프라인에 작업 성공 반환

| SCA CodePipeline 통합

: 최종 선정한 Dependency-Track을 CodePipeline에 통합한 자동화 구현

◆ Dependency-Track



▶ Lambda 작동 과정 정리

1. Lambda 동작
2. event에서 Job ID 추출 → CodePipeline의 세부 정보 조회
3. CodePipeline의 정보로 부터 Repository 정보 추출
4. get_projects() 함수를 통해 Dependency-track 내부에 해당 Repository 명으로 생성된 프로젝트가 있는지 확인
 - a. 해당 Repository 명으로 생성된 프로젝트가 있을 경우 해당 프로젝트의 uuid를 반환
 - b. 해당 Repository 명으로 생성된 프로젝트가 없을 경우 create_project() 함수를 호출하여 프로젝트 생성 후, 생성된 프로젝트의 uuid 반환
5. 반환된 uuid를 활용하여 analyze() 함수 호출 후 sca 분석 수행
 - a. sca 분석을 위해 run command를 작성한 ssm 명과 대상 인스턴스를 설정 후 sca 분석에 필요한 파라미터 전송
6. 분석 완료 후 완료 되었다는 메시지 출력과 함께 파이프라인에 성공을 전송

| SCA CodePipeline 통합

: 최종 선정한 Dependency-Track을 CodePipeline에 통합한 자동화 구현

◆ Dependency-Track



▶ Lambda 권한



AWS CodePipeline
4 actions, 1 resources



AWS Systems Manager
1 actions, 1 resources



Amazon CloudWatch Logs
1 actions, 1 resources



Amazon EC2
8 actions, 1 resources

| SCA CodePipeline 통합

: 최종 선정한 Dependency-Track을 CodePipeline에 통합한 자동화 구현

◆ Dependency-Track

▶ CodePiepline 통합

작업 이름
작업의 이름을 선택합니다.

100자 이하

작업 공급자

리전

입력 아티팩트

이 작업의 입력 아티팩트를 선택합니다. [자세히 알아보십시오](#)

SourceArtifact X
Source에 의해 정의됨

100자 이하

함수 이름

AWS Lambda 콘솔에서 이미 생성한 함수를 선택합니다. 또는 AWS Lambda 콘솔에서 함수를 생성한 후 이 작업으로 돌아옵니다.

 X C

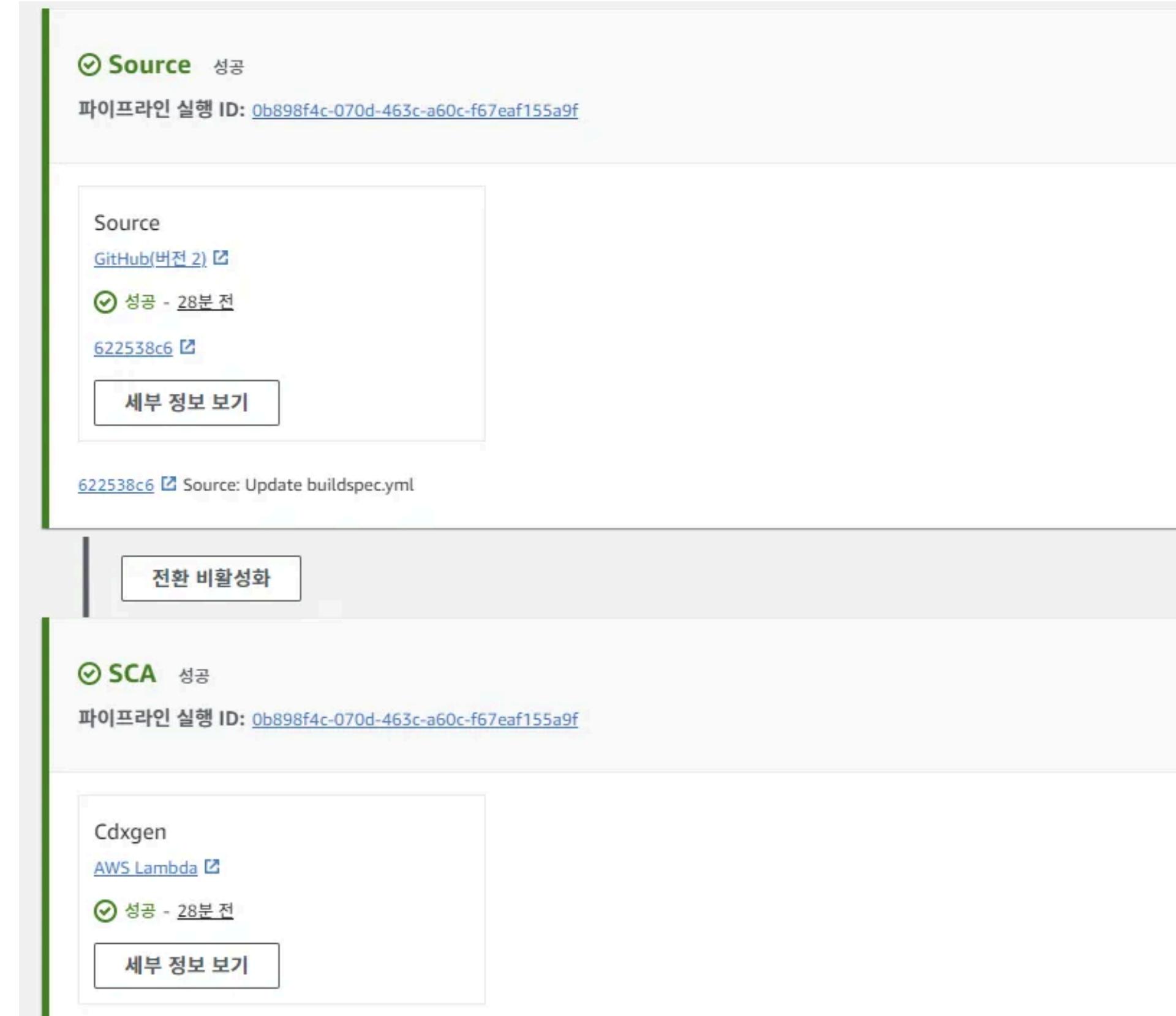
함수 이름에는 공백 없이 문자, 숫자, 하이픈 또는 밑줄만 포함됩니다. 함수 별칭 또는 함수 ARN은 포함되지 않습니다.

사용자 파라미터 - 선택 사항

| SCA CodePipeline 통합

: 최종 선정한 Dependency-Track을 CodePipeline에 통합한 자동화 구현

◆ Dependency-Track ▶ CodePiepline 통합



| SCA CodePipeline 통합

: 최종 선정한 Dependency-Track을 CodePipeline에 통합한 자동화 구현

◆ Dependency-Track

▶ CodePiepline 통합



| DAST 분석

: DAST 분석의 개요

◆ DAST(Dynamic Application Security Testing) 분석?

: 웹 애플리케이션을 실행 중인 상태에서 보안 취약점을 탐지하는 테스트 방식

- 실행 중인 애플리케이션을 대상으로 보안 테스트를 수행
- 블랙박스 테스트 방식으로, 소스 코드를 보지 않고 외부에서 웹 애플리케이션을 분석
- 실제 운영 중인 서비스에서 공격 시나리오를 시뮬레이션하여 취약점을 확인

◆ DAST 분석 도구 종류



OWASP-ZAP vs. Arachni

: OWASP-ZAP와 Arachni의 정성적인 비교 분석 지표

	OWASP-ZAP	Arachni
취약점 탐지	다양한 웹 취약점 (XSS, SQL Injection, 보안 헤더 미설정 등)	폼 기반 공격 (CSRF, 암호화되지 않은 전송 등) 및 세부적인 취약점 탐지에 강점
오픈 소스 커뮤니티	넓고 활발한 오픈 소스 커뮤니티 (OWASP 지원)	비교적 작은 커뮤니티
플랫폼 및 배포 방식	Windows, macOS, Linux	Linux, macOS (Windows는 공식 지원 미비)
웹 인터페이스	미지원	웹 UI 통해 스캔 관리 가능
업데이트	정기적인 업데이트 및 OWASP 지원으로 지속적인 기능 개선	2022년 이후 업데이트 지원 중단으로 인해, 최신 보안 위협에 대한 대응이 어려울 수 있음
다른 도구와의 통합	RESTful API를 통해 다른 도구와 쉽게 통합 가능	RPC API로 다른 시스템과 연동 및 자동화 가능
라이선스	무료	무료

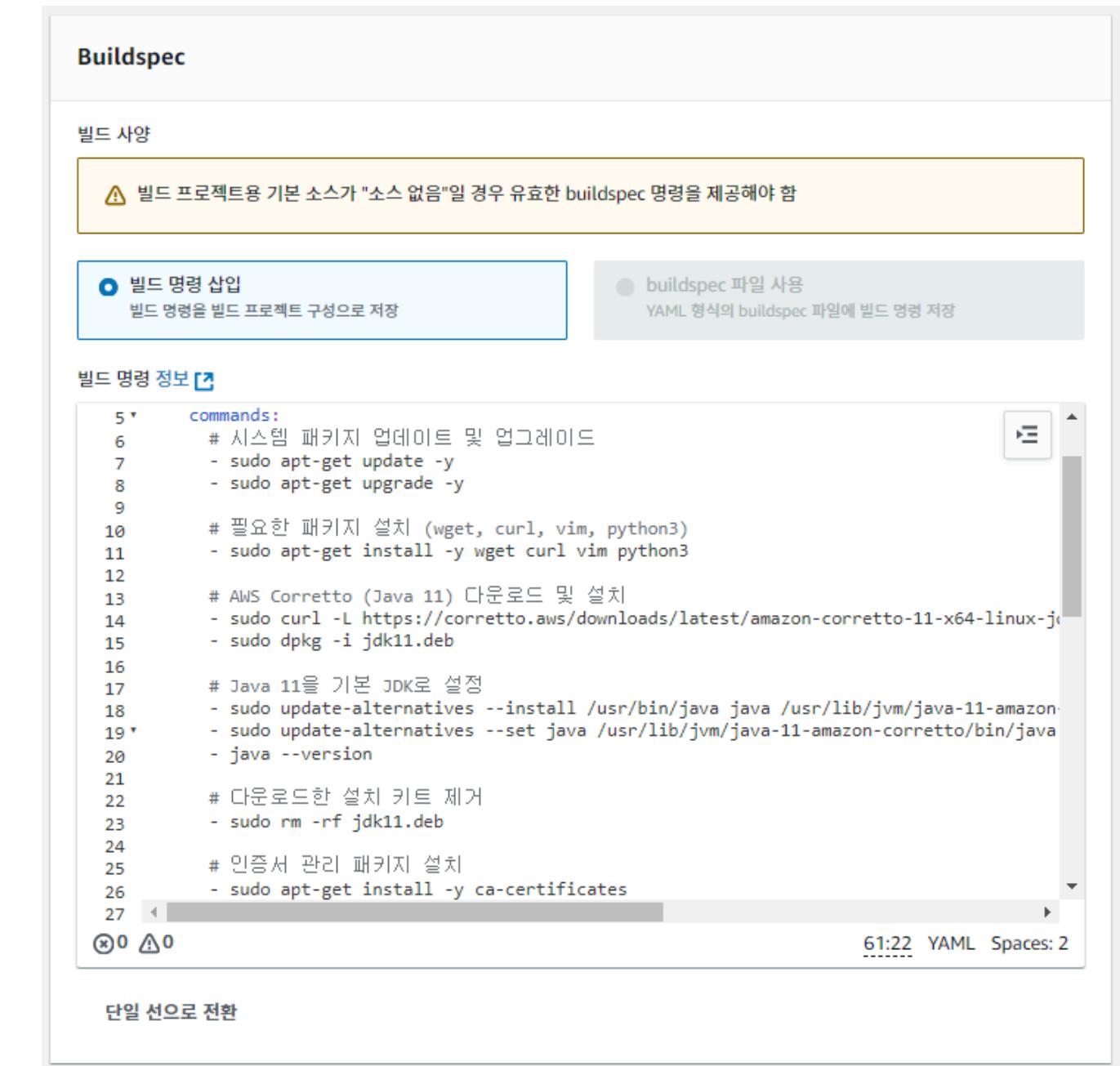
OWASP-ZAP vs. Arachni

: OWASP-ZAP와 Arachni의 정량적 비교분석을 위한 테스트 구현

◆ OWASP-ZAP

Buildspec.yml

OWASP ZAP을 이용하여 WebGoat를 빠르게 스캔하고
보안 리포트를 생성하는 파일



The screenshot shows the Arachni build configuration interface. The 'Buildspec' tab is selected. A warning message in a yellow box states: '빌드 프로젝트용 기본 소스가 "소스 없음"일 경우 유효한 buildspec 명령을 제공해야 함'. Below this, a radio button is selected for '빌드 명령 삽입', with the sub-instruction '빌드 명령을 빌드 프로젝트 구성으로 저장'. Another radio button is available for 'buildspec 파일 사용', which would store the build command in a YAML-formatted buildspec file. The 'Build 명령 정보' section contains a code editor with the following buildspec YAML code:

```
5 * commands:
6   # 시스템 패키지 업데이트 및 업그레이드
7   - sudo apt-get update -y
8   - sudo apt-get upgrade -y
9
10  # 필요한 패키지 설치 (wget, curl, vim, python3)
11  - sudo apt-get install -y wget curl vim python3
12
13  # AWS Corretto (Java 11) 다운로드 및 설치
14  - sudo curl -L https://corretto.aws/downloads/latest/amazon-corretto-11-x64-linux-jdk11.deb
15  - sudo dpkg -i jdk11.deb
16
17  # Java 11을 기본 JDK로 설정
18  - sudo update-alternatives --install /usr/bin/java java /usr/lib/jvm/java-11-amazon-corretto/bin/java
19  - sudo update-alternatives --set java /usr/lib/jvm/java-11-amazon-corretto/bin/java
20  - java --version
21
22  # 다운로드한 설치 키트 제거
23  - sudo rm -rf jdk11.deb
24
25  # 인증서 관리 패키지 설치
26  - sudo apt-get install -y ca-certificates
27
```

The code editor shows lines 5 through 27 of the buildspec file. The status bar at the bottom right indicates '61:22 YAML Spaces: 2'.

OWASP-ZAP vs. Arachni

: OWASP-ZAP와 Arachni의 정량적 비교분석을 위한 테스트 구현

◆ OWASP-ZAP

S3버킷에서 분석 리포트 결과 확인 가능



Site: <http://BlueGreen-jw-87223296.ap-northeast-2.elb.amazonaws.com>

Generated on Wed, 9 Oct 2024 09:20:27

ZAP Version: 2.15.0

ZAP is supported by the [Crash Override Open Source Fellowship](#)

Summary of Alerts

Risk Level	Number of Alerts
High	0
Medium	3
Low	1
Informational	1
False Positives:	0

Alerts

Name	Risk Level	Number of Instances
Absence of Anti-CSRF Tokens	Medium	1
Content Security Policy (CSP) Header Not Set	Medium	1
Missing Anti-clickjacking Header	Medium	1
X-Content-Type-Options Header Missing	Low	1
User Agent Fuzzer	Informational	12

Alert Detail

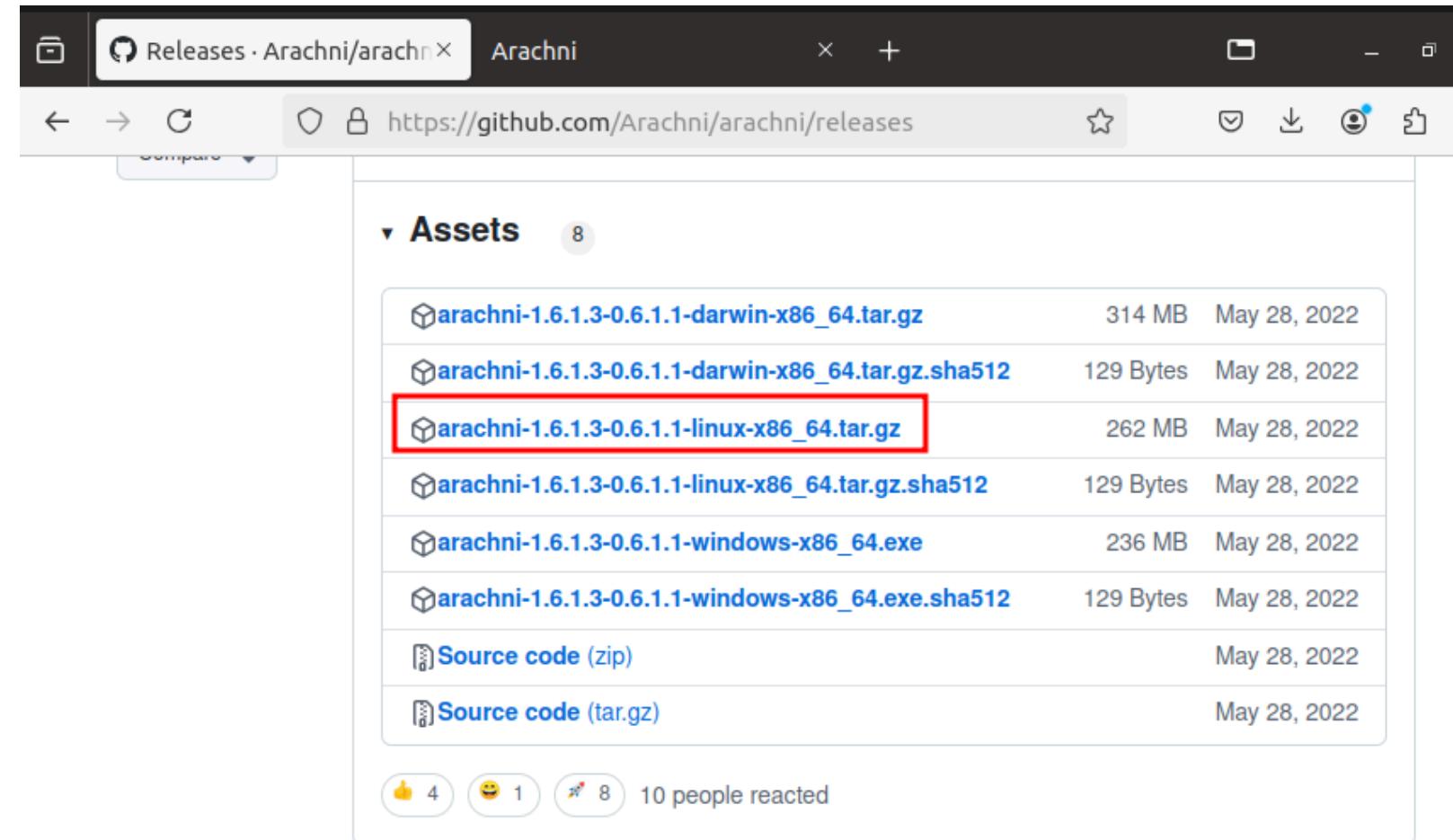
OWASP-ZAP vs. Arachni

: OWASP-ZAP와 Arachni의 정량적 비교분석을 위한 테스트 구현

◆ Arachni

리눅스 환경에서 진행

GitHub에서 Arachni 패키지를 다운로드



user 권한으로 로그인

Administrator account

E-mail: `admin@admin.admin`

Password: `administrator`

Regular user account

E-mail: `user@user.user`

Password: `regular_user`

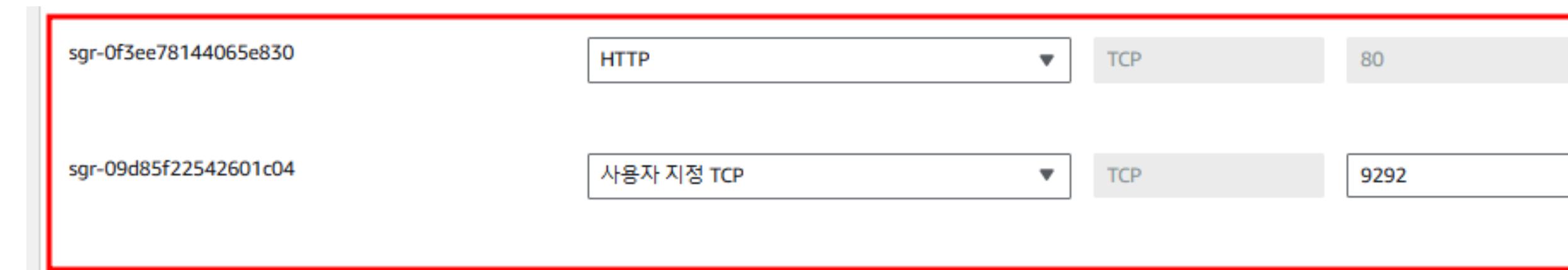
OWASP-ZAP vs. Arachni

3. OWASP-ZAP와 Arachni의 정량적 비교분석을 위한 테스트 구현

Arachni

```
smkim@ubuntu:~/Downloads/arachni-1.6.1.3-0.6.1.1$ ./bin/arachni_web
Puma starting in single mode...
* Puma version: 5.6.4 (ruby 2.7.5-p203) ("Birdie's Version")
* Min threads: 0
* Max threads: 5
* Environment: development
* PID: 3420
* Listening on http://127.0.0.1:9292
* Listening on http://[::1]:9292
Use Ctrl-C to stop
127.0.0.1 - [10/Oct/2024:11:48:41 -0700] "GET /unauthenticated HTTP/1.1"
```

LB 보안그룹에 aranchni 포트(9292 포트)를
인바운드 규칙에 추가



OWASP-ZAP vs. Arachni

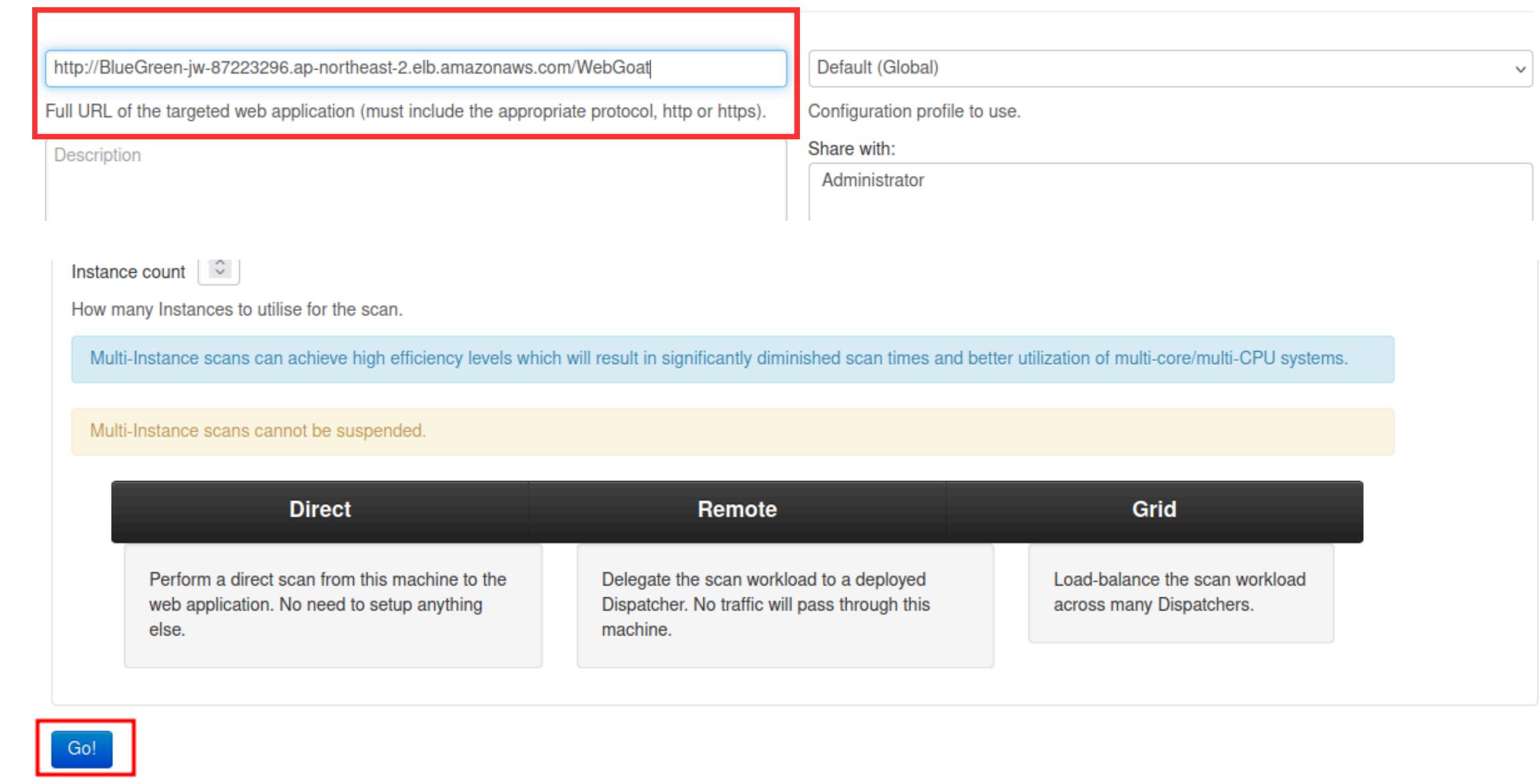
: OWASP-ZAP와 Arachni의 정량적 비교분석을 위한 테스트 구현

◆ Arachni

배포 성공한 로드밸런서 dns주소

Start a scan

The only thing you need to do is provide some basic information and make a simple choice about the type of scan you want to perform.



http://BlueGreen-jw-87223296.ap-northeast-2.elb.amazonaws.com/WebGoat

Full URL of the targeted web application (must include the appropriate protocol, http or https).

Description

Default (Global)

Configuration profile to use.

Share with:

Administrator

Instance count

How many Instances to utilise for the scan.

Multi-Instance scans can achieve high efficiency levels which will result in significantly diminished scan times and better utilization of multi-core/multi-CPU systems.

Multi-Instance scans cannot be suspended.

Direct

Remote

Grid

Perform a direct scan from this machine to the web application. No need to setup anything else.

Delegate the scan workload to a deployed Dispatcher. No traffic will pass through this machine.

Load-balance the scan workload across many Dispatchers.

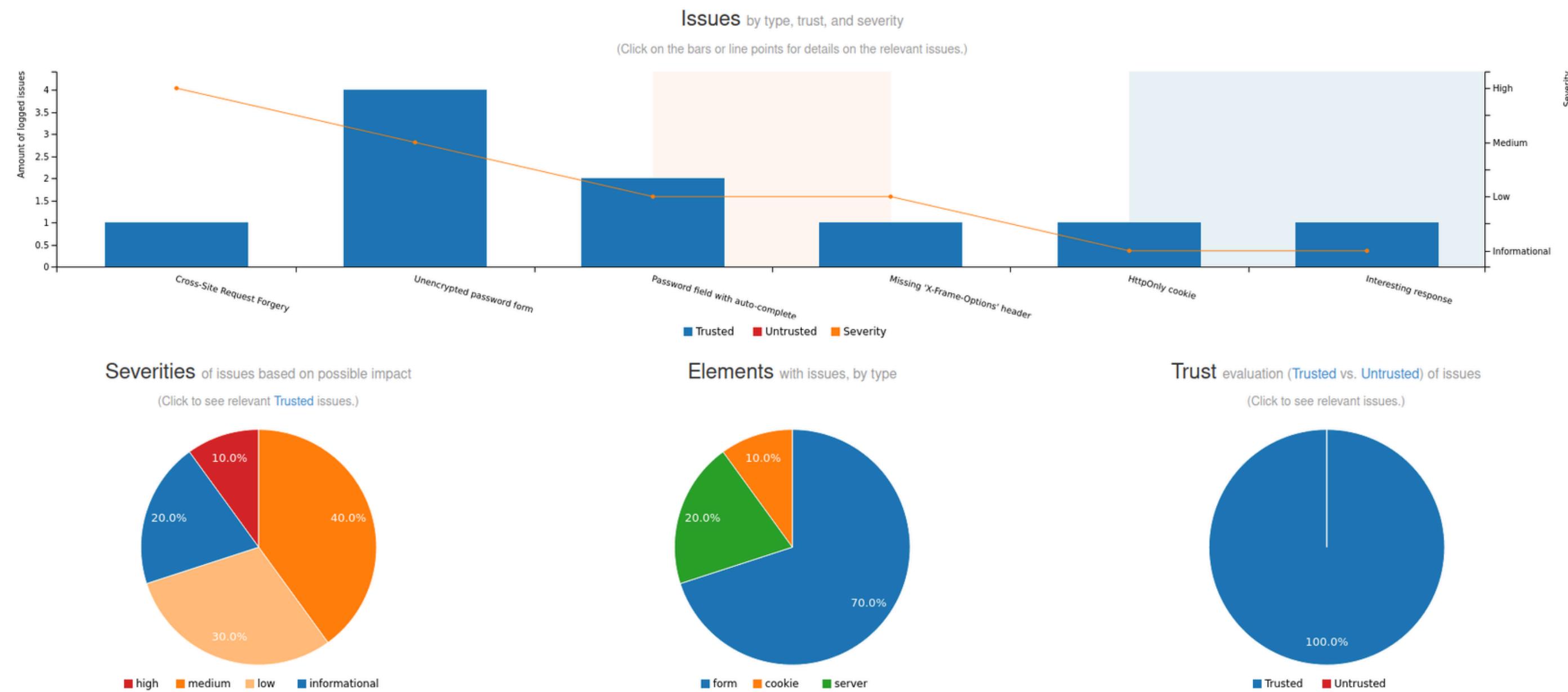
Go!

OWASP-ZAP vs. Arachni

: OWASP-ZAP와 Arachni의 정량적 비교분석을 위한 테스트 구현

◆ Arachni

시각화 그래프로 분석 결과 확인 가능



OWASP-ZAP vs. Arachni

: OWASP-ZAP와 Arachni의 정량적인 비교 분석 지표

	OWASP-ZAP	Arachni
Severity (심각도) 분류	High, Medium, Low, Informational, False Positives 총 5단계로 분류	High, Medium, Low, Informational 총 4단계로 분류
분석 소요 시간	약 2분	약 2분
총 탐지된 취약점 개수	6개	7개
제공 기능	각 취약점에 대한 해결책 명시	감지된 취약점의 구체적인 위치와 관련된 기술적 세부 사항을 제공하여, 수동 검토자가 취약점을 검토하고 해결할 수 있도록 도움

OWASP-ZAP vs. Arachni

: OWASP-ZAP와 Arachni의 정량적인 비교 분석 지표



VS



OWASP-ZAP vs. Arachni

: OWASP-ZAP와 Arachni의 정량적인 비교 분석 지표



VS

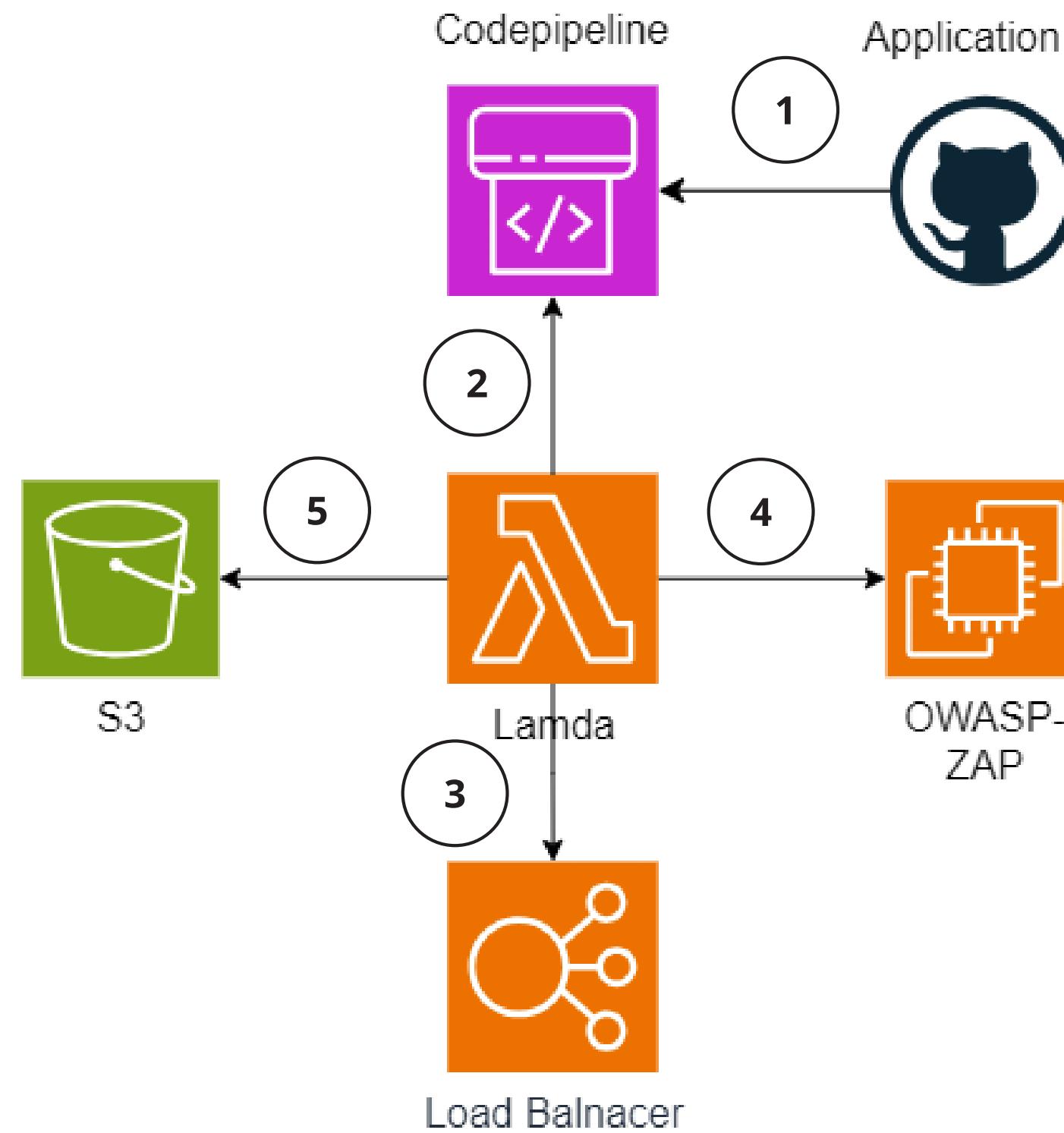


- 다양한 웹 취약점을 탐지
- 정기적인 업데이트와 OWASP 커뮤니티의 지원을 통해 지속적인 기능 개선
- 업데이트 중단으로 인해 Arachni는 최신 취약점을 반영하지 못한다는 단점

OWASP-ZAP vs. Arachni

: OWASP-ZAP와 Arachni의 정량적인 비교 분석 지표

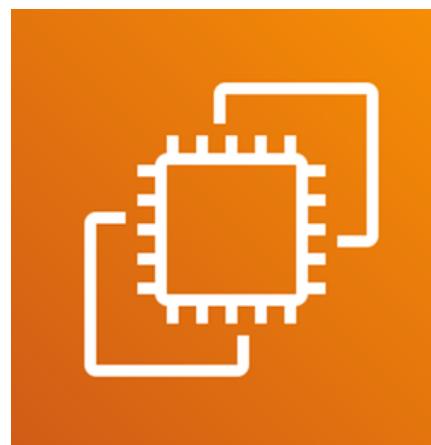
◆ OWASP-ZAP



| DAST CodePipeline 통합

: 최종 선정한 OWASP-ZAP를 CodePipeline에 통합한 자동화 구현

◆ OWASP-ZAP



Amazon EC2



```
docker run -u zap -p 8080:8080 -i zaproxy/zap-stable zap.sh -daemon -host 0.0.0.0 -port 8080 -config  
api.addrs.addr.name=.* -config api.addrs.addr.regex=true -config api.key=1234
```

| DAST CodePipeline 통합

: 최종 선정한 OWASP-ZAP를 CodePipeline에 통합한 자동화 구현

◆ OWASP-ZAP



Lambda

배포 후 생성된 로드밸런서 DNS 주소를 분석
→ 람다로 해당 DNS 주소 가져옴

Step 1.

Spider Scan을 통해 분석 정보를 가져오는 단계

```
# 스파이더링 실행
scanID = zap.spider.scan(target)
while int(zap.spider.status(scanID)) < 100:
    print('Spider progress %: {}'.format(zap.spider.status(scanID)))
```

| DAST CodePipeline 통합

: 최종 선정한 OWASP-ZAP를 CodePipeline에 통합한 자동화 구현

◆ OWASP-ZAP



Lambda

배포 후 생성된 로드밸런서 DNS 주소를 분석
→ 람다로 해당 DNS 주소 가져옴

Step 2.

Active Scan으로 취약점 분석을 진행하는 단계

```
# Active Scan 실행
print('Active Scanning target {}'.format(target))
active_scan_id = zap.ascan.scan(target)
while int(zap.ascan.status(active_scan_id)) < 100:
    print('Active Scan progress %: {}'.format(zap.ascan.status(active_scan_id)))
```

| DAST CodePipeline 통합

: 최종 선정한 OWASP-ZAP를 CodePipeline에 통합한 자동화 구현

◆ OWASP-ZAP



Lambda

배포 후 생성된 로드밸런서 DNS 주소를 분석
→ 람다로 해당 DNS 주소 가져옴

Step 3.

분석 후 json 형식의 Report를 생성하는 단계

```
# 취약점 스캔 결과 출력
print('Hosts: {}'.format(', '.join(zap.core.hosts)))
pprint(zap.core.alerts(baseurl=target))
# JSON 리포트 생성
report_json = zap.core.jsonreport()
with open(report_file_path, 'w') as report_file:
    report_file.write(report_json)
```

| DAST CodePipeline 통합

: 최종 선정한 OWASP-ZAP를 CodePipeline에 통합한 자동화 구현

◆ OWASP-ZAP



Lambda

배포 후 생성된 로드밸런서 DNS 주소를 분석
→ 람다로 해당 DNS 주소 가져옴

Step 4.

Report를 s3에 업로드하는 단계

```
# S3에 리포트 업로드
s3_report_key = 'zap_reports/zap_report.json'
s3_client.upload_file(report_file_path, s3_bucket_name, s3_report_key)
print('Report uploaded to S3 at {}{}'.format(s3_bucket_name, s3_report_key))
```

| DAST CodePipeline 통합

: 최종 선정한 OWASP-ZAP를 CodePipeline에 통합한 자동화 구현

◆ OWASP-ZAP



필요한 패키지 설치(레이어 추가)

- zap api clone
- request clone
- lambda_function.py

→ .zip으로 압축하여 람다에 업로드

```
PS C:\Users\USER\Downloads\lambda_function> ls

디렉터리 : C:\Users\USER\Downloads\lambda_function
```

Mode	LastWriteTime	Length	Name
----	-----	-----	-----
d----	2024-10-14 오 후 11:46		zap-api-python
d----	2024-10-14 오 후 11:46		zapv2
-a----	2024-10-14 오 후 11:54	2154	lambda_function.py

| DAST CodePipeline 통합

: 최종 선정한 OWASP-ZAP를 CodePipeline에 통합한 자동화 구현

◆ OWASP-ZAP

IAM 역할 생성

1. ec2 네트워크 인터페이스와 관련된 작업 권한

```
AWSLambdaVPCAccessExecutionRole

1 {{"Version": "2012-10-17",
2 "Statement": [
3 {
4 {
5 "Effect": "Allow",
6 "Action": [
7 "ec2:CreateNetworkInterface",
8 "ec2:DescribeNetworkInterfaces",
9 "ec2:DeleteNetworkInterface"
10 ],
11 "Resource": "*"
12 }
13 ]
14 }}
```

2. 람다 함수 codepipeline 통합 시 결과 업데이트 권한

```
CodePipeline

1 {{"Version": "2012-10-17",
2 "Statement": [
3 {
4 {
5 "Effect": "Allow",
6 "Action": [
7 "codepipeline:PutJobFailureResult",
8 "codepipeline:PutJobSuccessResult"
9 ],
10 "Resource": "*"
11 }
12 ]
13 }}
```

| DAST CodePipeline 통합

: 최종 선정한 OWASP-ZAP를 CodePipeline에 통합한 자동화 구현

◆ OWASP-ZAP

IAM 역할 생성

3. 로드밸런서 정보 가져오는 권한

```
LB
1 [ {
2   "Version": "2012-10-17",
3   "Statement": [
4     {
5       "Effect": "Allow",
6       "Action": "elasticloadbalancing:DescribeLoadBalancers",
7       "Resource": "*"
8     }
9   ]
10 } ]
```

4. s3 버킷 업로드 허용 권한

```
S3
1 [ {
2   "Version": "2012-10-17",
3   "Statement": [
4     {
5       "Effect": "Allow",
6       "Action": [
7         "s3:PutObject"
8       ],
9       "Resource": "arn:aws:s3:::your-bucket-name/zap_reports/*"
10     }
11   ]
12 } ]
```

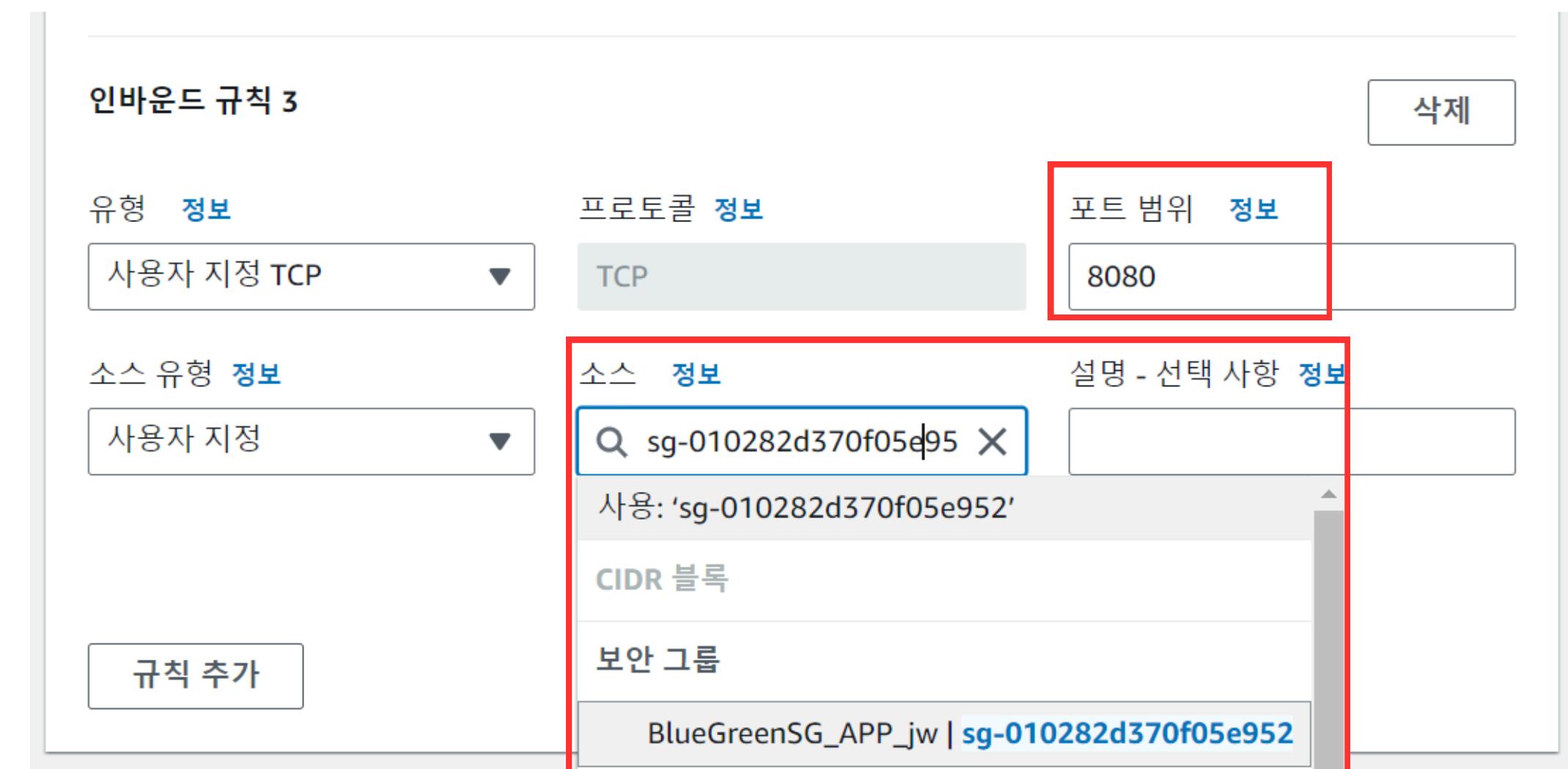
| DAST CodePipeline 통합

: 최종 선정한 OWASP-ZAP를 CodePipeline에 통합한 자동화 구현

◆ OWASP-ZAP

vpc 보안그룹 설정

ZAP 서버와 다른 인스턴스 간의 통신을 허용하기 위한
Security Chain 설정 필요



| DAST CodePipeline 통합

: 최종 선정한 OWASP-ZAP를 CodePipeline에 통합한 자동화 구현

◆ OWASP-ZAP



Lambda

람다에서 vpc 설정

VPC 정보 편집

VPC
vpc-06c90836fbda77c4d (10.0.0.0/16) | BlueGreen-jw

서브넷

- IPv6 트래픽 허용 = false
- subnet-0b98c7e84f46eac0e (10.0.3.0/24) | ap-northeast-2c, PrivateSubnet2_jw
- subnet-0ddce741b93403224 (10.0.2.0/24) | ap-northeast-2a, PrivateSubnet1_jw

보안 그룹

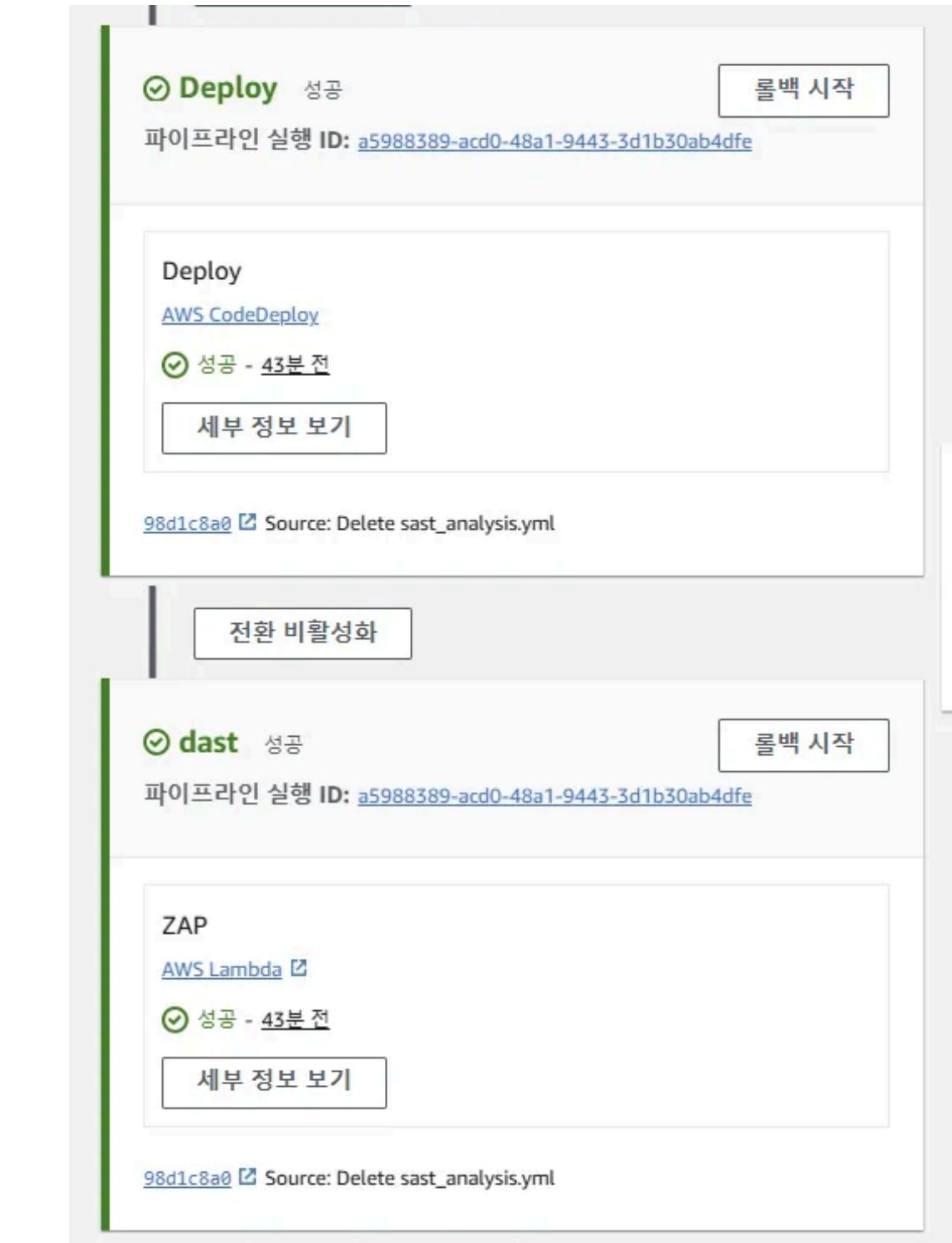
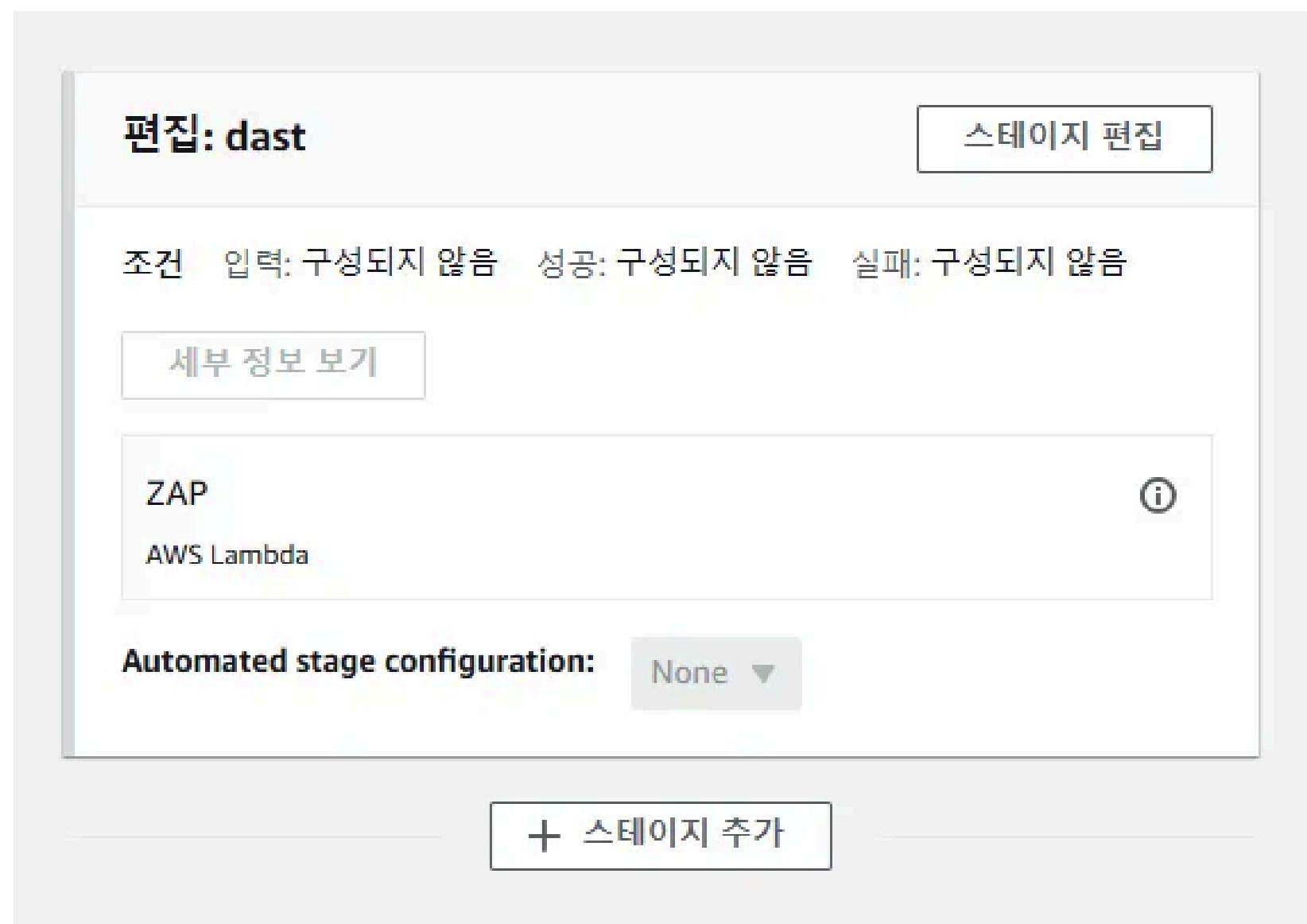
- sg-010282d370f05e952 (BlueGreenSG_APP_jw)

| DAST CodePipeline 통합

: 최종 선정한 OWASP-ZAP를 CodePipeline에 통합한 자동화 구현

◆ OWASP-ZAP

파이프라인 통합



DAST CodePipeline 통합

: 최종 선정한 OWASP-ZAP를 CodePipeline에 통합한 자동화 구현

◆ OWASP-ZAP



S3 버킷에서 Report 확인

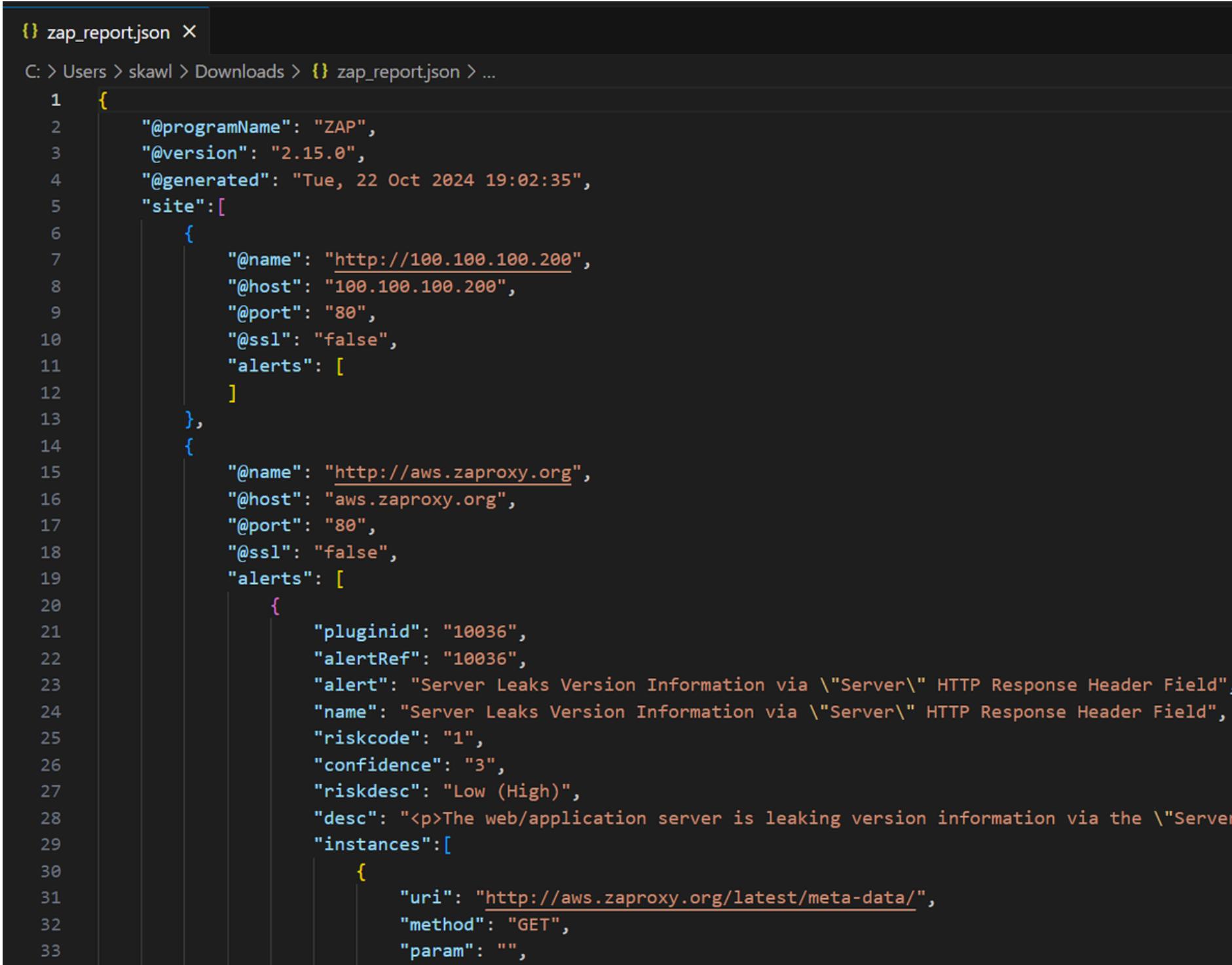
The screenshot shows the AWS S3 console interface. At the top, there is a navigation bar with various buttons: 'C' (Copy), 'S3 URI 복사' (Copy S3 URI), 'URL 복사' (Copy URL), '다운로드' (Download), '열기' (Open), '삭제' (Delete), '작업' (Actions), '폴더 만들기' (Create Folder), and '업로드' (Upload). Below the navigation bar, a message in Korean states: '객체 (1) 정보' (Object (1) Information), ' 객체는 Amazon S3에 저장되어 있는 기본 엔터티입니다. Amazon S3 인벤토리 를 사용하여 버킷에 있는 모든 객체의 목록을 얻을 수 있습니다. 다른 사용자가 객체에 액세스할 수 있게 하려면 명시적으로 권한을 부여해야 합니다. 자세히 알아보기' (The object is a basic entity stored in Amazon S3. You can use the Amazon S3 Inventory to get a list of all objects in the bucket. If you want other users to access the object, you must grant them explicit permissions. See details). A search bar below the message contains the text '접두사로 객체 찾기' (Find objects by prefix). The main table displays one object: 'zap_reports/' (Type: 폴더). At the bottom of the table, there is a note: '작성자' (Owner) and '작성일' (Last modified). The date shown is '2024. 10. 23. am 4:02:37 AM KST' and the size is '25.5KB'.

The screenshot shows the AWS S3 console interface, identical to the one above but with a different object list. The table now displays a single object: 'zap_report.json' (Type: json). The details for this object are: '작성자' (Owner) and '작성일' (Last modified). The date shown is '2024. 10. 23. am 4:02:37 AM KST' and the size is '25.5KB'.

DAST CodePipeline 통합

: 최종 선정한 OWASP-ZAP를 CodePipeline에 통합한 자동화 구현

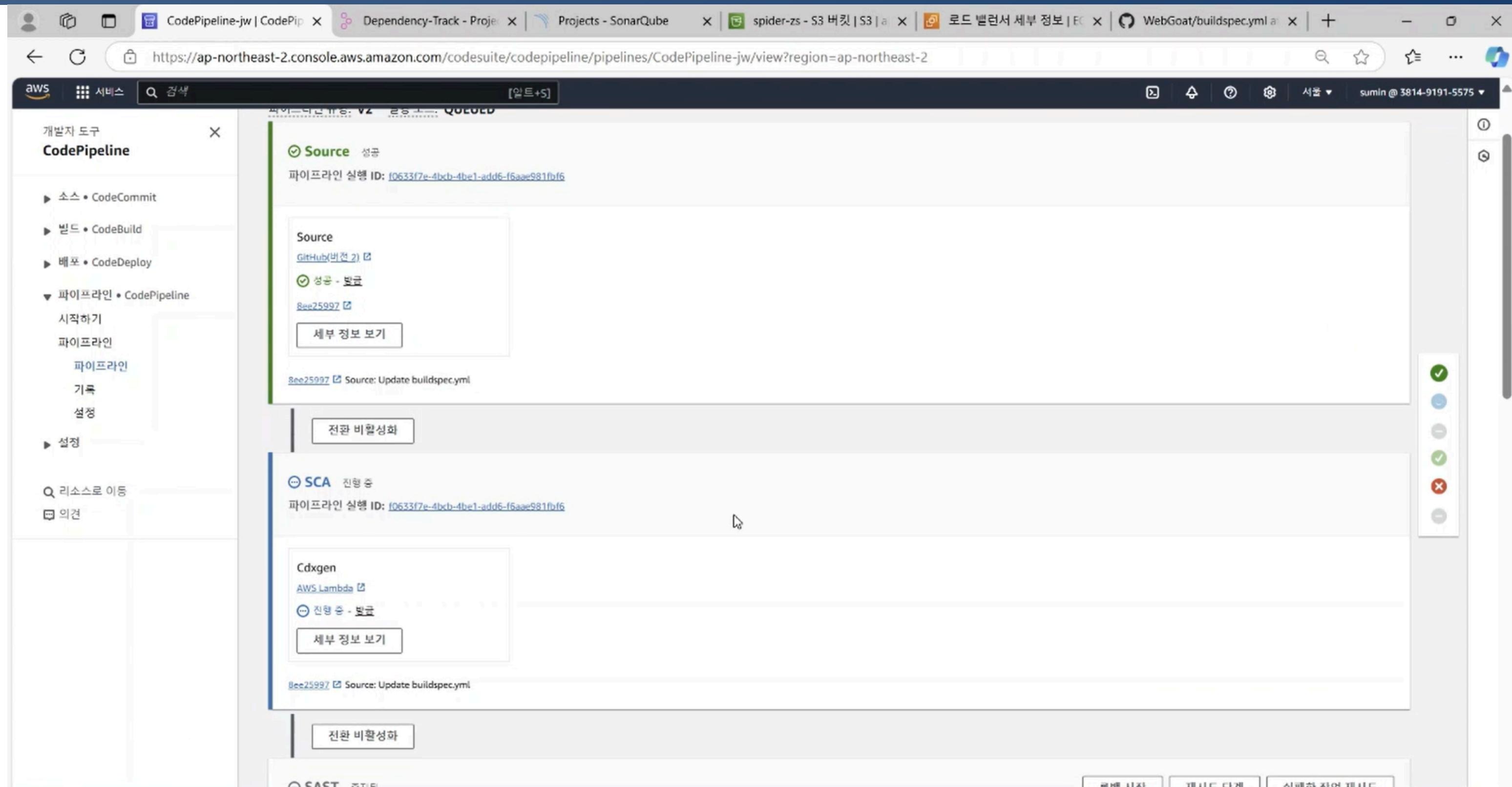
◆ OWASP-ZAP



```
{} zap_report.json x
C: > Users > skawl > Downloads > {} zap_report.json > ...
1  {
2      "@programName": "ZAP",
3      "@version": "2.15.0",
4      "@generated": "Tue, 22 Oct 2024 19:02:35",
5      "site": [
6          {
7              "@name": "http://100.100.100.200",
8              "@host": "100.100.100.200",
9              "@port": "80",
10             "@ssl": "false",
11             "alerts": [
12                 ]
13             },
14             {
15                 "@name": "http://aws.zaproxy.org",
16                 "@host": "aws.zaproxy.org",
17                 "@port": "80",
18                 "@ssl": "false",
19                 "alerts": [
20                     {
21                         "pluginid": "10036",
22                         "alertRef": "10036",
23                         "alert": "Server Leaks Version Information via \"Server\" HTTP Response Header Field",
24                         "name": "Server Leaks Version Information via \"Server\" HTTP Response Header Field",
25                         "riskcode": "1",
26                         "confidence": "3",
27                         "riskdesc": "Low (High)",
28                         "desc": "<p>The web/application server is leaking version information via the \"Server\" HTTP Response Header Field</p>",
29                         "instances": [
30                             {
31                                 "uri": "http://aws.zaproxy.org/latest/meta-data/",
32                                 "method": "GET",
33                                 "param": ""
34                             }
35                         ]
36                     }
37                 ]
38             }
39         ]
40     }
41 }
```

시연 영상

| AWS DevOps 환경에서의 CICD Security Best..? Practice





INFINITE

Q & A



INFINITE

Thank you
감사합니다.

Team. INFINITE

멘토 | 권현준 멘토님
팀원 | 김수민 김수민 남지우 천예슬