제 목	04장 백엔드 CI/CD 적용하기	
상세내용	백엔드(Spring Boot) 프로젝트에 CI/CD 적용하기	

방법 1 - 개인 프로젝트에서 많이 쓰는 CI/CD 구축 방법 (Github Actions)

✓ 전체적인 흐름

✓ 장점

- git pull을 활용해서 변경된 부분의 프로젝트 코드에 대해서만 업데이트 하기 때문에 CI/CD 속도가 빠르다.

  - 대부분의 CI/CD 방식들은 전체 프로젝트를 통째로 갈아끼우는 방식을 사용한다.

- CI/CD 툴로 Github Actions만 사용하기 때문에 인프라 구조가 복잡하지 않고 간단하다.

✓ 단점

- 빌드 작업을 EC2에서 직접 진행하기 때문에 운영하고 있는 서버의 성능에 영향을 미칠 수 있다.

- Github 계정 정보가 해당 EC2에 저장되기 때문에 개인 프로젝트 또는 믿을만한 사람들과 같이 진행하는 토이 프로젝트에서만 사용해야 한다.

✓ 이 방법은 언제 주로 쓰는 지

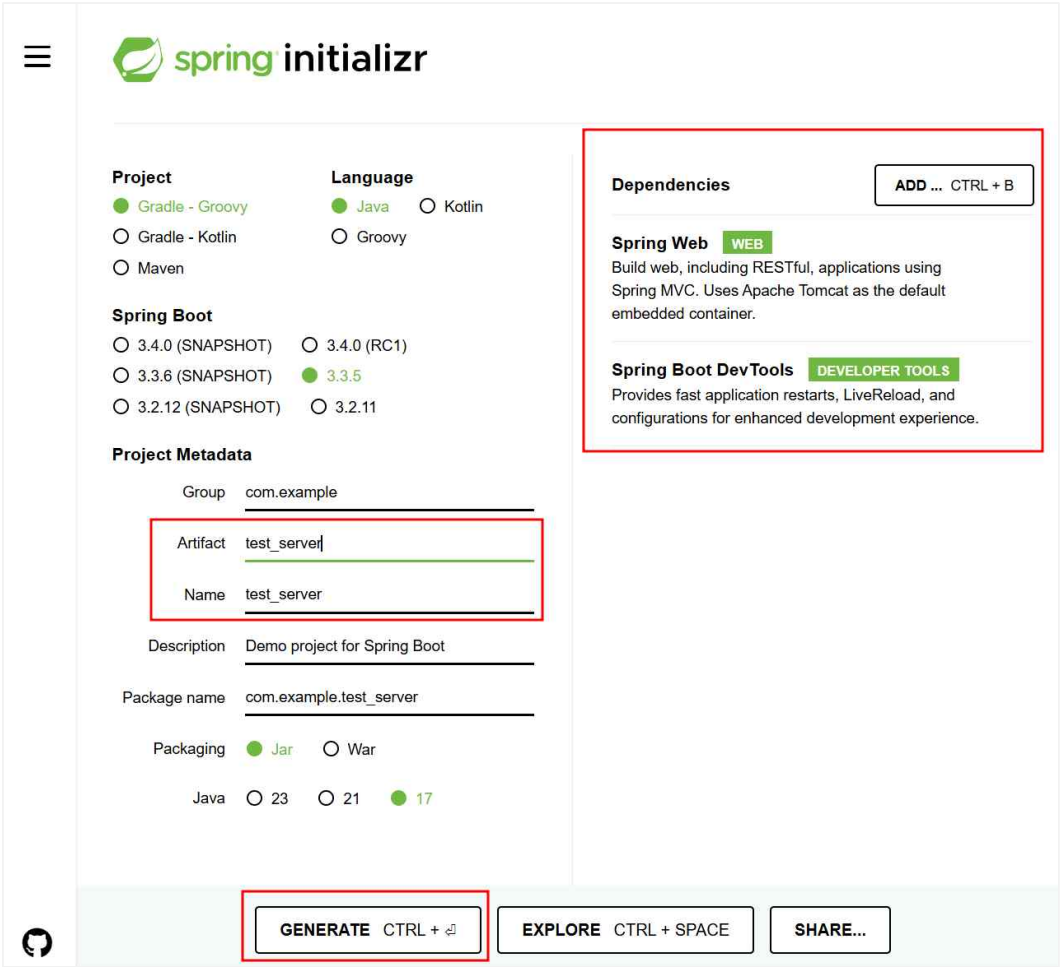
- 주로 개인 프로젝트에서 CI/CD를 심플하고 빠르게 적용시키고 싶을 때 적용한다.

[실습] 개인 프로젝트에서 많이 쓰는 CI/CD 구축 방법

✓ 1. Spring Boot 프로젝트 셋팅

1. 프로젝트 셋팅

▶ spring demo 다운로드: https://start.spring.io/



※ JDK 17로 설치하자. 이후 과정이 전부 JDK 17에 맞춰서 진행을 할 예정이다.

2. 간단한 코드 작성

AppController

```
@RestController
public class AppController {
  @GetMapping("/")
  public String home() {
    return "Hello, World!";
  }
}
```

✓ 2. Github에 프로젝트 올리기

1. Github에서 Repository 만들기

   실무에서는 Private Repository를 쓰는 경우가 많으므로 Private Repository로 만들자.

2. Github Repository에 프로젝트 코드 올리기

```
$ cd instagram-server
$ git init
$ git add .
$ git commit -m "first commit"
$ git branch -M main
$ git remote add origin _______
$ git push -u origin main
```

✓ 3. EC2에 들어가서 기본 환경 구성

1. EC2 구성하기

   프리티어인 t2.micro로 인스턴스를 생성하게 되면 메모리 부족으로 인해 CI/CD 과정 도중에 EC2가 멈추는 현상이 가끔 발생한다. 따라서 t3a.small로 인스턴스를 생성할 예정이다.

   만약 비용이 부담스러워서 프리티어 t2.micro를 유지한 채로 테스트하고 싶다면 아래 링크를 참고하자.

   ▶ EC2 메모리 부족 현상[swap 생성] : https://bssssss.tistory.com/1189

2. 보안그룹 8080번 포트 열기

   Spring Boot는 기본적으로 8080번 포트에서 실행된다.

3. JDK 설치

```
$ sudo apt update && /
sudo apt install openjdk-17-jdk -y

$ java -version # 잘 설치됐는 지 확인
```

4. Git clone을 활용해 프로젝트 다운받기

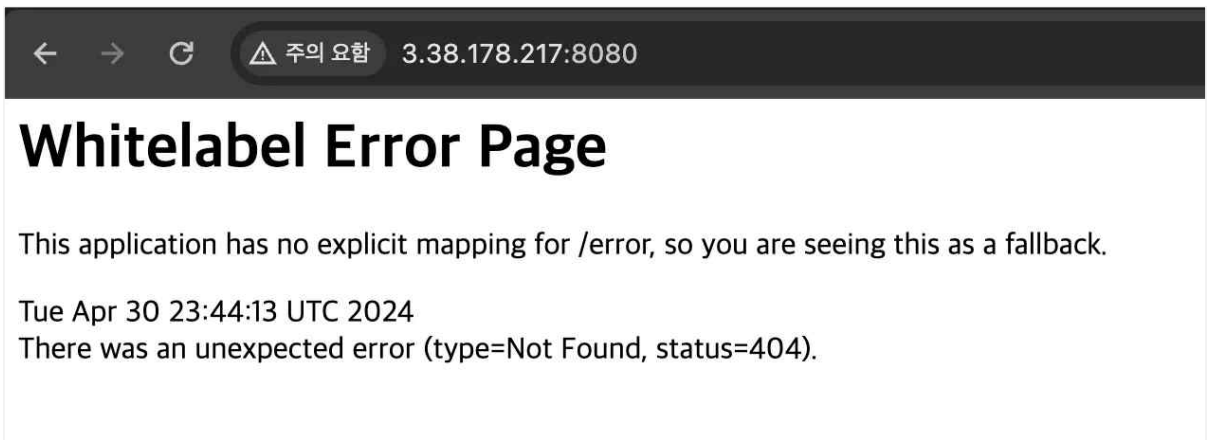
```
$ git clone {git repository clone 주소}
```

참고) Private Repository를 clone할 경우 Github의 계정과 비밀번호를 입력해야 한다. 이 때, 비밀번호는 실제 Github 계정의 비밀번호가 아닌 Github 토큰을 입력해야 한다. 토큰을 발급받는 방법은 아래 링크를 참고하자.

▶ github 토큰 발급하기: https://tinyurl.com/2xowkctv

5. EC2에서 clone 받은 서버가 잘 작동하는 지 확인하기

```
$ cd {프로젝트 경로}
$ ./gradlew clean build
$ cd build/libs
$ nohup java -jar _________.jar &

$ sudo lsof -i:8080 # 8080번 포트에 Spring Boot가 실행되고 있는 지 확인
```



✓ 4. 실제 코드가 업데이트 될 때 어떤 과정을 거쳐서 배포하는 지 짚어보기

1. 내 컴퓨터에서 새로운 코드 작성
2. Commit 찍은 뒤 Github에 Push하기
3. EC2에 들어가서 Git Pull 받기

```
$ cd {프로젝트 경로}
$ git pull origin main
```

4. 빌드 후 재배포하기

```
$ sudo lsof -i:8080 # 8080번 포트에 Spring Boot가 실행되고 있는 지 확인
$ sudo fuser -k -n tcp 8080 # 8080번 포트에 실행되고 있는 프로세스 종료
$ ./gradlew clean build
$ cd build/libs
$ nohup java -jar _________.jar &
```

✓ 5. 매번 Github 계정과 비밀번호를 치는 과정 없애기

배포를 할 때마다 Github 계정과 비밀번호를 일일이 치는 과정이 포함되어 있으면 배포를 자동화할 수가 없다. 따라서 최초 한 번만 작성하고 그 이후에는 Github 계정과 비밀번호를 입력하지 않아도 되게 만들어보자.

```
$ git config --global credential.helper store
$ git pull origin main
# Github 계정 및 비밀번호 입력

$ git pull origin main # 더 이상 비밀번호를 안 묻는 걸 확인할 수 있다.
```

▶ 참고: Git, Pull/Push할 때 id password 묻지 않게 하기
=> https://pinedance.github.io/blog/2019/05/29/Git-Credential

※ 이 방식은 **~/.git-credentials**에 로그인 정보를 저장해둠으로써 github 계정과 비밀번호를 따로 묻지 않는 방식이다. 이 방식의 단점은 EC2에 접근할 수 있는 모든 사용자가 내 Github 정보를 볼 수 있다는 점이 단점이다.

✓ 6. 지금까지 했던 코드 배포 과정을 자동화하기

1. **.github/workflows/deploy.yml** 작성하기

```
name: Deploy To EC2

on:
  push:
    branches:
      - main

jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - name: SSH로 EC2에 접속하기
        uses: appleboy/ssh-action@v1.0.3
        with:
          host: ${{ secrets.EC2_HOST }} # EC2의 주소
          username: ${{ secrets.EC2_USERNAME }} # EC2 접속 username
          key: ${{ secrets.EC2_PRIVATE_KEY }} # EC2의 Key 파일의 내부 텍스트
          # 아래 script 중 실패하는 명령이 하나라도 있으면 실패로 처리
          script_stop: true
          script: |
            # 여기 경로는 자신의 EC2에 맞는 경로로 재작성하기
            cd /home/ubuntu/instagram-server
            git pull origin main
            ./gradlew clean build
            sudo fuser -k -n tcp 8080 || true
            # || true를 붙인 이유는 8080에 종료시킬 프로세스가 없더라도 실패로 처리하지 않기 위해서이다.
            # jar 파일을 실행시키는 명령어이다. 그리고 발생하는 로그들을 ./output.log 파일에 남기는 명령어이다.
            nohup java -jar build/libs/*SNAPSHOT.jar > ./output.log 2>&1 &
```

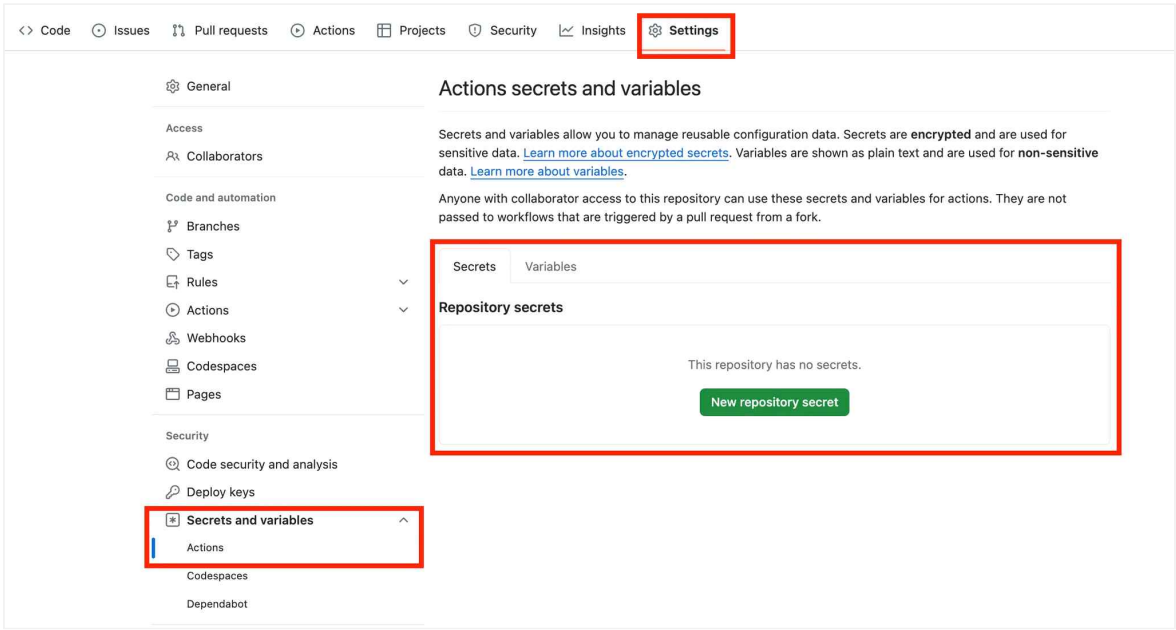
- SSH란 다른 컴퓨터로 원격 접속하는 방식을 의미한다.
- appleboy/ssh-action의 공식 문서

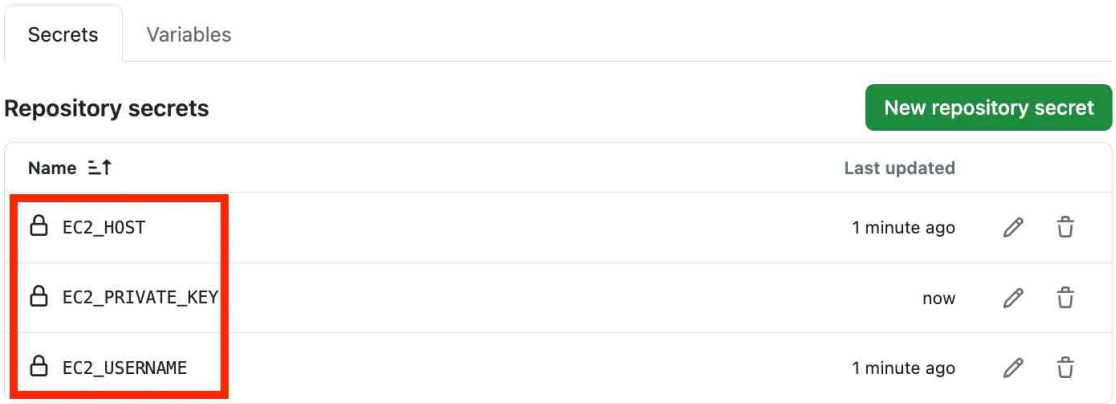
  ▶ https://github.com/marketplace/actions/ssh-remote-commands

- nohup ... & 부분의 코드를 수동으로 배포할 때와 조금 다르게 작성한 이유

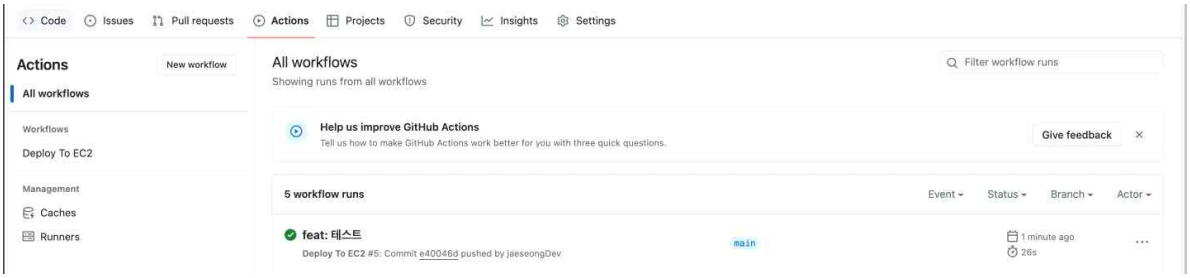
  ▶ https://github.com/appleboy/ssh-action/issues/67

2. Github에 Secret값 넣어주기





3. Github에 Push해서 Github Actions 잘 작동하는 지 확인하기



4. 실제 서버로 접속해서 잘 작동하는 지 확인하기

세팅이 되었는지 확인

✓ 7. application.yml 파일 넣는 과정 자동화시키기

현업 프로젝트에서는 민감한 값을 따로 application.yml로 분리하는 경우가 많다. 민감한 값이기에 .gitignore에 추가해서 application.yml가 버전관리 되지 않게 셋팅한다. 이 때문에 배포를 할 때 application.yml를 따로 넣어주어야 하는 귀찮은 과정이 포함된다. 이 과정을 자동화시켜보자

1. .gitignore에 application.yml 추가

.gitignore

```
...

application.yml
```

2. application.yml 파일 만들기

src/main/resources/application.yml

```
aws:
  access-key: ABCDEFG
  secret-key: HIJKLMN
```

3. Github에 resources 폴더가 push 되도록 임의의 파일 하나 만들어주기

src/main/resources/empty.txt

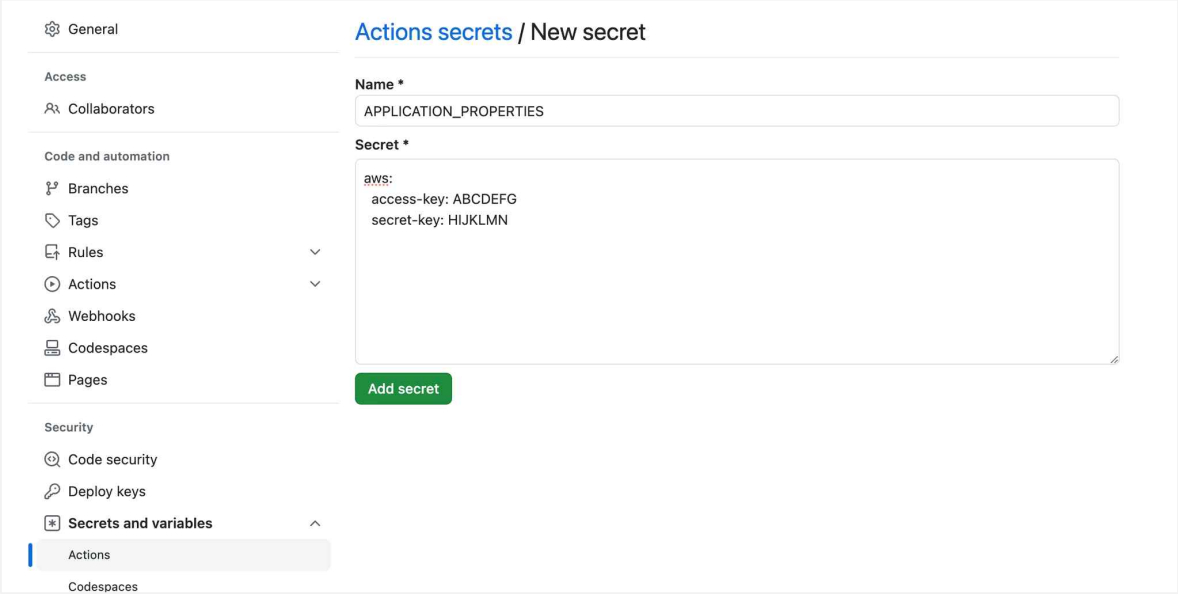
```

```

4. Gitbub Actions 코드 수정하기

.github/workflows/deploy.yml

```
name: Deploy To EC2

on:
  push:
    branches:
      - main

jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - name: SSH로 EC2에 접속하기
        uses: appleboy/ssh-action@v1.0.3
        env:
          APPLICATION_PROPERTIES: ${{ secrets.APPLICATION_PROPERTIES }}
        with:
          host: ${{ secrets.EC2_HOST }} # EC2의 주소
          username: ${{ secrets.EC2_USERNAME }} # EC2 접속 username
          key: ${{ secrets.EC2_PRIVATE_KEY }} # EC2의 Key 파일의 내부 텍스트
          envs: APPLICATION_PROPERTIES
          script_stop: true
          # 아래 script 중 실패하는 명령이 하나라도 있으면 실패로 처리

          script: |
            cd /home/ubuntu/instagram-server
            # 여기 경로는 자신의 EC2에 맞는 경로로 재작성하기
            rm -rf src/main/resources/application.yml
            git pull origin main
            echo "$APPLICATION_PROPERTIES" > src/main/resources/application.yml
            ./gradlew clean build
            sudo fuser -k -n tcp 8080 || true
            # || true를 붙인 이유는 8080에 종료시킬 프로세스가 없더라도 실패로
처리하지 않기 위해서이다.
            nohup java -jar build/libs/*SNAPSHOT.jar > ./output.log 2>&1 &
```

5. Github에 Secret 값 넣어주기



6. 실제 EC2에 application.yml 파일도 같이 배포 잘 됐는 지 체크하기

✓ 8. 테스트 코드 실패하면 CI/CD 과정이 실패하는 지 확인하기

./gradlew clean build의 과정에 테스트 코드를 실행시키는 과정이 포함되어 있다. CI/CD를 할 때 테스트가 실패했는데 자동으로 배포가 되면 안 된다. 테스트를 실패하면 자동 배포가 진행되지 않고 중단되는 지 확인해보자.

1. Spring Boot 프로젝트에 내장되어 있는 테스트 코드가 잘 작동하는 지 실행해보기

```
$ ./gradlew test
```

2. 테스트 실패하게 수정하기

test/.../__ApplicationTests.java

```
package com.example.demo;

import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest
class DemoApplicationTests {

	@Test
	void contextLoads() {
		throw new RuntimeException("Fail!");
	}
}
```

3. 테스트 코드 실패하면 배포가 안 되는 지 확인해보기

```
./gradlew clean build
sudo fuser -k -n tcp 8080 || true # || true를 붙인 이유는 8080에 종료시킬 프로세스가 없더라도 실패로 처리하지 않기 위해서이다.
nohup java -jar build/libs/*SNAPSHOT.jar > ./output.log 2>&1 &
======END======
err: From https://github.com/jaeseongDev/spring-test
err:  * branch            main       -> FETCH_HEAD
err:    55427a4..14c35a9  main       -> origin/main
out: Updating 55427a4..14c35a9
out: Fast-forward
out:  src/test/java/com/example/demo/DemoApplicationTests.java | 1 +
out:  1 file changed, 1 insertion(+)
out: > Task :clean
out: > Task :compileJava
out: > Task :processResources
out: > Task :classes
out: > Task :resolveMainClassName
out: > Task :bootJar
out: > Task :jar
out: > Task :assemble
out: > Task :compileTestJava
out: > Task :processTestResources NO-SOURCE
out: > Task :testClasses
err: OpenJDK 64-Bit Server VM warning: Sharing is only supported for boot loader classes because bootstrap classpath has been appended
out: > Task :test
err: 1 test completed, 1 failed
out: DemoApplicationTests > contextLoads() FAILED
out:     java.lang.RuntimeException at DemoApplicationTests.java:11
err: FAILURE: Build failed with an exception.
out: > Task :test FAILED
err: * What went wrong:
err: Execution failed for task ':test'.
out: Deprecated Gradle features were used in this build, making it incompatible with Gradle 9.0.
err: > There were failing tests. See the report at: file:///home/***/spring-test/build/reports/tests/test/index.html
err: * Try:
err: > Run with --scan to get full insights.
err: BUILD FAILED in 9s
out: You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own scripts or plugins.
out: For more on this, please refer to https://docs.gradle.org/8.7/userguide/command_line_interface.html#sec:command_line_warnings in the Gradle documentation.
out: 8 actionable tasks: 8 executed
2024/05/01 11:21:18 Process exited with status 1
```

ㄴ

방법 2-일반 프로젝트에서 많이 쓰는 CI/CD 구축 방법 (Github Actions, SCP)

✓ 전체적인 흐름

✓ 장점

- 빌드 작업을 Github Actions에서 하기 때문에 운영하고 있는 서버의 성능에 영향을 거의 주지 않는다.
- CI/CD 툴로 Github Actions만 사용하기 때문에 인프라 구조가 복잡하지 않고 간단하다.

✓ 단점

- 무중단 배포를 구현하거나 여러 EC2 인스턴스에 배포를 해야 하는 상황이라면, 직접 Github Actions에 스크립트를 작성해서 구현해야 한다. 직접 구현을 해보면 알겠지만 생각보다 꽤 복잡하다.

✓ 이 방법은 언제 주로 쓰는 지

- 현업에서 초기 서비스를 구축할 때 이 방법을 많이 활용한다.

  처음 서비스를 구현할 때는 대규모 서비스에 적합한 구조로 구현하지 않는다. 즉, 오버 엔지니어링을 하지 않는다. 확장의 필요성이 있다고 느끼는 시점에 인프라를 고도화하기 시작한다. 왜냐하면 복잡한 인프라 구조를 갖추고 관리하는 건 생각보다 여러 측면에서 신경쓸 게 많아지기 때문이다

  - 인프라 구조를 변경할 때 시간이 많이 들어감
  - 에러가 발생했을 때 트러블 슈팅의 어려움
  - 팀원이 인프라 구조를 이해하기 어려워 함
  - 기능을 추가하거나 수정할 때 더 많은 시간이 들어감
  - 금전적인 비용이 더 많이 발생

[실습] 일반 프로젝트에서 많이 쓰는 CI/CD 구축 방법

✅ 이전 실습했던 내용 정리

- 서버 종료
- 프로젝트 폴더 삭제

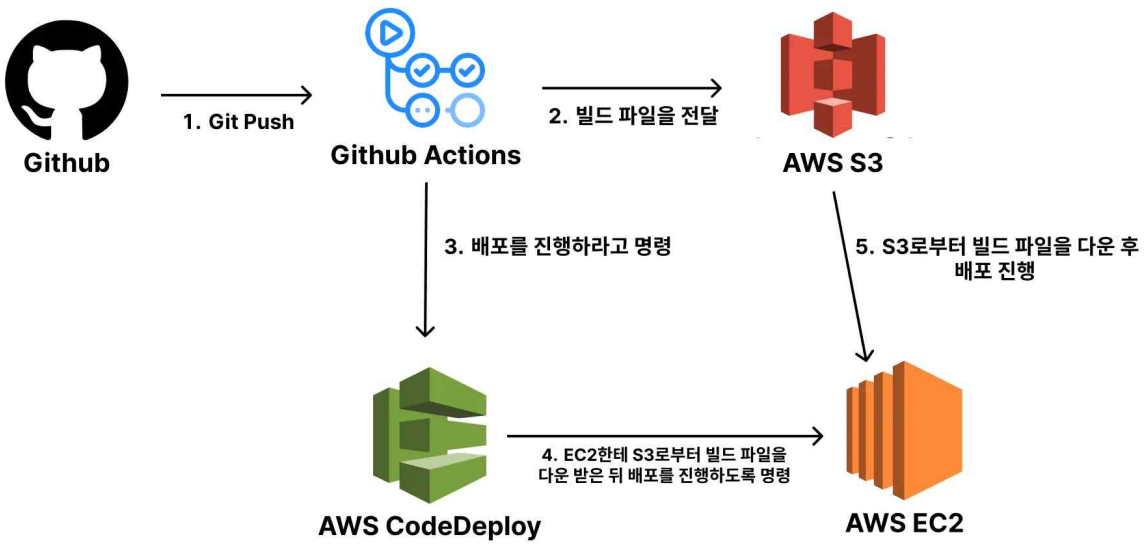
✅ Github Actions 코드 수정

```
name: Deploy To EC2

on:
  push:
    branches:
      - main

jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - name: Github Repository 파일 불러오기
        uses: actions/checkout@v4

      - name: JDK 17버전 설치
        uses: actions/setup-java@v4
        with:
          distribution: temurin
          java-version: 17

      - name: application.yml 파일 만들기
        run: echo "${{ secrets.APPLICATION_PROPERTIES }}" > ./src/main/resources/application.yml

      - name: 테스트 및 빌드하기
        run: ./gradlew clean build

      - name: 빌드된 파일 이름 변경하기
        run: mv ./build/libs/*SNAPSHOT.jar ./project.jar
```

```
      - name: SCP로 EC2에 빌드된 파일 전송하기
        uses: appleboy/scp-action@v0.1.7
        with:
          host: ${{ secrets.EC2_HOST }}
          username: ${{ secrets.EC2_USERNAME }}
          key: ${{ secrets.EC2_PRIVATE_KEY }}
          source: project.jar
          target: /home/ubuntu/instagram-server/tobe

      - name: SSH로 EC2에 접속하기
        uses: appleboy/ssh-action@v1.0.3
        with:
          host: ${{ secrets.EC2_HOST }}
          username: ${{ secrets.EC2_USERNAME }}
          key: ${{ secrets.EC2_PRIVATE_KEY }}
          script_stop: true
          script: |
            rm -rf /home/ubuntu/instagram-server/current
            mkdir /home/ubuntu/instagram-server/current
            mv /home/ubuntu/instagram-server/tobe/project.jar
/home/ubuntu/instagram-server/current/project.jar
            cd /home/ubuntu/instagram-server/current
            sudo fuser -k -n tcp 8080 || true
            nohup java -jar project.jar > ./output.log 2>&1 &
            rm -rf /home/ubuntu/instagram-server/tobe
```

방법 3 - 확장성을 고려한 프로젝트에서 많이 쓰는 CI/CD 구축 방법 (Code Deploy)

✓ 전체적인 흐름



✓ CodeDeploy를 사용하는 이유

- CodeDeploy는 수많은 AWS EC2에 배포를 쉽게 할 수 있도록 도와준다.
- CodeDeploy에 무중단 배포 기능이 내재되어 있어 손쉽게 무중단 배포를 진행할 수 있다.
- 이 외에도 다양한 장점이 존재한다.

▶ https://docs.aws.amazon.com/ko_kr/codedeploy/latest/userguide/welcome.html

✓ 장점

- 서버가 여러 대이더라도 쉽게 자동 배포를 구축할 수 있다.
- 쉽게 무중단 배포를 적용시킬 수 있다.

✓ 단점

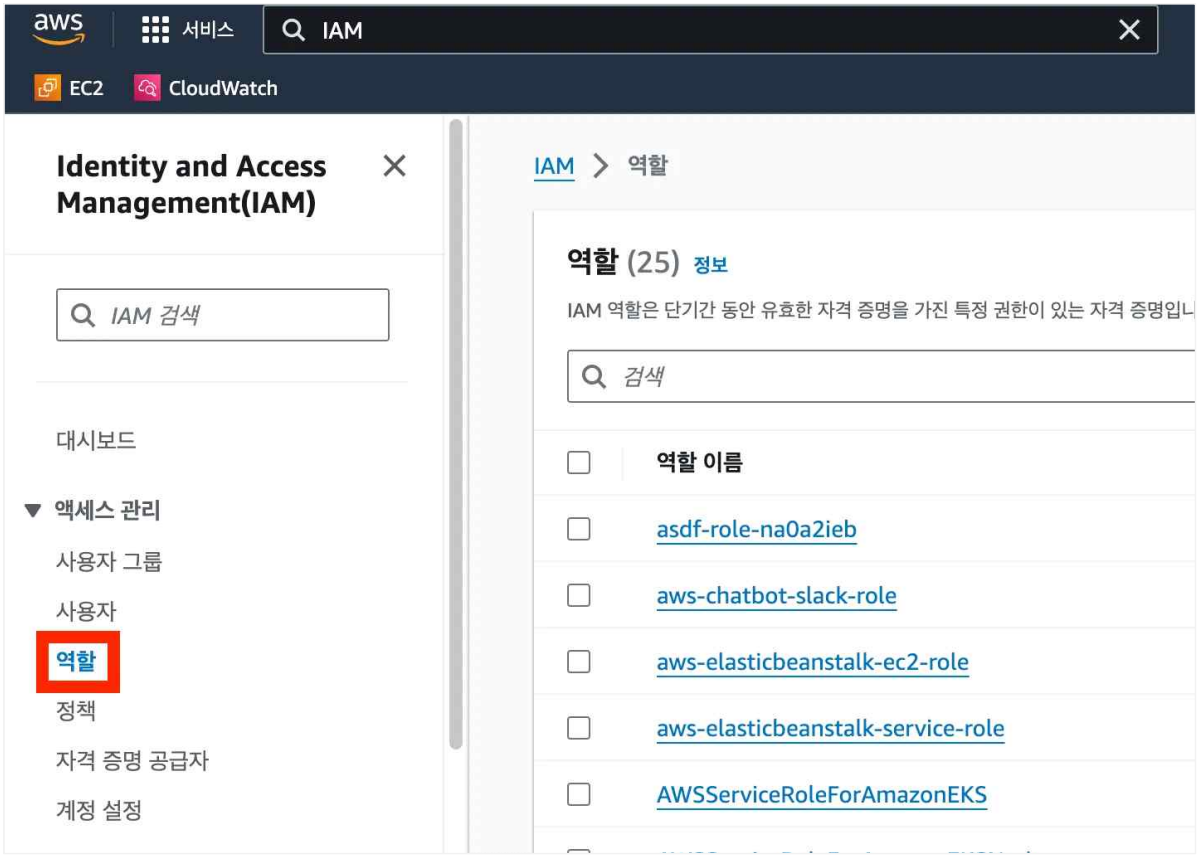
- CodeDeploy를 사용함으로써 인프라 구조가 복잡해졌다. 구조가 복잡해짐에 따라 관리 비용, 유지보수 비용, 난이도, 트러블 슈팅 어려움, 복잡도가 증가했다.

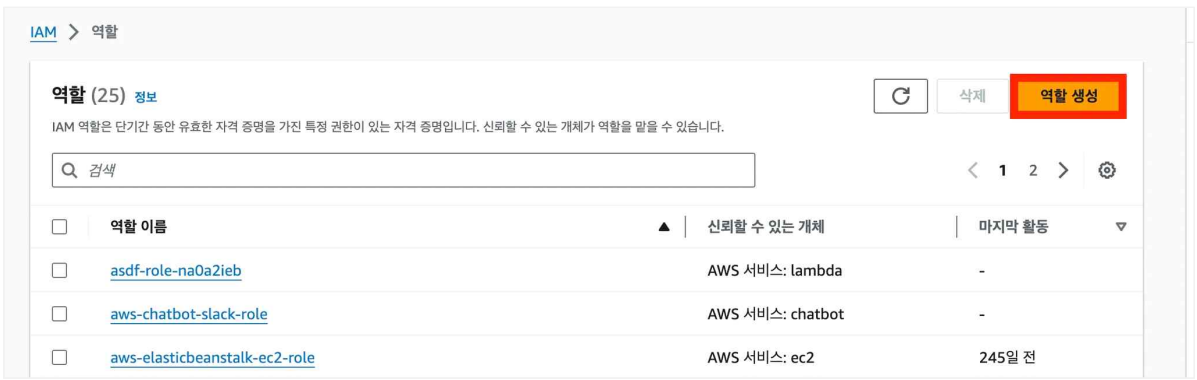
✓ 이 방법은 언제 주로 쓰는 지

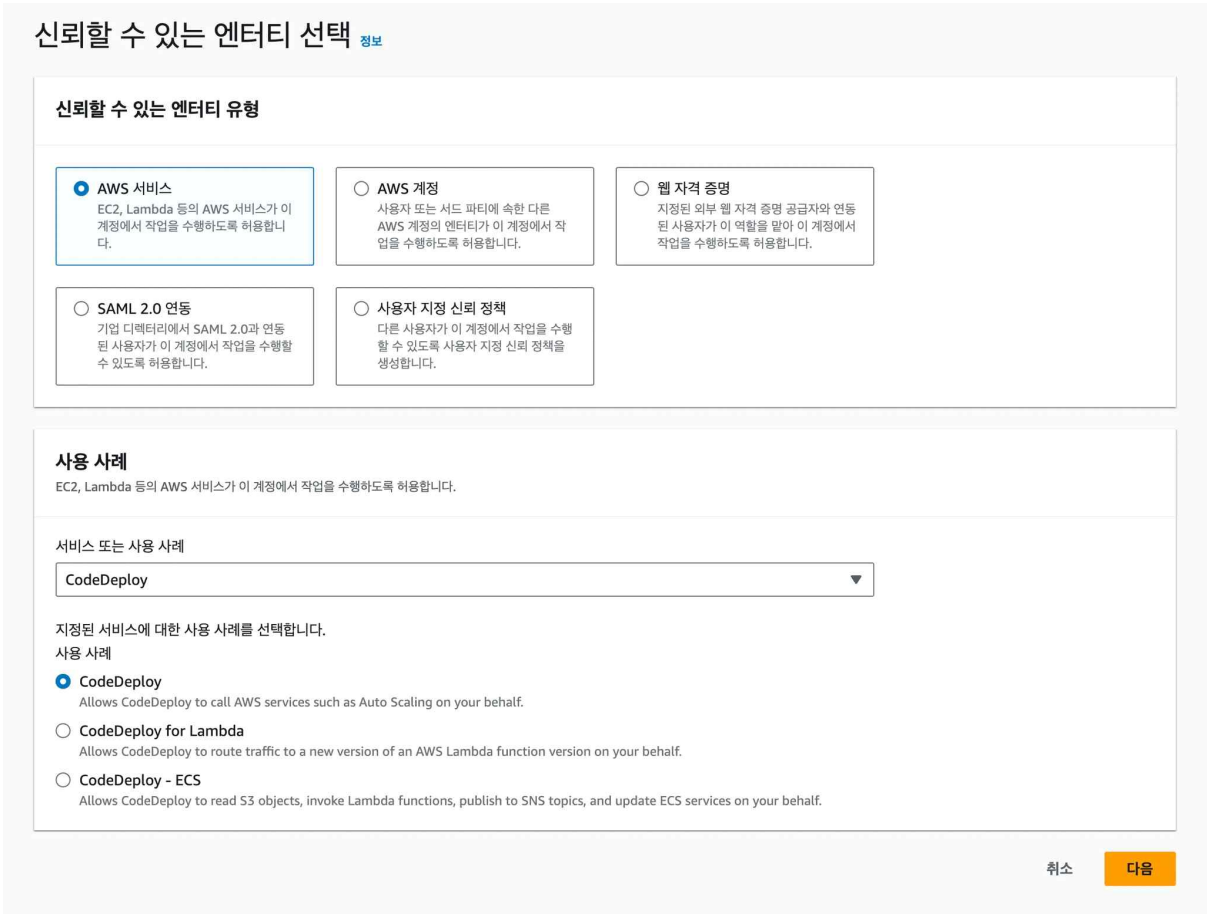
- 서버를 여러 대 이상 구동해야 하거나 무중단 배포가 중요한 서비스일 때 주로 활용한다.

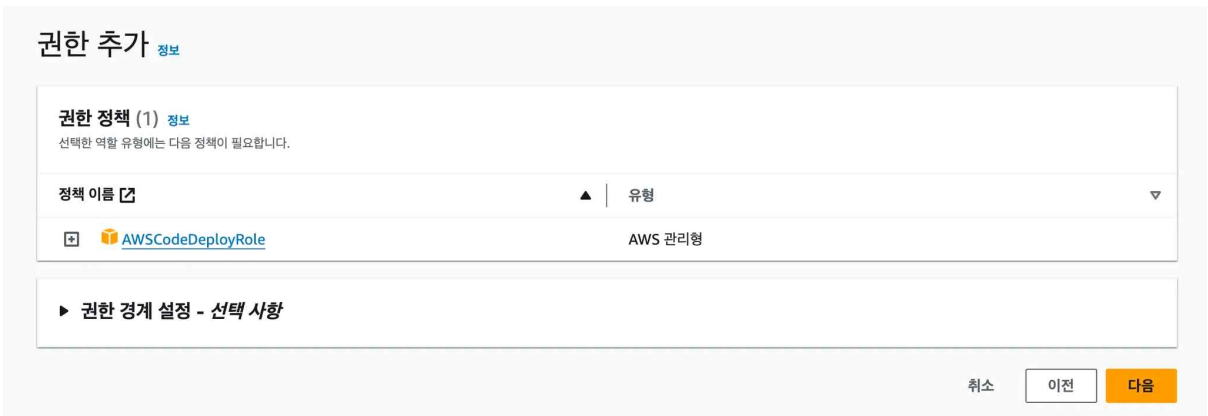
[실습] CodeDeploy 셋팅 / IAM 설정

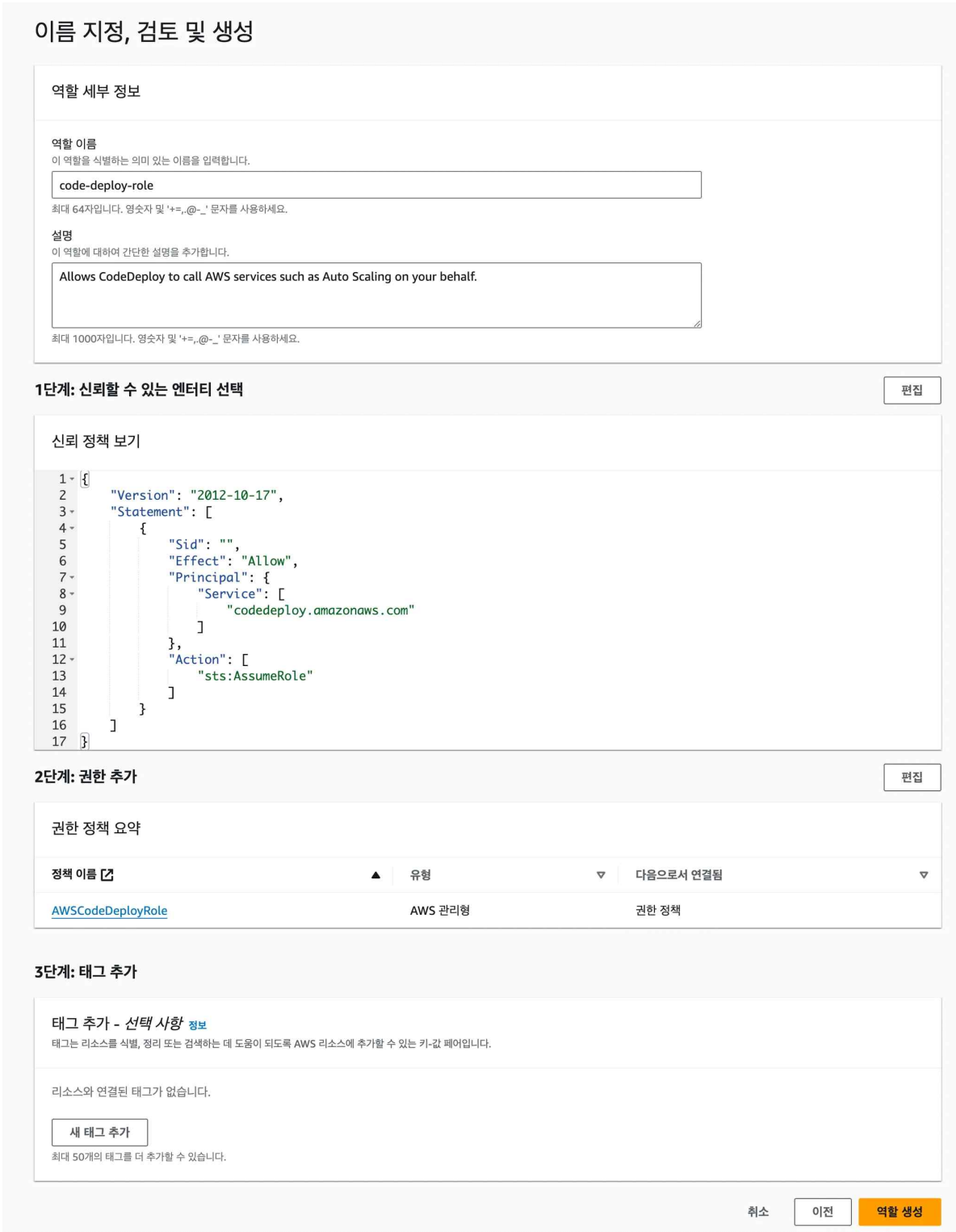
▶ Code Deploy가 다른 AWS Resource에 접근하려면 권한이 필요하다. 그 권한을 부여해주는 기능이 IAM의 역할(Role)이다





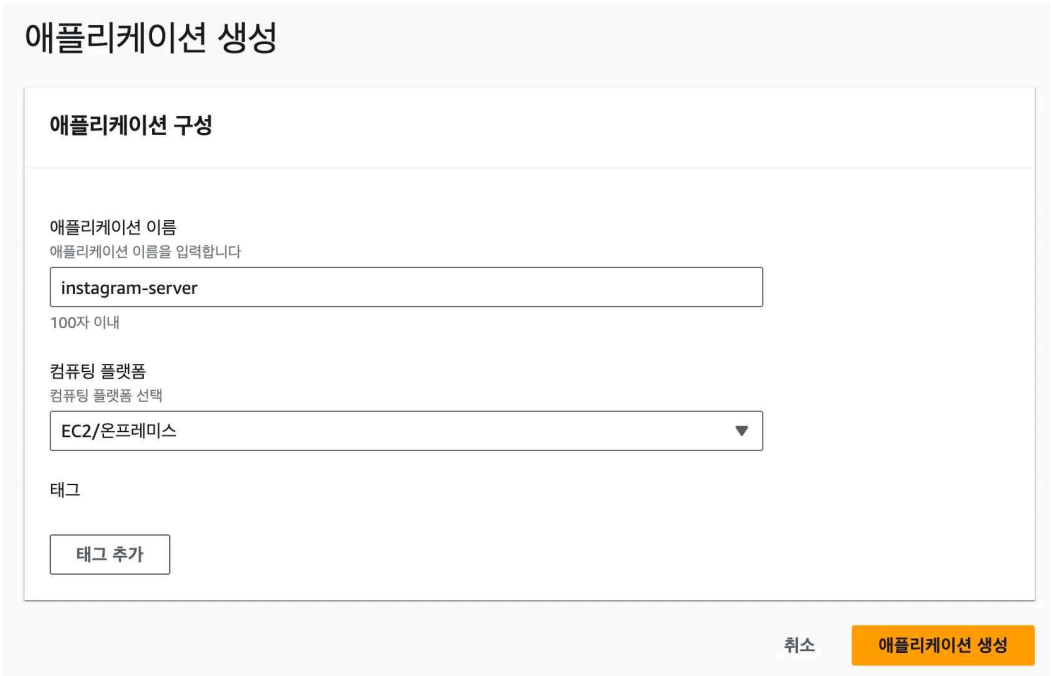

신뢰할 수 있는 엔터티 선택 정보
신뢰할 수 있는 엔터티 유형
AWS 서비스
EC2, Lambda 등의 AWS 서비스가 이 계정에서 작업을 수행하도록 허용합니다.
AWS 계정
사용자 또는 서드 파티에 속한 다른 AWS 계정의 엔터티가 이 계정에서 작업을 수행하도록 허용합니다.
웹 자격 증명
지정된 외부 웹 자격 증명 공급자와 연동된 사용자가 이 역할을 맡아 이 계정에서 작업을 수행하도록 허용합니다.
SAML 2.0 연동
기업 디렉터리에서 SAML 2.0과 연동된 사용자가 이 계정에서 작업을 수행할 수 있도록 허용합니다.
사용자 지정 신뢰 정책
다른 사용자가 이 계정에서 작업을 수행할 수 있도록 사용자 지정 신뢰 정책을 생성합니다.
사용 사례
EC2, Lambda 등의 AWS 서비스가 이 계정에서 작업을 수행하도록 허용합니다.
서비스 또는 사용 사례
CodeDeploy
지정된 서비스에 대한 사용 사례를 선택합니다.
사용 사례
CodeDeploy
Allows CodeDeploy to call AWS services such as Auto Scaling on your behalf.
CodeDeploy for Lambda
Allows CodeDeploy to route traffic to a new version of an AWS Lambda function version on your behalf.
CodeDeploy - ECS
Allows CodeDeploy to read S3 objects, invoke Lambda functions, publish to SNS topics, and update ECS services on your behalf.
취소
다음


권한 추가 정보
권한 정책 (1) 정보
선택한 역할 유형에는 다음 정책이 필요합니다.
정책 이름
유형
AWSCodeDeployRole
AWS 관리형
권한 경계 설정 - 선택 사항
취소
이전
다음

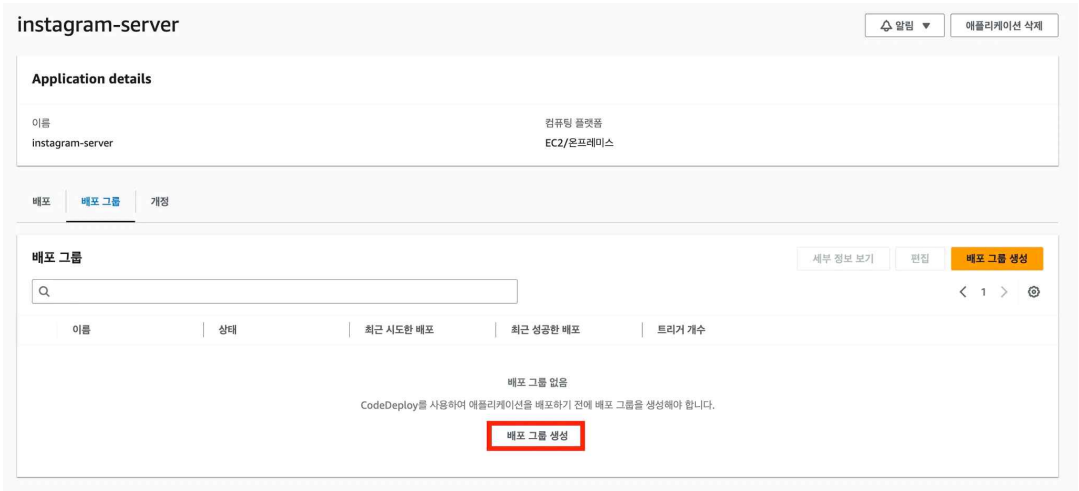

이름 지정, 검토 및 생성
역할 세부 정보
역할 이름
이 역할을 식별하는 의미 있는 이름을 입력합니다.
code-deploy-role
최대 64자입니다. 영숫자 및 '+=,.@-_' 문자를 사용하세요.
설명
이 역할에 대하여 간단한 설명을 추가합니다.
Allows CodeDeploy to call AWS services such as Auto Scaling on your behalf.
최대 1000자입니다. 영숫자 및 '+=,.@-_' 문자를 사용하세요.
1단계: 신뢰할 수 있는 엔터티 선택
편집
신뢰 정책 보기
{
"Version": "2012-10-17",
"Statement": [
{
"Sid": "",
"Effect": "Allow",
"Principal": {
"Service": [
"codedeploy.amazonaws.com"
]
},
"Action": [
"sts:AssumeRole"
]
}
]
}
2단계: 권한 추가
편집
권한 정책 요약
정책 이름
유형
다음으로서 연결됨
AWSCodeDeployRole
AWS 관리형
권한 정책
3단계: 태그 추가
태그 추가 - 선택 사항
정보
태그는 리소스를 식별, 정리 또는 검색하는 데 도움이 되도록 AWS 리소스에 추가할 수 있는 키-값 페어입니다.
리소스와 연결된 태그가 없습니다.
새 태그 추가
최대 50개의 태그를 더 추가할 수 있습니다.
취소
이전
역할 생성

✓ 2. CodeDeploy 생성하기

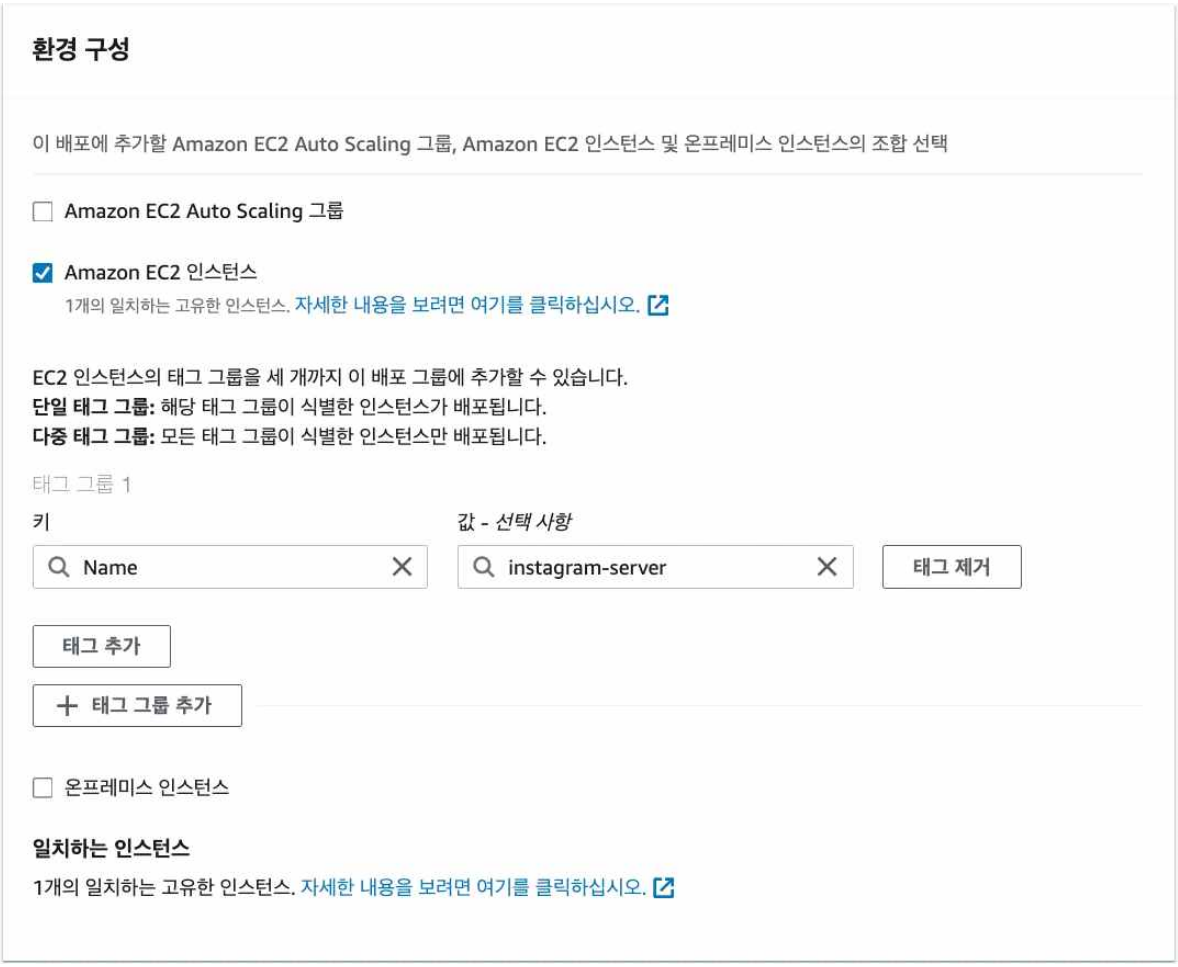
1. CodeDeploy 애플리케이션 생성



2. CodeDeploy 배포그룹 생성




배포 그룹 생성
애플리케이션
애플리케이션
instagram-server
컴퓨팅 유형
EC2/온프레미스
배포 그룹 이름
배포 그룹 이름 입력
Production
100자 제한
서비스 역할
서비스 역할 입력
AWS CodeDeploy가 대상 인스턴스에 액세스하도록 허용하는 CodeDeploy 권한이 있는 서비스 역할을 입력합니다.
arn:aws:iam::002177417362:role/codedeploy-240507
배포 유형
애플리케이션 배포 방법 선택
현재 위치
배포 그룹의 인스턴스를 최신 애플리케이션 개정으로 업데이트합니다. 배포 중에 각 인스턴스가 업데이트를 위해 잠시 오프라인 상태로 전환됩니다.
블루/그린
배포 그룹의 인스턴스를 새 인스턴스로 교체하고 최신 애플리케이션 개정을 해당 인스턴스에 배포합니다. 대체 환경의 인스턴스가 로드 밸런서에 등록된 후 원본 환경의 인스턴스는 등록 취소되고 종료할 수 있습니다.

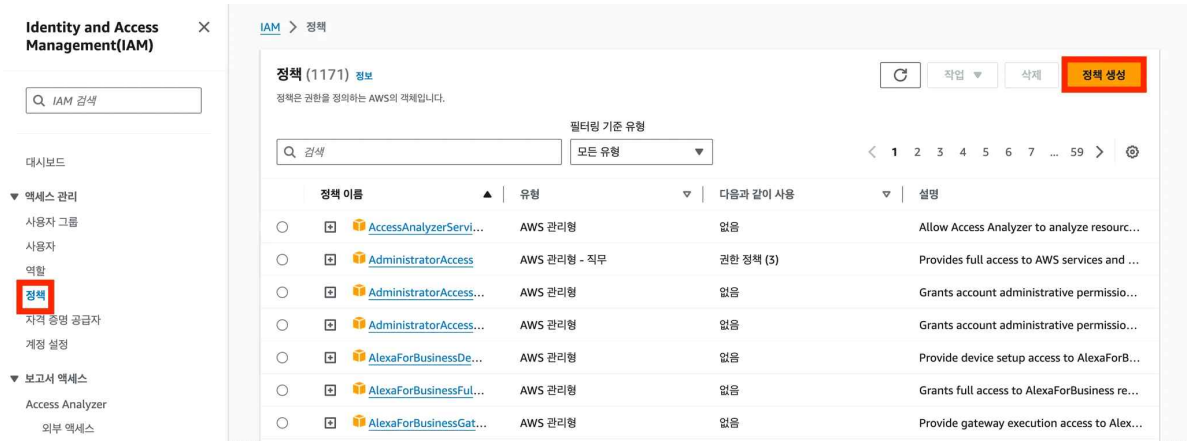

환경 구성
이 배포에 추가할 Amazon EC2 Auto Scaling 그룹, Amazon EC2 인스턴스 및 온프레미스 인스턴스의 조합 선택
Amazon EC2 Auto Scaling 그룹
Amazon EC2 인스턴스
1개의 일치하는 고유한 인스턴스. 자세한 내용을 보려면 여기를 클릭하십시오.
EC2 인스턴스의 태그 그룹을 세 개까지 이 배포 그룹에 추가할 수 있습니다.
단일 태그 그룹: 해당 태그 그룹이 식별한 인스턴스가 배포됩니다.
다중 태그 그룹: 모든 태그 그룹이 식별한 인스턴스만 배포됩니다.
태그 그룹 1
키
값 - 선택 사항
Name
instagram-server
태그 제거
태그 추가
태그 그룹 추가
온프레미스 인스턴스
일치하는 인스턴스
1개의 일치하는 고유한 인스턴스. 자세한 내용을 보려면 여기를 클릭하십시오.


배포 설정
배포 구성
기본 및 사용자 지정 배포 구성 목록에서 선택합니다. 배포 구성은 애플리케이션이 배포되는 속도와 배포 성공 또는 실패 조건을 결정하는 규칙 세트입니다.
CodeDeployDefault.AllAtOnce
또는
배포 구성 만들기
로드 밸런서
배포 프로세스 중에 수신 트래픽을 관리할 로드 밸런서를 선택합니다. 로드 밸런서는 배포 중인 각 인스턴스에서 트래픽을 차단하고 배포 성공 후 인스턴스에 대한 트래픽을 다시 허용합니다.
로드 밸런싱 활성화

✓ 3. EC2 역할(Role) 생성하기

EC2가 빌드된 파일을 S3로부터 다운받아야 한다. 이 때, EC2가 S3에 접근하려면 권한이 필요하다. 그 권한을 부여해주는 기능이 IAM의 역할(Role)이다.

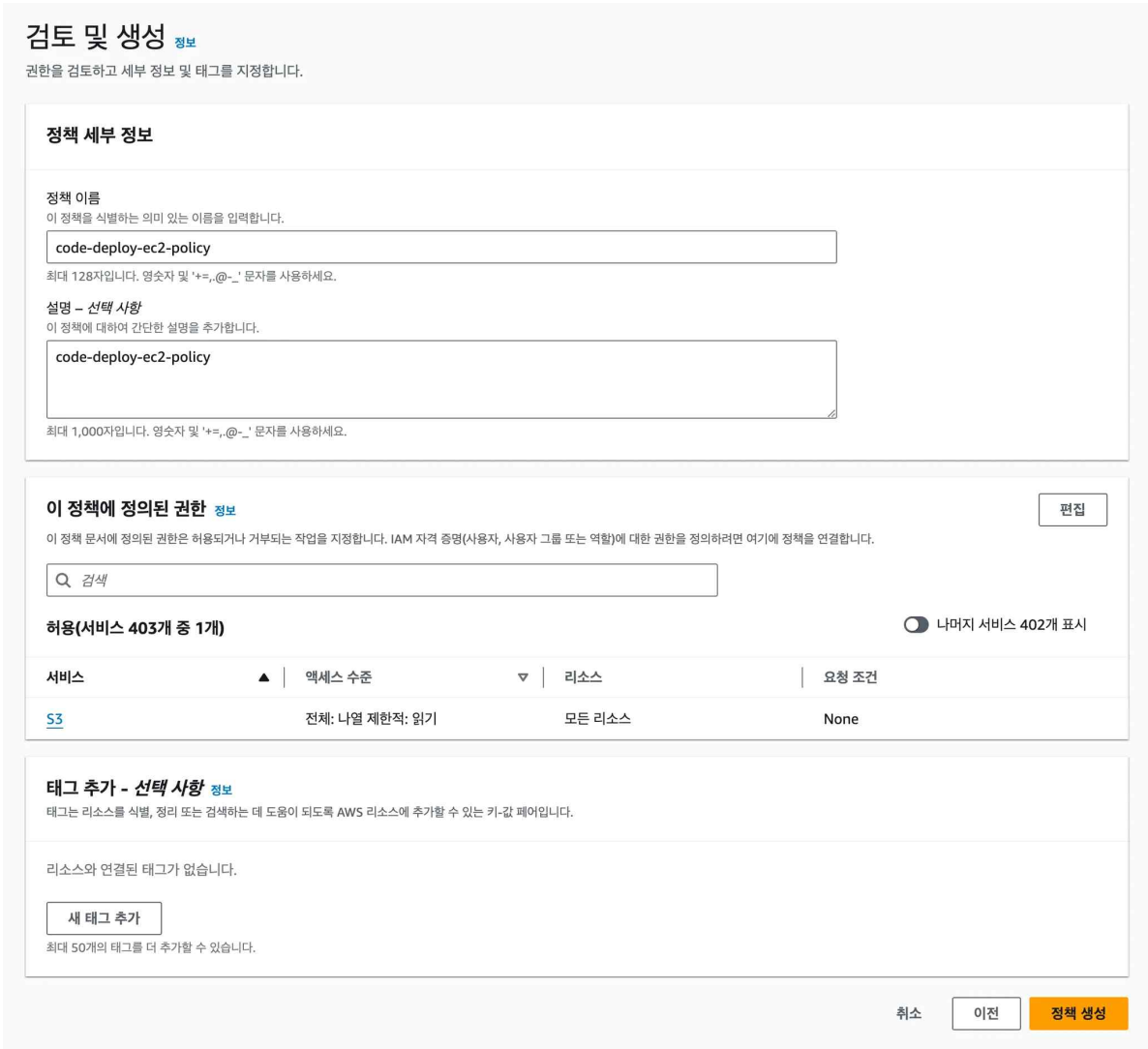
1. 정책 생성



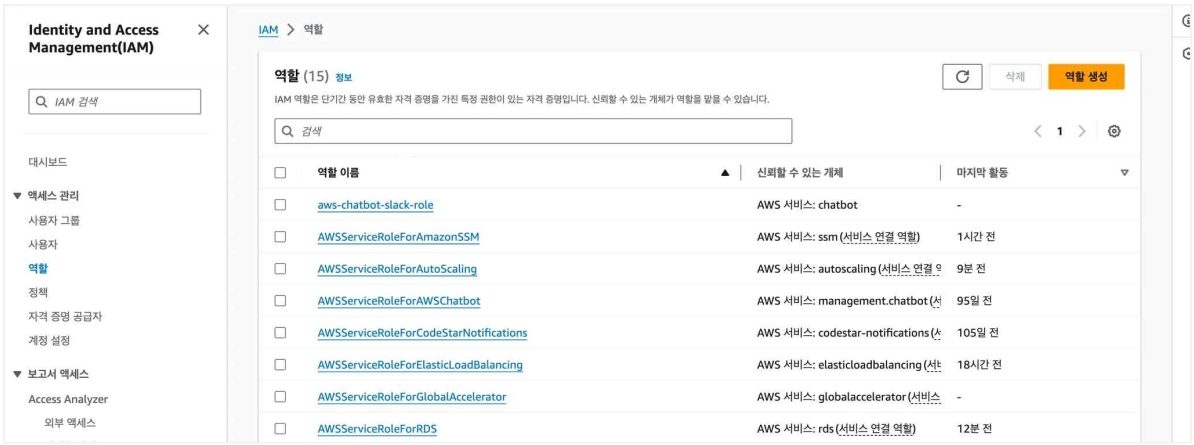
2. 권한 지정하기

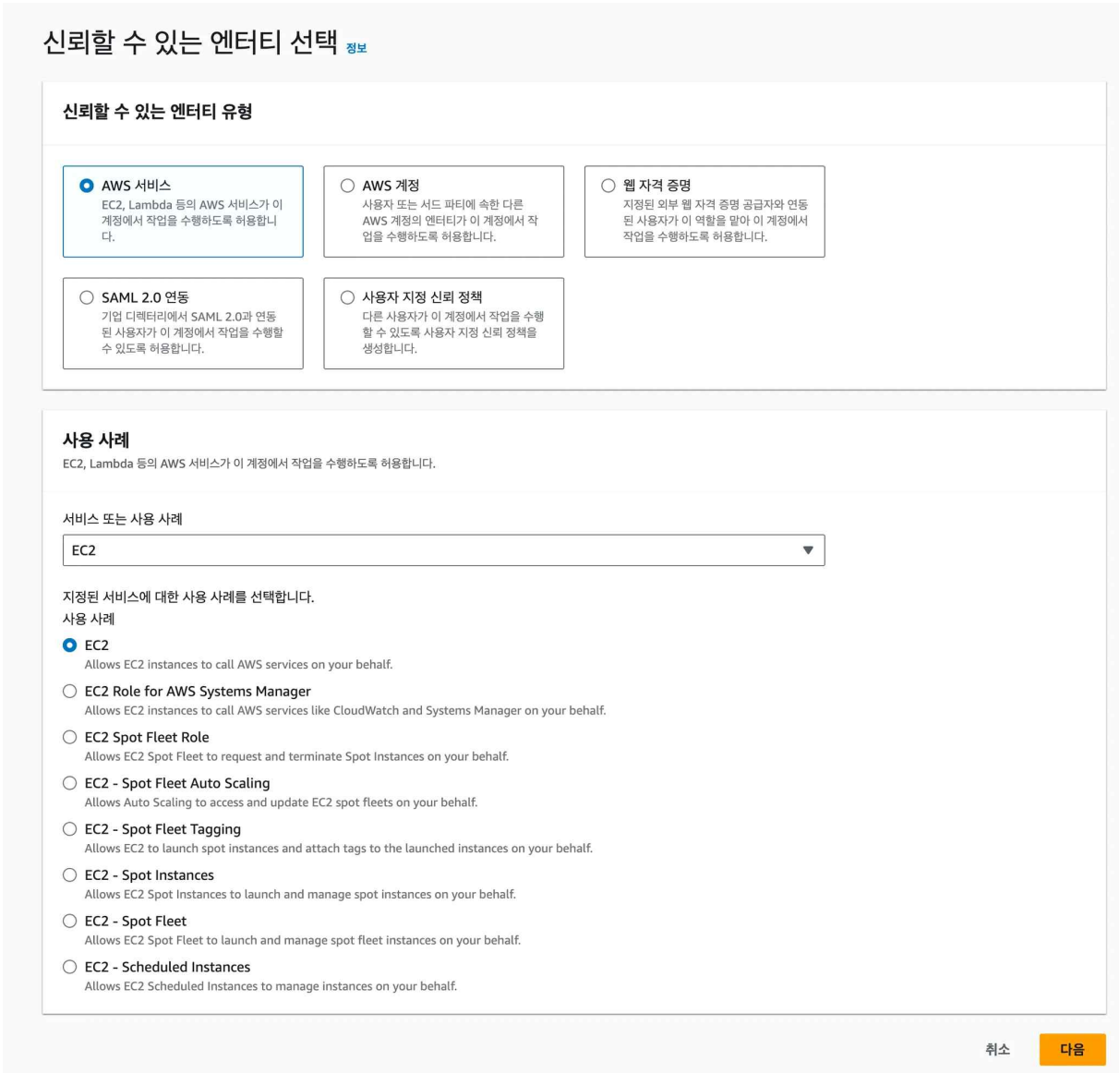
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "s3:Get*",
                "s3:List*"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

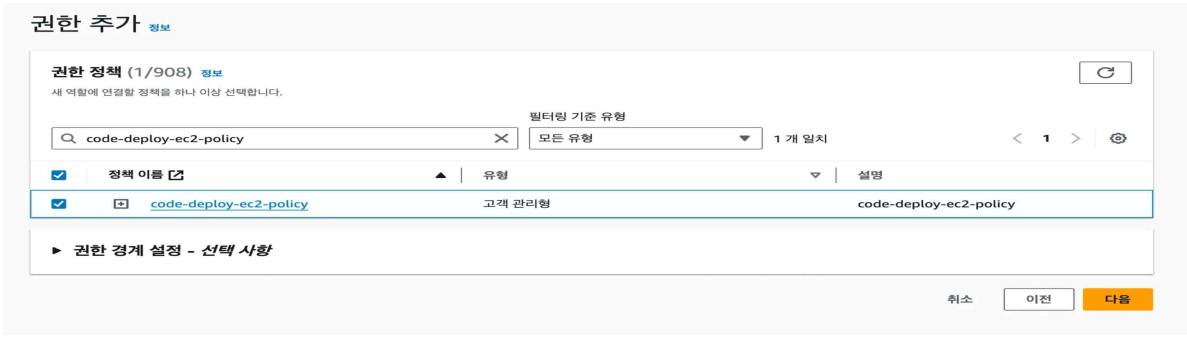
3. 정책 세부 설정

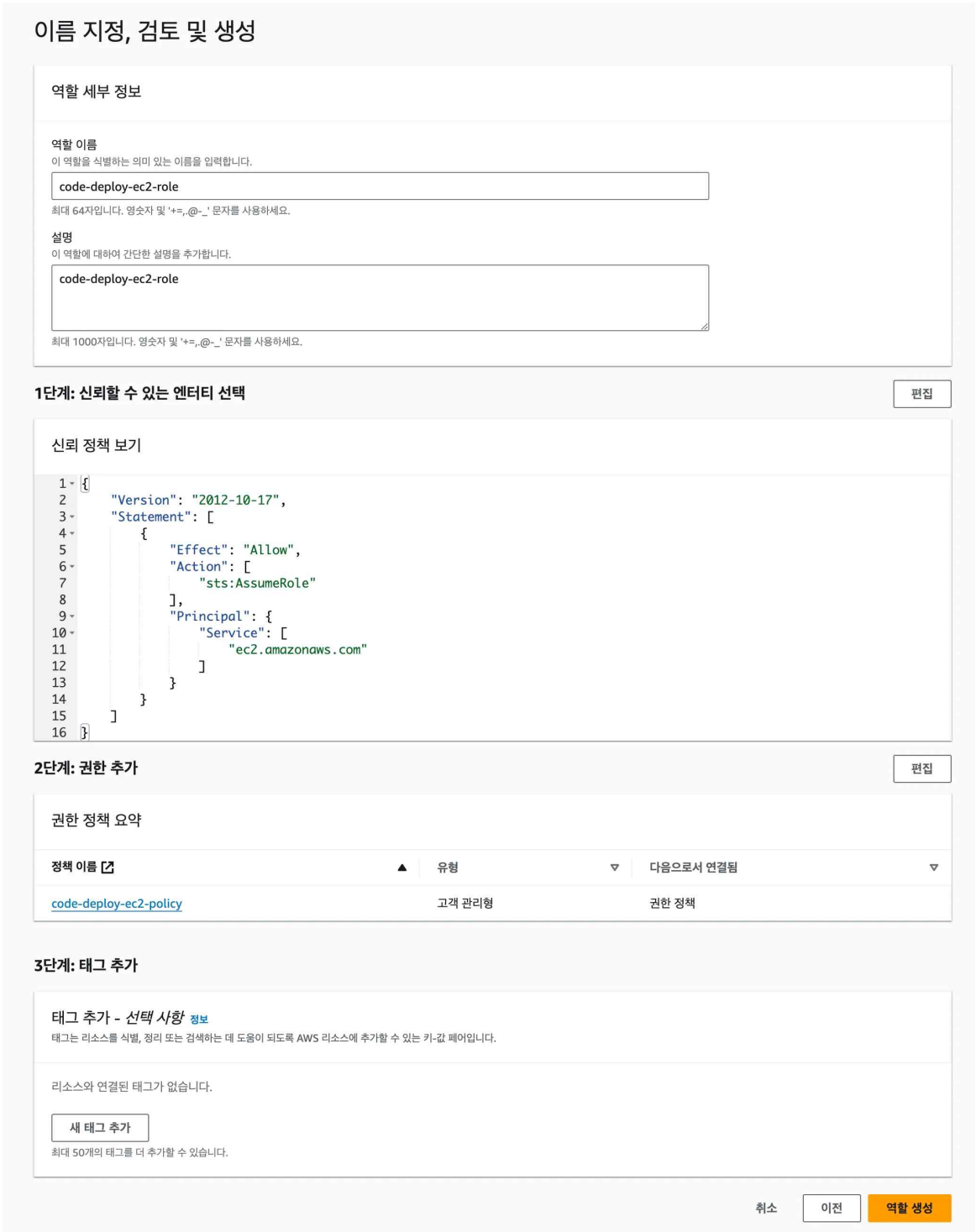


4. 역할 생성


Identity and Access Management(IAM)
IAM 검색
대시보드
액세스 관리
사용자 그룹
사용자
역할
정책
자격 증명 공급자
계정 설정
보고서 액세스
Access Analyzer
외부 액세스
IAM > 역할
역할 (15) 정보
IAM 역할은 단기간 동안 유효한 자격 증명을 가진 특정 권한이 있는 자격 증명입니다. 신뢰할 수 있는 개체가 역할을 맡을 수 있습니다.
삭제
역할 생성
검색
역할 이름
신뢰할 수 있는 개체
마지막 활동
aws-chatbot-slack-role
AWS 서비스: chatbot
AWSServiceRoleForAmazonSSM
AWS 서비스: ssm (서비스 연결 역할)
1시간 전
AWSServiceRoleForAutoScaling
AWS 서비스: autoscaling (서비스 연결
9분 전
AWSServiceRoleForAWSChatbot
AWS 서비스: management.chatbot (서
95일 전
AWSServiceRoleForCodeStarNotifications
AWS 서비스: codestar-notifications (서
105일 전
AWSServiceRoleForElasticLoadBalancing
AWS 서비스: elasticloadbalancing (서비
18시간 전
AWSServiceRoleForGlobalAccelerator
AWS 서비스: globalaccelerator (서비스
AWSServiceRoleForRDS
AWS 서비스: rds (서비스 연결 역할)
12분 전


신뢰할 수 있는 엔터티 선택 정보
신뢰할 수 있는 엔터티 유형
AWS 서비스
EC2, Lambda 등의 AWS 서비스가 이 계정에서 작업을 수행하도록 허용합니다.
AWS 계정
사용자 또는 서드 파티에 속한 다른 AWS 계정의 엔터티가 이 계정에서 작업을 수행하도록 허용합니다.
웹 자격 증명
지정된 외부 웹 자격 증명 공급자와 연동된 사용자가 이 역할을 맡아 이 계정에서 작업을 수행하도록 허용합니다.
SAML 2.0 연동
기업 디렉터리에서 SAML 2.0과 연동된 사용자가 이 계정에서 작업을 수행할 수 있도록 허용합니다.
사용자 지정 신뢰 정책
다른 사용자가 이 계정에서 작업을 수행할 수 있도록 사용자 지정 신뢰 정책을 생성합니다.
사용 사례
EC2, Lambda 등의 AWS 서비스가 이 계정에서 작업을 수행하도록 허용합니다.
서비스 또는 사용 사례
EC2
지정된 서비스에 대한 사용 사례를 선택합니다.
사용 사례
EC2
Allows EC2 instances to call AWS services on your behalf.
EC2 Role for AWS Systems Manager
Allows EC2 instances to call AWS services like CloudWatch and Systems Manager on your behalf.
EC2 Spot Fleet Role
Allows EC2 Spot Fleet to request and terminate Spot Instances on your behalf.
EC2 - Spot Fleet Auto Scaling
Allows Auto Scaling to access and update EC2 spot fleets on your behalf.
EC2 - Spot Fleet Tagging
Allows EC2 to launch spot instances and attach tags to the launched instances on your behalf.
EC2 - Spot Instances
Allows EC2 Spot Instances to launch and manage spot instances on your behalf.
EC2 - Spot Fleet
Allows EC2 Spot Fleet to launch and manage spot fleet instances on your behalf.
EC2 - Scheduled Instances
Allows EC2 Scheduled Instances to manage instances on your behalf.
취소
다음


권한 추가 정보
권한 정책 (1/908) 정보
새 역할에 연결할 정책을 하나 이상 선택합니다.
필터링 기준 유형
code-deploy-ec2-policy
모든 유형
1 개 일치
정책 이름
유형
설명
code-deploy-ec2-policy
고객 관리형
code-deploy-ec2-policy
권한 경계 설정 - 선택 사항
취소
이전
다음


이름 지정, 검토 및 생성
역할 세부 정보
역할 이름
이 역할을 식별하는 의미 있는 이름을 입력합니다.
code-deploy-ec2-role
최대 64자입니다. 영숫자 및 '+=,.@-_' 문자를 사용하세요.
설명
이 역할에 대하여 간단한 설명을 추가합니다.
code-deploy-ec2-role
최대 1000자입니다. 영숫자 및 '+=,.@-_' 문자를 사용하세요.
1단계: 신뢰할 수 있는 엔터티 선택
편집
신뢰 정책 보기
{
"Version": "2012-10-17",
"Statement": [
{
"Effect": "Allow",
"Action": [
"sts:AssumeRole"
],
"Principal": {
"Service": [
"ec2.amazonaws.com"
]
}
}
]
}
2단계: 권한 추가
편집
권한 정책 요약
정책 이름
유형
다음으로서 연결됨
code-deploy-ec2-policy
고객 관리형
권한 정책
3단계: 태그 추가
태그 추가 - 선택 사항 정보
태그는 리소스를 식별, 정리 또는 검색하는 데 도움이 되도록 AWS 리소스에 추가할 수 있는 키-값 페어입니다.
리소스와 연결된 태그가 없습니다.
새 태그 추가
최대 50개의 태그를 더 추가할 수 있습니다.
취소
이전
역할 생성

5. EC2에 생성한 IAM 역할 연결하기

참고) 혹시나 Code Deploy Agent를 먼저 설치한 뒤에 EC2에 IAM 역할을 부여했다면 EC2에서 아래 명령어로 codedeploy-agent를 재시작시켜주어야 한다.

```
$ sudo systemctl restart codedeploy-agent
```

▶ 참고: https://tinyurl.com/2bfchtck

✓ 4. Code Deploy Agent 설치하기

Code Deploy Agent가 다른 AWS Resource에 접근하려면 권한이 필요하다. 그 권한을 부여해주는 기능이 IAM의 역할(Role)이다.

▶ https://tinyurl.com/2avof82q

[Ubuntu 22.04. 기준]

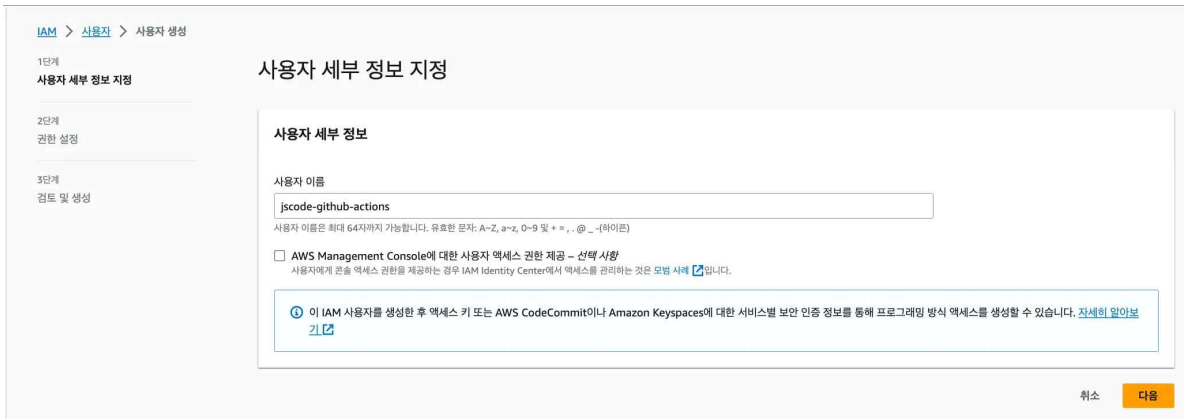
```
$ sudo apt update && \
sudo apt install -y ruby-full wget && \
cd /home/ubuntu && \
wget https://aws-codedeploy-ap-northeast-2.s3.ap-northeast-2.amazonaws.com
/latest/install && \
chmod +x ./install && \
sudo ./install auto
```

[Code Deploy Agent가 정상적으로 실행되고 있는 지 확인]

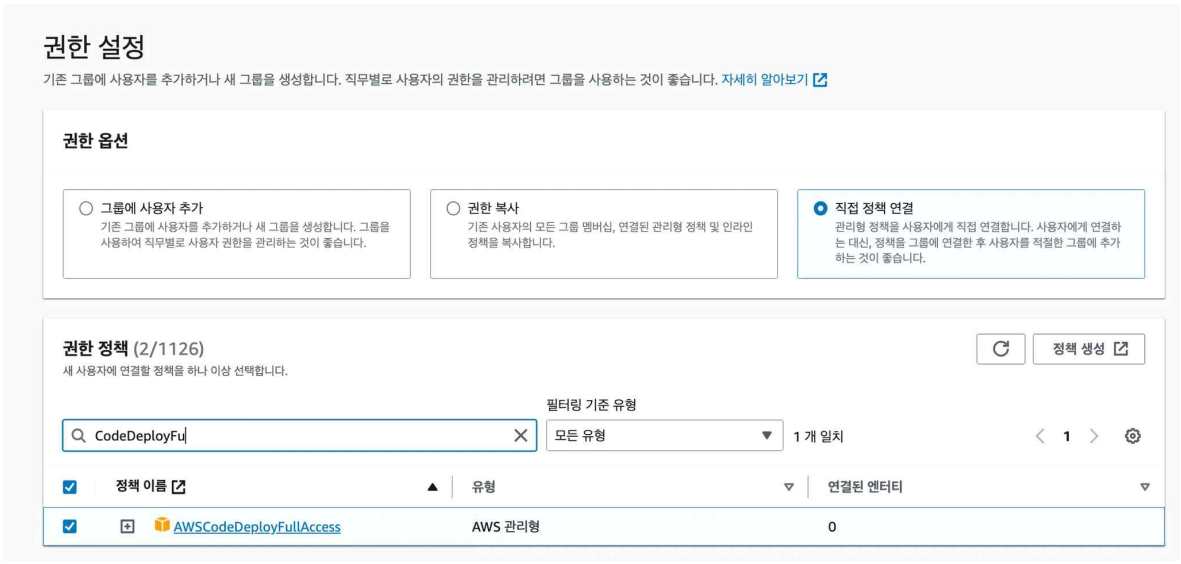
```
$ systemctl status codedeploy-agent
```

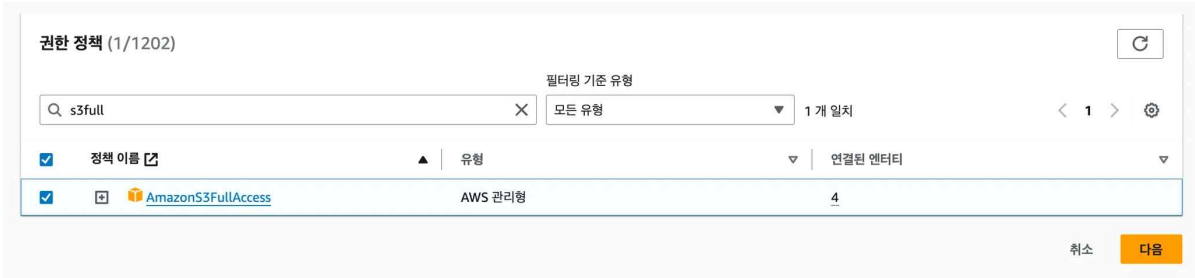
✓ 5. Github Actions가 CodeDeploy, S3에 접근할 수 있게 IAM 발급

1. IAM 사용자 생성

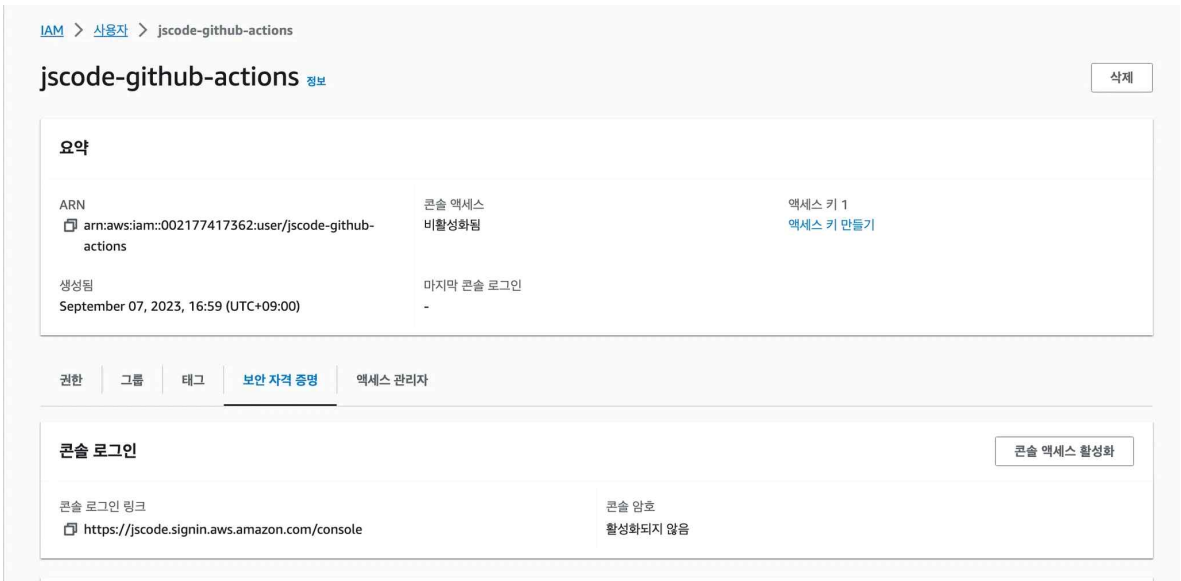


2. 직접 정책 연결 (AWSCodeDeployFullAccess, AmazonS3FullAccess)

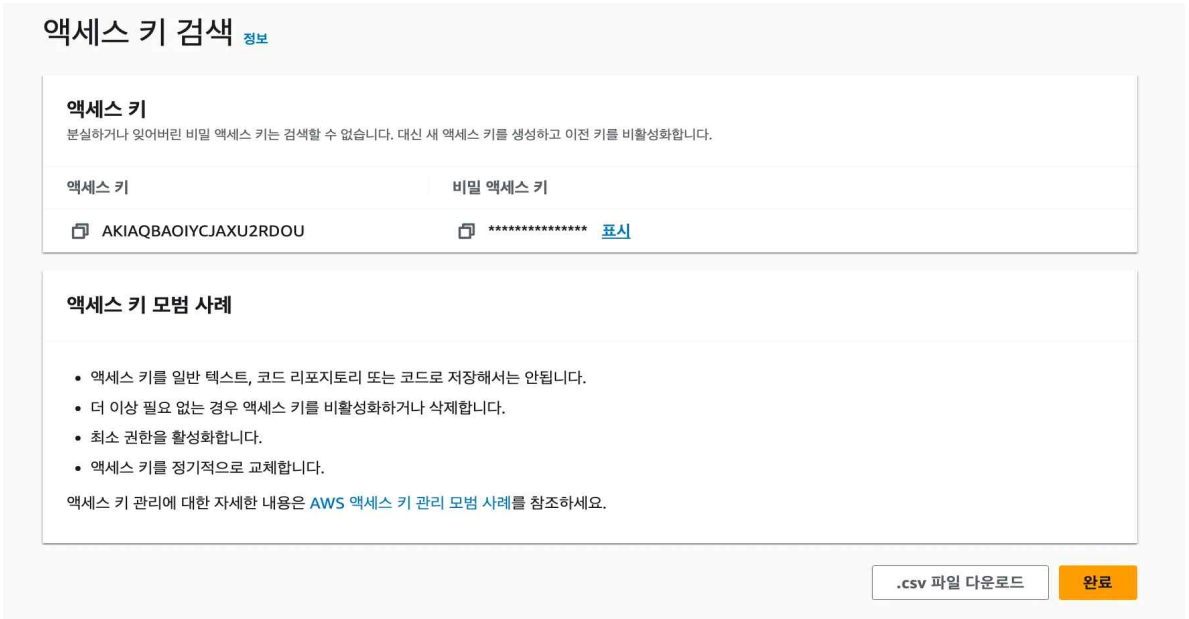




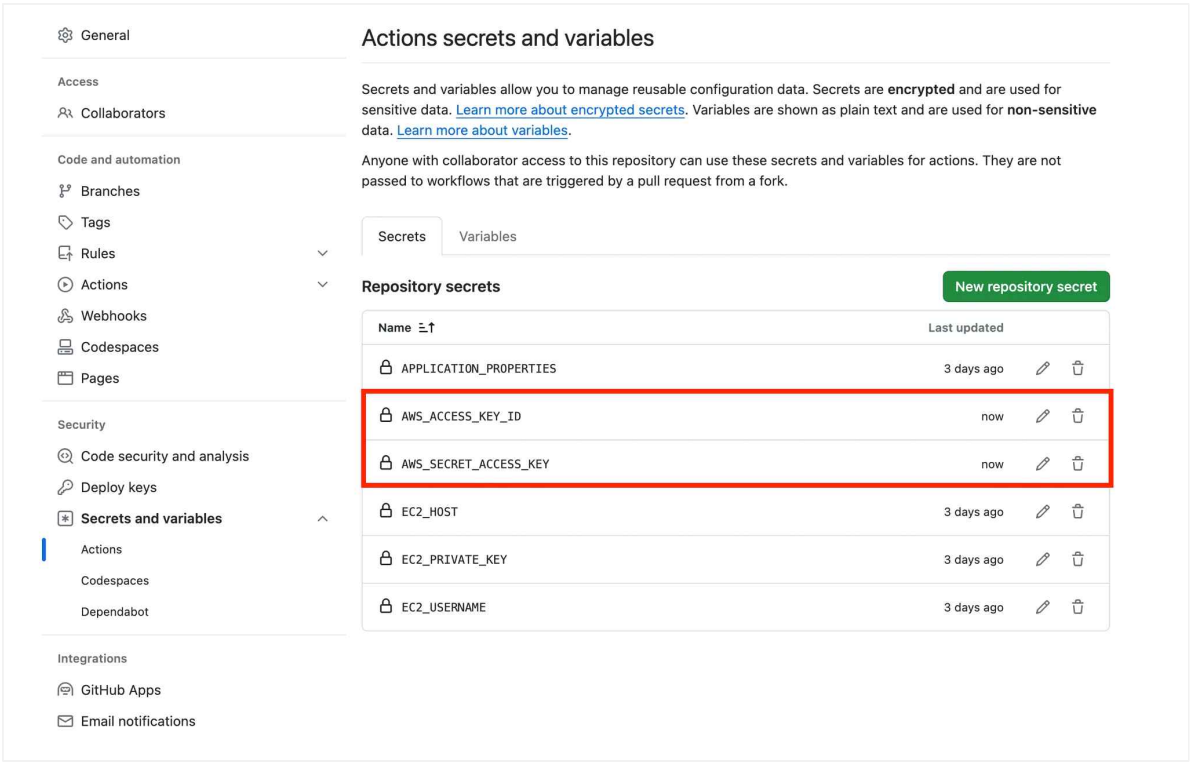
3. 보안 자격 증명에 들어가서 액세스 키 만들기



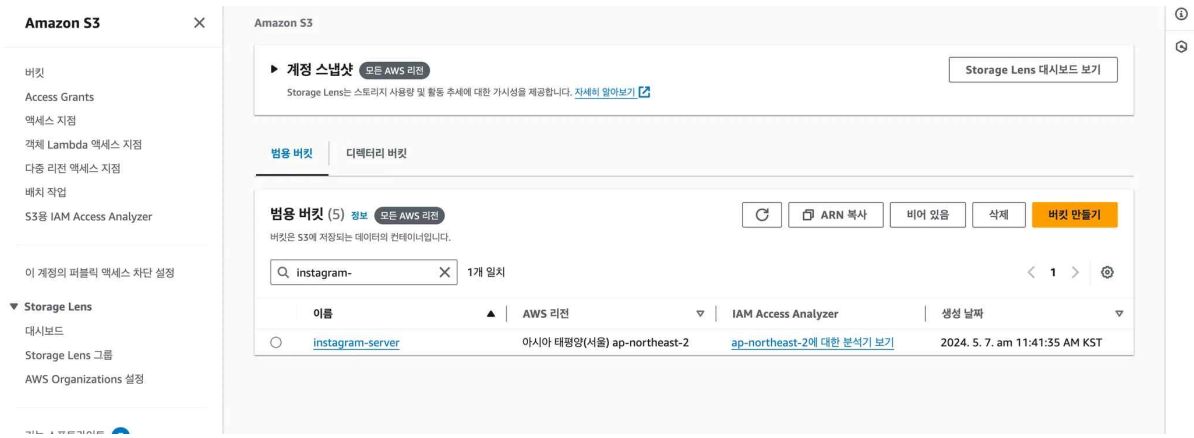
4. 액세스 키와 비밀 액세스 키 잘 보관해두기



✓ 6. Github Actions의 Secret Key에 저장해두기



✓ 7. 배포 전에 프로젝트 파일 저장할 S3 만들기



[참고] CodeDeploy 로그 확인하는 방법

▶ https://tinyurl.com/284mbgpu

[실습] 확장성을 고려한 프로젝트에서 많이 쓰는 CI/CD 구축 방법

✅ 이전 실습했던 내용 정리

- 서버 종료
- 프로젝트 폴더 삭제

✅ 1. appspec.yml, 스크립트 파일 작성하기

appspec.yml은 CodeDeploy가 실행될 때 필수적으로 존재해야 하는 파일이다. CodeDeploy는 이 설정 파일을 기반으로 실행한다.

appspec.yml

```
version: 0.0
os: linux

files:
  # S3에 저장한 파일들 중 destination(AWS EC2)으로 이동시킬 대상을 지정한다.
  # / 이라고 지정하면 S3에 저장한 전체 파일을 뜻한다.
  - source: /
    # EC2의 어떤 경로에 저장할 지 지정한다.
    destination: /home/ubuntu/instagram-server

permissions:
  - object: /
    owner: ubuntu
    group: ubuntu

hooks:
  ApplicationStart:
    - location: scripts/start-server.sh
      timeout: 60
      runas: ubuntu
```

scripts/start-server.sh

```
#!/bin/bash

echo "--------------- 서버 배포 시작 -----------------"
cd /home/ubuntu/instagram-server
sudo fuser -k -n tcp 8080 || true
nohup java -jar project.jar > ./output.log 2>&1 &
echo "--------------- 서버 배포 끝 -----------------"
```

✓ 2. Github Actions 코드 작성하기

.github/workflows/deploy.yml

```
name: Deploy To EC2

on:
  push:
    branches:
      - main

jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - name: Github Repository 파일 불러오기
        uses: actions/checkout@v4

      - name: JDK 17버전 설치
        uses: actions/setup-java@v4
        with:
          distribution: temurin
          java-version: 17

      - name: application.yml 파일 만들기
        run: echo "${{ secrets.APPLICATION_PROPERTIES }}" > ./src/main/reso
urces/application.yml
```

```
      - name: 테스트 및 빌드하기
        run: ./gradlew clean build

      - name: 빌드된 파일 이름 변경하기
        run: mv ./build/libs/*SNAPSHOT.jar ./project.jar

      - name: 압축하기
        run: tar -czvf $GITHUB_SHA.tar.gz project.jar appspec.yml scripts

      - name: AWS Resource에 접근할 수 있게 AWS credentials 설정
        uses: aws-actions/configure-aws-credentials@v4
        with:
          aws-region: ap-northeast-2
          aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
          aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}

      - name: S3에 프로젝트 폴더 업로드하기
        run: aws s3 cp --region ap-northeast-2 ./$GITHUB_SHA.tar.gz s3://in
stagram-server/$GITHUB_SHA.tar.gz

      - name: Code Deploy를 활용해 EC2에 프로젝트 코드 배포
        run: aws deploy create-deployment
          --application-name instagram-server
          --deployment-config-name CodeDeployDefault.AllAtOnce
          --deployment-group-name Production
          --s3-location  bucket=instagram-server,bundleType=tgz,key=$GITHUB
_SHA.tar.gz
```

✓ 3. CodeDeploy가 잘 실행됐는 지 확인하기

아래 경로로 들어가면 CodeDeploy가 실행시킨 스크립트 로그 파일을 열 수 있다.

/opt/codedeploy-agent/deployment-root/{deployment-group-ID}/{deployment-ID}/logs/scripts.log