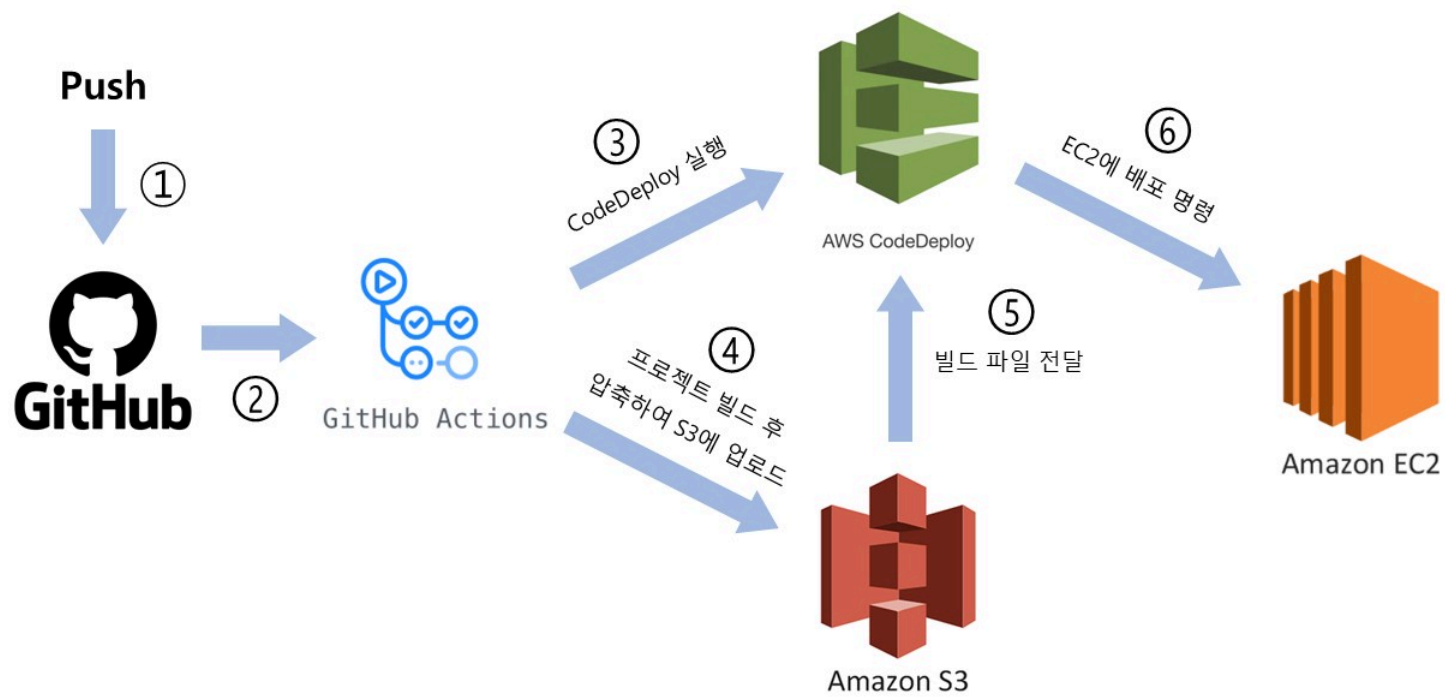


0. 개발환경

- Spring Boot
- GitHub Actions
- AWS CodeDeploy
- AWS EC2
- AWS S3

전체적인 흐름

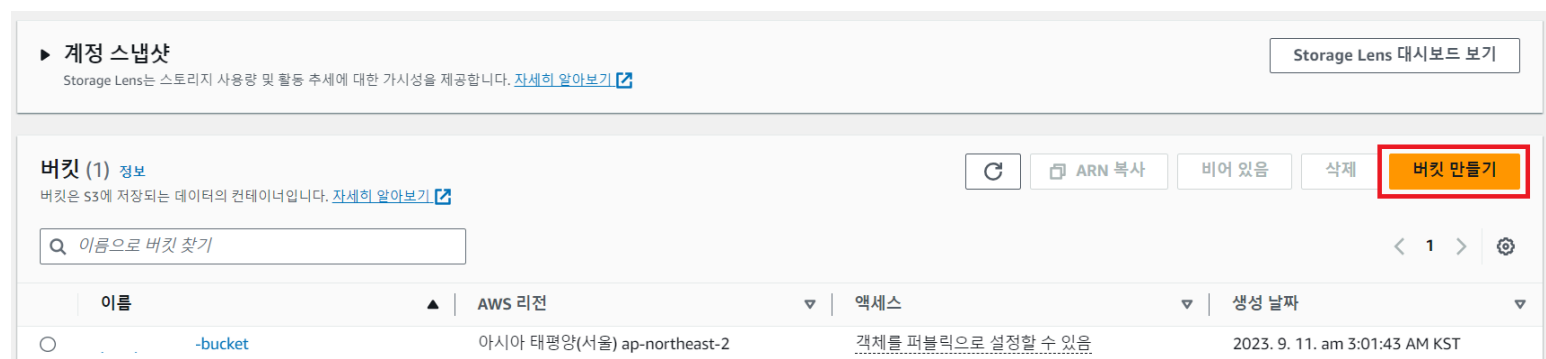


1. S3 생성

1.1 S3 버킷 생성

aws S3 → 버킷 → 버킷 만들기

나는 이미 버킷을 생성해둔 상태다.



버킷 생성 시 리전은 꼭 EC2 인스턴스와 같은 것으로 지정해야 한다.

만약 리전이 다르다면 S3 리전에서 EC2 리전으로 데이터를 옮기는 데에 💰 비용이 청구된다.

(그게 나야,, 둠빠둠빠두비두밥,,)

버킷 만들기 정보

버킷은 S3에 저장되는 데이터의 컨테이너입니다. [자세히 알아보기](#)

일반 구성

버킷 이름

myawsbucket

버킷 이름은 글로벌 네임스페이스 내에서 고유해야 하며 버킷 이름 지정 규칙을 따라야 합니다. [버킷 이름 지정 규칙 보기](#)

AWS 리전

미국 동부(버지니아 북부) us-east-1

기존 버킷에서 설정 복사 - **선택 사항**

다음 구성의 버킷 설정만 복사됩니다.

버킷 선택

나머지는 모두 디폴트 값으로 놔두고 버킷 생성!

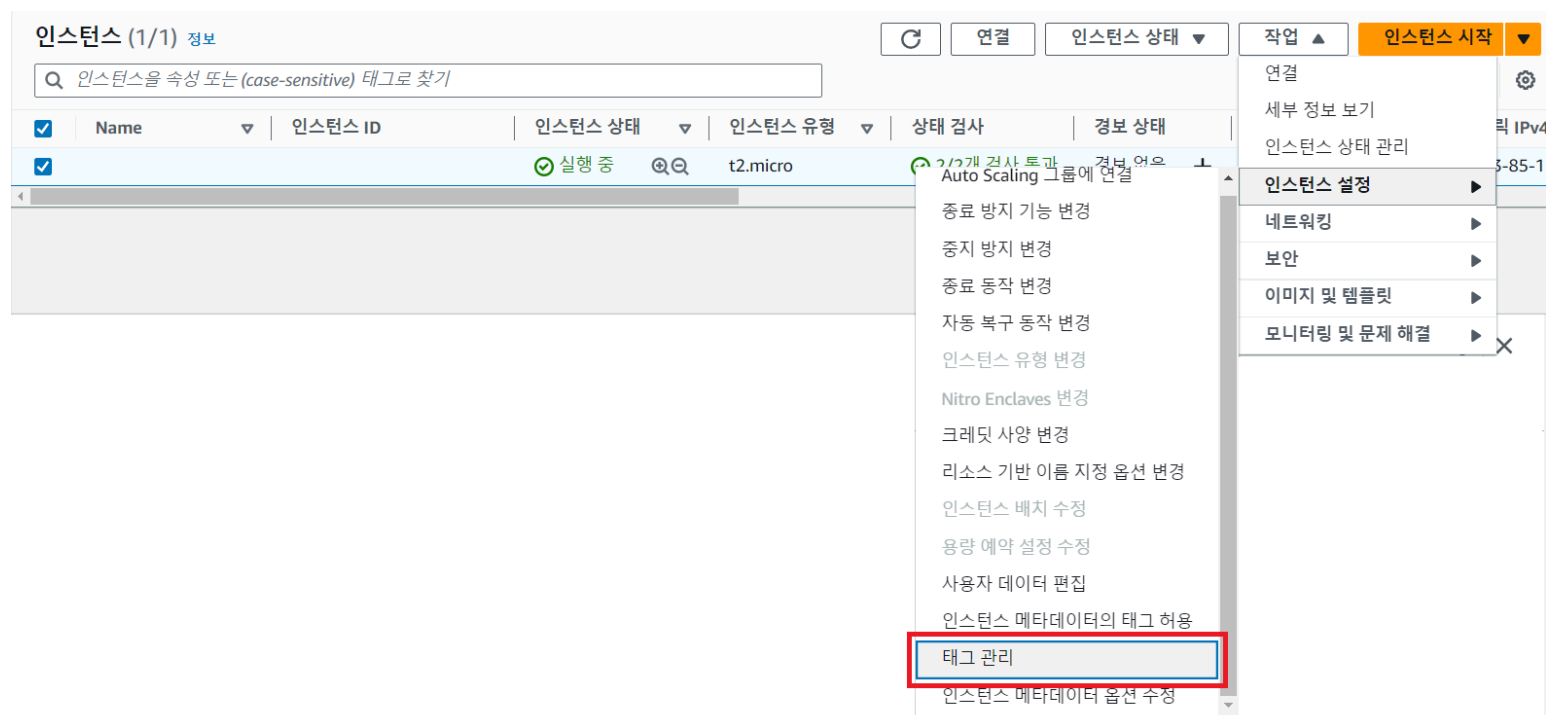
2. EC2 설정 추가

1. 태그 추가
2. IAM 역할 추가
3. EC2 서버에 CodeDeploy 에이전트 설치

2.1 인스턴스 태그 추가

CodeDeploy를 생성할 때 수행할 인스턴스를 지정해줘야하기 때문에 인스턴스들을 구분할 태그를 추가해줘야 한다.

인스턴스 선택 → 작업 → 인스턴스 설정 → 태그 관리

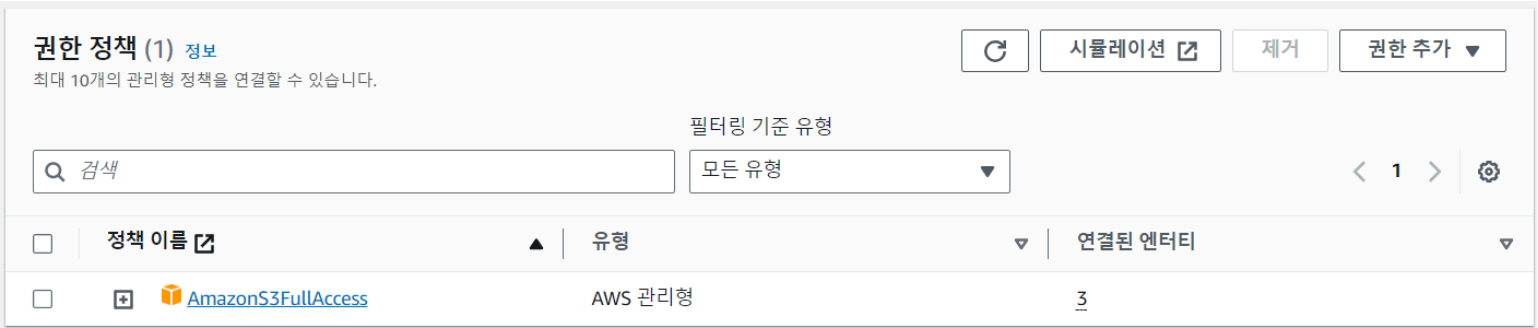


- 태그 확인

태그	
<input type="text"/>	
Key	Value
my-ec2-tag-key	

2.2 IAM 역할 추가

EC2 인스턴스에서 S3에 접근할 수 있도록 **AmazonS3FullAccess** 권한을 추가한 뒤, 인스턴스에 연결한다.



2.3 CodeDeploy 에이전트 설치

EC2 서버에 접속한 뒤, 똑같이 입력하여 설치해준다.

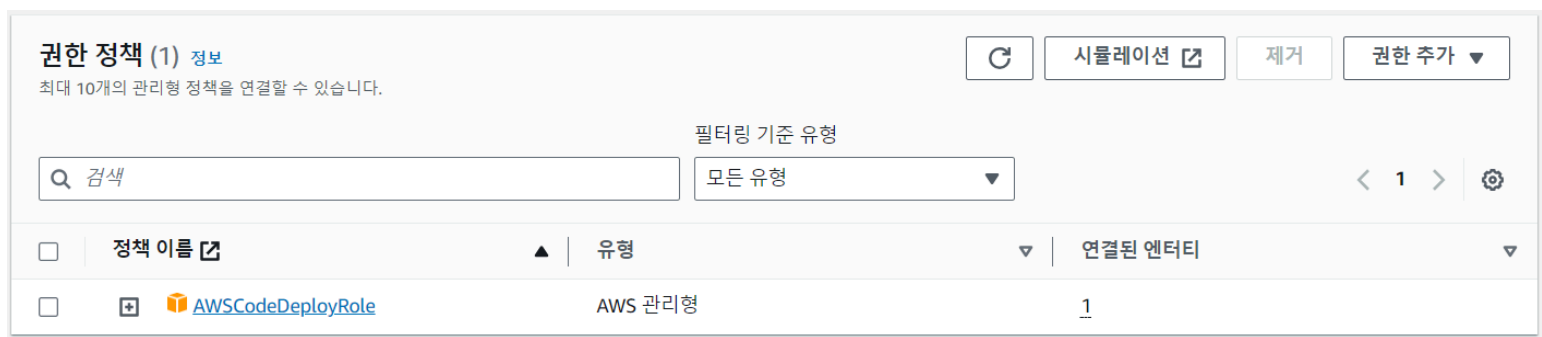
```
$ sudo apt update
$ sudo apt install ruby-full
$ sudo apt install wget
$ cd /home/ubuntu
$ wget https://aws-codedeploy-ap-northeast-2.s3.ap-northeast-2.amazonaws.com/latest/install
$ chmod +x ./install
$ sudo ./install auto > /tmp/logfile
$ sudo service codedeploy-agent status
```

3. CodeDeploy 생성

1. IAM 역할 추가
2. CodeDeploy 애플리케이션 생성
3. CodeDeploy 배포 그룹 생성

3.1 IAM 역할 추가

AWSCodeDeployRole 권한을 추가하여 역할을 생성한다.



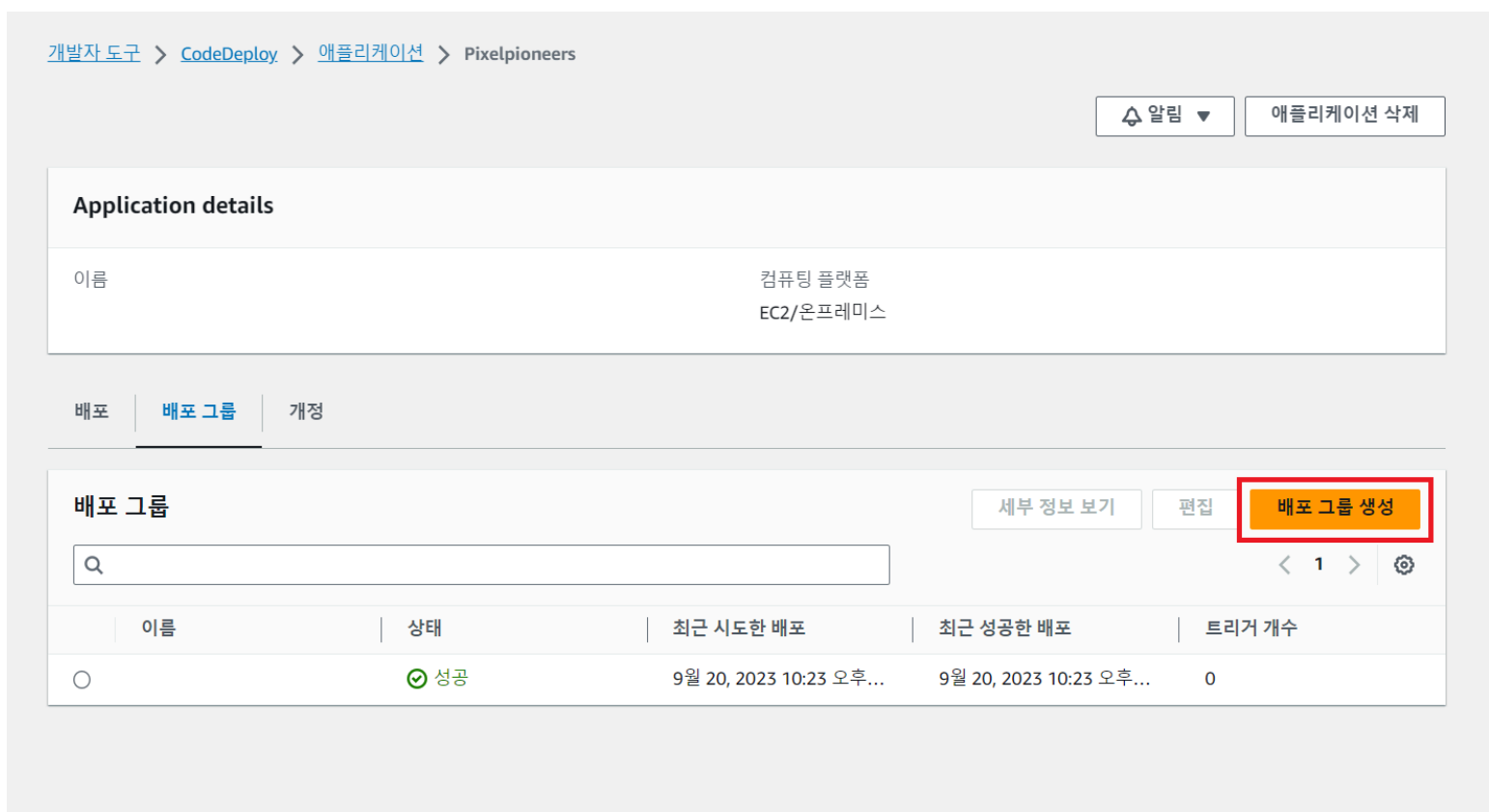
3.2 CodeDeploy 애플리케이션 생성

컴퓨팅 플랫폼은 EC2/온프레미스를 선택하여 생성한다.



3.3 CodeDeploy 배포 그룹 생성

위에서 생성한 CodeDeploy 애플리케이션 이름을 눌러 들어가면 배포 그룹을 생성할 수 있다.



서비스 역할에서는 3.1에서 생성한 AWSCodeDeployRole 권한을 가진 IAM을 선택한다.

배포 그룹 이름

배포 그룹 이름 입력

100자 제한

서비스 역할

서비스 역할 입력

AWS CodeDeploy가 대상 인스턴스에 액세스하도록 허용하는 CodeDeploy 권한이 있는 서비스 역할을 입력합니다.

my-codedeploy-iam

arn:aws:iam::829643732047:role/my-codedeploy-iam

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.

다음 환경 구성에서는 Amazon EC2 인스턴스를 선택하고 2.1에서 태그를 추가했다면 아래와 같이 해당 태그의 인스턴스를 선택할 수 있다.

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

태그 제거

태그 추가

+ 태그 그룹 추가

☐ 온프레미스 인스턴스

일치하는 인스턴스

1개의 일치하는 고유한 인스턴스. [자세한 내용을 보려면 여기를 클릭하십시오.](#)

나머지 설정은 아래와 같이 해주고 배포 그룹을 생성한다.

AWS Systems Manager를 사용한 에이전트 구성 정보

AWS Systems Manager가 CodeDeploy 에이전트를 설치하기 전에 필요한 사전 요구 사항을 완료합니다.
AWS Systems Manager Agent가 모든 인스턴스에 설치되어 있는지 확인하고 필요한 IAM 정책을 인스턴스에 연결합니다. [자세히 알아보기](#)

AWS CodeDeploy 에이전트 설치

- ☐ 안 함
- ☒ 한 번만
- ☐ 지금 업데이트 및 업데이트 일정 예약

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

4. Github Actions에서 사용할 IAM 사용자 추가

1. IAM 사용자 생성
2. Access Key, Secret Key 생성 및 확인

4.1 IAM 사용자 생성

IAM → 액세스 관리 → 사용자 → 사용자 생성

[IAM](#) > 사용자

사용자 (2) 정보

IAM 사용자는 계정에서 AWS와 상호 작용하는 데 사용되는 장기 자격 증명을 가진 자격 증명입니다.

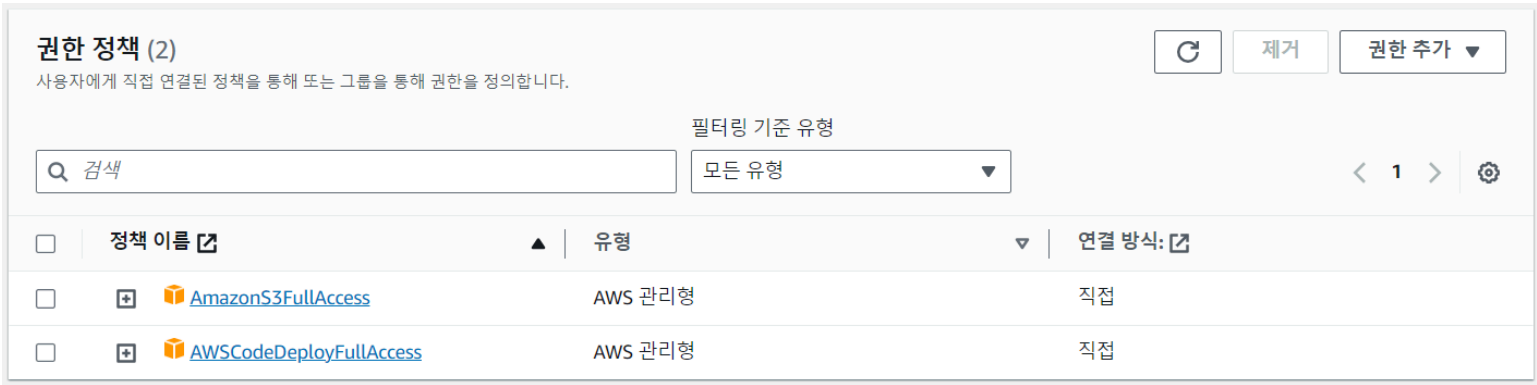
[새로고침]
[삭제]
사용자 생성

검색

<input type="checkbox"/>	사용자 이름 ▲	경로 ▼	그룹 ▼	마지막 활동 ▼	MFA ▼	암호 수명 ▼	콘솔 마지막 로그인
<input type="checkbox"/>	my-github-actions-iam-user	/	0	✔ 3시간 전	-	-	-
<input type="checkbox"/>		/	0	✔ 2시간 전	-	-	-

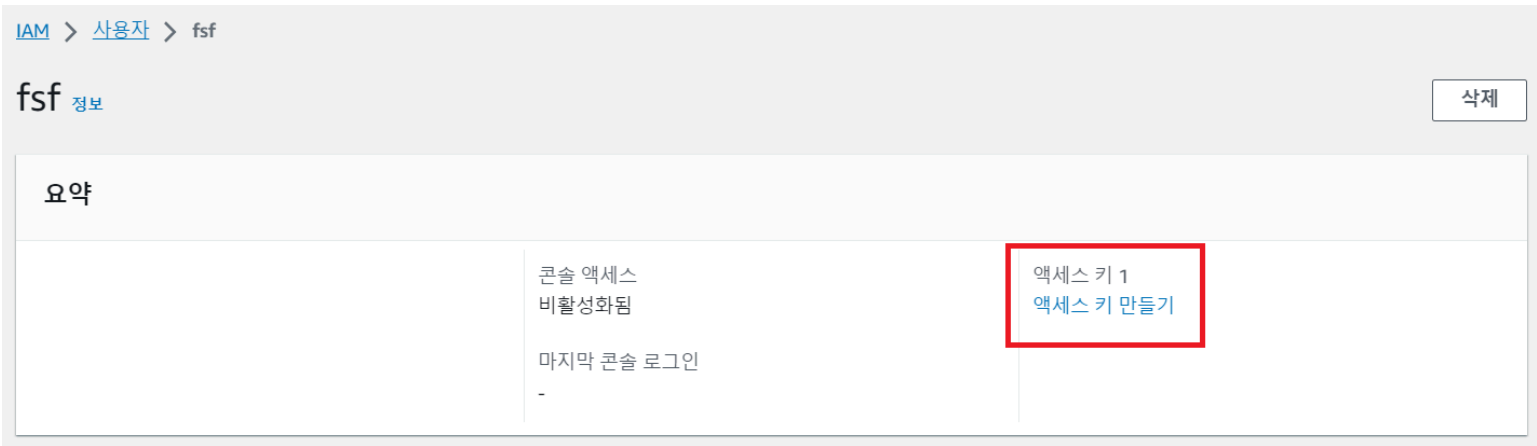
직접 정책 연결을 선택하고

AmazonS3FullAccess, AWSCodeDeployRole 권한을 추가하여 사용자를 생성한다.



4.2 Access Key, Secret Key 생성 및 확인

생성한 사용자 이름을 클릭하여 들어가서 아래와 같이 액세스 키 만들기를 클릭한다.

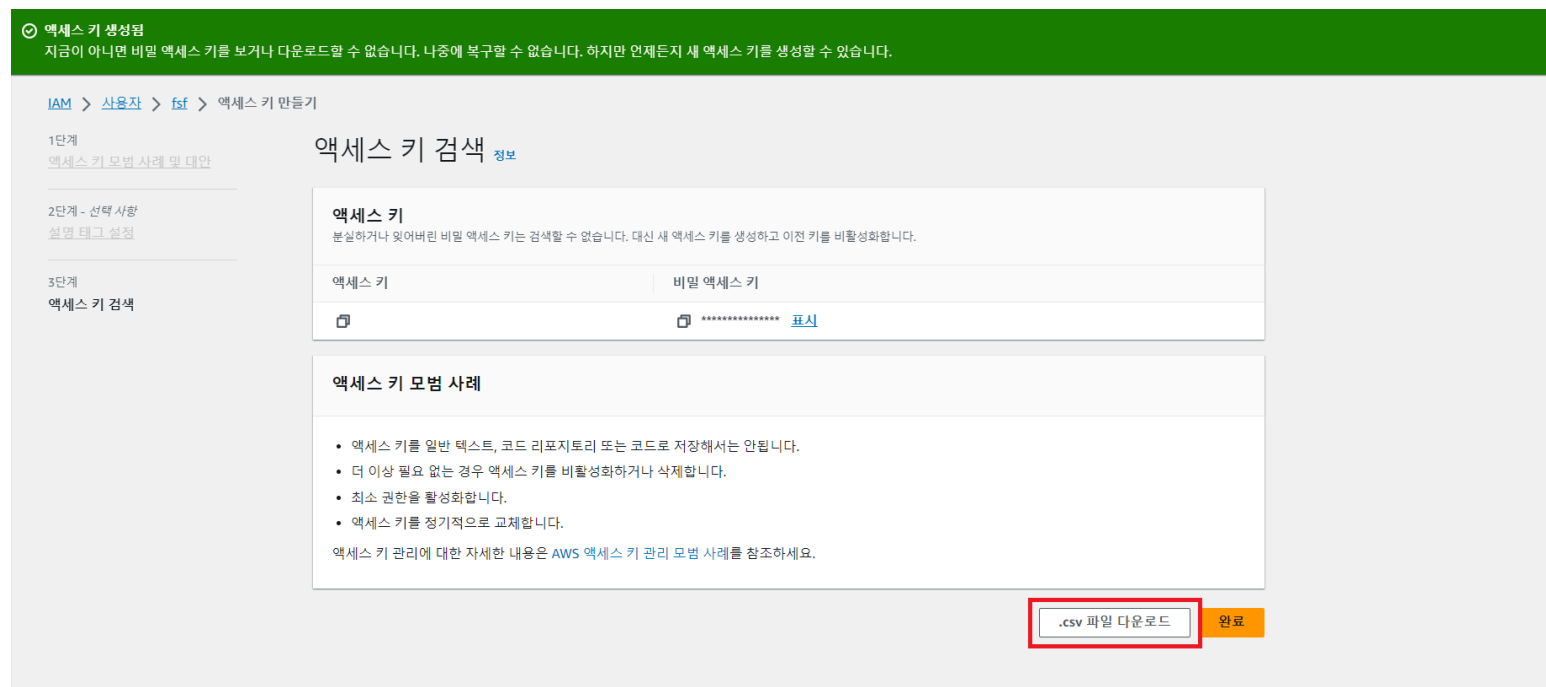


그러면 아래와 같은 화면이 뜨는데,
이 화면에서는 아무거나 선택해도 무방하다.



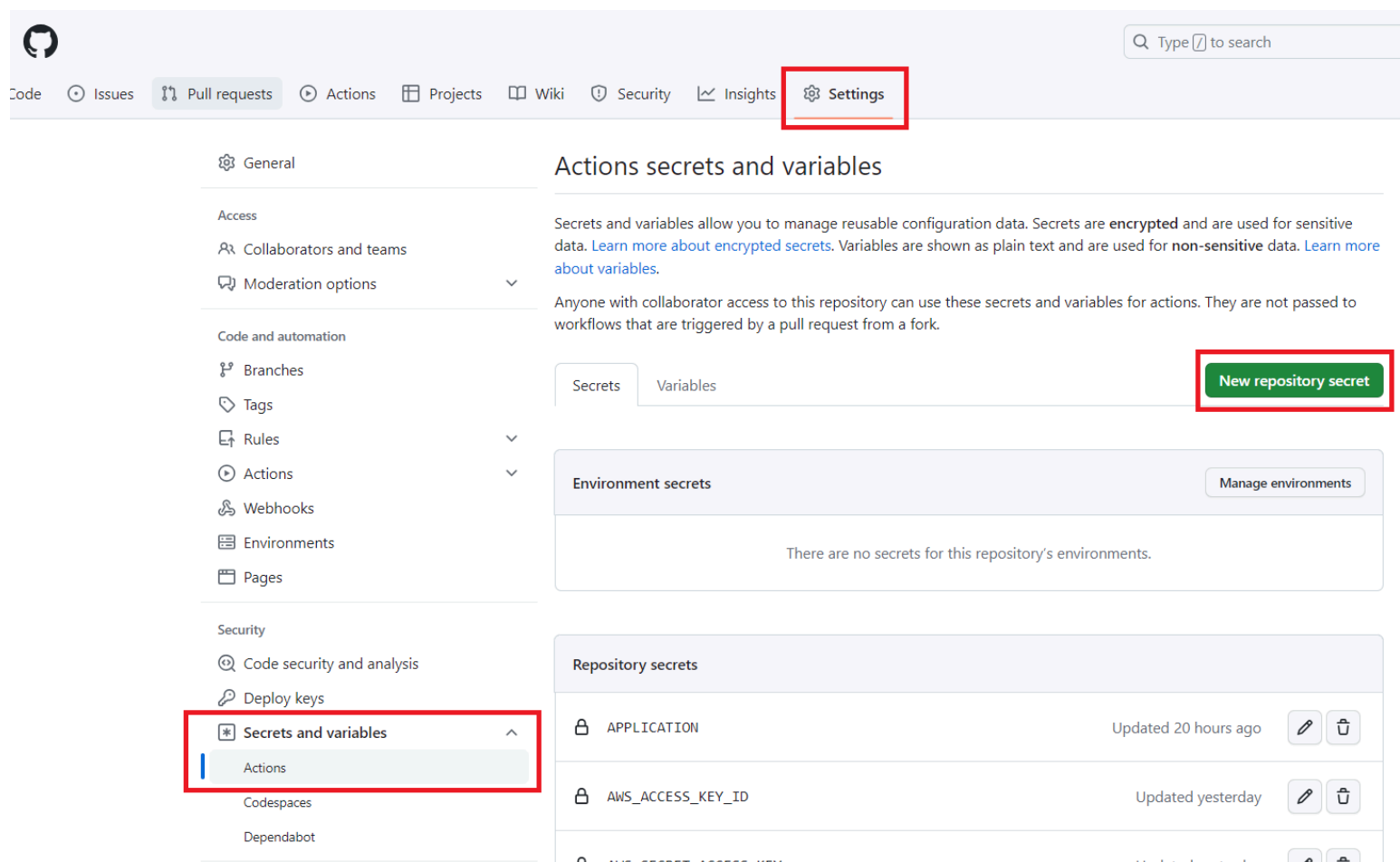
액세스 키를 생성하게 되면
Access Key, Secret Key가 발급이 되는데

! 이때, Secret Key는 이 화면을 나가게 되면 더이상 알 수 있는 방법이 없기 때문에 꼭 csv파일로 가지고 있는 것을 추천한다.












5. Github Repository 의 Secrets 추가

Github Repository → Settings → Secrets and variables → Actions → New repository secret



Access Key, Secret Key를 등록해준다.

Repository secrets		
 APPLICATION	Updated 20 hours ago	 
 AWS_ACCESS_KEY_ID	Updated yesterday	 
 AWS_SECRET_ACCESS_KEY	Updated yesterday	 

! 여기서 APPLICATION 키는 뭘까?

APPLICATION 키는 바로 Spring 프로젝트 내의 `application.properties` 파일의 내용을 가지고 있는 키다.

Spring 프로젝트 내에 있는 `application.properties` 파일은 각종 민감정보를 담고있는 파일이기 때문에 깃허브에 올릴 때는 `.gitignore` 에 작성해서 깃허브에 올라가지 않도록 설정하는데, github action을 사용해서 깃허브에 푸쉬한 코드로 빌드를 해야하기 때문에 `application.properties` 파일이 필요하다!

그렇기 때문에 우리는 Scrots에 `application.properties` 파일의 내용을 저장해두고 빌드할 때마다 파일을 생성하여 빌드해주려고 한다.

- 깃허브에 올라가는 것은 아니다!

`application.properties` 파일의 내용을 모두 복붙하여 키를 생성해두면 된다.

6. appspec.yml 파일 작성

Spring 프로젝트 루트 디렉토리에 `appspec.yml` 파일을 생성하고 아래와 같이 작성한다.

```
# appspec.yml

version: 0.0
os: linux

files:
  - source: /
    destination: /home/ubuntu/app
    overwrite: yes

permissions:
  - object: /
    pattern: "*"
    owner: ubuntu
    group: ubuntu

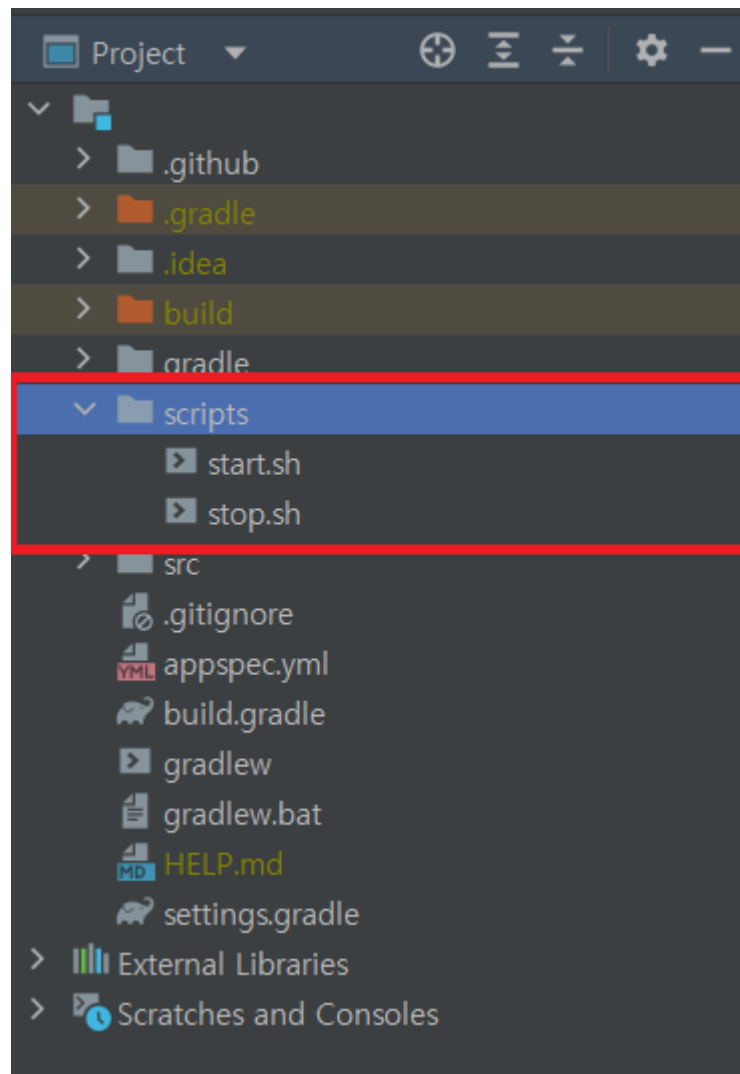
hooks:
  AfterInstall:
    - location: scripts/stop.sh
      timeout: 60
      runas: ubuntu
  ApplicationStart:
```

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

7. 배포 스크립트 작성

1. start.sh 작성
2. stop.sh 작성
3. build.gradle 파일 수정

아래와 같이 프로젝트 루트 디렉토리 아래 scripts 디렉토리를 만들고 그 안에 start.sh 와 stop.sh 를 작성했다.



7.1 start.sh 작성

```
#!/usr/bin/env bash

PROJECT_ROOT="/home/ubuntu/app"
JAR_FILE="$PROJECT_ROOT/spring-webapp.jar"

APP_LOG="$PROJECT_ROOT/application.log"
ERROR_LOG="$PROJECT_ROOT/error.log"
DEPLOY_LOG="$PROJECT_ROOT/deploy.log"

TIME_NOW=$(date +%c)

# build 파일 복사
echo "$TIME_NOW > $JAR_FILE 파일 복사" >> $DEPLOY_LOG
cp $PROJECT_ROOT/build/libs/*.jar $JAR_FILE
```

```
# jar 파일 실행
echo "$TIME_NOW > $JAR_FILE 파일 실행" >> $DEPLOY_LOG
nohup java -jar $JAR_FILE > $APP_LOG 2> $ERROR_LOG &

CURRENT_PID=$(pgrep -f $JAR_FILE)
echo "$TIME_NOW > 실행된 프로세스 아이디 $CURRENT_PID 입니다." >> $DEPLOY_LOG
```

7.2 stop.sh 작성

```
#!/usr/bin/env bash

PROJECT_ROOT="/home/ubuntu/app"
JAR_FILE="$PROJECT_ROOT/spring-webapp.jar"

DEPLOY_LOG="$PROJECT_ROOT/deploy.log"

TIME_NOW=$(date +%c)

# 현재 구동 중인 애플리케이션 pid 확인
CURRENT_PID=$(pgrep -f $JAR_FILE)

# 프로세스가 켜져 있으면 종료
if [ -z $CURRENT_PID ]; then
    echo "$TIME_NOW > 현재 실행중인 애플리케이션이 없습니다" >> $DEPLOY_LOG
else
    echo "$TIME_NOW > 실행중인 $CURRENT_PID 애플리케이션 종료 " >> $DEPLOY_LOG
    kill -15 $CURRENT_PID
fi
```

7.3 build.gradle 파일 수정

build.gradle에 아래 코드를 추가해준다.

```
jar {
    enabled = false
}
```

8. Github Actions Workflow 작성

1. Github Actions Workflow 생성
2. deploy.yml 파일 작성

8.1 Github Actions Workflow 생성

Github Repository → Actions → Simple workflow 선택

Get started with GitHub Actions

Build, test, and deploy your code. Make code reviews, branch management, and issue triaging work the way you want. Select a workflow to get started.

Skip this and [set up a workflow yourself](#) →

Categories

Found 4 workflows

Simple workflow

By GitHub

Start with a file with the minimum necessary structure.

Configure

clj-holmes

By Matheus Bernardes

A Static Application Security Testing tool to find vulnerable Clojure code via rules that use a simple pattern language.

Requires [GitHub Advanced Security](#)

Flawfinder

By David A. Wheeler

Flawfinder is a simple program that scans C/C++ source code and reports potential security flaws.

Requires [GitHub Advanced Security](#)

8.2 deploy.yml 파일 작성

```
name: Deploy to Amazon EC2

on:
  push:
    branches:
      - main #브랜치명

# 본인이 설정한 값을 여기서 채워넣는다.
env:
  AWS_REGION: us-east-1 #리전
  S3_BUCKET_NAME: test-bucket #버킷 이름
  CODE_DEPLOY_APPLICATION_NAME: test-app #CodeDeploy 애플리케이션 이름
  CODE_DEPLOY_DEPLOYMENT_GROUP_NAME: test-deployment-group #CodeDeploy 배포 그룹 이름

permissions:
  contents: read

jobs:
  deploy:
    name: Deploy
    runs-on: ubuntu-latest
    environment: production

    steps:
      # (1) 기본 체크아웃
      - name: Checkout
        uses: actions/checkout@v3

      # (2) application.properties 설정
      - uses : actions/checkout@v3
      - run: touch ./src/main/resources/application.properties
      - run: echo "${{ secrets.APPLICATION }}" > ./src/main/resources/application.properties
      - run: cat ./src/main/resources/application.properties

      # (3) gradlew 권한 추가
      - name: Run chmod to make gradlew executable
```

```

run: chmod +x ./gradlew

# (4) JDK 11 세팅
- name: Set up JDK 11
  uses: actions/setup-java@v3
  with:
    distribution: 'temurin'
    java-version: '11'

# (5) Gradle build (Test 제외)
- name: Build with Gradle
  uses: gradle/gradle-build-action@0d13054264b0bb894ded474f08ebb30921341cee
  with:
    arguments: clean build -x test

# (6) AWS 인증 (IAM 사용자 Access Key, Secret Key 활용)
- name: Configure AWS credentials
  uses: aws-actions/configure-aws-credentials@v1
  with:
    aws-access-key-id: ${ secrets.AWS_ACCESS_KEY_ID }
    aws-secret-access-key: ${ secrets.AWS_SECRET_ACCESS_KEY }
    aws-region: ${ env.AWS_REGION }

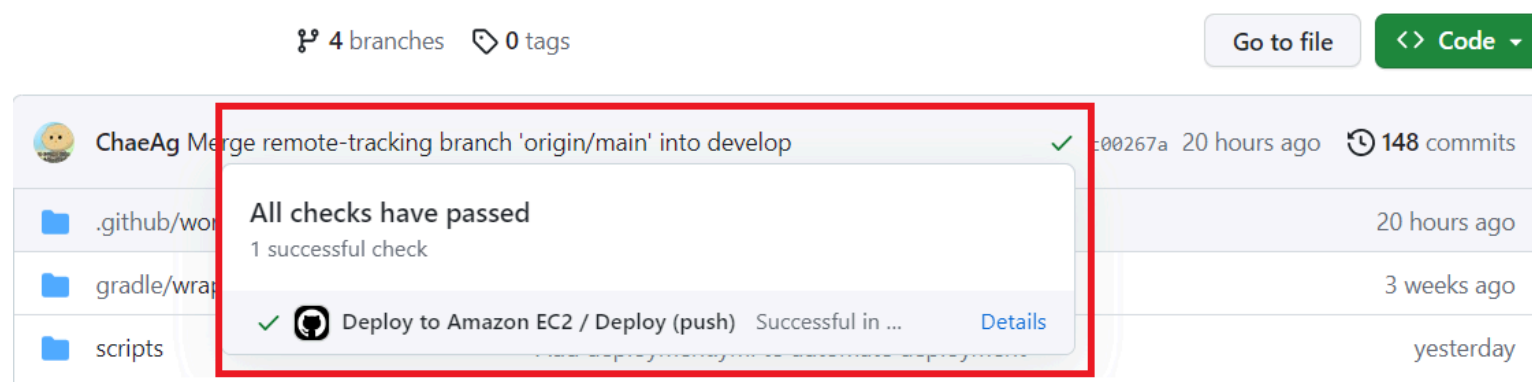
# (7) 빌드 결과물을 S3 버킷에 업로드
- name: Upload to AWS S3
  run: |
    aws deploy push \
      --application-name ${ env.CODE_DEPLOY_APPLICATION_NAME } \
      --ignore-hidden-files \
      --s3-location s3://$S3_BUCKET_NAME/$GITHUB_SHA.zip \
      --source .

# (8) S3 버킷에 있는 파일을 대상으로 CodeDeploy 실행
- name: Deploy to AWS EC2 from S3
  run: |
    aws deploy create-deployment \
      --application-name ${ env.CODE_DEPLOY_APPLICATION_NAME } \
      --deployment-config-name CodeDeployDefault.AllAtOnce \
      --deployment-group-name ${ env.CODE_DEPLOY_DEPLOYMENT_GROUP_NAME } \
      --s3-location bucket=$S3_BUCKET_NAME,key=$GITHUB_SHA.zip,bundleType=zip

```

9. 세팅 끝. 배포 테스트

action이 일어나면 워크 플로우가 동작하도록 설정한 브랜치에 push를 한 뒤,
워크 플로우가 정상적으로 수행이 끝나면 아래와 같이 ✓표시가 뜬다.



Actions 탭에서 보고 싶은 워크 플로우를 클릭하면 아래와 같이 수행 과정을 볼 수 있다.
만약 에러가 난다면 어떤 부분에서 에러가 났는지도 확인이 가능하다.

The screenshot shows the GitHub Actions interface for a workflow named "Deploy to Amazon EC2". The workflow is in a "Completed" state, indicated by a green checkmark and the number "#14". The left sidebar shows the "Summary" tab selected. The main area displays a list of steps that were executed successfully:

- Set up job
- Checkout
- Run actions/checkout@v3
- Run touch ./src/main/resources/application.properties
- Run echo *****
- Run cat ./src/main/resources/application.properties
- Run chmod to make gradlew executable
- Set up JDK 11
- Build with Gradle
- Configure AWS credentials
- Upload to AWS S3
- Deploy to AWS EC2 from S3
- Post Configure AWS credentials
- Post Build with Gradle
- Post Set up JDK 11
- Post Run actions/checkout@v3
- Post Checkout
- Complete job

배포가 완료된 후 AWS S3에는 zip형식의 빌드파일이 저장되어 있고

The screenshot shows the AWS S3 console for a bucket named "-bucket". The "Objects" tab is selected, showing a list of objects. One object is highlighted with a red box:

이름	유형	마지막 수정	크기	스토리지 클래스
418e5ac535191c15ac2f9de3c608f2fa2b269eee.zip	zip	2023. 9. 20. pm 10:23:24 PM KST	52.1MB	Standard

AWS CodeDeploy에서 배포 내역을 확인할 수 있다.

