

프로젝트를 진행하면서, CI/CD 설정을 언젠간 해보고 싶다는 생각을 했습니다. 전 회사에서도 배포 비슷한 것을 해보았지만 직접 파일을 서버로 옮긴 뒤 실행하는 것이 다였던 터라, 자동 CI/CD에 대해 굉장히 궁금한 점이 많았습니다.

그래서! 요번에 한 번 해보기로 했습니다. 다만 CI 보다는 CD에 초점을 맞춘 점을 미리 말씀드립니다:)

## 어떻게 배포할 것인가

여러 방법이 있겠지만, 최종적으로 아래의 2가지 중에 하나를 선택하기로 했습니다.

### 1. Github Action + AWS S3 + AWS CodeDeploy + AWS EC2

- Github action을 통해 코드를 빌드하고, 이를 압축하여 Aws S3에 업로드, 그리고 원격 서버에서 S3로부터 파일을 가져다가 압축 해제 후 배포하도록 하는 방식

### 2. Github Action + Docker + Docker Compose + AWS EC2

- Github action을 통해 코드를 빌드하고, 이를 도커 레포지토리에 push, 그리고 ssh로 원격 서버에 접속하여 도커의 이미지를 pull하여 Docker-Compose를 통해 실행시키는 방식

## 무엇을, 왜 선택했는가

저는 여기서 2번을 선택했는데요, 이유는 다음과 같습니다.

먼저 **Docker 및 Docker Compose**를 사용할 경우 확장성 면에서 더 낫다고 생각되었습니다.

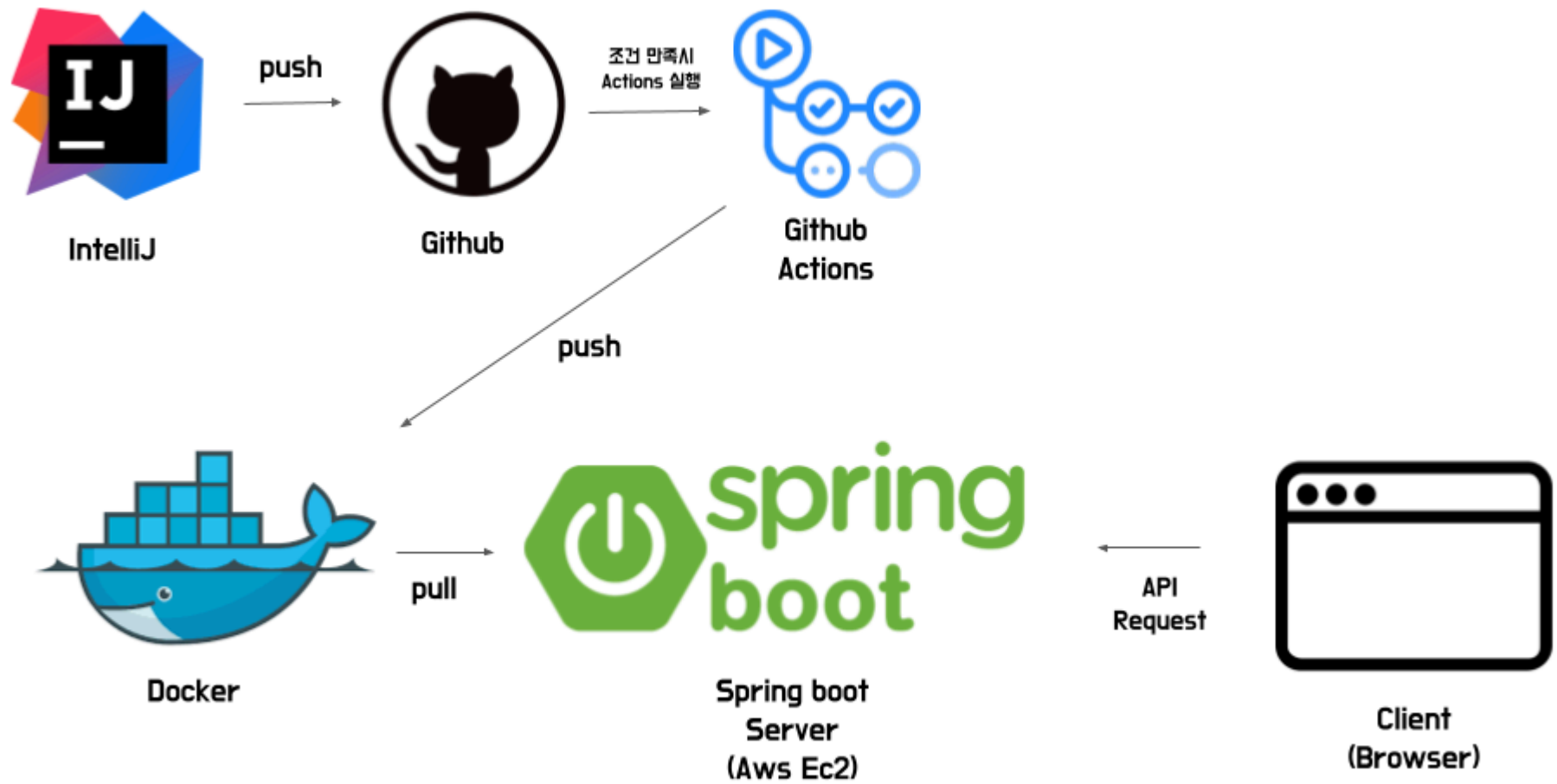
1번의 경우, 단순히 빌드된 코드를 서버에 전달하고, 이를 배포하는데 집중하고 있습니다. 만약 다른 연계되는 프로그램들이 있다면, 추가적인 설정이 필요합니다.

반면에 2번은 Docker-Compose를 사용하고 있는데, docker-compose.yml 파일의 설정을 통해서 다른 프로그램들과의 연계를 수월하게 가져갈 수 있습니다. 예를 들어서 빌드된 코드의 프로그램과 Redis, mysql을 연동해야 한다면, compose를 통해 연계에 대해 설정해줄 수 있는 것이죠.

한편으로는 전에 Aws S3는 사용해보기도 해서, 사용하지 않았던 Docker를 써보고 싶었던 마음도 있었습니다 🙌

## 배포 프로세스 살펴보기

계획한 배포 프로세스를 아래처럼 도식화 해보았습니다.

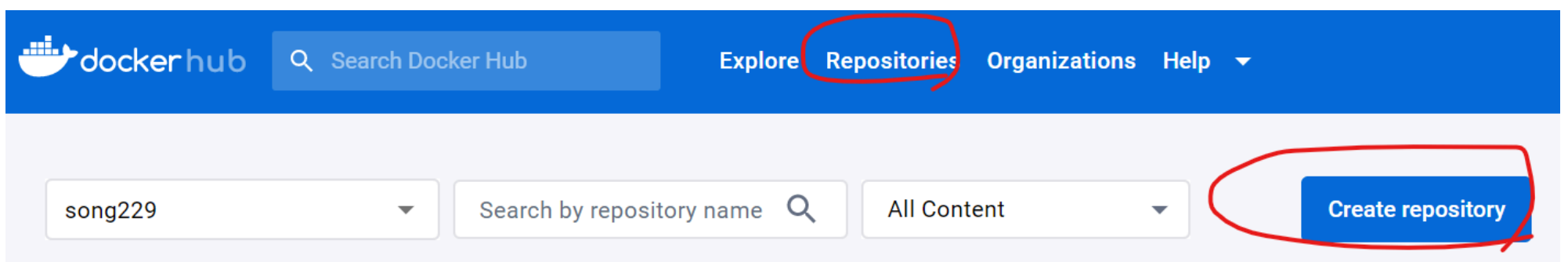


1. 인텔리제이에서 코드를 작성 후, Github으로 Push.
2. Github에서 조건을 만족할 경우, Github Actions 실행.  
(저는 main branch에 코드가 Push, Pull Request될 경우로 설정했습니다.)
3. Github Actions는 미리 작성해 둔 .yml 파일의 내용 대로 프로세스가 진행되는데요, 저는  
1. 빌드 2. 도커 repository에 이를 push 3. ssh로 서버에서 도커 이미지 pull 받아 실행  
의 프로세스를 만족하도록 파일을 작성했습니다. 어떻게 했는지에 대한 자세한 내용은 아래에서 살펴보기로 하시죠!

## 구체적인 설정들

### 1. 도커 - Repository 만들기

- 도커를 사용할 것이기 때문에, 미리 이미지를 저장할 Repository를 만들어줍니다.
- 로그인 후(회원 가입 필요), Repository로 가서 Create repository를 합니다. 이 때 만든 Repository Name을 계속 사용할 것이므로 기억합니다.



### 2. Github Actions 시 실행될 프로세스를 담은 yml 파일 만들기

- 여기서 만드는 파일의 내용을 통해 Github Actions가 무엇을 수행할지를 정의할 수 있습니다. 다음은 제가 작성한 yml 파일입니다. 주석을 통해 내용을 설명해보겠습니다.

```
name: CD with Gradle
```

```
# 언제 이 파일의 내용이 실행될 것인지 정의합니다.
```

```

# 여기서는 main 브랜치에 코드가 push 되거나 pull_request되었을 때 실행할 것을 말하고 있습니다.
on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]

# 코드의 내용을 이 파일을 실행하여 action을 수행하는 주체(Github Actions에서 사용하는 VM)가 읽을 수 있도록 허용합니다.
permissions:
  contents: read

# 실제 실행될 내용들을 정의합니다.
jobs:
  build:
    runs-on: ubuntu-latest # ubuntu 최신 버전에서 script를 실행

    steps:
      # 지정한 저장소(현재 REPO)에서 코드를 워크플로우 환경으로 가져오도록 하는 github action
      - uses: actions/checkout@v3

      # open jdk 17 버전 환경을 세팅
      - name: Set up JDK 17
        uses: actions/setup-java@v3
        with:
          java-version: '17'
          distribution: "adopt"

      # Github secrets로부터 데이터를 받아서, 워크 플로우에 파일을 생성
      - name: Make application.properties
        run: |
          cd ./src/main/resources
          touch ./application.properties
          echo "${{ secrets.PROPERTIES }}" > ./application.properties
        shell: bash

      # gradle을 통해 소스를 빌드.
      - name: Build with Gradle
        run: |
          chmod +x ./gradlew
          ./gradlew clean build -x test

      # dockerfile을 통해 이미지를 빌드하고, 이를 docker repo로 push 합니다.
      # 이 때 사용되는 ${ secrets.DOCKER_REPO }}/directors-dev 가 위에서 만든 도커 repository 입니다.
      - name: Docker build & push to docker repo
        run: |
          docker login -u ${ secrets.DOCKER_USERNAME } -p ${ secrets.DOCKER_PASSWORD }
          docker build -f Dockerfile -t ${ secrets.DOCKER_REPO }}/directors-dev .
          docker push ${ secrets.DOCKER_REPO }}/directors-dev

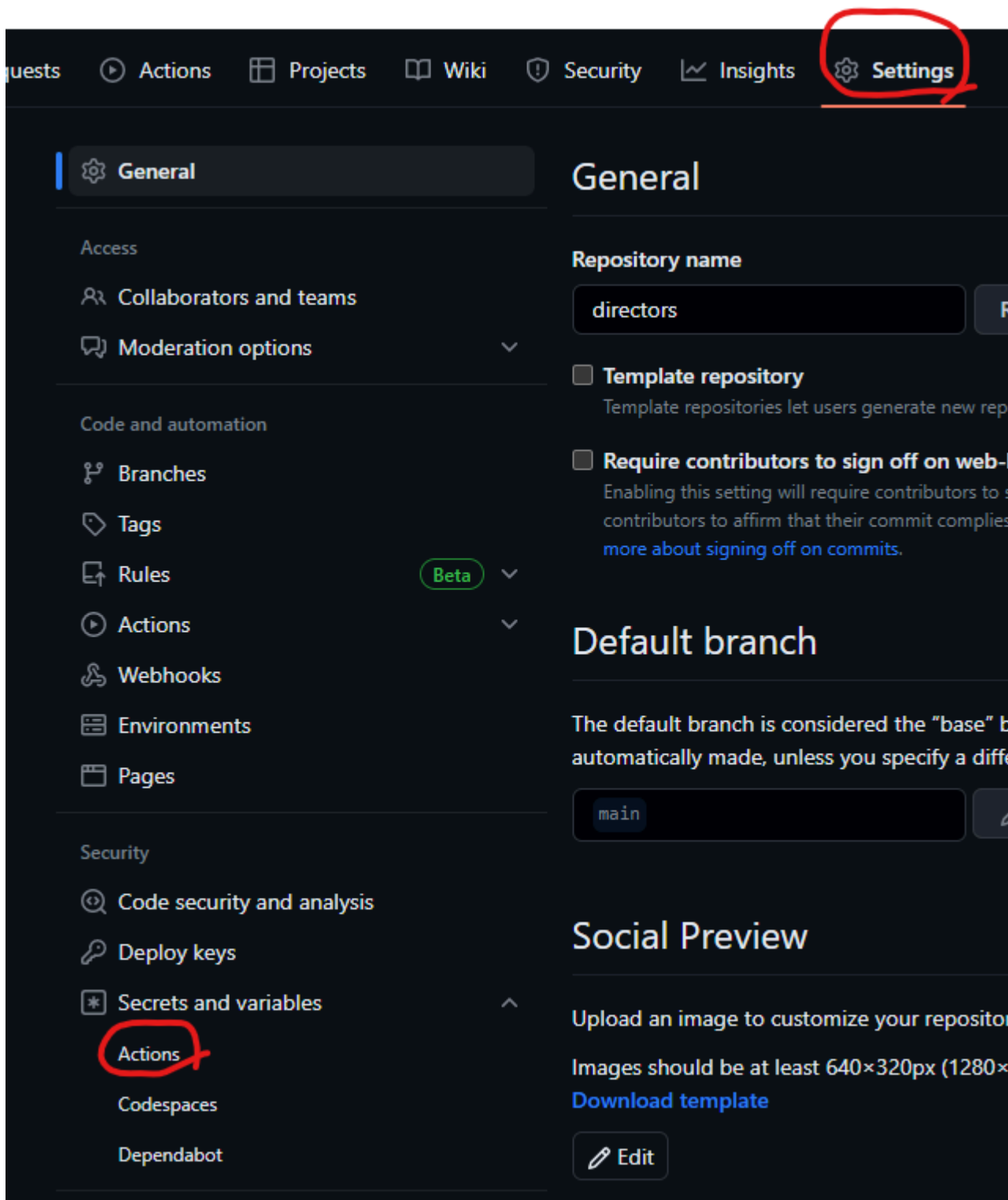
      # appleboy/ssh-action@master 액션을 사용하여 지정한 서버에 ssh로 접속하고, script를 실행합니다.
      # script의 내용은 도커의 기존 프로세스들을 제거하고, docker repo로부터 방금 위에서 push한 내용을 pull 받아 실행하는 것입니다.
      # 실행 시, docker-compose를 사용합니다.
      - name: Deploy to server
        uses: appleboy/ssh-action@master
        id: deploy
        with:
          host: ${ secrets.HOST }
          username: ubuntu
          key: ${ secrets.KEY }
          envs: GITHUB_SHA
        script: |
          sudo docker rm -f $(docker ps -qa)
          sudo docker pull ${ secrets.DOCKER_REPO }}/directors-dev
          docker-compose up -d
          docker image prune -f

```

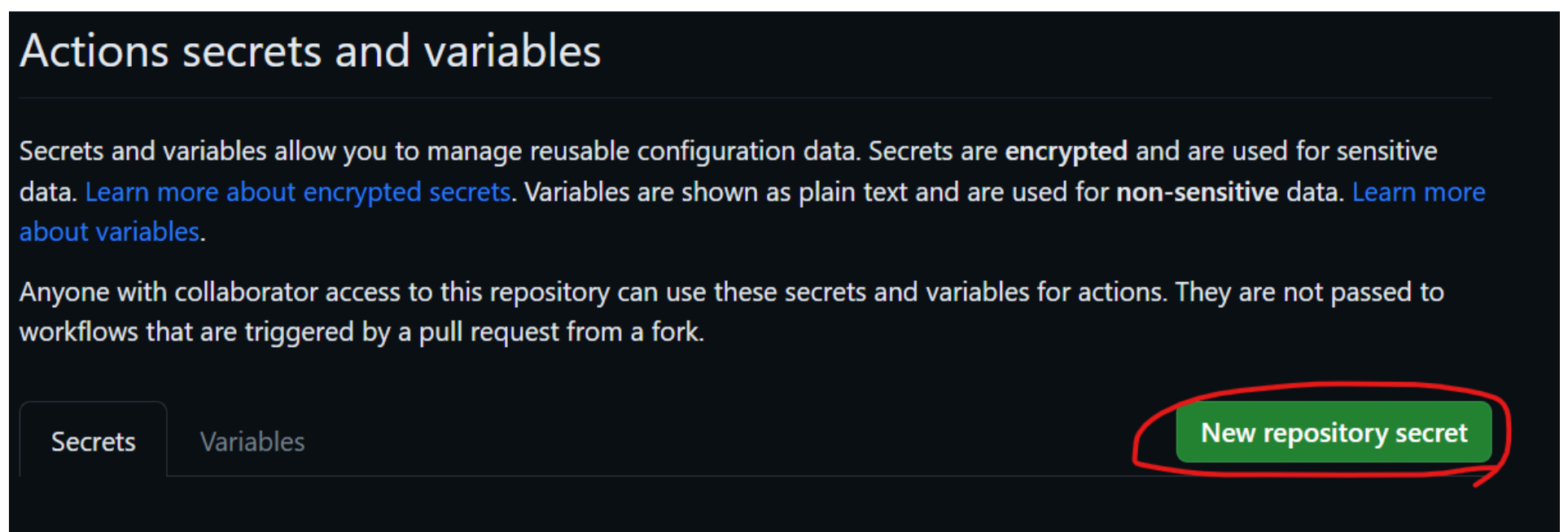
- 여기서 나오는 `${ secrets.~}` 로 이루어진 데이터들은, Github secrets에 의해 저장된 변수를 사용합니다. Github secrets는 Github Repository에서 직접 입력할 수 있습니다. 자세한 내용은 아래에서 설명하겠습니다.
- 이 파일은 미리 github의 repository의 `/.github/workflows/파일명.yml`로 업로드되어있어야 합니다. 깃허브는 이 파일이 repository에 존재한 이후에만 github actions를 실행하기 때문입니다.

### 3. Github Secrets 설정하기



















- 위의 yml에서 사용되는 secrets 변수들을 github repository를 통해 넣어줄 수 있습니다.



- repository의 Settings - Secrets and variables - Actions로 이동합니다.



- 그리고 New repository secret을 통해 secret들을 넣어줍니다.

| Repository secrets   |                 |   |
|--|-----------------|---|
|   | DOCKER_PASSWORD | Updated 12 hours ago    |
|   | DOCKER_REPO     | Updated 12 hours ago    |
|   | DOCKER_USERNAME | Updated 12 hours ago    |
|   | HOST            | Updated 12 hours ago    |
|   | KEY             | Updated 12 hours ago    |
|  | PROPERTIES      | Updated 3 hours ago   |

- 이제 yml은 여기 있는 secret들을 참조하여 사용할 수 있습니다.
- 가끔 Secrets를 참조하지 못하는 경우가 발생하기도 합니다. 그 이유를 알고 싶으시다면 ==> Github Secrets가 동작하지 않는다고요? 그건 아마도..

## 4. 빌드 결과물을 배포할 서버 세팅

- 저는 AWS의 EC2 서버(Ubuntu 22.04)를 사용했습니다. 여기서 EC2 세팅에 대한 부분을 자세히 다루지는 않겠습니다.
- Docker를 사용할 것이므로 서버에 접속하여 Docker를 미리 설치해줍니다.

```
sudo apt-get update
sudo apt-get install docker.io
sudo service docker start
```

- Docker-compose도 설치해줍니다.

```
sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
```

- docker-compose --version 을 통해 잘 설치되었는지 확인합니다.
- docker-compose를 사용하기 위해서, 이 역시 yml 파일을 세팅해줄 필요가 있습니다.
  - 서버의 루트 디렉토리로 이동합니다.
  - docker-compose.yml 파일을 생성합니다.(sudo vi docker-compose.yml)

```
version: '3'

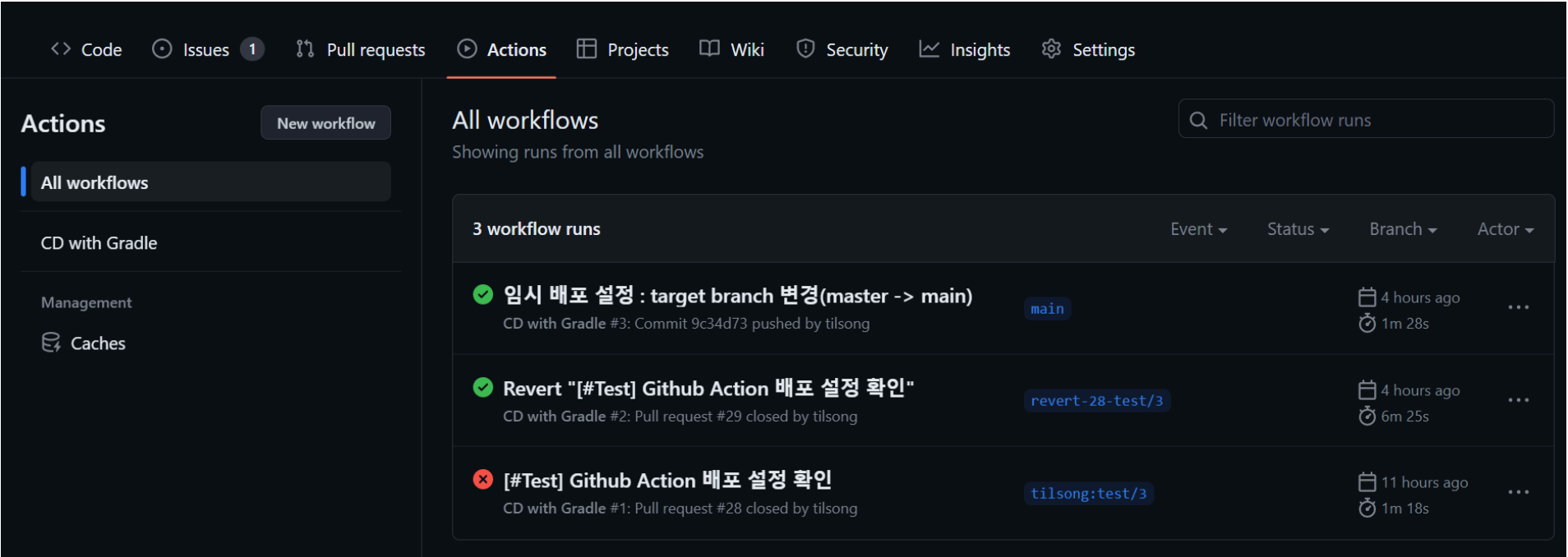
services:

  web:
    container_name: web
    image: song229/springboot-github-actions-test
    expose:
      - 8080
    ports:
      - 8080:8080
```

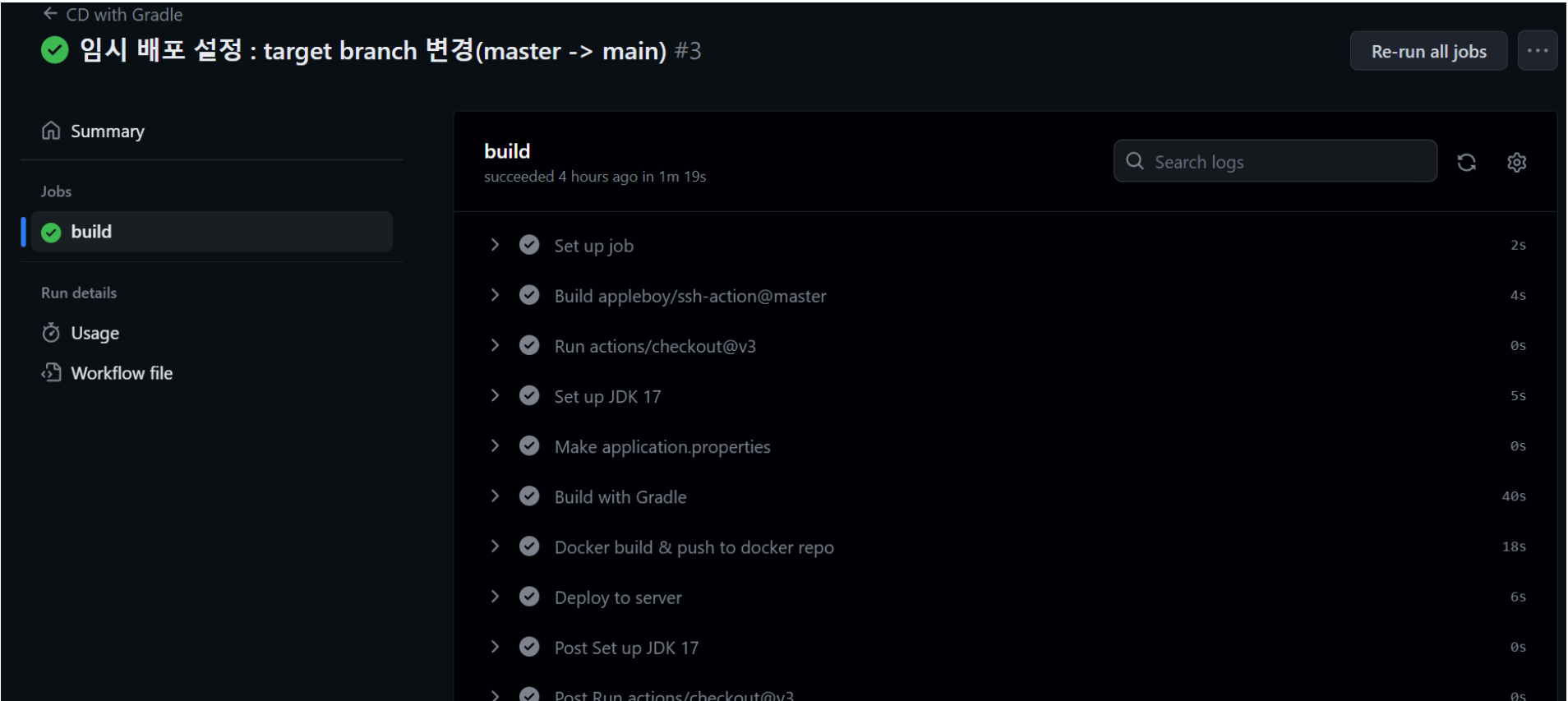
- 이제 서버의 준비도 거의 끝났습니다. 아래 2가지만 더 확인해줍니다.
  - 생성한 서버에 대한 주소 및 사용되는 키를 Github Secrets에서 넣어주기(위의 HOST, KEY)
  - Github Actions에서 ssh를 통해 원격 서버에 접속해야 하므로 22번 port는 0.0.0.0에 대하여 인바운드를 열어두기

# 실행하기

- 필요한 세팅은 거의 완료된 것 같습니다. 이제 코드를 로컬 레포지토리에서 원격으로 push해봅시다. 현재 yml파일은 push나 pull\_request에 반응하여 github actions 프로세스를 실행하므로 정상 작동할 것입니다.
- 확인하기
  - 프로세스가 잘 돌아가고 있는지 확인하고 싶다면, github repository의 Actions를 접속하면 됩니다.



- 실시간으로 github action이 실행되는 모습을 확인할 수 있습니다.



# 자동 배포 프로세스 완료!

해두기는 귀찮지만, 한 번 해두면 두고두고 잘 쓰는 CD 프로세스를 작성해보았습니다.