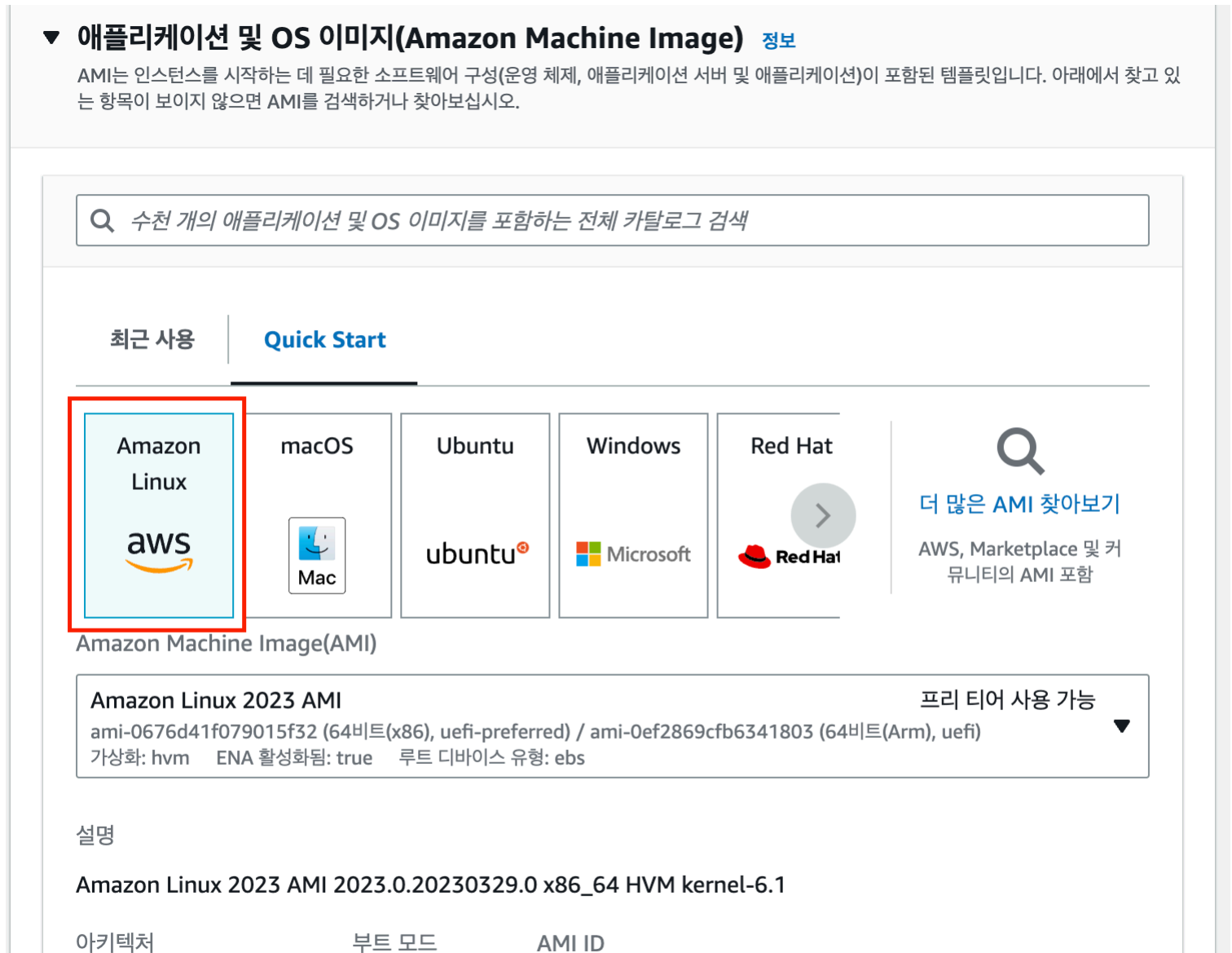
AWS + Docker + Github Action 사용한 서버 자동배포 [Spring CICD]

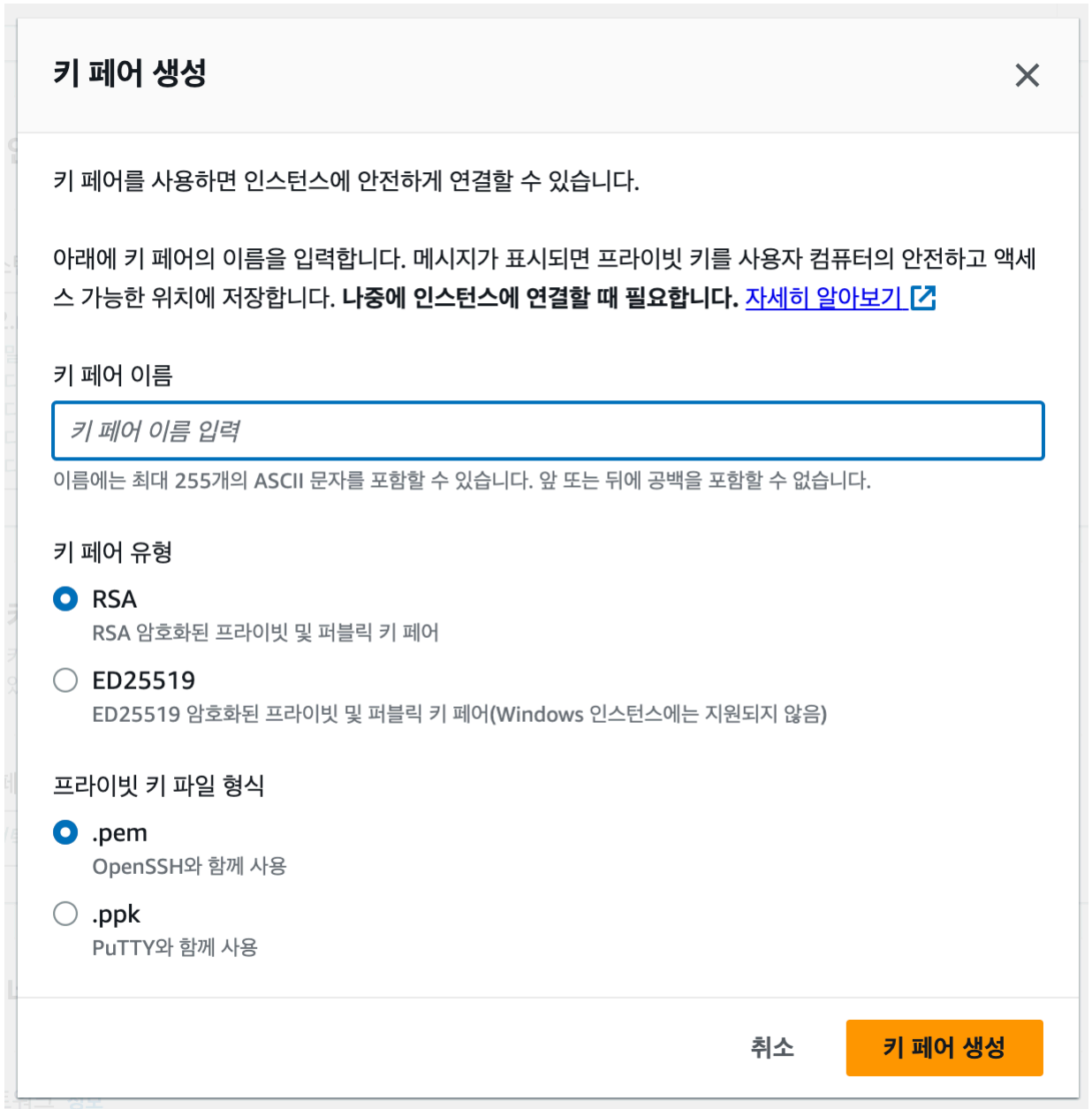
⚙️ **개발 환경**

- Docker
- AWS EC2 Amazone Linux 2
- Github Action
- Spring boot
- Java 17
- Gradle

📍 AWS EC2 인스턴스 생성하기



Amazone Linux 이미지를 프리티어로 사용함



키페어 생성해서 저장해놓기 (지워버리지 않도록 유의하기)

그 외에는 스토리지만 20GB로 변경하고 인스턴스 생성함
이후 보안 그룹 설정해주기 (개발 시 다른 개발자도 접근 가능하도록 모든 위치에서 접근 가능하도록 설정함)

📍 AWS EC2 인스턴스에 도커 설치하기

키페어가 있는 폴더 내부에서 아래의 명령어를 통해 ssh 접속

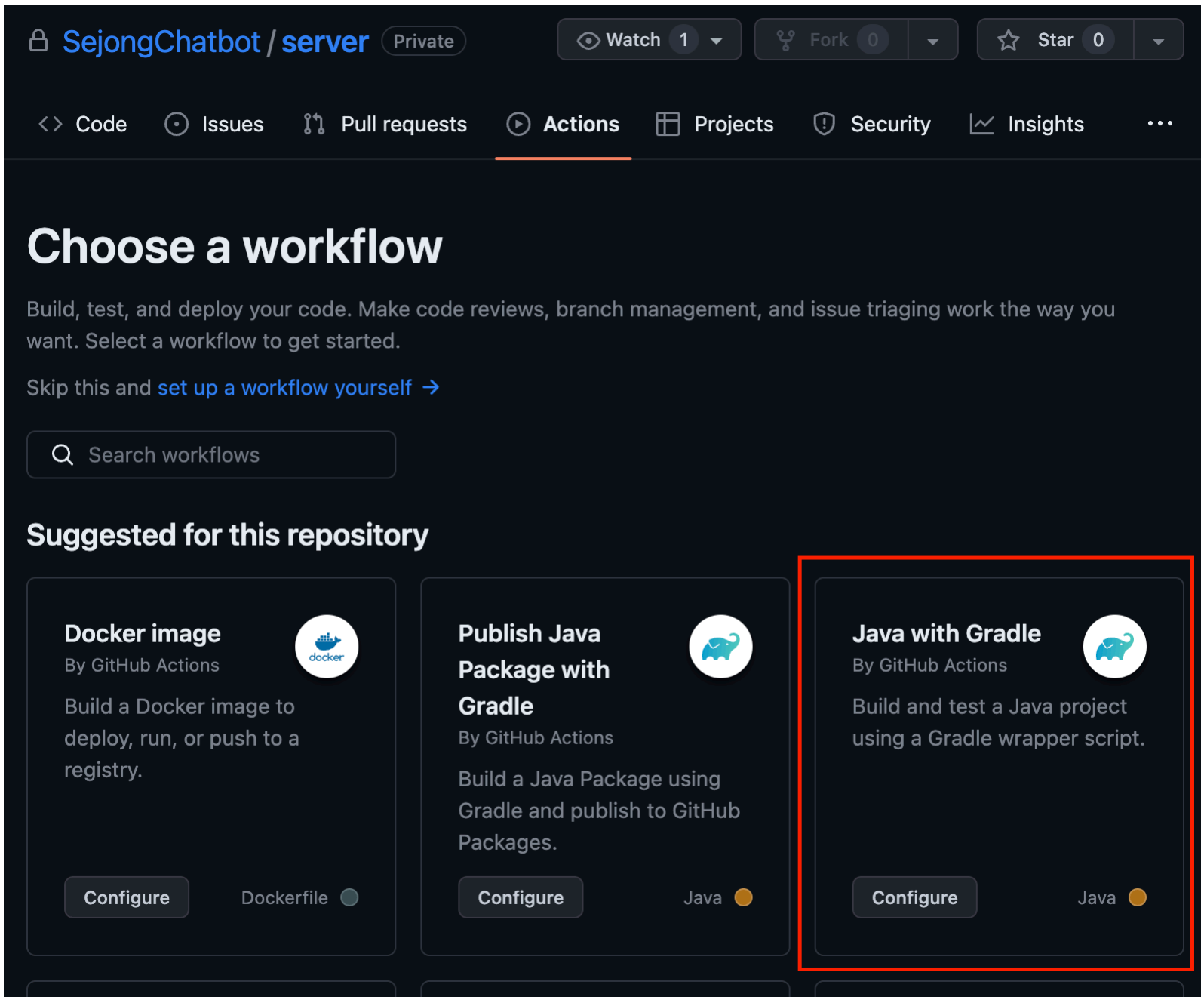
```
ssh -i "키페어 파일이름" "퍼블릭 DNS 주소"
```

이후 아래의 과정을 통해 docker, docker-compose 설치

```
//도커 설치
sudo yum install docker -y

//도커 실행
sudo service docker start

//도커 상태 확인
systemctl status docker.service

//도커 관련 권한 추가
sudo chmod 666 /var/run/docker.sock
docker ps

//최신 버전 docker-compose 설치
sudo curl -L https://github.com/docker/compose/releases/latest/download/docker-compose-$(uname -s)-$(uname -m) -o /usr/local/bin/docker-compose

//권한 추가
sudo chmod +x /usr/local/bin/docker-compose

//버전 확인
docker-compose --version
```

Github-Actions 스크립트 파일 생성

Github repository - Actions - Java with Gradle 선택



이후 `gradle.yml` 이라는 파일을 생성하게 됨

Spring 배포를 위해 작성한 코드는 아래와 같음

```yaml
name: Java CI with Gradle

on:
  push:
    branches: [ "main" , "dev" ]
  pull_request:
    branches: [ "main" , "dev" ]

permissions:
  contents: read

jobs:
  build:

    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v3
    - name: Set up JDK 17
      uses: actions/setup-java@v3
      with:
        java-version: '17'
        distribution: 'temurin'


    - name: make application-prod.yml
```

```
    run: |
      cd ./src/main/resources
      touch ./application-prod.yml
      echo "${{ secrets.APPLICATION_PROD }}" > ./application-prod.yml


  - name: Grant execute permission for gradlew
    run: chmod +x gradlew

  - name: Build with Gradle
    run: ./gradlew build -x test

  - name: Docker build
    run: |
      docker login -u ${{ secrets.DOCKER_USERNAME }} -p ${{ secrets.DOCKER_PASSWORD }}
      docker build -t app .
      docker tag app ${{ secrets.DOCKER_USERNAME }}/sejongmate:latest
      docker push ${{ secrets.DOCKER_USERNAME }}/sejongmate:latest

  - name: Deploy
    uses: appleboy/ssh-action@master
    with:
      host: ${{ secrets.HOST }} # EC2 인스턴스 퍼블릭 DNS
      username: ec2-user
      key: ${{ secrets.PRIVATE_KEY }} # pem 키
      # 도커 작업
      script: |
        docker pull ${{ secrets.DOCKER_USERNAME }}/sejongmate:latest
        docker stop $(docker ps -a -q)
        docker run -d --log-driver=syslog -p 8080:8080 ${{ secrets.DOCKER_USERNAME }}/sejongmate:latest
        docker rm $(docker ps --filter 'status=exited' -a -q)
        docker image prune -a -f
```

위 코드를 순서대로 설명해보면, 아래와 같음

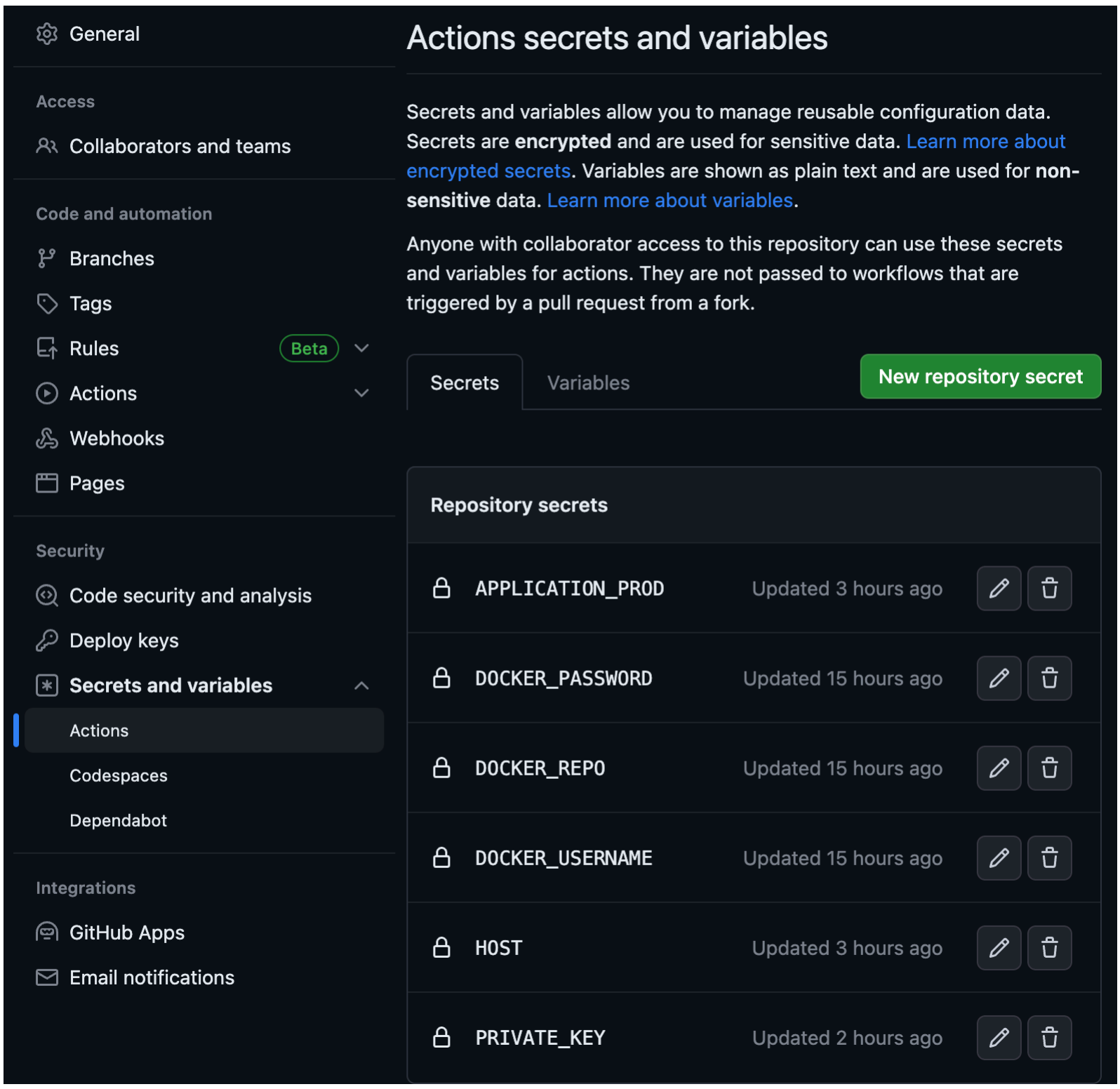
1. 배포를 위한 application-prod.yml 파일 생성하기 : 개발에서 사용되는 rds 주소 및 비밀번호를 노출할 수 없기 때문에 해당 파일은 .gitignore 해두고 배포 시 secret 키 이용해서 생성함
2. jar 파일 빌드 : 빌드 전 권한 설정해주기
3. docker build : docker hub 로그인 -> build -> push
4. 배포 : docker hub에서 image pull -> 이전에 올라와 있던 것 stop -> docker run -> 사용 중이 아닌 이미지 삭제

Dockerfile 코드

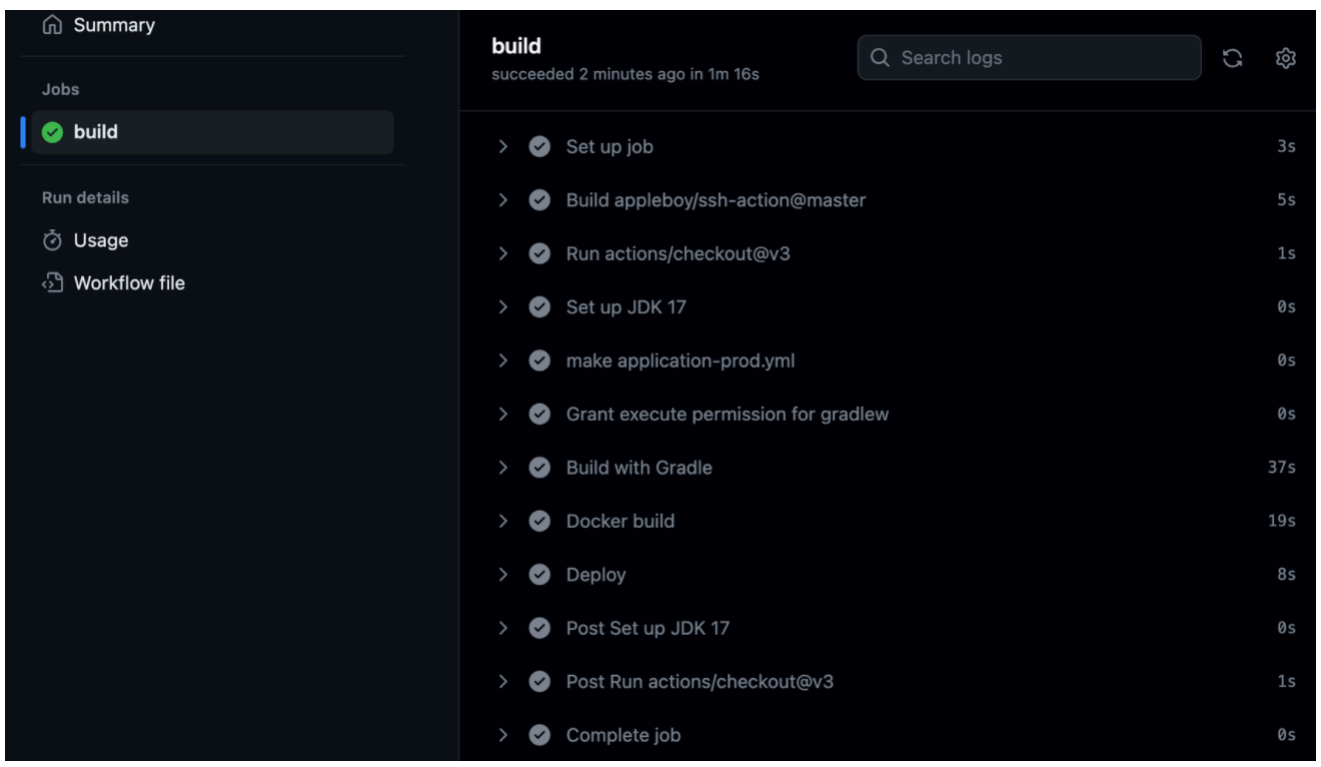
```
FROM openjdk:17-alpine

ARG JAR_FILE=/build/libs/sejongmate-0.0.1-SNAPSHOT.jar

COPY ${JAR_FILE} /sejongmate.jar

ENTRYPOINT ["java","-jar","-Dspring.profiles.active=prod", "/sejongmate.jar"]
```

Github Action 비밀키 생성법

Github Repository > Settings > Secrets and variables > Actions > New repository secret 버튼을 통해 생성 가능



📌 배포 완료 확인



🟩🟦 Trouble Shooting

build 시 발생한 문제

- 스프링 부트 gradle 플러그인 2.5 버전부터 gradle 빌드 시 JAR 파일이 2개 생성된다.
  - 프로젝트 이름-버전 - .jar
  - 프로젝트 이름-버전 - plain.jar

- build.gradle 에 아래 코드 삽입

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

- 명확히 하기 위해 Dockerfile에서 build 할 jar 파일 이름으로 지정해줌