

Docker + Github Action을 통한 CICD 자동배포 환경 구축

AWS Cloud Club

DDWU ACC Crew

신이현

Docker + Github Action을 통한 자동 배포 환경 구축

Git



<https://github.com/ddwu-aws-cloud-club/cicd-practice.git>

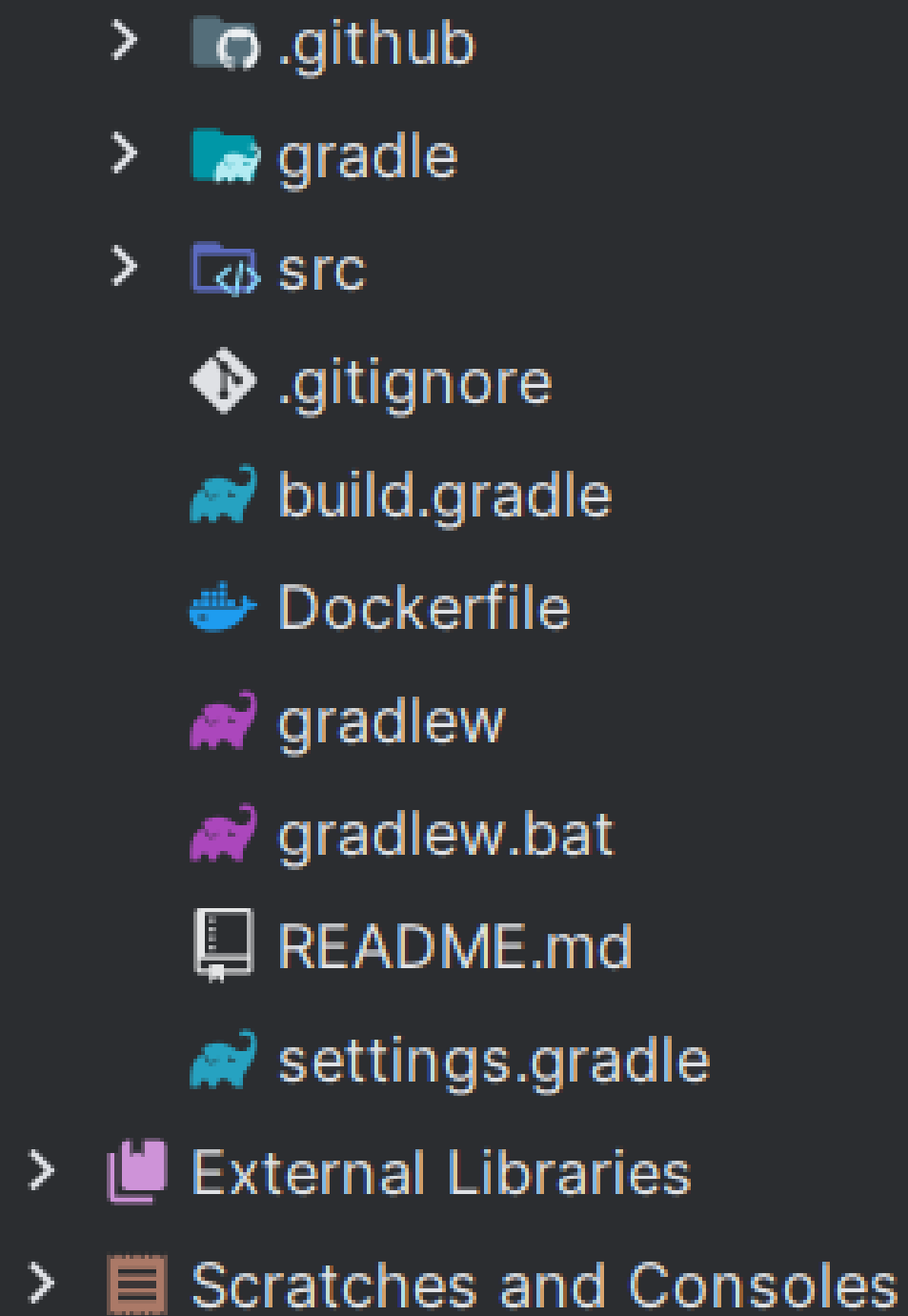
해당 레포지토리 fork 받아서 실습 진행해주세요
Docker hub 아이디와 public repository를 미리 생성해주세요

Docker + Github Action을 통한 자동 배포 환경 구축

Docker File 생성

Docker File을 해당 위치에 생성해주고 아래 코드를 작성해주세요

```
FROM openjdk:17-jdk
ARG JAR_FILE=./build/libs/*-SNAPSHOT.jar
COPY ${JAR_FILE} app.jar
ENTRYPOINT [ "java", "-jar", "/app.jar" ]
```



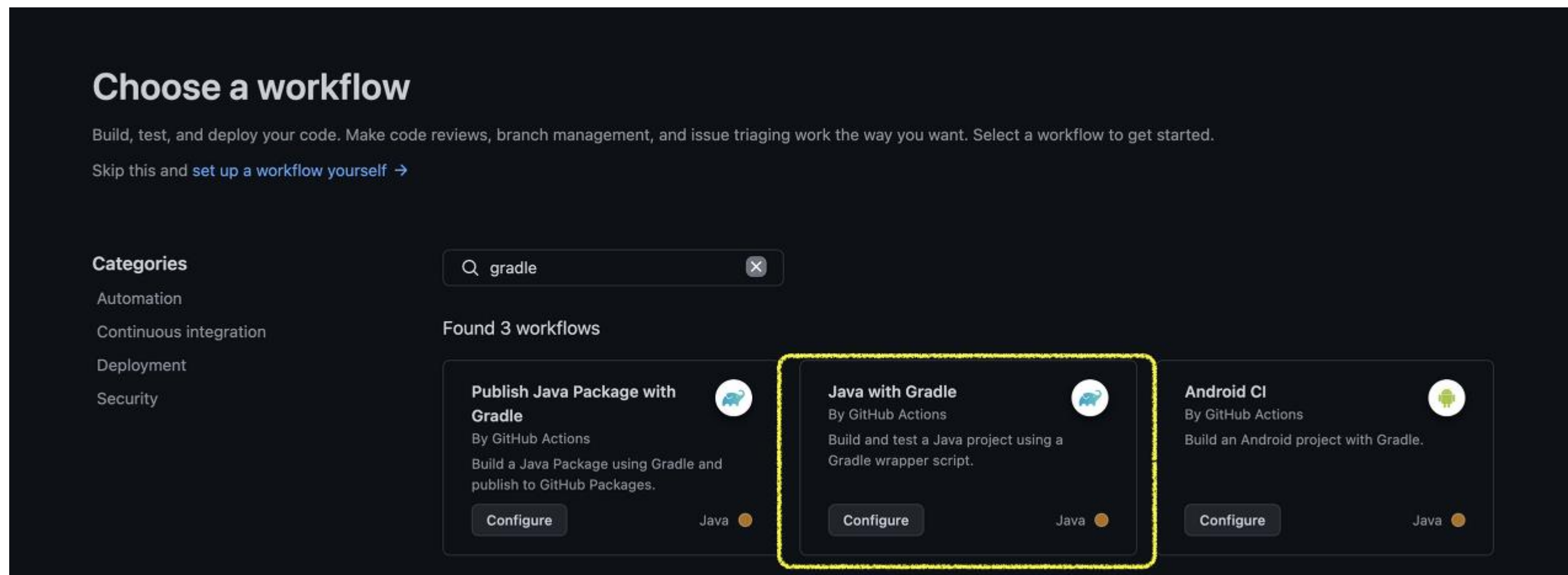
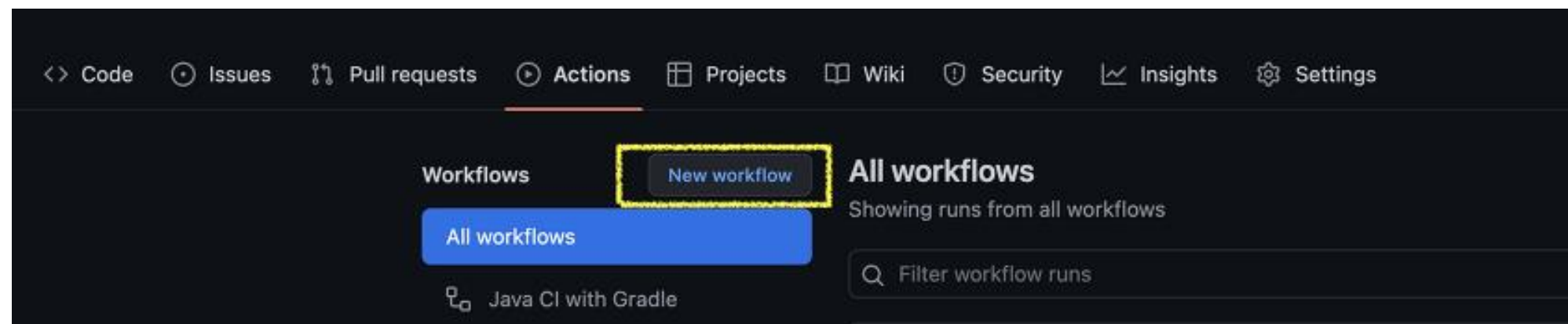
A screenshot of a file explorer interface with a dark background. It shows a project structure with the following items: a folder named '.github', a folder named 'gradle', a folder named 'src', a file named '.gitignore', a file named 'build.gradle', a file named 'Dockerfile', a file named 'gradlew', a file named 'gradlew.bat', a file named 'README.md', and a file named 'settings.gradle'. At the bottom, there are two expandable sections: 'External Libraries' and 'Scratches and Consoles'.

- > .github
- > gradle
- > src
- .gitignore
- build.gradle
- Dockerfile
- gradlew
- gradlew.bat
- README.md
- settings.gradle
- > External Libraries
- > Scratches and Consoles

Docker + Github Action을 통한 자동 배포 환경 구축

Github Action 작성

Github Action은 특정 레포지토리의 동작을 트래킹 해서 개발자가 작성한 workflow대로 행동하는 CI/CD 툴



Docker + Github Action을 통한 자동 배포 환경 구축

Github Action 작성

Github Action은 특정 레포지토리의 동작을 트래킹 해서 개발자가 작성한 workflow대로 행동하는 CI/CD 툴

The screenshot displays the GitHub Actions workflow editor interface. The main editor shows a workflow file named `gradle.yml` in the `main` branch. The workflow is configured to trigger on a push to the `main` branch and to build a Java project using Gradle. The workflow file content is as follows:

```
1 # This workflow uses actions that are not certified by GitHub.
2 # They are provided by a third-party and are governed by
3 # separate terms of service, privacy policy, and support
4 # documentation.
5 # This workflow will build a Java project with Gradle and cache/restore any dependencies to improve the workflow execution time
6 # For more information see: https://help.github.com/actions/language-and-framework-guides/building-and-testing-java-with-gradle
7
8 name: Java CI with Gradle
9
10 on:
11   push:
12     branches: [ "main" ]
13   pull_request:
14     branches: [ "main" ]
15
16 permissions:
17   contents: read
18
19 jobs:
20   build:
21
22     runs-on: ubuntu-latest
23
24     steps:
25     - uses: actions/checkout@v3
26     - name: Set up JDK 11
27       uses: actions/setup-java@v3
28       with:
29         java-version: '11'
30         distribution: 'temurin'
31     - name: Build with Gradle
32       uses: gradle/gradle-build-action@67421db6bd0bf253fb4bd25b31ebb98943c375e1
33       with:
34         arguments: build
35
```

The right sidebar shows the 'Marketplace' tab, which lists featured actions. The actions listed are:

- Upload a Build Artifact** (1.7k stars): Upload a build artifact that can be used by subsequent workflow steps.
- Close Stale Issues** (712 stars): Close issues and pull requests with no recent activity.
- Download a Build Artifact** (660 stars): Download a build artifact that was previously uploaded in the workflow by the upload-artifact action.
- Setup .NET Core SDK** (543 stars): Used to build and publish .NET source. Set up a specific version of the .NET and authentication to private NuGet repository.
- First interaction** (402 stars): Greet new contributors when they create their first issue or open their first pull request.

The interface also includes a 'Cancel changes' button and a 'Start commit' button in the top right corner.

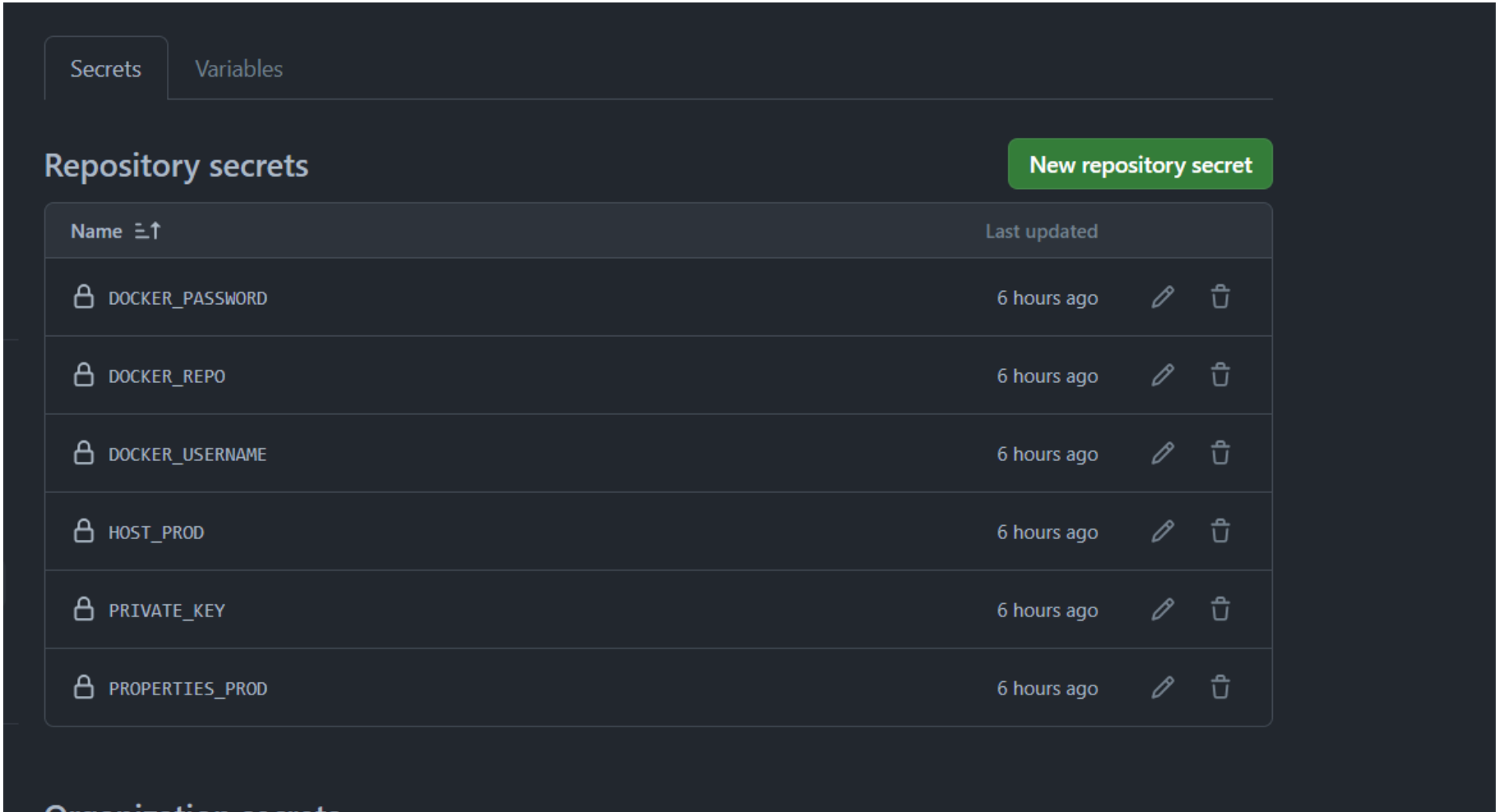
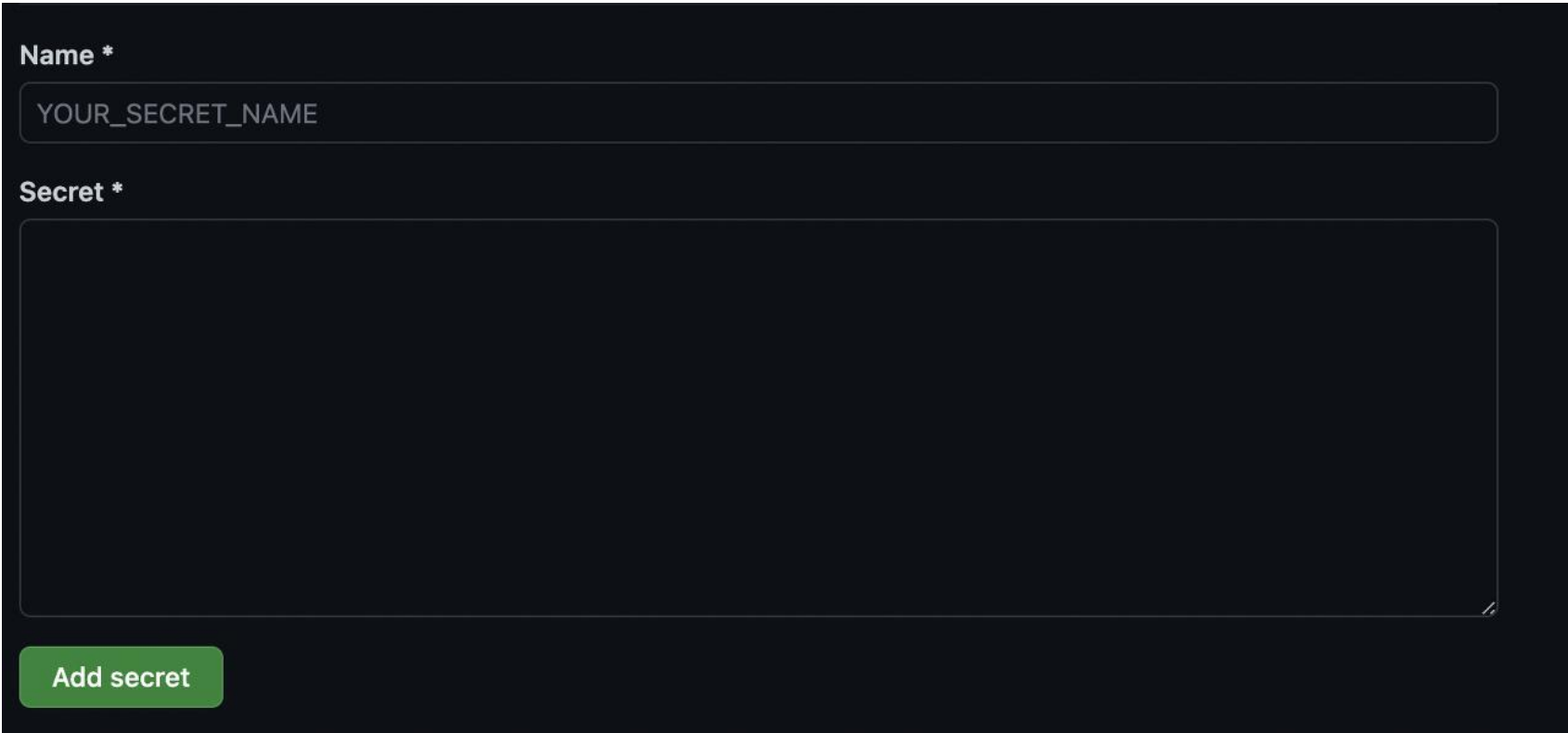
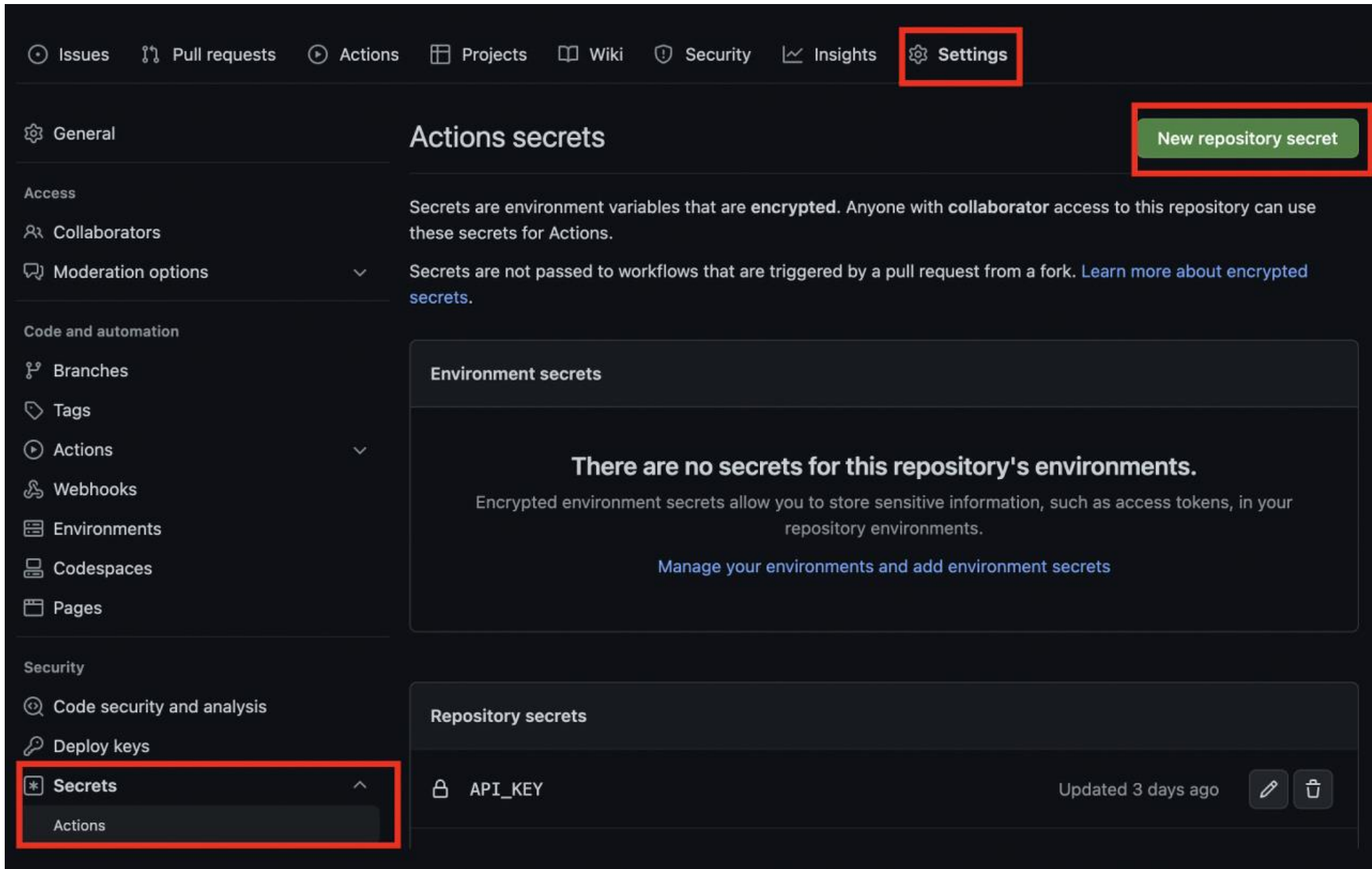
Docker + Github Action을 통한 자동 배포 환경 구축

Github Action 작성

<pre># github repository Actions 페이지에 나타낼 이름 name: CI/CD # event trigger on: push: branches: ["main"] pull_request: branches: ["main"] permissions: contents: read jobs: build: runs-on: ubuntu-22.04 steps: ## jdk setting - uses: actions/checkout@v3 - name: Set up JDK 17 uses: actions/setup-java@v3 with: java-version: '17' distribution: 'temurin' # https://github.com/actions/setup-java ## gradle caching - name: Gradle Caching uses: actions/cache@v3 with: path: ~/.gradle/caches ~/.gradle/wrapper key: \${{ runner.os }}-gradle-\${{ hashFiles('**/*.gradle*', '**/gradle-wrapper.properties') }} restore-keys: \${{ runner.os }}-gradle- - name: Grant execute permission for gradlew run: chmod +x gradlew - name: Build with Gradle run: ./gradlew build -x test shell: bash</pre>	<pre>## create application-prod.yml - name: create application-prod.yml if: contains(github.ref, 'main') run: cd ./src/main mkdir resources cd ./resources touch ./application.yml ls * echo "\${{ secrets.PROPERTIES_PROD }}" > ./application.yml shell: bash - name: Build With Gradle if: contains(github.ref, 'main') run: ./gradlew build -x test ## docker build & push to production - name: Docker build & push to prod if: contains(github.ref, 'main') run: echo "\${{ secrets.DOCKER_PASSWORD }}" docker login -u \${{ secrets.DOCKER_USERNAME }} --password-stdin docker build -f Dockerfile -t \${{ secrets.DOCKER_REPO }} . docker push \${{ secrets.DOCKER_REPO }} ## deploy to production - name: Deploy to prod uses: appleboy/ssh-action@v0.1.6 id: deploy-prod if: contains(github.ref, 'main') with: host: \${{ secrets.EC2_HOST_PROD }} username: \${{ secrets.EC2_USERNAME }} key: \${{ secrets.EC2_PRIVATE_KEY }} port: 22 envs: GITHUB_SHA script: echo test1234 > test.txt sudo docker rm -f \$(docker ps -qa) sudo docker pull \${{ secrets.DOCKER_REPO }} docker-compose up -d docker image prune -f</pre>
--	--

Docker + Github Action을 통한 자동 배포 환경 구축

Secrets 환경 변수 설정 DockerHub 가입하여 레포를 봤다는 전제 하에 진행된다.



- DOCKER_PASSWORD: 도커 계정 패스워드
 - DOCKER_REPO: 도커 레포지토리
 - DOCKER_USERNAME: 도커 ID
 - EC2_HOST_PROD : prod 환경의 EC2 인스턴스 ip
 - EC2_USERNAME : ubuntu
 - EC2_PRIVATE_KEY : EC2 PEM키를 복사한 내용
 - PROPERTIES_PROD : application.properties 파일 내용
- 위와 같은 정보들이 필요해서 설정을 해 주었는데, 본인의 상황에 따라 필요한 값들만 설정해주시면 된다.

Docker + Github Action을 통한 자동 배포 환경 구축

EC2 내부 Docker 설치

도커에 필요한 패키지 설치

```
$ sudo apt-get install apt-transport-https ca-certificates curl gnupg-agent software-properties-common
```

Docker의 공식 GPG키를 추가

Docker의 공식 apt 저장소를 추가

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -  
sudo add-apt-repository "deb [arch=amd64]  
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
```

시스템 패키지 업데이트

```
$ sudo apt-get update
```

Docker 설치

```
$ sudo apt-get install docker-ce
```

도커 실행상태 확인

```
$ sudo systemctl status docker
```

도커 권한 추가

```
$ sudo chmod 666 /var/run/docker.sock  
$ docker ps
```

도커 컴포즈 설치

```
$ sudo curl -L "https://github.com/docker/compose/releases/download/1.26.2/docker-  
compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

컴포즈 권한 추가

```
$ sudo chmod +x /usr/local/bin/docker-compose
```

버전 확인

```
$ docker-compose --version
```


Docker + Github Action을 통한 자동 배포 환경 구축

Docker-compose.yml 작성

Docker Compose란 복수의 컨테이너를 정의하고 실행하기 위한 도구

```
version: '3'
services:
  서비스 이름:
    image: 도커이미지(= 도커 계정명/레포지토리)
    container_name: 컨테이너 이름
    restart: always
    ports:
      - 8080:8080
```

restart: 컨테이너가 종료되거나 실패했을 때 동작

no, always, unless-stopped 등이 있다.

ports: 호스트와 컨테이너 간의 포트 매핑을 정의한다. 호스트의 8080 포트와 컨테이너의 8080 포트를 연결하고 있다.

이를 통해 호스트에서 8080포트로 들어오는 요청이 컨테이너의 8080포트로 전달된다.

Docker + Github Action을 통한 자동 배포 환경 구축

EC2 보안 규칙 설정

8080포트를 통해 접근할 것이니 인바운드 규칙을 편집해준다.

보안 그룹 규칙 ID

유형 정보

프로토콜 정보

포트 범위 정보

소스 정보

설명 - 선택 사항 정보

sgr-00c8979b1eed7e23e	SSH ▼	TCP	22	사용... ▼	<div>Q</div> <div>0.0.0.0/0 ✕</div>	<div></div> <div>삭제</div>
sgr-0802788edcb4b9716	HTTPS ▼	TCP	443	사용... ▼	<div>Q</div> <div>0.0.0.0/0 ✕</div>	<div></div> <div>삭제</div>
sgr-040bee6a96ce6f323	MYSQL/Aurora ▼	TCP	3306	사용... ▼	<div>Q</div> <div>0.0.0.0/0 ✕</div>	<div></div> <div>삭제</div>
sgr-024cf661500c4f0ad	사용자 지정 TCP ▼	TCP	8080	사용... ▼	<div>Q</div> <div>0.0.0.0/0 ✕</div>	<div></div> <div>삭제</div>
sgr-057dba9a4a91e670d	HTTP ▼	TCP	80	사용... ▼	<div>Q</div> <div>0.0.0.0/0 ✕</div>	<div></div> <div>삭제</div>

규칙 추가

Docker + Github Action을 통한 자동 배포 환경 구축

EC2 방화벽 설정

8080포트를 통해 접근할 것이니 인바운드 규칙을 편집해준다.

8080 포트 상태 확인하기

```
$ sudo ufw status
```

allow해주기

```
$ sudo ufw allow 8080
```

```
$ sudo ufw allow OpenSSH
```

```
$ sudo ufw enable
```

```
$ sudo ufw status
```

Docker + Github Action을 통한 자동 배포 환경 구축

EC2에 Nginx 설치

```
#
# Read up on ssl_ciphers to ensure a secure configuration.
# See: https://bugs.debian.org/765782
#
# Self signed certs generated by the ssl-cert package
# Don't use them in a production server!
#
# include snippets/snakeoil.conf;

root /var/www/html;

# Add index.php to the list if you are using PHP
index index.html index.htm index.nginx-debian.html;

server_name _;

include /etc/nginx/conf.d/service-url.inc;
location / {
    # First attempt to serve request as file, then
    # as directory, then fall back to displaying a 404.
    #try_files $uri $uri/ =404;

    proxy_pass $service_url;
}

# pass PHP scripts to FastCGI server
#
#location ~ \.php$ {
#    include snippets/fastcgi-php.conf;
#
```

nginx 설치

```
$ sudo apt install nginx
```

Nginx 실행 확인

```
$ sudo systemctl start nginx
```

```
$ sudo systemctl status nginx
```

nginx.conf 수정

conf 파일을 수정하기 위해 편집기를 연다.

```
$ sudo vi /etc/nginx/sites-enabled/default
```

```
include /etc/nginx/conf.d/service-url.inc;
proxy_pass $service_url;
```

이 둘을 해당 위치에 추가한다

Docker + Github Action을 통한 자동 배포 환경 구축

EC2에 Nginx 설치

ervice-url.inc 파일을 추가한다.

이 파일이 nginx가 자동으로 8080 포트로 포워딩할 수 있도록 해준다고 보면 된다고한다.

파일 열기

```
$ sudo vi /etc/nginx/conf.d/service-url.inc
```

파일 내용 추가

```
$ set $service_url http://127.0.0.1:8080;
```

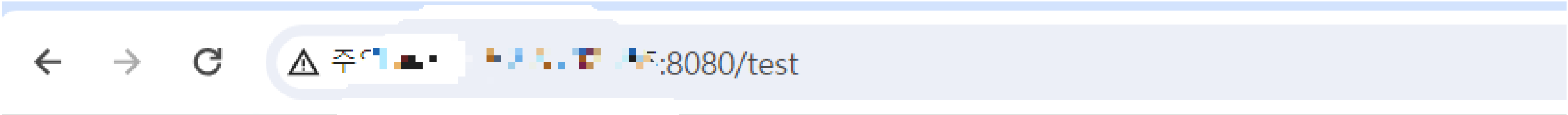
아래의 파일 내용 추가 후 재시작

```
$ sudo service nginx restart
```


Docker + Github Action을 통한 자동 배포 환경 구축

실행결과

Public ip: 8080/test를 url에 입력해준다.



Hello, World!