**포팅 메뉴얼**

**SSAFY 서울캠퍼스 7기 자율프로젝트** (22.10.11~22.11.21)

**A101 정예원 김미애 김용환 김지훈 박종선 이창현**

<< 목차 >>

1. 기술 스택
2. 빌드 상세 내용
3. 배포 특이 사항
4. 외부 서비스

**1. 프로젝트 기술 스택**

가. 이슈관리 : Jira

나. 형상관리 : Gitlab

다. 커뮤니케이션 : Mattermost, Notion

라. 개발 환경

1) OS : Windows 10

2) IDE : IntelliJ, Visual Studio Code

3) Server : AWS EC2 Ubuntu(20.04 LTS), Amazon S3, Docker, Jenkins

마. 기술 스택

1) Client : React, Three.js, Unity, node.js

2) Backend : Java, Gradle, Springboot, JPA, Spring Data JPA, nginx

3) DB: MySQL(8.0.28)

**2. 빌드 상세 내용**

배포는 아래 링크를 참고하여 진행했다.

<https://github.com/hjs101/CICD_manual>

Jenkins docker 안에 docker 를 설치해서 gitlab develop에 머지될 때마다 자동으로 도커 파일을 읽어 빌드하여 배포하는 방식을 택했다. Docker in Docker 방식이라 설정 변경에 있어서 복잡하다.

AWS EC2 server에 docker-compose.yml를 작성한다.

|  |
| --- |
| version: '3'  services:  jenkins:  image: jenkins/jenkins:lts  container\_name: jenkins  volumes:  - /var/run/docker.sock:/var/run/docker.sock  - /jenkins:/var/jenkins\_home  ports:  - "9090:8080"  privileged: true  user: root  mysql:  container\_name: mysql  image: mysql:8.0.28  env\_file:  - "/home/ubuntu/.env"  ports:  - "3306:3306" |

docker-compose를 사용해서 MySQL 컨테이너를 올릴 때 필요한 .env 파일을 /home/ubuntu/ 에 작성한다.

MySQL에서 연결할 데이터베이스 이름은 folaroid, 접속할 root 비밀번호는 123456789 다.

|  |
| --- |
| MYSQL\_DATABASE=folaroid  MYSQL\_ROOT\_PASSWORD=123456789 |

sudo docker-compose up -d 명령어로 Jenkins, MySQL 컨테이너를 실행한다. 참고한 자료대로 진행하고 Build Steps의 Execute shell에 아래 내용을 입력한다.

Server는 그냥 빌드시에는 권한 오류 문제가 발생하므로 Gradle에 권한 부여를 한 후 빌드한다.

|  |
| --- |
| mkdir -p /var/jenkins\_home/images\_tar  docker image prune -a --force  cd /var/jenkins\_home/workspace/S07P31A103/backend/portfolio/  chmod 777 gradlew  ./gradlew clean build  docker build -t server .  docker save server > /var/jenkins\_home/images\_tar/server.tar  cd /var/jenkins\_home/workspace/S07P31A103/frontend/folaroid/  docker build -t react .  docker save react > /var/jenkins\_home/images\_tar/react.tar |

빌드 후 조치에서는

Send build artifacts over SSH를 통해 Exec command로 아래 내용을 입력한다.

React 이미지를 run 할 때는 볼륨으로 SSL 관련 폴더를 전달한다. 해당 설정은 외부 서비스 다의 SSL 관련 폴더를 NGINX에서 접근할 때 필요하다.

|  |
| --- |
| docker load < /jenkins/images\_tar/server.tar  docker load < /jenkins/images\_tar/react.tar  if (docker ps | grep "server"); then docker stop server; fi  if (docker ps | grep "react"); then docker stop react; fi  docker run -d --rm -p 8080:8080 --name server server  docker run -d --rm -p 80:80 -p 443:443 -v "/home/ubuntu/certbot/conf:/etc/letsencrypt" -v "/home/ubuntu/certbot/log:/var/log/letsencrypt" -v "/home/ubuntu/certbot/www:/var/www/certbot" --name react react |

Nginx.conf은 참고한 자료에서 제시한 대로 위치시키되 파일 내용은 아래와 같이 설정해준다.

|  |
| --- |
| upstream backend{  ip\_hash;  server 168.126.63.1:8080;  }  map $http\_upgrade $connection\_upgrade {  default upgrade;  '' close;  }  server {  listen 80;  listen [::]:80;  server\_name folaroid.com;  location / {  return 301 <https://$host$request_uri>; # 80번 포트로 오는 경우에는 https로 redirect 해준다.  }  }  server {  listen 443 ssl;  listen [::]:443 ssl ipv6only=on;  server\_name folaroid.com;  access\_log /var/log/nginx/access.log;  error\_log /var/log/nginx/error.log;  # SSL 인증, docker run에서 volume을 통해 전달해주었었다.  ssl\_certificate /etc/letsencrypt/live/folaroid.com/fullchain.pem;  ssl\_certificate\_key /etc/letsencrypt/live/folaroid.com/privkey.pem;  ssl\_protocols TLSv1 TLSv1.1 TLSv1.2 SSLv3;  ssl\_ciphers ALL;  location / {  root /usr/share/nginx/html;  index index.html index.htm;  proxy\_redirect off;  charset urf-8;  try\_files $uri $uri/ /index.html; # React Router에서 하위 페이지를 인식하지 못하는 오류가 뜨기에 추가하여 해결했다.  proxy\_http\_version 1.1;  proxy\_set\_header Upgrade $http\_upgrade;  proxy\_set\_header Connection "upgrade";  proxy\_set\_header Host $host;  proxy\_set\_header X-Real-IP $remote\_addr;  proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;  proxy\_set\_header X-Forwarded-Proto $scheme;  proxy\_set\_header X-Nginx-Proxy true;  }  location /api {  rewrite ^/api/(.\*|$) /$1 break;  proxy\_pass http://backend/;  proxy\_redirect off;  charset utf-8;  proxy\_http\_version 1.1;  proxy\_set\_header Upgrade $http\_upgrade;  proxy\_set\_header Connection $connection\_upgrade;  proxy\_set\_header Host $http\_host;  proxy\_set\_header X-Real-IP $remote\_addr;  proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;  proxy\_set\_header X-Forwarded-Proto $scheme;  proxy\_set\_header X-Nginx-Proxy true;  }  error\_page 500 502 503 504 /50x.html;  location = /50x.html {  root /usr/share/nginx/html;  }  } |

**Dockerfile 내용은 gitlab에 코드가 있으므로 생략하겠다. Frontend와 backend로 나누어 각각 빌드한다. Frontend의 dockerfile을 읽을 때 nginx도 설치되며 작성한 nginx.conf을 전달하는 방식이다.**

**Develop에 정상적으로 merge된 후 모든 자동 배포 작업이 끝나면 Ubuntu 서버에서 sudo docker ps 명령어를 통해 현재 동작중인 container들의 목록을 볼 수 있고 다음과 같다.**

|  |
| --- |
| **~$ sudo docker ps**  **CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES**  **b5c684a041ce react "/docker-entrypoint.…" 21 minutes ago Up 21 minutes 0.0.0.0:80->80/tcp, :::80->80/tcp, 0.0.0.0:443->443/tcp, :::443->443/tcp react**  **94f0d158c6ec server "java -jar /app.jar" 21 minutes ago Up 21 minutes 0.0.0.0:8080->8080/tcp, :::8080->8080/tcp server**  **e9fbb2d283ef mysql:8.0.28 "docker-entrypoint.s…" 47 hours ago Up 47 hours 0.0.0.0:3306->3306/tcp, :::3306->3306/tcp, 33060/tcp mysql**  **17c14a836acc jenkins/jenkins:lts "/usr/bin/tini -- /u…" 47 hours ago Up 47 hours 50000/tcp, 0.0.0.0:9090->8080/tcp, :::9090->8080/tcp jenkins** |

만약 정상적으로 jenkins를 통해 빌드가 안됐다면 로컬에서 직접 빌드를 해보면서 오류를 해결했고 빌드는 됐지만 빌드 후 조치 과정에서 정상적으로 컨테이너가 올라가지 않는다면 docker run –it 명령어를 활용하여 직접 컨테이너를 올렸을 때 발생하는 로그를 파악해서 문제를 해결하는 방식으로 진행했다.

**3. 배포 특이사항**

별도의 계정 관련 설정, 여러 아이디 비밀번호 관련하여 보안적으로 보호할 필요성이 있는 부분은 gitlab에 올리지 않고 별도로 로컬에서 파일로 만들어서 서버로 전송하여 필요한 곳에 배치시키는 방식으로 진행했다.

로컬의 C:\Users\SSAFY\Desktop\security\server 에 application-security.properties 를 작성했다.

|  |
| --- |
| spring.datasource.url=jdbc:mysql://<< 도메인 >>:<< mysql 포트, 일반적으로 3306 >> /folaroid?useSSL=false&&allowPublicKeyRetrieval=true&useUnicode=true&serverTimezone=Asia/Seoul  spring.datasource.username=<< MySQL 계정, 일반적으로 root >>  spring.datasource.password=<< MySQL 계정의 비밀번호>>  #Github  client-id=<< client-id >>  client-secret=<< client-secret >>  #S3 Bucket  # AWS Account Credentials  cloud.aws.credentials.accessKey=<< accessKey >>  cloud.aws.credentials.secretKey=<< secretKey >>  # AWS S3 bucket Info  cloud.aws.s3.bucket.url=https://s3.<< << region 이름 >> >>.amazonaws.com/<< bucket 이름 >>  cloud.aws.s3.bucket=<< bucket 이름 >>  cloud.aws.region.static=<< region 이름 >>  cloud.aws.region.auto=false  cloud.aws.stack.auto=false  # ssl  server.ssl.enable=true  server.ssl.key-store=classpath:keystore.p12  server.ssl.key-store-password = << sudo passwd ubuntu 비밀번호로 설정한 비밀번호 >>  server.ssl.keyStoreType = PKCS12 |

Github, Amazon S3 Bucket, SSL 관련 부분은 외부 서비스를 활용한 것이므로 뒤에서 설명하겠다.

Scp 명령어를 통해 로컬에서 서버의 /home/ubuntu/security 로 전달했다. 전달할 때 Ubuntu 서버의 IP 주소를 파악하기 위해 cmd 에서 nslookup << 도메인 명 >> 명령어를 사용한다.

|  |
| --- |
| scp -i << pem키 이름>>.pem -r C:\Users\SSAFY\Desktop\security\server\application-security.properties ubuntu@<< Ubuntu 서버의 IP 주소 >>:/home/ubuntu/security |

application-security.properties 파일을 sudo docker cp 명령어를 통해 Jenkins docker 안의 필요한 곳에 배치시켰다. 해당 파일의 경우에는 /var/jenkins\_home/workspace/S07P31A103/backend/portfolio/src/main/resources 였다.

|  |
| --- |
| sudo docker cp /home/ubuntu/security/application-security.properties jenkins:/var/jenkins\_home/workspace/S07P31A103/backend/portfolio/src/main/resources |

같은 위치에 .env.local를 저장하고 같은 방법으로 scp을 통해 서버로 파일을 보낸다. React에서는 8080 포트로 요청을 보내며 client\_id와 client\_secret는 외부 서비스인 github REST API로부터 정보를 얻어올 때 사용하는 값이기에 해당 부분에서 설명하겠다.

|  |
| --- |
| REACT\_APP\_CLIENT\_ID=<< client-id >>  REACT\_APP\_CLIENT\_SECRET=<< client-secret >>  REACT\_APP\_BASE\_URL=https://<< 도메인 >>:8080 |

같은 방법으로 Jenkins docker 안의 필요한 곳에 배치시켰다. 해당 파일의 경우에는 /var/jenkins\_home/workspace/S07P31A103/frontend/folaroid 였다.

|  |
| --- |
| sudo docker cp /home/ubuntu/security/.env.local jenkins:/var/jenkins\_home/workspace/S07P31A103/frontend/folaroid |

**4. 외부 서비스**

가. Github login을 위해 Github REST API 로부터 client-id, client-secret를 받아야 한다.

Settings - Developer settings - OAuth Apps – New OAuth App 클릭

Application name에 folaroid 입력

Homepage URL에 <https://folaroid.com> 입력

Authorization callback URL에 <https://folaroid.com/callback> 입력

Client secrets를 발급받아 복사한다. (발급 후 사라지니 주의)

## Client ID 를 복사한다.

/home/ubuntu/security 의 application-security.properties 에서

client-id=<< client-id >>

client-secret=<< client-secret >>

뒤에 해당 값들을 채워 넣는다.

.env.local 에서

REACT\_APP\_CLIENT\_ID=<< client-id >>

REACT\_APP\_CLIENT\_SECRET=<< client-secret >>

## 뒤에 해당 값들을 채워 넣는다.

나. 이미지 파일을 별도로 저장하기 위해서 Amazon S3를 활용한다. Bucket을 생성하여 region 이름과 bucket 이름을 가져와

/home/ubuntu/security 에 application-security.properties에서

cloud.aws.s3.bucket.url=https://s3.<< << region 이름 >> >>.amazonaws.com/<< bucket 이름 >>

cloud.aws.s3.bucket=<< bucket 이름 >>

cloud.aws.region.static=<< region 이름 >>

뒤에 해당 값들을 채워 넣는다.

AWS Account Credentials 정보를 가져와

cloud.aws.credentials.accessKey=<< accessKey >>

cloud.aws.credentials.secretKey=<< secretKey >>

뒤에 해당 값들을 채워 넣는다.

다. SSL

Https를 위한 SSL 설정은 <https://galid1.tistory.com/612> 를 참고하여 진행했다.

<https://certbot.eff.org/> 에서 아래 설정으로 선택한 후



하단에 갱신된 절차대로 이행한다.

Install snapd 단계는 아래의 다음 링크를 참고하여 설치했다.

<https://www.cyberithub.com/5-best-steps-to-install-snap-command-on-ubuntu-20-04/>

7번의 sudo certbot certonly —standalone 를 실행하고 나서는 Y, N를 차례로 입력하고 SSL을 발급받고자 하는 도메인인 folaroid.com을 입력했다.

Keystore 를 로컬로 옮기는 과정은 scp를 활용했다.

Permission denied로 막히는 부분은 전부 chmod를 활용하여 권한 부여하였다.

Springboot 설정 관련 부분에서 server.port 부분은 제외하고 입력하였다. Server는 443이 아닌 8080으로 접속해야하기 때문이다. 배포 관련된 부분은 따라하지 않고 기존에 진행하던 부분으로 대체하였다.