포팅 메뉴얼

태그

목차

1. 개발 환경

- Server : Ubuntu 20.04.6 LTS
- JDK : OpenJDK17
- Node.js : 18.17.1
- Nginx : 1.18.0
- Mysql: 8.0.34
- Jenkins : 2.422
- Docker : 24.0.6

- 포트 번호

Port	Usage
22	SSH
80	HTTP
443	HTTPS
9090	Jenkins
3000	React
8080	SpringBoot
6379	Redis
8000	Django
27017	MongoDB

```
sudo ufw allow [port number] //포트 허용

sudo ufw status
```

2. Docker 및 젠킨스 설치, redis 설치

도커

```
sudo apt install apt-transport-https ca-certificates curl software-properties-common
```

apt-transport-https

: https를 통해 데이터나 패키지에 접근할 수 있다.

ca-certificates

: SSL기반 웹 애플리케이션이 SSL연결의 진위여부를 판별할 수 있게 해준다.

curl

: 링크로 데이터를 다운 받을 수 있게 해주는 도구

software-properties-common

: 우분투에서 PPA를 사용하기 위한 패키지

```
// Docker GPG key 추가
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

// apt repository에 Docker 다운로드 경로 추가
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic stable"

// 도커 다운 & 설치
sudo apt-cache policy docker-ce
sudo apt install docker-ce

sudo apt update //업데이트

sudo docker version //설치 잘 되어있나 확인
```

젠킨스 설치(도커 이미지 활용)

```
//젠킨스 이미지 pull
docker pull jenkins/jenkins:lts


//젠킨스 컨테이너 설치 및 실행
docker run -p 9090:8080 -p 50000:50000 -v /home/ubuntu/jenkinsData:/var/jenkins_home -v /var/run/docker.sock:/var/run/docker.sock --name jenkins -u root  jenkins/jenkins:lts
```

Redis 설치 (도커 이미지 활용)

```
docker pull redis //레디스 도커로 설치

docker run -p 6379:6379 --name=redis redis

docker exec -it redis /bin/sh //레디스 컨테이너 접속

redis-cli //레디스 명령어

//spring boot application.properties

spring.data.redis.host=localhost
spring.data.redis.port=6379
```

3. EC2에 mysql 설치

- mysql 설치 및 스프링부트에 연동

```
sudo apt-get update
apt-get install mysql-server
CREATE DATABASE db이름 default CHARACTER SET UTF8;
use Db이름;
//사용자 설정
CREATE USER 'root'@'%' identified by '비밀번호';
//사용자 권한 설정
GRANT ALL PRIVILEGES ON db이름.* TO 사용자@localhost IDENTIFIED BY '비번';


//spring boot application.properties
spring.datasource.url=jdbc:mysql://localhost:3306(포트번호)/Db이름?serverTimezone=UTC&useUnicode=true&characterEncoding=utf8
spring.datasource.username=사용자이름
spring.datasource.password=비밀번호
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
```

4. nginx 설정

```
server {

        listen 80;

        server_name k9d101.p.ssafy.io;
        server_tokens off;
```

```
        access_log /var/log/nginx/reverse-access.log;
        error_log /var/log/nginx/reverse-error.log;

        underscores_in_headers on;
        client_max_body_size 100M;

        return 301 https://$server_name$request_uri;

}

server {

        listen 443 ssl;
        server_name k9d101.p.ssafy.io;
        server_tokens off;

        ssl_certificate /etc/letsencrypt/live/k9d101.p.ssafy.io/fullchain.pem; # managed by Certbot
        ssl_certificate_key /etc/letsencrypt/live/k9d101.p.ssafy.io/privkey.pem; # managed by Certbot

        ssl_protocols TLSv1 TLSv1.1 TLSv1.2 SSLv3;
        ssl_ciphers ALL;

        ssl_prefer_server_ciphers off;

        underscores_in_headers on;
        client_max_body_size 100M;

       //스프링부트
        location /api/v1/ {
                proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
                proxy_set_header Host $http_host;
                proxy_set_header X-Real-IP $remote_addr;
                proxy_set_header X-Scheme $scheme;
                proxy_pass http://localhost:8080/api/v1/;

                proxy_set_header Authorization $http_authorization;

        }

//Django
        location /opencv/ {
                proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
                proxy_set_header Host $http_host;
                proxy_set_header X-Real-IP $remote_addr;
                proxy_set_header X-Scheme $scheme;
                proxy_pass http://0.0.0.0:8000/;
                proxy_set_header Authorization $http_authorization;

        }

//웹소켓
        location /ws/chat/ {
                proxy_pass http://localhost:8080/ws/chat/;
                proxy_http_version 1.1;
                proxy_set_header Upgrade $http_upgrade;
                proxy_set_header Connection "Upgrade";
                proxy_set_header Host $host;
        }

//리액트
        location / {

                proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
                proxy_set_header Host $http_host;
                proxy_set_header X-Real-IP $remote_addr;
                proxy_set_header X-Scheme $scheme;
                root /usr/share/nginx/html;
                index index.html index.html;
                proxy_redirect off;
                charset utf-8;
                try_files $uri $uri/ /index.html;
                #proxy_pass http://localhost:3000/;
        }


}
```

5. SpringBoot 배포

스프링부트 도커 파일 작성

```
FROM gradle:jdk17 as builder //gradle과 jdk17이 설치된 도커 이미지를 생성
WORKDIR /build //작업 디렉토리를 /build로 설정

# 그래들 파일이 변경되었을 때만 새롭게 의존패키지 다운로드 받게함.
COPY build.gradle settings.gradle /build/
RUN gradle build -x test --parallel --continue > /dev/null 2>&1 || true

# -x test : 텍스트 작업 건너 뜀
# --parallel : 가능하면 병렬처리를 하여 빌드 시간 축소

# 빌더 이미지에서 애플리케이션 빌드
COPY . /build  // 모든 소스코드가 /build에 복사
```

```
RUN gradle build -x test --parallel

FROM openjdk:17.0-slim
WORKDIR /app

# 빌더 이미지에서 jar 파일만 복사
COPY --from=builder /build/build/libs/프로젝트 이름-0.0.1-SNAPSHOT.jar .

EXPOSE 8080

# root 권한으로 실행
USER root
CMD [ \
    "java", \
    "-jar", \
    "-Djava.security.egd=file:/dev/./urandom", \
    "-Dsun.net.inetaddr.ttl=0", \
    "프로젝트이름-0.0.1-SNAPSHOT.jar"\
]
```

📌에러

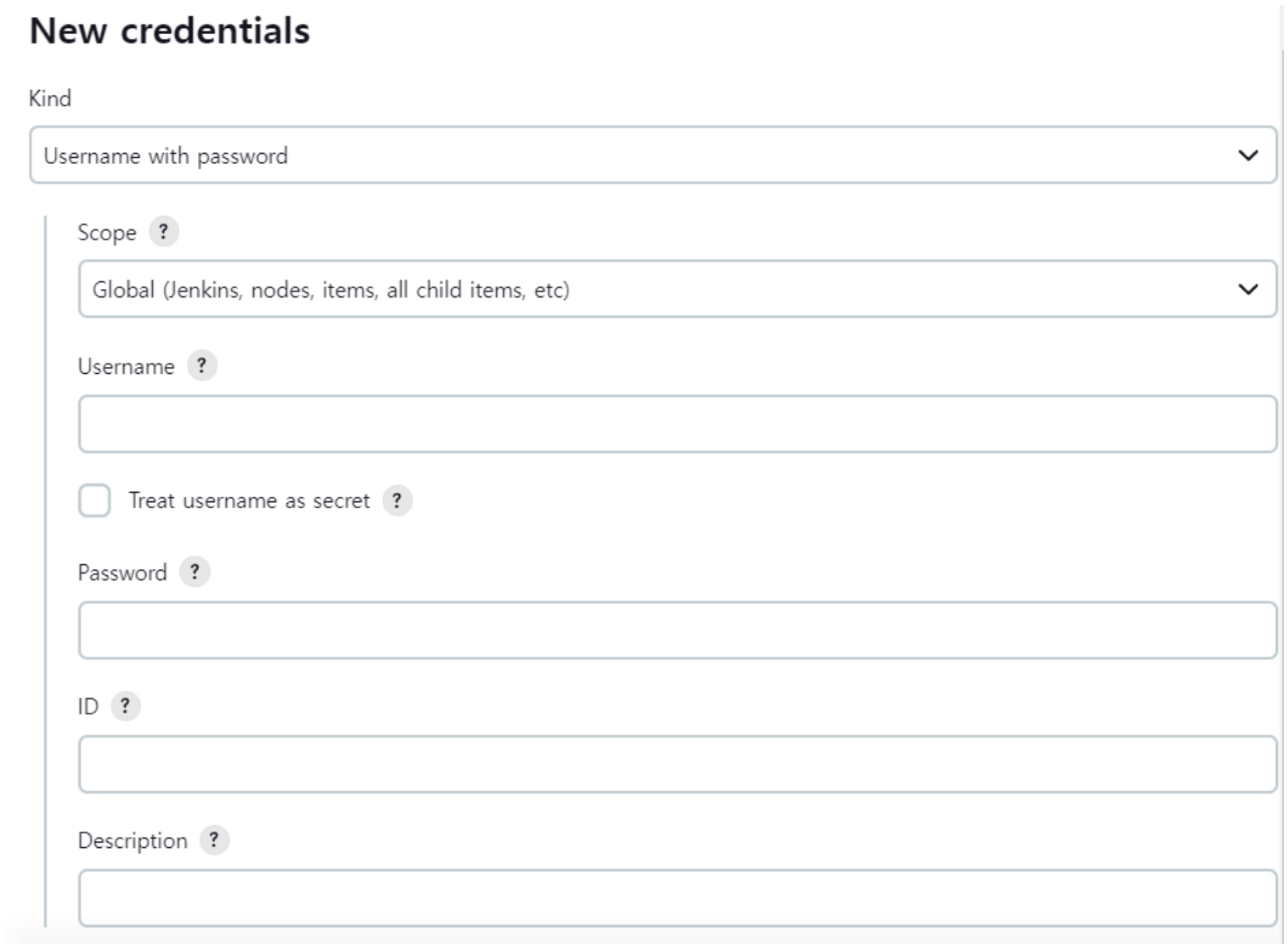
```
Step 9/12 : COPY --from=builder /build/build/libs/demo-0.0.1-SNAPSHOT.jar .
COPY failed: stat build/build/libs/demo-0.0.1-SNAPSHOT.jar: file does not exist
```

- 도커 파일 작성할 때 빌드로 생성되는 jar파일 이름을 확인하지 못해서 생긴 문제(기본 demo로 도커파일 작성)
- 스프링부트 프로젝트 이름으로 변경하여 해결

- 젠킨스 접속 후 젠킨스 플러그인 설치.

```
Docker

Docker Pipeline

Docker-build-step

GitLab Api Plugin

GitLab Authentication plugin

GitLab Plugin
```

- Jenkins 관리 화면에서 Credentials 생성



Username, Password는 깃랩 ID, 비밀번호.

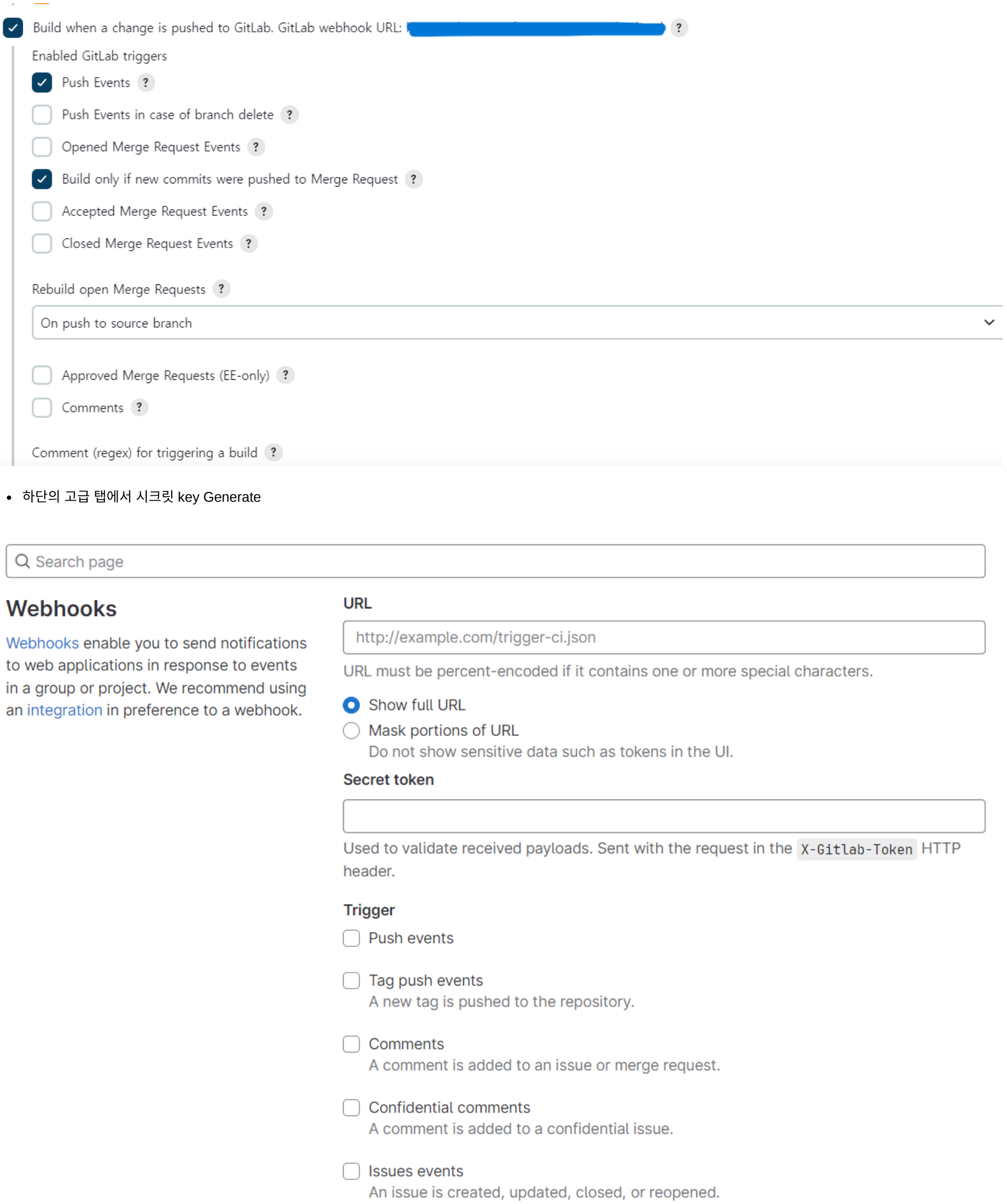
ID의 경우 구별값. 원하는 id부여하면 됨.(비울시 랜덤 생성)

→ 이걸로 깃에 연동함.

백엔드 파이프라인

```
pipeline {
    agent any

    environment {
        imagename = "backendServer"
        BUILD_DIR = "/var/laams/backend"
        DOCKER_IMAGE_NAME = "laams_backend"
        DOCKER_CONTAINER_NAME = "laams_backend_app"

    }

    stages {
        stage('Git Clone') {
          steps {
            git branch: 'be',
              credentialsId: '32298597-8536-4853-a089-bd3c8fd9d08a',
              url : 'https://lab.ssafy.com/s09-final/S09P31S107.git'
            }
            post {
             success {
               echo 'Successfully Cloned Repository'
             }
             failure {
               error 'This pipeline stops at cloning Git Repo'
             }
          }
        }

        stage('Build image') {
            steps {
                sh 'pwd'
                sh 'ls -al'
                sh 'chmod +x spring/laams/'
                sh "docker build -t $DOCKER_IMAGE_NAME spring/laams/"
            }
            post {
             success {
               echo 'Successfully Build Docker image'
             }
             failure {
               error 'This pipeline stops at building docker image'
             }
            }

        }
        stage('Deploy Container') {
            steps {
                sh "docker stop $DOCKER_CONTAINER_NAME || true"  // 기존 컨테이너가 없을 경우를 위해 무시
                sh "docker rm $DOCKER_CONTAINER_NAME || true"    // 기존 컨테이너가 없을 경우를 위해 무시

                sh "docker run -d --name $DOCKER_CONTAINER_NAME -p 8080:8080 -e JASYPT_ENCRYPTOR_PASSWORD=ssafy1234! -e TZ=Asia/Seoul --network=host $DOCKER_IMAGE_NAME"
            }
            post {
                success {
                    echo 'Successfully Deployed Docker Container'
                }
                failure {
                    error 'This pipeline stops at deploying docker container'
                }
            }
        }
    }
}
```

깃랩 웹훅 설정



- 하단의 고급 탭에서 시크릿 key Generate

- URL에 고급 탭의 파란색 블러처리한 부분 넣고, 시크릿 토큰에 Generate했던 토큰 값 넣어주기
- Trigger 설정(특정 브랜치 push만 받기 위해서 정규표현식 선택)

6. LAAMS 페이지(React) 배포

리액트 CI/CD

리액트 프로젝트 내에 도커 파일 생성

- 처음 시도한 세팅(이후 변경함)

```
# Builder stage
# FROM node:18 as builder

# WORKDIR /app

# COPY ./package*.json ./

# RUN npm install

# COPY . .

# RUN npm run build


# # Runner stage
# FROM node:18-alpine

# WORKDIR /app

# COPY --from=builder /app .

# EXPOSE 3000

# CMD [ "npm", "start"]
```

- npm start를 통해서 리액트 화면을 보여주는 것은 개발단계에서는 문제 없지만, 실제 배포환경에서는 nginx를 사용해서 정적인 페이지를 보여주고, 로드 밸런싱 및 캐싱의 기능을 활용하는 것이 일반적임.
- nginx를 활용하는 방식으로 변경한 도커파일

```
#  Dockerfile
FROM node:18 as builder //node.js 이미지 실행

WORKDIR /app // 파일 경로 설정

COPY ./package*.json ./ // 패키지 복사

RUN npm install

COPY . . // 전체 복사

RUN npm run build // 빌드

FROM nginx:latest  //nginx 사용

// 도커 컨테이너 내에 /etc/letsencrypt/live/k9d101.p.ssafy.io/ 안에 key들 복사
COPY ./fullchain.pem /etc/letsencrypt/live/k9d101.p.ssafy.io/fullchain.pem
COPY ./privkey.pem /etc/letsencrypt/live/k9d101.p.ssafy.io/privkey.pem

//프로젝트 내의 nginx.conf 파일을 컨테이너 내 react.conf에 복사
COPY ./nginx.conf /etc/nginx/conf.d/react.conf

// 빌드된 결과물을 아래 경로에 복사
COPY --from=builder /app/build /usr/share/nginx/html

EXPOSE 3000
// nginx 실행
CMD ["nginx", "-g", "daemon off;"]
```

- 아직 해결하지 못한 문제: 보안상 pem key들을 도커파일에 넣는 것은 좋지 않아서 외부에서 주입받는 방법이 안전한데, 시도해봤으나 제대로 되지 않아서 이렇게 진행.
- COPY pem구문 삭제 : 직접 젠킨스 컨테이너에서 workspace로 들어가서 파일 작성하는 방식으로 수정

젠킨스 파이프라인

```
pipeline {
    agent any

    environment {
        imagename = "frontend"
        BUILD_DIR = "/var/laams/frontend"
        DOCKER_IMAGE_NAME = "laams_frontend"
        DOCKER_CONTAINER_NAME = "laams_frontend_app"
    }

    stages {
        stage('Git Clone') {
          steps {
            git branch: 'FE',
              credentialsId: '32298597-8536-4853-a089-bd3c8fd9d08a',
              url : 'https://lab.ssafy.com/s09-final/S09P31S107.git'
            }
            post {
             success {
               echo 'Successfully Cloned Repository'
             }
             failure {
               error 'This pipeline stops at cloning Git Repo'
             }
          }
        }

        stage('Build image') {
```

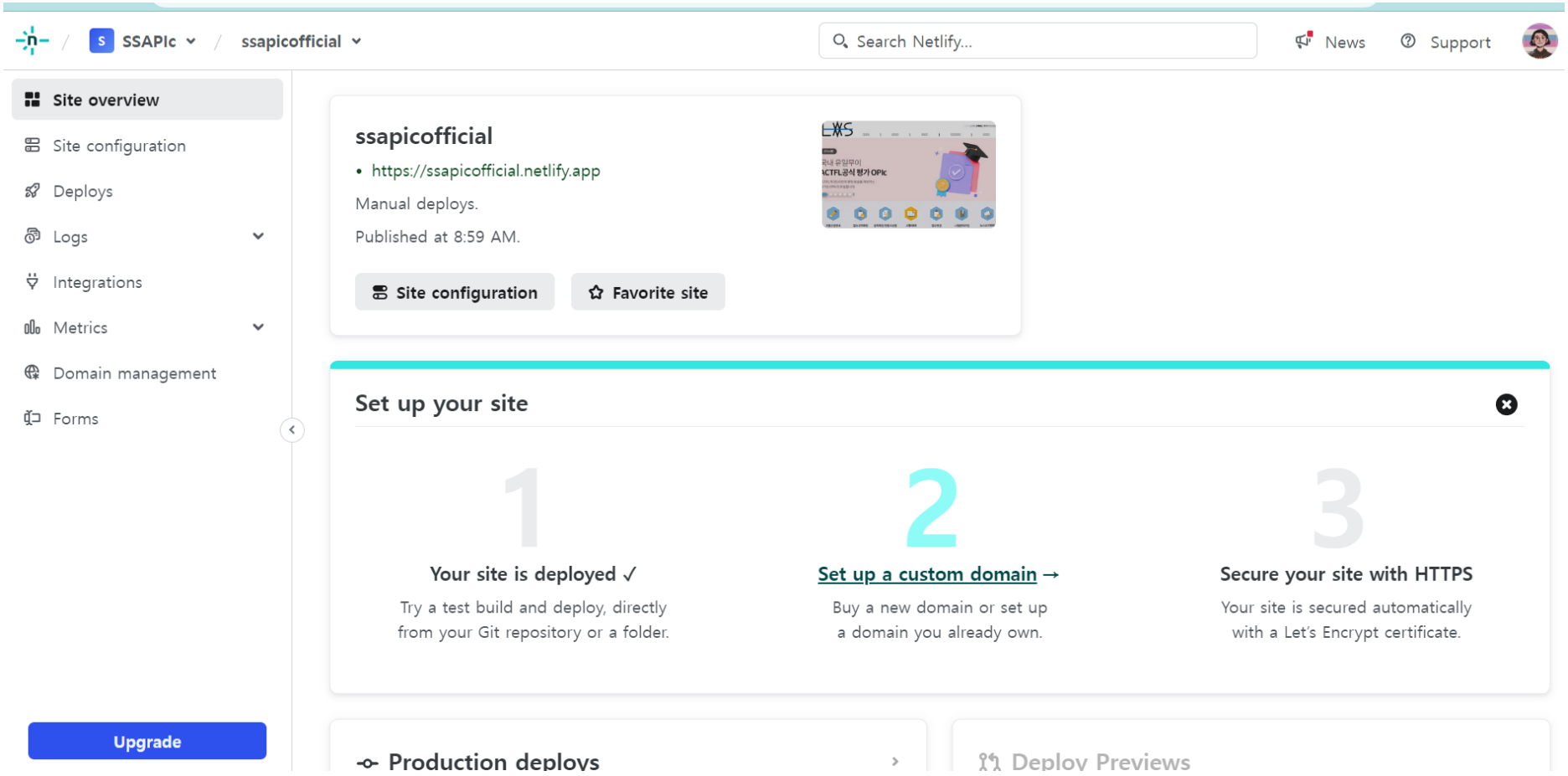
```
            steps {
                sh 'pwd'
                sh 'ls -al'
                sh 'chmod +x FE/laams/'
                sh "docker build -t $DOCKER_IMAGE_NAME FE/laams/"
            }
            post {
             success {
               echo 'Successfully Build Docker image'
             }
             failure {
               error 'This pipeline stops at building docker image'
             }
            }

        }
        stage('Deploy Container') {
            steps {
                sh "docker stop $DOCKER_CONTAINER_NAME || true"  // 기존 컨테이너가 없을 경우를 위해 무시
                sh "docker rm $DOCKER_CONTAINER_NAME || true"    // 기존 컨테이너가 없을 경우를 위해 무시

                sh "docker run -d --name $DOCKER_CONTAINER_NAME -p 8080:8080 --network=host $DOCKER_IMAGE_NAME"
            }
            post {
                success {
                    echo 'Successfully Deployed Docker Container'
                }
                failure {
                    error 'This pipeline stops at deploying docker container'
                }

            }
        }
    }
```

7. 장고 배포

```
# ./Dockerfile
FROM python:3
WORKDIR /usr/src/app

## Install packages
COPY requirements.txt ./
RUN pip install --upgrade pip && \
    pip install -r requirements.txt

## Copy all src files
COPY . .

## Run the application on the port 8080
EXPOSE 8000

# gunicorn 배포 명령어
# CMD ["gunicorn", "--bind", "허용하는 IP:열어줄 포트", "project.wsgi:application"]
CMD ["gunicorn", "--bind", "0.0.0.0:8000", "laams.wsgi:application"]
```

8. 응시자 페이지(리액트) 배포

```
// netlify를 통해 배포

npm run build를 통해서 생성된 build 폴더를 올려서 배포.
```


SSAPIc
ssapicofficial
Search Netlify...
News
Support
Site overview
Site configuration
Deploys
Logs
Integrations
Metrics
Domain management
Forms
Upgrade
ssapicofficial
https://ssapicofficial.netlify.app
Manual deploys.
Published at 8:59 AM.
Site configuration
Favorite site
Set up your site
1
Your site is deployed ✓
Try a test build and deploy, directly from your Git repository or a folder.
2
Set up a custom domain →
Buy a new domain or set up a domain you already own.
3
Secure your site with HTTPS
Your site is secured automatically with a Let's Encrypt certificate.
Production deploys
Deploy Previews