포팅 메뉴얼

1. 프로젝트 기술스택

개발환경

FrontEnd

- Node.js: `18.15.0`
- npm: `8.19.3`
- NextJs:

DevOps

- Docker: `23.0.1`
- Jenkins: `2.387.1`
- Nginx: `nginx/1.18.0`

Server

- AWS EC2: `ubuntu 20.04`
- AWS S3
- IntelliJ: `IDEA 2022.3.1`
- SpringBoot: `2.7.10`
- JDK: `OpenJDK 11.0.17`

Database

- MySQL: `8.0.32`
- MongoDB

관리

- GitLab
- Jira
- Notion
- Slack

2. 서버 세팅

▼ jenkins 서버에 구축 및 root 권한 부여 (Docker로 안 만듦)

1. Ubuntu 20.04 설치

```
sudo apt-get update -y
sudo apt-get upgrade -y
sudo apt install openjdk-11-jdk
wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo apt-key add -
sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'
sudo apt update
```

```
sudo apt install jenkins
#public key 오류가 나타남.
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys #위에서 알려준 public key
systemctl status jenkins #잘 동작하는 지 확인용
sudo ufw allow 8080
sudo ufw enable
```

2. root 권한 부여

```
vi /etc/sudoers
# root 계정 아래에 아래 내용을 추가
jenkins ALL= NOPASSWD: ALL
```

3. 빌드 상세내용

▼ 무중단 배포하기

1. 특화 때 CI/CD 구축 내용을 보고 Nginx 서버에 구축하기
2. Flow
   a. 두개의 docker-compose.yml 을 준비한다.
   b. 동작하고 있던 Port의 반대 Port를 빌드한다.
   c. 새로 올린 Port가 잘 동작하면, 이전 Port를 내린다.
   d. nginx의 포트를 이전 port에서 새로운 port로 변경하고 reload한다.
   e. 그러면 된다.
3. docker-compose
   a. docker-compose.first.yml

```
version: "3"

services:
  spring_first:
    container_name: spring_first
    build: ./backend/xyz
    ports:
      - "8081:8081"
    volumes:
      - /spring:/image
    restart: on-failure
  nestjs_first:
    container_name: nextJS_first
    build: ./frontend/xyz
    ports:
      - "3000:3000"
    volumes:
      - /nextJS:/image
    restart: on-failure
```

   b. docker-compose.second.yml

```
version: "3"

services:
  spring_second:
    container_name: spring_second
    build: ./backend/xyz
    ports:
      - "8082:8081"
    volumes:
      - /spring:/image
    restart: on-failure
  nestjs_second:
    container_name: nextJS_second
    build: ./frontend/xyz
```

```
    ports:
      - "3001:3000"
    volumes:
      - /nextJS:/image
    restart: on-failure
```

4. deploy.sh

```
#!/bin/bash
EXIST_FIRST=$(docker-compose -f docker-compose.first.yml ps | grep Up)

if [ -z "$EXIST_FIRST" ]; then # second -> first
    echo "first UP!!!"
    docker-compose -f docker-compose.first.yml up -d --build
    BEFORE_COMPOSE_NAME="second"
    AFTER_COMPOSE_NAME="first"
    BEFORE_SPRING_PORT_NUMBER=8082
    AFTER_SPRING_PORT_NUMBER=8081
    BEFORE_NEXTJS_PORT_NUMBER=3001
    AFTER_NEXTJS_PORT_NUMBER=3000
    #next js포트도 추가하자
else
    echo "second UP!!!"
    docker-compose -f docker-compose.second.yml up -d --build
    BEFORE_COMPOSE_NAME="first"
    AFTER_COMPOSE_NAME="second"
    BEFORE_SPRING_PORT_NUMBER=8081
    AFTER_SPRING_PORT_NUMBER=8082
    BEFORE_NEXTJS_PORT_NUMBER=3000
    AFTER_NEXTJS_PORT_NUMBER=3001
    #next js포트도 추가하자
fi

echo "${AFTER_COMPOSE_NAME} server up(port:${AFTER_SPRING_PORT_NUMBER} and ${AFTER_NEXTJS_PORT_NUMBER})"

# 2
for cnt in {1..10}
do
    echo "서버 응답 확인중..(${cnt}/10)";
    UP=$(curl -s http://localhost:${AFTER_NEXTJS_PORT_NUMBER}/)
    if [ -z "${UP}" ]
        then
      sleep 10
      continue
        else
            break
    fi

if [ $cnt -eq 10 ]
then
    echo "서버가 정상적으로 구동되지 않았습니다."
    exit 1
fi
done

# 3
sudo sed -i "s/${BEFORE_SPRING_PORT_NUMBER}/${AFTER_SPRING_PORT_NUMBER}/" /etc/nginx/nginx.conf
sudo sed -i "s/${BEFORE_NEXTJS_PORT_NUMBER}/${AFTER_NEXTJS_PORT_NUMBER}/" /etc/nginx/nginx.conf
sudo nginx -s reload
echo "Deploy Completed!!"

# 4
echo "$BEFORE_COMPOSE_NAME server down(port:${BEFORE_SPRING_PORT_NUMBER} and ${BEFORE_NEXTJS_PORT_NUMBER})"
if [ "${BEFORE_COMPOSE_NAME}" = "first" ]; then
docker stop spring_first nextJS_first
docker rm spring_first nextJS_first
else
docker stop spring_second nextJS_second
docker rm spring_second nextJS_second
fi

echo "success"
```

5. nginx

```
user www-data;
worker_processes auto;
pid /run/nginx.pid;

events {
  worker_connections 768;
```

```
  # multi_accept on;
}

http {

  ##
  # Basic Settings
  ##

  sendfile on;
  tcp_nopush on;
  tcp_nodelay on;
  keepalive_timeout 65;
  types_hash_max_size 2048;
  # server_tokens off;

  # server_names_hash_bucket_size 64;
  # server_name_in_redirect off;

  include /etc/nginx/mime.types;
  default_type application/octet-stream;

  ##
  # SSL Settings
  ##

  ssl_protocols TLSv1 TLSv1.1 TLSv1.2 TLSv1.3; # Dropping SSLv3, ref: POODLE
  ssl_prefer_server_ciphers on;

  ##
  # Logging Settings
  ##

  access_log /var/log/nginx/access.log;
  error_log /var/log/nginx/error.log;

  ##
  # Gzip Settings
  ##

  gzip on;

  # gzip_vary on;
  # gzip_proxied any;
  # gzip_comp_level 6;
  # gzip_buffers 16 8k;
  # gzip_http_version 1.1;
  # gzip_types text/plain text/css application/json application/javascript text/xml application/xml application/xml+rss text/j

  ##
  # Virtual Host Configs
  ##

        client_max_body_size 500M;

  server {
    server_name xyz-gen.com www.xyz-gen.com;

    location / {
      proxy_pass http://localhost:3001/;
    }

    location /frontend/ {
      proxy_pass http://localhost:3002/;
    }

    location /sonar {
      proxy_pass http://localhost:9000/sonar;
      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_connect_timeout   150;
      proxy_send_timeout      100;
      proxy_read_timeout      100;
      proxy_buffers           4 32k;
      client_max_body_size    8m;
      client_body_buffer_size 128k;
    }

    location /api/ {
      rewrite ^/(.*)$ /$1 break;
      proxy_pass http://localhost:8082/;
      proxy_http_version 1.1;
      proxy_set_header Upgrade $http_upgrade;
      proxy_set_header Connection "upgrade";
    }
```

```
    location /backend/ {
      rewrite ^/backend/(.*)$ /$1 break;
      proxy_pass http://localhost:8083/;
      proxy_http_version 1.1;
      proxy_set_header Upgrade $http_upgrade;
      proxy_set_header Connection "upgrade";
    }

    error_page 500 502 503 504 /50x.html;
    location = 50x.html {
      root /usr/share/nginx/html;
    }

    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/xyz-gen.com/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/xyz-gen.com/privkey.pem; # managed by Certbot
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot


}




  server {
      if ($host = www.xyz-gen.com) {
          return 301 https://$host$request_uri;
      } # managed by Certbot


      if ($host = xyz-gen.com) {
          return 301 https://$host$request_uri;
      } # managed by Certbot




    listen  80;
    server_name xyz-gen.com www.xyz-gen.com;
    return 404; # managed by Certbot



}}
```

6. pipeline

```
pipeline {
    agent any
    stages {
        stage('Init') {
            steps {
                // catchError {
                //     deleteDir()
                // }
                sh "ls"
            }
        }
        stage('GitHub Repository Clone') {
            steps {
                git branch: 'develop', credentialsId: 'b5f8c9c4-964d-45d9-a484-197e53a49f0f', url: 'https://lab.ssafy.com/s08-
                sh "cp -rpf /home/properties/application.yml /var/lib/jenkins/workspace/XYZ/backend/xyz/src/main/resources/"
                sh "cp -rpf /home/properties/env /var/lib/jenkins/workspace/XYZ/frontend/xyz/.env"
            }
        }
        stage('Spring docker') {
            steps {
                dir("./backend/xyz"){
                    echo "Spring"
                    sh "chmod +x gradlew"
                    sh "./gradlew clean build --exclude-task test"
                    sh "ls"
                }
            }
        }
        stage('React Docker') {
      steps {
        dir("./frontend/xyz"){
                    echo "React"
                    sh "npm i"
```

```
                    sh "npm run build"
                }
      }
    }
  //    stage('Docker compose down') {
  //             steps {
  //                 catchError {
  //                     echo 'docker compose down'
  //                     // sh 'sudo docker-compose -f docker-compose.yml down'
  //                 }
  //             }
  //    }
  //         stage('docker-compose') {
  //             steps {
  //                 dir("./"){
  //                     echo 'docker compose'
  //                     // sh "sudo docker-compose -f docker-compose.yml up -d --build"
  //                     sh "sudo docker-compose -f docker-compose.first.yml up -d --build"
  //                 }
  //             }
  //         }
        stage('server update') {
            steps {
                dir("./"){
                    echo 'docker compose'
                    // sh "sudo docker-compose -f docker-compose.yml up -d --build"
                    sh "cp -rpf /home/properties/deploy.sh ./"
                    sh "sudo sh ./deploy.sh"
                }
            }
        }
    }
}
```

▼ 채팅서버 구축하기(post api → kafka → sse방식)

1) nginx.conf (chat으로 시작함)

```
user www-data;
worker_processes auto;
pid /run/nginx.pid;

events {
  worker_connections 768;
  # multi_accept on;
}

http {

  ##
  # Basic Settings
  ##

  sendfile on;
  tcp_nopush on;
  tcp_nodelay on;
  keepalive_timeout 65;
  types_hash_max_size 2048;
  # server_tokens off;

  # server_names_hash_bucket_size 64;
  # server_name_in_redirect off;

  include /etc/nginx/mime.types;
  default_type application/octet-stream;

  ##
  # SSL Settings
  ##

  ssl_protocols TLSv1 TLSv1.1 TLSv1.2 TLSv1.3; # Dropping SSLv3, ref: POODLE
  ssl_prefer_server_ciphers on;

  ##
  # Logging Settings
  ##
```

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

##
# Gzip Settings
##

gzip on;

# gzip_vary on;
# gzip_proxied any;
# gzip_comp_level 6;
# gzip_buffers 16 8k;
# gzip_http_version 1.1;
# gzip_types text/plain text/css application/json application/javascript text/xml application/xml application/xml+rss text/javasc

##
# Virtual Host Configs
##

      client_max_body_size 500M;

upstream communication {
      server localhost:8089;
      server localhost:8088;
  }

server {
  server_name xyz-gen.com www.xyz-gen.com;

  location / {
    proxy_pass http://localhost:3001/;
  }

  location /frontend/ {
    rewrite ^/frontend/(.*)$ /$1 break;
    proxy_pass http://localhost:3002/;
    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;
  }

  location /chat/ {
    rewrite ^/(.*)$ /$1 break;
              proxy_pass http://communication/chat;
        proxy_http_version 1.1;
        proxy_buffering off;
        proxy_read_timeout 24h;
        proxy_set_header Connection '';
        }

  location /sonar {
    proxy_pass http://localhost:9000/sonar;
    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_connect_timeout   150;
    proxy_send_timeout      100;
    proxy_read_timeout      100;
    proxy_buffers           4 32k;
    client_max_body_size    8m;
    client_body_buffer_size 128k;
  }

  location /api/ {
    rewrite ^/(.*)$ /$1 break;
    proxy_pass http://localhost:8082/;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
        proxy_read_timeout 24h;
  }


  location /backend/ {
    rewrite ^/backend/(.*)$ /$1 break;
    proxy_pass http://localhost:8083/;
    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_read_timeout 24h;
    }

    error_page 500 502 503 504 /50x.html;
    location = 50x.html {
      root /usr/share/nginx/html;
    }

    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/xyz-gen.com/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/xyz-gen.com/privkey.pem; # managed by Certbot
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot


}

  server {
      if ($host = www.xyz-gen.com) {
          return 301 https://$host$request_uri;
      } # managed by Certbot


      if ($host = xyz-gen.com) {
          return 301 https://$host$request_uri;
      } # managed by Certbot


    listen  80;
    server_name xyz-gen.com www.xyz-gen.com;
    return 404; # managed by Certbot

}}
```

▼ 2) docker-compose.yml

```
version: "3"
services:
  zookeeper:
    image: wurstmeister/zookeeper
    container_name: zookeeper
    ports:
      - "2181:2181"
  kafka:
    image: wurstmeister/kafka
    container_name: kafka
    ports:
      - "9092:9092"
    environment:
      KAFKA_ADVERTISED_HOST_NAME: 3.38.168.160
      KAFKA_CREATE_TOPICS: "Topic:1:1"
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
    volumes:
      - /var/run/docker.sock
    depends_on:
      - zookeeper
  spring_test:
    container_name: spring_test
    build: ./xyz
    ports:
      - "8083:8081"
    volumes:
      - /spring_test:/image
    restart: on-failure
  spring_chat_first:
    container_name: spring_chat_first
    build: ./xyz-chat
    ports:
      - "8089:8080"
    volumes:
      - /spring_chat_first:/image
    restart: on-failure
  spring_chat_second:
    container_name: spring_chat_second
    build: ./xyz-chat
    ports:
      - "8088:8080"
    volumes:
      - /spring_chat_second:/image
    restart: on-failure
```

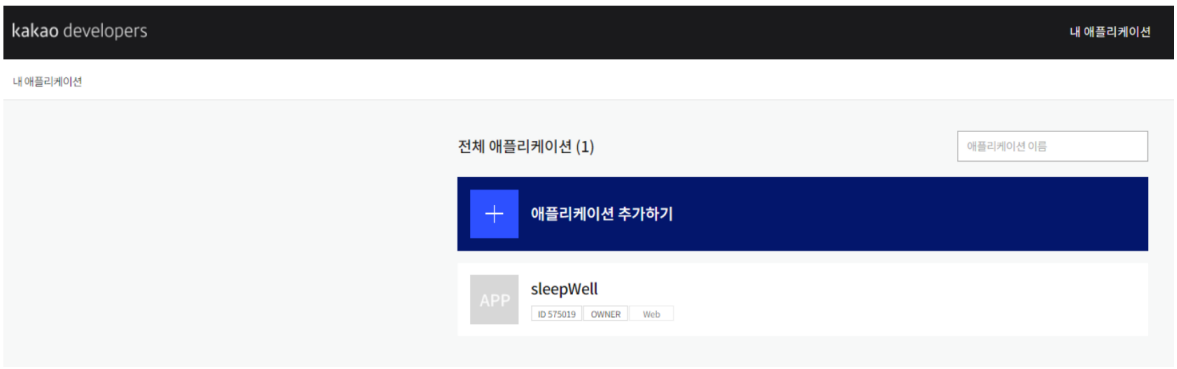
3) Dockfile

```
FROM azul/zulu-openjdk:11
WORKDIR /spring
COPY ./build/libs/WebSocketAndKafka-0.0.1-SNAPSHOT.jar server.jar
ENTRYPOINT ["java", "-jar", "server.jar"]
```

4. 외부 서비스

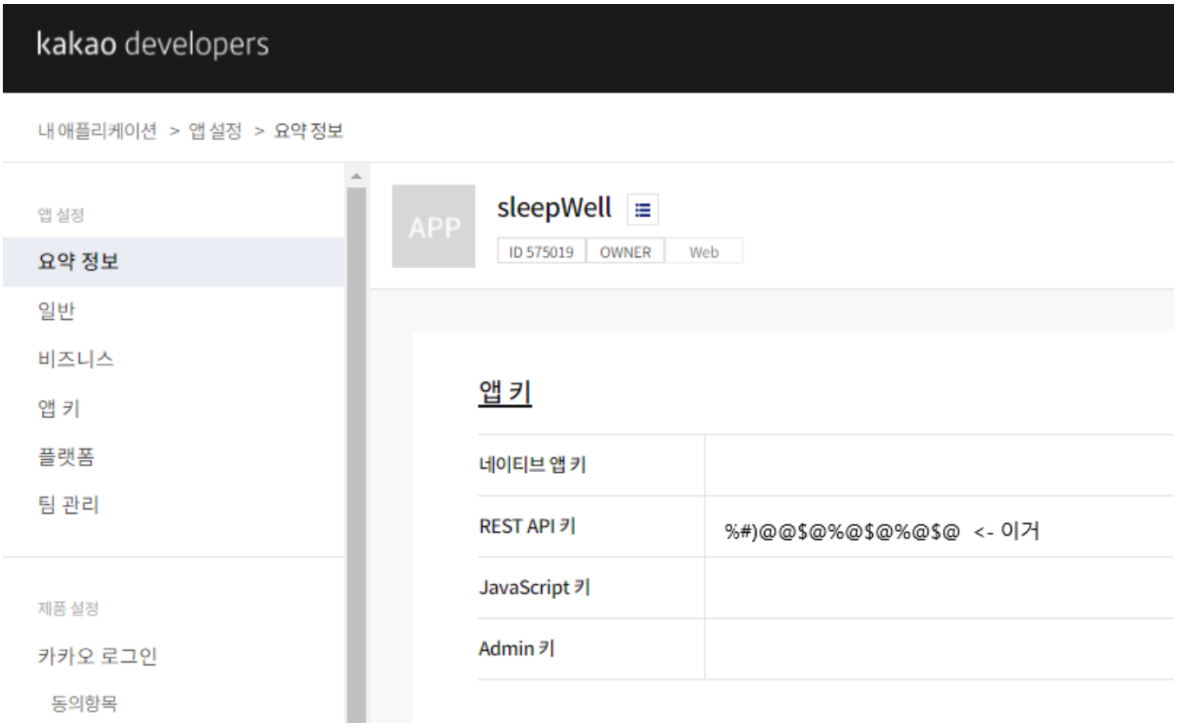
▼ 1) kakao API

애플리케이션

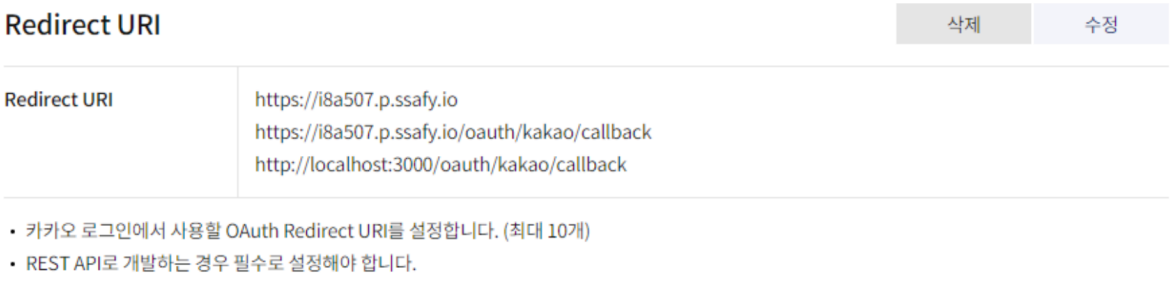


- cliend_id와 redirect_uri를 받아와서 {REST_API_KEY}와 {REDIRECT_URI}에 채워주어야 한다.
- 두 가지를 얻으려면 우선 애플리케이션을 생성한다.

Redirect Url 추가

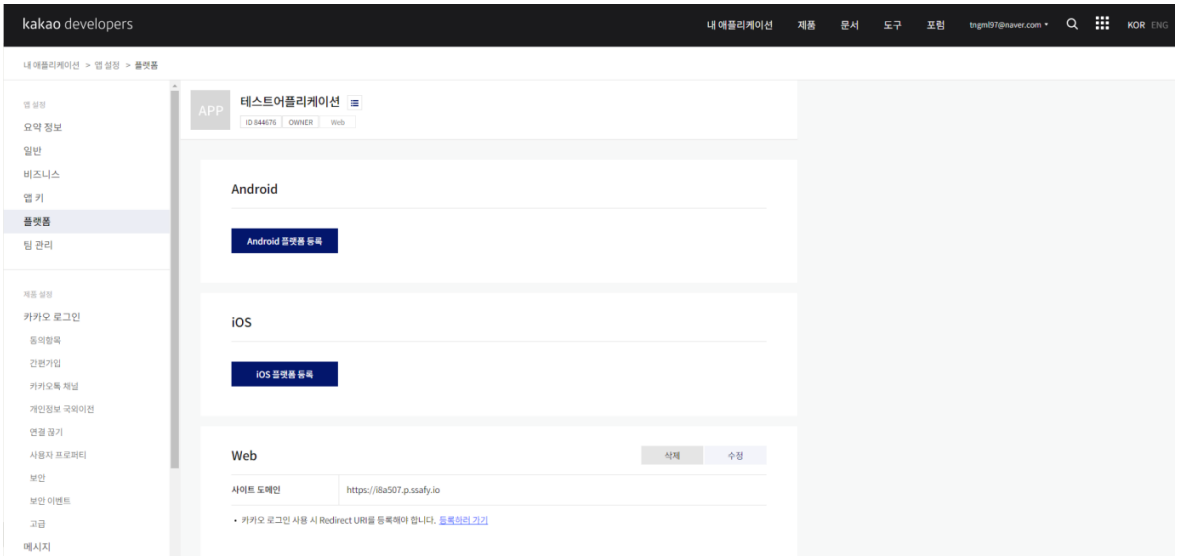


- cliend_id는 kakao developers에서 내 애플리케이션을 추가했을 때 생기는 REST_API 키를 넣어주면 되고, Redirect URI는 카카오 로그인 메뉴에 들어가서 추가를 해준다.



- Redirect URI는 반드시 프론트에서 접근할 수 있는 Host로 지정해주어야 한다.
- 왜냐하면 여기에서 인가 코드 받고 넘기고 등등 모든 작업이 이루어져야 하는데 프론트엔드가 접근할 수 없는 Host로 지정을 해버리면 말 그대로 접근을 못하니 아무것도 할 수 없다.
  (localhost:8080 등... 대신 이건 백엔드에서 자체 테스트할 때 사용할 수 있다)

플랫폼 추가



- Web에서 사용할 것이기 때문에 Web 플랫폼에 사이트 도메인을 추가한다.

▼ 2) SonarQube 설정하기

▼ EC2 설치하기

1. 기본 세팅 (가상 메모리 할당량 늘리기)

```
sysctl -w vm.max_map_count=262144
sysctl -w fs.file-max=65536
ulimit -n 65536
ulimit -u 4096
```

2. docker-compose.yml
   a. volume을 설정하면 plugin이 설치 되지 않는 오류가 발생하여 제거 하였습니다.

```
version: "3.1"
services:
  sonarqube:
          #image: sonarqube:7.9.1-community
    image: sonarqube:8.3-community
    container_name: sonar
```

```
    ports:
      - "9000:9000"
      - "9092:9092"
    networks:
      - sonarnet
    environment:
      - SONAR_HOME=/opt/sonarqube
      - SONAR_JDBC_USERNAME=sonar
      - SONAR_JDBC_PASSWORD=sonar
      - SONAR_JDBC_URL=jdbc:postgresql://db:5432/sonar
      - SONAR_WEB_CONTEXT=/sonar
        #volumes:
        #- /app/sonarqube/conf:/opt/sonarqube/conf
        #- /app/sonarqube/data:/opt/sonarqube/data
        #- /app/sonarqube/logs:/opt/sonarqube/logs
        #- /app/sonarqube/extensions:/opt/sonarqube/extensions

  db:
    image: postgres
    container_name: postgres
    networks:
      - sonarnet
    environment:
      - POSTGRES_USER=sonar
      - POSTGRES_PASSWORD=sonar
        #volumes:
        #- /app/sonarqube/postgres:/var/lib/postgresql/data

networks:
  sonarnet:
    driver: bridge
```

3. docker-compose up -d —build 명령어로 설치
4. 로그인 하고, spring과 nextJS 프로젝트를 만듭니다. 그 과정에서 설정값들은 기억해둡니다.
5. 기본 ID, PASSWORD는 admin입니다.

▼ Spring 설정

1. build.gradle

```
buildscript {
  dependencies {
    classpath "org.sonarsource.scanner.gradle:sonarqube-gradle-plugin:2.8"
  }
}

...원래 있던 것들 ....

apply plugin: "org.sonarqube"
sonarqube {
  properties {
    property "sonar.host.url", "https://xyz-gen.com/sonar"
    property "sonar.login", "admin"        // 로그인 id
    property "sonar.password", "admin"    // 로그인 비번
    property "sonar.projectKey", "spring"
    property "sonar.projectName", "spring"
    property "sonar.projectVersion", "1.0"
    property "sonar.sourceEncoding", "UTF-8"
    //property "sonar.sources", "src/"
    property "sonar.profile", "Sonar way"
    property "sonar.coverage.jacoco.xmlReportPaths", "build/reports/coverageReport/coverageReport.xml"    // Test Coverage
  }
}
```

2. pipeline

```
pipeline {
    agent any
    stages {
        stage('Init') {
            steps {
                // catchError {
                //     deleteDir()
```

```
                // }
                sh "ls"
            }
        }
        stage('GitHub Repository Clone') {
            steps {
                git branch: 'backend', credentialsId: 'b5f8c9c4-964d-45d9-a484-197e53a49f0f', url: 'https://lab.ssafy.com/s
                sh "cp -rpf /home/properties_backend/application.yml /var/lib/jenkins/workspace/XYZ_BACKEND/backend/xyz/src
                sh "cp -rpf /home/properties_backend/build.gradle /var/lib/jenkins/workspace/XYZ_BACKEND/backend/xyz/"
            }
        }
        stage('Spring docker') {
            steps {
                dir("./backend/xyz"){
                    echo "Spring"
                    sh "chmod +x gradlew"
                    sh "./gradlew clean build --exclude-task test"
                    sh "ls"
                }
            }
        }
        stage('Spring Analyze') {
            steps {
                dir("./backend/xyz"){
                    echo "Spring Analyze"
                    sh "chmod +x gradlew"
                    sh "./gradlew sonarqube "
                }
            }
        }
        stage('docker compose down') {
            steps {
                dir("./backend/"){
                    echo 'docker compose down'
                    sh "sudo docker-compose -f docker-compose.yml down"
                }
            }
        }
        stage('docker compose up') {
            steps {
                dir("./backend/"){
                    echo 'docker compose up'
                    sh "sudo docker-compose -f docker-compose.yml up -d --build"
                }
            }
        }
    }
}
```

3. 완성

▼ NextJS 설정(ETC)

1. Sonar-Scanner 설치
   a. 기본 세팅

```
apt-get update -y
apt-get upgrade -y
apt-get install unzip wget nodejs
```

   b. 설치 및 default 세팅

```
mkdir /downloads/sonarqube -p
cd /downloads/sonarqube
wget https://binaries.sonarsource.com/Distribution/sonar-scanner-cli/sonar-scanner-cli-4.2.0.1873-linux.zip
unzip sonar-scanner-cli-4.2.0.1873-linux.zip
mv sonar-scanner-4.2.0.1873-linux /opt/sonar-scanner
```

   c. 환경설정 추가

```
vi /opt/sonar-scanner/conf/sonar-scanner.properties
# 아래 내용 추가
sonar.host.url=http://localhost:9000
sonar.sourceEncoding=UTF-8
```

d. 부팅시 자동 설정을 위함(생략가능 : export 명령어를 직접 cmd에 치면 됨)

```
vi /etc/profile.d/sonar-scanner.sh
# 아래 내용 추가
#/bin/bash
export PATH="$PATH:/opt/sonar-scanner/bin"
```

e. 동작확인

```
sonar-scanner -v
```

```
INFO: Scanner configuration file: /opt/sonar-scanner/conf/sonar-scanner.properties
INFO: Project root configuration file: NONE
INFO: SonarQube Scanner 4.2.0.1873
INFO: Java 11.0.3 AdoptOpenJDK (64-bit)
INFO: Linux 5.3.0-18-generic amd64
```

f. 동작 완료

2. pipeline

```
pipeline {
    agent any
    stages {
        stage('Init') {
            steps {
                // catchError {
                //     deleteDir()
                // }
                sh "ls"
            }
        }
        stage('GitHub Repository Clone') {
            steps {
                git branch: 'frontend', credentialsId: 'b5f8c9c4-964d-45d9-a484-197e53a49f0f', url: 'https://lab.ssafy.com/
                sh "cp -rpf /home/properties/env /var/lib/jenkins/workspace/XYZ_FRONTEND/frontend/xyz/.env"
            }
        }
        stage('nestJS Docker') {
      steps {
        dir("./frontend/xyz"){
                    echo "nestJS"
                    sh "npm i"
                    sh "npm run build"
                }
      }
    }
        stage('nestJS Analyze') {
      steps {
        dir("./frontend/xyz"){
                    echo "nestJS Analyze"
                    sh "export PATH='$PATH:/opt/sonar-scanner/bin'"
                    sh "sudo /opt/sonar-scanner/bin/sonar-scanner -Dsonar.projectKey=nextJS -Dsonar.sources=. -Dsonar.host
                }
      }
    }
        stage('docker compose down') {
            steps {
                dir("./frontend/"){
                    echo 'docker compose down'
                    sh "sudo docker-compose -f docker-compose.yml down"
                }
            }
        }
        stage('docker compose up') {
            steps {
                dir("./frontend/"){
                    echo 'docker compose up'
                    sh "sudo docker-compose -f docker-compose.yml up -d --build"
                }
```

```
            }
        }
    }
}
```

3. 완성