포팅 메뉴얼

태그

▼ EC2 서버 환경 설정

▼ 1. 기본 환경 설정

```
# 패키지 업데이트
sudo apt update
sudo apt upgrade

# 우분투 서버 한국 표준시로 변경
sudo timedatectl set-timezone Asia/Seoul

# 확인 (KST)
date

# 카카오 미러 서버 변경
sudo vim /etc/apt/sources.list

# 문자 일괄 변경
:%s/ap-northeast-2.ec2.archive.ubuntu.com/mirror.kakao.
:wq
```

▼ 2. Swap 영역 할당

```
# swap 메모리 할당
# 메모리 용량 확인
free -h

# 메모리에 맞는 SWAP 파일 생성 (2GB)
sudo dd if=/dev/zero of=/swapfile bs=128M count=64
# sudo fallocate -l 8G /swapfile -> 이 명령어로도 사용 가능

# SWAP 파일 권한 설정(루트 사용자만)
sudo chmod 600 /swapfile
```

```
# SWAP 영역 설정
sudo mkswap /swapfile  # 활성화 준비

# SWAP 영역 활성화
sudo swapon /swapfile  # 활성화

# etc/fstab 파일을 편집하여 부팅 시 스왑 파일을 활성화한다.
sudo vi /etc/fstab
# 아래의 한 줄을 아랫칸에 추가한다.
/swapfile swap swap defaults 0 0
```

▼ 3. 방화벽 설정

```
# UFW(방화벽) 상태 확인
sudo ufw status

# UFW 활성화
sudo ufw enable

# 특정 포트 허용하기 (http : 80, https : 443, SSH : 22)
sudo ufw allow 80
sudo ufw allow 443
sudo ufw allow 22

# UFW 상태 확인
sudo ufw status

# 포트 삭제
sudo ufw status numbered
sudo ufw status $number
```

▼ 4. Nginx 설치

▼ Nginx 설치

```
# Nginx 설치
sudo apt-get -y install nginx

# Certbot 설치
```

```
sudo snap install --classic certbot

# Certbot으로 ssl 인증서 발급, 적용
sudo certbot --nginx -d ${domain}
```

▼ Nginx 설정

```
server {
    listen 80 default_server;
    listen [::]:80 default_server;

    root /var/www/html;

    index index.html index.htm index.nginx-debian.ht

    server_name _;

    location / {
        try_files $uri $uri/ =404;
    }

}


server {

    root /var/www/html;

    index index.html index.htm index.nginx-debian.ht
    server_name www.!{domain}.com;

    include /etc/nginx/conf.d/client-url.inc;
    include /etc/nginx/conf.d/service-url.inc;




    location / {
        proxy_pass $client_url;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header Host $http_host;
        proxy_set_header X-Forwarded-For $proxy_add_
    }

    location /api {
        proxy_pass $service_url;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header Host $http_host;
        proxy_set_header X-Forwarded-For $proxy_add_

        proxy_set_header Connection '';
        proxy_http_version 1.1;
        chunked_transfer_encoding off;
        proxy_buffering off;
        proxy_cache off;
    }

    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/www.!{doma
    ssl_certificate_key /etc/letsencrypt/live/www.!{
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; #

}

server {

    listen 443 ssl; # managed by Certbot
    listen [::]:443 ssl;
    ssl_certificate /etc/letsencrypt/live/jenkins.!{
    ssl_certificate_key /etc/letsencrypt/live/jenkin
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; #



    server_name jenkins.!{domain}.com;
    location / {
        proxy_pass http://localhost:9090;

    proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_
        proxy_set_header X-Forwarded-Proto $scheme;



    }

}

server {

    listen 443 ssl; # managed by Certbot
    listen [::]:443 ssl;
    ssl_certificate /etc/letsencrypt/live/sonarqube.
    ssl_certificate_key /etc/letsencrypt/live/sonarq
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; #


    server_name sonarqube.!{domain}.com;

    location / {
        proxy_pass http://localhost:53000;

        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_
        proxy_set_header X-Forwarded-Proto $scheme;

    }

}

server {
    listen 443 ssl; # managed by Certbot
    listen [::]:443 ssl;
    ssl_certificate /etc/letsencrypt/live/prometheus
    ssl_certificate_key /etc/letsencrypt/live/promet
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; #


    server_name prometheus.!{domain}.com;

    location / {
        proxy_pass http://localhost:59090;

        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_
        proxy_set_header X-Forwarded-Proto $scheme;

    }

}

server {

    listen 443 ssl; # managed by Certbot
    listen [::]:443 ssl;
    ssl_certificate /etc/letsencrypt/live/grafana.!{
    ssl_certificate_key /etc/letsencrypt/live/grafan
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; #

    server_name grafana.!{domain}.com;

    location / {
        proxy_pass http://localhost:53001;

        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_
        proxy_set_header X-Forwarded-Proto $scheme;

    }

}

server {

    listen 443 ssl; # managed by Certbot
    listen [::]:443 ssl;
    ssl_certificate /etc/letsencrypt/live/loki.!{dom
    ssl_certificate_key /etc/letsencrypt/live/loki.!
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; #

    server_name loki.!{domain}.com;

    location / {
        proxy_pass http://localhost:53100;

        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_
        proxy_set_header X-Forwarded-Proto $scheme;

    }

}

server {
    if ($host = !{domain}) {
        return 301 https://www.!{domain}.com;
    } # managed by Certbot

    server_name !{domain};
    listen 80;
    return 404; # managed by Certbot

}

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

    server_name www.!{domain}.com;
    listen 80;
    return 404; # managed by Certbot

}

server {
    if ($host = grafana.!{domain}.com) {
        return 301 https://$host$request_uri;
    } # managed by Certbot


    if ($host = loki.!{domain}.com) {
        return 301 https://$host$request_uri;
    } # managed by Certbot


    if ($host = prometheus.!{domain}.com) {
        return 301 https://$host$request_uri;
    } # managed by Certbot


    if ($host = sonarqube.!{domain}.com) {
        return 301 https://$host$request_uri;
    } # managed by Certbot


    if ($host = jenkins.!{domain}.com) {
        return 301 https://$host$request_uri;
    } # managed by Certbot


    listen 80;
    server_name .!{domain}.com;

    return 301 https://www.!{domain}.com$request_uri

}
```

▼ 5. docker 설치

```
# 도커 패키지 설치
sudo apt install apt-transport-https ca-certificates cu

# 도커 공식 GPG 키 추가
curl -fsSL https://download.docker.com/linux/ubuntu/gpg

# 도커 저장소 설정
echo "deb [arch=amd64 signed-by=/usr/share/keyrings/doc

# 업데이트
sudo apt update

# 도커 설치
sudo apt install docker-ce docker-ce-cli containerd.io

# ubuntu 유저에게 권한 부여 -> sudo 없이 docker 명령어 사용 가
sudo usermod -aG docker ubuntu

# Docker Socket 설정
sudo chmod 666 /var/run/docker.sock

# docker 서비스 재시작
sudo service docker restart
```

▼ 6. docker-compose 설치

```
# 도커 컴포즈 설치
sudo curl -L "https://github.com/docker/compose/release

# 실행 권한 부여
sudo chmod +x /usr/local/bin/docker-compose

# 설치 확인
docker-compose --version
```

▼ 7. jenkins 설치

  ▼ jenkins 설치

    ▼ run 방식

```
# jenkins 설치
docker run -d --name jenkins -p 8080:8080 -p 5000

# jenkins 도커에 접속
# docker exec -it jenkins bash       # 기본 사용자
docker exec -it -u root jenkins bash # 루트 사용자

# 업데이트
apt-get update

# 업그레이드
apt-get upgrade

# 필요한 패키지 설치
apt-get install -y apt-transport-https ca-certifi

# 도커 공식 GPG 키 추가
curl -fsSL https://download.docker.com/linux/debi

# 도커 저장소 설정
echo "deb [arch=amd64 signed-by=/usr/share/keyrin

# 업데이트
apt-get update
```

```
# 업그레이드
apt-get upgrade

# 도커 설치
apt-get install -y docker-ce

# 사용자 추가
usermod -aG docker jenkins
```

▼ docker-compose 방식

▼ docker-compose.yml

```
version: '3'
services:
  jenkins:
    image: jenkins/jenkins:lts
    container_name: jenkins
    ports:
      - "8080:8080"
      - "50000:50000"
    volumes:
      - /var/run/docker.sock:/var/run/docker.s
      - jenkins_home:/var/jenkins_home
    restart: always
    user: root
    command: |
      sh -c "
        apt-get update &&
        apt-get install -y apt-transport-https
        curl -fsSL https://download.docker.com
        echo 'deb [arch=amd64 signed-by=/usr/s
        apt-get update &&
        apt-get install -y docker-ce-cli &&
        jenkins.sh
      "
```

```
volumes:
  jenkins_home:
```

```
# docker-dompose.yml 작성 후 작성된 폴더로 이동
# docker-compose up로 컨테이너 생성
docker-compose up -d
```

▼ jenkins 설정

```
# jenkins 컨테이너 포트를 통해 접속
# 토큰을 입력하는 화면이 등장
# docker run 혹은 docker-compose.yml로 생성된 컨테이너 로
docker logs jenkins

# 출력된 토큰 복사 후 페이지에 붙여넣기
# admin 계정 등록 후 기본 패키지 설치
# 기본 패키지 설치 후 추가로 패키지 설치
pipeline
gitlab
mattermost notification
ssh agent
```

▼ 파이프라인 설정

```
// 다음 포트를 선택
def getNextVersion(currentVersion) {
    return currentVersion == '60000' ? '60001' : '60
}
// 다음 포트를 선택
def getActuatorPort(currentVersion) {
    return currentVersion == '60000' ? '58001' : '58
}

pipeline {
    agent any
    environment {
        imageName = "backend-test"
        containerName= "backend-test"
        branch="dev-be"
        gitUrl=credentials("gitUrl")

        releaseServerAccount = 'ubuntu'
        releaseServerUri = '!{ec2 adress}'
    }
    stages {
        stage('Cleanup'){
            steps{
                deleteDir()
            }
        }
        stage('clone'){
            steps{
                git branch:"$branch",
                changelog: false,
                credentialsId: 'git',
                poll: false,
                url: "$gitUrl"
            }
        }
        stage('Make Dir'){
            steps{
                dir ('backend') {
                    sh "mkdir -p ./gradle/wrapper"
                }
            }
        }
        stage('Add File'){
            steps{
                dir ('backend') {
                    withCredentials([file(credential
                        sh "cp ${application} ./src/
                    }
                    withCredentials([file(credential
                        sh "cp ${logback} ./src/main
                    }
                    withCredentials([file(credential
                        sh "cp ${wrapper} ./gradle/w
                    }
                    withCredentials([file(credential
                        sh "cp ${gradle} ./gradle/wr
                    }
                }
            }
        }
        stage('Jar Build') {
            steps {
                dir ('backend') {
                    sh 'chmod +x ./gradlew'
                    sh './gradlew clean bootJar -x t
                }
            }
        }
        stage('Set Port'){
            steps{
                script{
                    // 현재 도커에서 실행 중인 백엔드 컨테
                    def currentVersion = sh(script:
                    // 만약 컨테이너를 생성하지 않은 상태라
                    if (currentVersion.isEmpty()) {
                        env.CURRENT_VERSION = "60000
                    } else {
                        env.CURRENT_VERSION = curren
                    }
                    // 다음 버전의 번호 저장
                    env.NEXT_VERSION = getNextVersio
                    // actuator에 사용 할 포트 저장
                    env.ACTUATOR_PORT = getActuatorP
                }
            }
        }
        stage('Build') {
            steps {
                script {
                    echo "CURRENT_VERSION : ${env.CU
                    echo "NEXT_VERSION : ${env.NEXT_
                    echo "ACTUATOR_PORT : ${env.ACTU
                    // 이미지 이름과 다음 버전을 태그로 이
                    dir ('backend') {
                        dockerImage=docker.build("$i
                    }
                }
            }
        }
        stage('Run') {
            steps {
                script {
                    // 생성 한 이미지를 가지고 다음 버전의
                    sh "docker run --name $container
                }
            }
        }
        stage('SonarQube Analysis') {
            steps {
                dir ('backend') {
                    script {
                        withCredentials([
                            string(credentialsId: 's
                            string(credentialsId: 's
                            string(credentialsId: 's
                        ]) {
                            sh 'chmod +x ./gradlew'
                            sh "./gradlew sonarqube
                        }
                    }
                }
            }
        }
        stage('Health Check') {
            steps {
                script {
                    def healthCheckUrl = "!{EC2 addr
                    def healthCheckResult = sh(scrip

                    if (healthCheckResult.toString()
                        echo "${healthCheckResult.to
                        error "Health check failed.
                        sh "docker rm -f $containerN
                        sh "docker rmi $imageName:$N
                    }
                }
            }
        }
        stage('Switch to New Version') {
            steps {
                sshagent(credentials: ['EC2']) {
                    script {
                        // nginx의 service-url.inc 파
                        sh "ssh -o StrictHostKeyChec
                        // nginx를 재시작
                        sh "ssh -o StrictHostKeyChec
                    }
                }
            }
        }
        stage('Clean'){
            steps{
                script {
                    // Nginx의 포트를 변경하기 전 서버로
                    // 그래서 임의로 대기시간을 추가했음
                    sleep(time:5,unit:"SECONDS")
                    try {
                        sh "docker stop $containerNa
                        sh "docker rm $containerName
                        sh "docker rmi $imageName:$C
                    } catch (Exception e) {
                        echo "Failed to remove conta
                    }
                }
            }
        }
    }
    post {
        success {
            script {
                def Author_ID = sh(script: "git show
                def Author_Name = sh(script: "git sh
                mattermostSend (color: 'good',
                message: "back 빌드 성공: ${env.JOB_NA
                )
            }
        }
        failure {
            script {
                def Author_ID = sh(script: "git show
                def Author_Name = sh(script: "git sh
                mattermostSend (color: 'danger',
                message: "back 빌드 실패: ${env.JOB_NA
                )
            }
        }
    }
}
```

▼ 8. sonarqube 설치

```
# sonarqube 설치
docker run --name sonarqube -d -p 9000:9000

# 해당 포트로 접속 초기 로그인시 사용자명, 비밀번호 admin
# 로그인 후 비밀번호 재설정
# 프로젝트 생성에서 local project 선택
# 프로젝트 이름과 브랜치 이름 설정 후 생성
# 우상단의 계정 정보로 이동
# Security 선택 후 Generate Tokens에서 Type은 Project Anal
# Project는 생성한 프로젝트 선택
# Name 입력, Expires in 설정 후 생성
# 생성된 token, sonarqube url, project key를 jenkins cred
```

▼ 9. Grafana, Prometheus, Loki 설치

▼ 설정 파일들 작성

▼ docker-compose.yml 작성

```
version: '3'

services:
  prometheus:
    image: prom/prometheus:latest
    container_name: prometheus
    volumes:
      - ./prometheus.yml:/etc/prometheus/promethe
      - ./web.yml:/etc/prometheus/web.yml
      - prometheus_data:/prometheus
    command:
      - --config.file=/etc/prometheus/prometheus.
      - --storage.tsdb.path=/prometheus
      - --web.console.libraries=/usr/share/promet
      - --web.console.templates=/usr/share/promet
      - --web.config.file=/etc/prometheus/web.yml

    ports:
      - 127.0.0.1:!{prometheus port}:9090
    restart: always

  loki:
    image: grafana/loki:latest
    container_name: loki
    volumes:
      - ./loki-config.yml:/etc/loki/local-config.
      - ./data/loki/index:/loki/index
      - ./data/loki/chunks:/loki/chunks
      - ./data/loki/wal:/loki/wal
      - ./data/loki/rules:/loki/rules
      - ./data/loki/rules-temp:/loki/rules-temp
      - ./data/loki/compactor:/loki/compactor
    command: -config.file=/etc/loki/local-config.
    ports:
      - 127.0.0.1:!{loki port}:3100
    restart: always

  grafana:
    image: grafana/grafana:latest
    container_name: grafana
    volumes:
      - grafana_data:/var/lib/grafana
      - ./grafana/provisioning/:/etc/grafana/prov
    environment:
      - GF_SECURITY_ADMIN_USER=!{user}
      - GF_SECURITY_ADMIN_PASSWORD=!{password}
      - GF_SERVER_SERVE_FROM_SUB_PATH=true
    ports:
      - 127.0.0.1:!{grafana port}:3000
    restart: always

volumes:
  prometheus_data: {}
  grafana_data: {}
```

▼ loki-config.yml 작성

```
auth_enabled: false

server:
  http_listen_port: 3100

ingester:
  lifecycler:
    address: 127.0.0.1
    ring:
      kvstore:
        store: inmemory
      replication_factor: 1
    final_sleep: 0s
  chunk_idle_period: 5m
  chunk_retain_period: 30s
  max_transfer_retries: 0
  wal:
    dir: /loki/wal

schema_config:
  configs:
    - from: 2020-10-24
      store: boltdb
      object_store: filesystem
      schema: v11
      index:
        prefix: index_
        period: 24h

storage_config:
  boltdb:
    directory: /loki/index

  filesystem:
    directory: /loki/chunks

limits_config:
  enforce_metric_name: false
  reject_old_samples: true
  reject_old_samples_max_age: 168h

chunk_store_config:
  max_look_back_period: 0s

table_manager:
  retention_deletes_enabled: false
  retention_period: 0s

compactor:
  working_directory: /loki/compactor
  shared_store: filesystem

ruler:
  storage:
    type: local
    local:
      directory: /loki/rules
  rule_path: /loki/rules-temp
  alertmanager_url: http://localhost:9093
  ring:
    kvstore:
      store: inmemory
  enable_api: true
```

▼ prometheus.yml 작성

```
global:
  scrape_interval: 15s
  evaluation_interval: 15s

scrape_configs:
  - job_name: 'prometheus'
    static_configs:
      - targets: ['localhost:9090']

  - job_name: 'spring-boot-app'
    metrics_path: '/info/actuator/prometheus'
    scrape_interval: 5s
    static_configs:
      - targets:
          - '!{domain}:58000'
          - '!{domain}:58001'

  - job_name: 'grafana'
    static_configs:
      - targets: ['grafana:3000']
```

▼ web.yml 작성 (프로메테우스 비밀번호 설정)

```
basic_auth_users:
    !{prometheus user}: !{prometheus user passwor
```

▼ Grafana 설정

```
# grafana 접속
# docker-compose.yml에 작성한 user, password 입력
# 좌측 상단에서 Connections 선택
# Add data source 선택
# Prometheus 선택
# 설정한 prometheus 도메인 설정
# Authentication에 Basic authentication 선택 후 web.ym
# 저장
# Loki 선택
# 설정한 Loki 도메인 설정
# 저장
# Alerting -> Contact points 선택
# 기본으로 생성된 default 설정 옆에 edit 선택
# 기본 설정 삭제 후 Add contact point integration 선택
# Integration에서 discord 혹은 slack(mattermost) 설정
# Webhook URL 설정
# Title, Text Body에 보낼 메세지 설정
# Alerting  -> Alert rules 선택
# New alert rule 선택
# 이름 설정 후 Define query and alert condition에서 추가
# Options 선택 후 Time Range선택 후 Specify time range
# Label filters에서 level, ERROR 선택
# Operations 기존에 생성된 내용 삭제 후
# Opetations 새로 추가 (Aggregations -> Count) 선택
# Expressions 기존에 있는 내용 삭제
# Set evaluation behavior에서 Folder, Evaluation grou
# Configure no data and error handling 선택
# Alert state if no data or all values are null에서 O
```

▼ 10. mysql 설치

▼ RDS Mysql master 설정

▼ Mysql 인스턴스 생성, 설정

```
# 적당한 성능의 인스턴스를 선택
# 계정과 비밀번호, 초기 데이터베이스 생성 설정
# 로그를 최소 1일 이상으로 설정 후 인스턴스 생성
# 생성 된 RDS Mysql에 접속
# 슬레이브에서 연결할 때 사용할 사용자 생성
create user'username'@'%' identified by 'password
grant replication slave on *.* to 'username'@'%';

# 그 후에 slave에서 master를 설정하기 바로 전에 실행
# 출력된 내용을 slave mysql에서 작성 해야한다 log_file,
show master status
```

▼ EC2 Mysql slave 설정

▼ Mysql 설치

```
docker run --name mysql-slave-container-name -e M
```

▼ Mysql 설정

```
# 생성 된 docker에 접속 후 편집기 설치 (다른 편집기도 상관
microdnf install nano

# 그 후에 설정 파일 편집
nano /etc/my.cnf

# [mysqld] 로 작성된 부분 과 [client] 사이에 작성
replicate-ignore-table=mysql.rds_heartbeat2
replicate-ignore-table=mysql.rds_sysinfo
transaction-isolation=REPETABLE-READ

# 작성 종료 후 mysql 재시작
docker restart mysql-slave
```

```
# 생성 된 docker에 접속 후 mysql 접속 root, 컨테이너 생
# 들어가서 RDS Mysql에서 생성한 데이터베이스의 이름과 똑같
# slave 설정
# MASTER_USER, MASTER_PASSWORD은 master에서 생성한 사
# MASTER_LOG_FILE, MASTER_LOG_POS은 master에서 조회
CHANGE MASTER TO
MASTER_HOST='RDS 엔드포인트',
MASTER_USER='username',
MASTER_PASSWORD='password',
MASTER_LOG_FILE='file name',
MASTER_LOG_POS=position;

# 작성 후 연결이 되었는지 확인
show slave status;

# Slave_IO_Running, Slave_SQL_Running 값이 Yes가 나
```

▼ 11. redis 설치

```
# redis 설치
docker run --name redis -p 6379:6379 --requirepass "pas
```

▼ 백엔드 설정

▼ application.yml

```
spring:
  datasource:
    master:
      hikari:
        jdbc-url: jdbc:mysql://!{RDS Mysql address}
        username: !{username}
        password: !{password}
        driver-class-name: com.mysql.cj.jdbc.Driver
    slave:
      hikari:
        jdbc-url: jdbc:mysql://!{EC2 Mysql address}
        username: !{username}
        password: !{password}
        driver-class-name: com.mysql.cj.jdbc.Driver
  jpa:
    hibernate:
      ddl-auto: update
    open-in-view: false
  security:
    oauth2:
      client:
        registration:
          google:
            client-id: !{google client id}
            client-secret: !{google client secret}
            redirect-uri: !{google redirect uri}
            scope:
              - email
          kakao:
            client-id: !{kakao client id}
            client-secret: !{kakao client secret}
            redirect-uri: !{kakao redirect uri}
            client-authentication-method: client_secret
            authorization-grant-type: authorization_cod
            scope:
              - account_email
            client-name: kakao
        provider:
          kakao:
            authorization-uri: !{kakao aurhorization ur
            token-uri: !{kakao token uri}
            user-info-uri: !{kaako user info uri}
            user-name-attribute: id
  data:
    redis:
      password: !{password}
      port: !{port}
      host: !{EC2 address}
  servlet:
    multipart:
      max-file-size: 10MB
      max-request-size: 10MB
      enabled: true
```

```yaml
jwt:
  secret-key: !{JWT secret key}

server:
  base-url: !{EC2 address}
  servlet:
    context-path: /api

management:
  endpoints:
    web:
      exposure:
        include: prometheus, health
      base-path: /info/actuator
    enabled-by-default: false
  endpoint:
    prometheus:
      enabled: true
    health:
      enabled: true
  server:
    port: !{actuator port}

cloud:
  aws:
    s3:
      bucket: !{bucket name}
      region: !{region}
      credentials:
        access-key: !{access key}
        secret-key: !{secret key}
      url: !{bucket url}
springdoc:
  api-docs:
    path: /v3/api-docs
  swagger-ui:
    path: /swagger-ui.html
```

```
redirect-url: !{redirect url}

feign-client:
  secret-code: !{feign client secret code}
  url: !{feign client url}
```

▼ logback-spring.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <include resource="org/springframework/boot/logging
    <include resource="org/springframework/boot/logging
    <springProperty scope="context" name="application_n

    <appender name="LOKI" class="com.github.loki4j.logb
        <http>
            <url>!{Loki address}</url>
        </http>
        <format>
            <label>
                <pattern>app=${application_name},host=$
            </label>
            <message>
                <pattern>[%thread] %m%n</pattern>
            </message>
            <sortByTime>true</sortByTime>
        </format>
    </appender>

    <root level="INFO">
        <appender-ref ref="LOKI"/>
        <appender-ref ref="CONSOLE"/>
    </root>
</configuration>
```

▼ build.gradle

```
plugins {
    id 'java'
    id 'org.springframework.boot' version '3.2.4'
    id 'io.spring.dependency-management' version '1.1.4
    id "org.sonarqube" version "5.0.0.4638"
}

group = 'group.name'
version = '0.0.1-SNAPSHOT'

java {
    sourceCompatibility = '17'
}

configurations {
    compileOnly {
        extendsFrom annotationProcessor
    }
}

repositories {
    mavenCentral()
}

dependencies {
    implementation 'org.springframework.boot:spring-boo
    implementation 'org.springframework.boot:spring-boo
    implementation 'org.springframework.boot:spring-boo
    implementation 'org.springframework.boot:spring-boo
    implementation 'org.springframework.boot:spring-boo
    implementation 'org.springframework.boot:spring-boo
    implementation 'org.springframework.boot:spring-boo
    implementation 'org.springframework.cloud:spring-cl
    testImplementation 'org.springframework.security:sp
    compileOnly 'org.projectlombok:lombok'
    annotationProcessor 'org.projectlombok:lombok'
    testImplementation 'org.springframework.boot:spring
    // swagger
    implementation 'org.springdoc:springdoc-openapi-sta

    // Querydsl
    implementation 'com.querydsl:querydsl-jpa:5.0.0:jak
    annotationProcessor "com.querydsl:querydsl-apt:${de
    annotationProcessor "jakarta.annotation:jakarta.ann
    annotationProcessor "jakarta.persistence:jakarta.pe
    runtimeOnly 'com.mysql:mysql-connector-j'

    // Jwt
    implementation 'io.jsonwebtoken:jjwt-api:0.12.5'
    runtimeOnly 'io.jsonwebtoken:jjwt-impl:0.12.5'
    runtimeOnly 'io.jsonwebtoken:jjwt-jackson:0.12.5'

    // Loki
    implementation 'com.github.loki4j:loki-logback-appe

    // prometheus
    implementation 'io.micrometer:micrometer-registry-p

    // mapstruct
    implementation 'org.mapstruct:mapstruct:1.5.5.Final
    annotationProcessor 'org.mapstruct:mapstruct-proces

    // AWS S3
    implementation 'com.amazonaws:aws-java-sdk-s3:1.12.


}

tasks.named('test') {
    useJUnitPlatform()
}

bootJar {
    archiveFileName = 'app.jar'
}
```

▼ Dockerfile

```
FROM openjdk:17-alpine
ADD ./build/libs/app.jar app.jar
ENTRYPOINT ["java", "-jar", "/app.jar"]
```

▼ 프론트 설정

▼ Dockerfile

```
FROM node:20.12.0-alpine AS build
WORKDIR /app
COPY package.json yarn.lock ./
RUN yarn installCOPY . .
RUN yarn build

FROM nginx:stable-alpine

COPY nginx.conf /etc/nginx/conf.d/default.conf
COPY --from=build /app/build /usr/share/nginx/html

CMD ["nginx","-g","daemon off;"]
```

▼ Jenkins 파이프라인

```
def getNextVersion(currentVersion) {
    return currentVersion == '3000' ? '3001' : '3000'
}




pipeline {
    agent any
    environment {
        imageName = "frontend-test"
        containerName= "frontend-test"
        gitUrl=credentials("gitUrl")
        releaseServerAccount = 'ubuntu'
        releaseServerUri = '3.36.120.189'
```

```
     }
     stages {
         stage('clone'){
             steps{
                 git branch:"dev-fe",
                 changelog: false,
                 credentialsId: 'git',
                 poll: false,
                 url: gitUrl
             }
         }
           stage('Set Port'){
             steps{
                 script{
                 dir ('front') {
                   def currentVersion = sh(script: "dock
                   if(currentVersion.isEmpty()){
                     env.CURRENT_VERSION="3000"
                   } else{
                     env.CURRENT_VERSION = currentVersio
                   }
                   env.NEXT_VERSION = getNextVersion(env
                 }
                 }
             }
         }
         stage('Image build'){
             steps{
                 dir('front'){
                     script{
                         dockerImage=docker.build("$imag
                     }
                 }
             }
         }
         stage('Run') {
             steps {
                 script {
```

```
                // 생성 한 이미지를 가지고 다음 버전의 컨터
                sh "docker run --name $containerNam
            }
        }
    }
    // stage('SonarQube Analysis') {
    //     steps {
    //         dir ('backend') {
    //             script {
    //                 withCredentials([
    //                     string(credentialsId: 's
    //                     string(credentialsId: 's
    //                     string(credentialsId: 's
    //                 ]) {
    //                     sh 'chmod +x ./gradlew'
    //                     sh "./gradlew sonarqube
    //                 }
    //             }
    //         }
    //     }
    // }
    stage('Switch to New Version') {
        steps {
            sshagent(credentials: ['EC2']) {
                script {
                    // nginx의 service-url.inc 파일을
                    sh "ssh -o StrictHostKeyCheckin
                    // nginx를 재시작
                    sh "ssh -o StrictHostKeyCheckin
                }
            }
        }
    }
    stage('Clean'){
        steps{
            script {
                try {
                    sh "docker stop $containerName-
```

```
                        sh "docker rm $containerName-$C
                        sh "docker rmi $imageName:$CURR
                    } catch (Exception e) {
                        echo "Failed to remove containe
                    }
                }
            }
        }
    }
    post {
        success {
            script {
                def Author_ID = sh(script: "git show -s
                def Author_Name = sh(script: "git show
                mattermostSend (color: 'good',
                message: "frontend 빌드 성공: ${env.JOB_N
                )
            }
        }
        failure {
            script {
                 try {
                        sh "docker stop $containerName-
                        sh "docker rm $containerName-${
                        sh "docker rmi $imageName:${env
                    } catch (Exception e) {
                        echo "Failed to remove containe
                    }
                def Author_ID = sh(script: "git show -s
                def Author_Name = sh(script: "git show
                mattermostSend (color: 'danger',
                message: "frontend 빌드 실패: ${env.JOB_N
                )
            }
        }
    }
}
```