포팅 매뉴얼

I. 기술 스택 & 개발 환경 🔧

1. 사용 도구

> 이슈 관리: JIRA
>
> 형상 관리: GitLab

> 커뮤니케이션: Notion, Mattermost
>
> 디자인: Figma, ERDCloud
>
> UCC: DaVinci Resolve
>
> CI/CD: EC2, Docker, Jenkins

2. 개발 환경

Front-End

> Visual Studio

Back-End

> JDK: Eclipse Temurin 17.0.11 (LTS)
>
> SpringBoot: 3.3.3
>
> Gradle: 8.10
>
> IntelliJ: 2024.1.4 (Ultimate Edition)

AI

> FastAPI
>
> Conda: 24.7.1
>
> Python: 3.6.13 / 3.8.19
>
> VS Code: 1.90.2

DB

> MySQL: 8.0.39
>
> MongoDB: 7.0.14

Infra

> AWS EC: Ubuntu 20.04.6 LTS
>
> Docker: 27.2.1
>
> Jenkins: 2.477
>
> Nginx: 1.18.0 (Ubuntu)

3. 외부 서비스

> X

II. 빌드 및 배포 👟

1. 빌드

docker-compose.yml

```
services:
  mysql:
    image: mysql:8.0
    container_name: mysql
    env_file:
      - /home/ubuntu/.env
    environment:
      - TZ=Asia/Seoul
      - LANG=ko_KR.UTF-8
      - MYSQL_ROOT_PASSWORD=${MYSQL_PASSWORD}
    volumes:
      - ./db/mysql/data:/var/lib/mysql
      - ./db/mysql/init:/docker-entrypoint-initdb.d
      - /home/ubuntu/db/mysql/my.cnf:/etc/my.cnf
    ports:
      - ${MYSQL_HOST_PORT}:${MYSQL_APP_PORT}
    networks:
      - backend-network
  mongodb:
    image: mongo:latest
```

```
    container_name: mongodb
    env_file:
      - /home/ubuntu/.env
    restart: always
    environment:
      - MONGO_URI=${MONGODB_URI}
      - MONGO_INITDB_ROOT_USERNAME=${MONGODB_USER_NAME}
      - MONGO_INITDB_ROOT_PASSWORD=${MONGODB_USER_PASSWORD}
    ports:
      - ${MONGODB_HOST_PORT}:${MONGODB_APP_PORT}
    volumes:
      - /home/ubuntu/mongodb/data:/data
    networks:
      - backend-network
  redis:
    image: redis:alpine
    container_name: redis
    env_file:
      - /home/ubuntu/.env
    ports:
      - ${REDIS_HOST_PORT}:${REDIS_APP_PORT}
    volumes:
      - /home/ubuntu/redis_data:/data
    environment:
      - REDIS_PASSWORD=${REDIS_PASSWORD}
    networks:
      - backend-network
    command: ["redis-server", "--requirepass", "${REDIS_PASSW
volumes:
  # mysql volumes
  mysql_data:
  mysql_init:
  db_config:

  # mongo volumes
  mongo_data:
```

```
networks:
  backend-network:
```

docker-compose.{Blue}.yml

```
services:
  spring-boot:
    image: {image}
    container_name: ulvan-${SPRING_BOOT_HOST_PORT_BLUE}
    environment:
      - TZ=Asia/Seoul
      - LANG=ko_KR.UTF-8
      - HTTP_PORT=${SPRING_BOOT_HOST_PORT_BLUE}
    env_file:
      - {env_root}
    ports:
      - ${SPRING_BOOT_HOST_PORT_BLUE}:${SPRING_BOOT_APP_PORT_
    networks:
      - backend-network

networks:
  backend-network:
```

docker-compose.{Green}.yml

```
services:
  spring-boot:
    image: {image}
    container_name: ulvan-${SPRING_BOOT_HOST_PORT_GREEN}
    env_file:
      - {env_root}
    environment:
      - TZ=Asia/Seoul
      - LANG=ko_KR.UTF-8
      - HTTP_PORT=${SPRING_BOOT_HOST_PORT_GREEN}
    ports:
      - ${SPRING_BOOT_HOST_PORT_GREEN}:${SPRING_BOOT_APP_PORT_
```

```
    networks:
      - backend-network

networks:
  backend-network:
```

docker-compose.ngrinder.yml

```
version: '3.8'
services:
  controller:
    image: ngrinder/controller
    container_name: ngrinder-controller
    restart: always
    ports:
            - {ngrinder_port}
    volumes:
      - /ngrinder-controller:/opt/ngrinder-controller
  agent:
    image: ngrinder/agent
    container_name: ngrinder-agent
    restart: always
    links:
      - controller
```

.env 환경 변수

```
### Spring Boot ###
# SPRING BOOT
SPRING_BOOT_HOST_PORT=호스트 주소
SPRING_BOOT_APP_PORT=앱 주소
SPRING_BOOT_CONTEXT_PATH=/smartMirrorApi

# SPRING BOOT BLUE
SPRING_BOOT_HOST_PORT_BLUE=블루 호스트 포트 주소
SPRING_BOOT_APP_PORT_BLUE=블루 호스트 포트 앱 주소
```

```
# SPRING BOOT GREEN
SPRING_BOOT_HOST_PORT_GREEN=그린 호스트 포트 주소
SPRING_BOOT_APP_PORT_GREEN=그린 앱 포트 주소

DOMAIN_NAME=도메인 주소
PORT_NAME=도메인 포트 번호

### DATABASE ###
# Main DB (프로젝트에 적용할 DB)
DB_TYPE=DB 종류
DB_DOMAIN_NAME=DB 도메인 이름
DB_PORT=DB 포트 주소
DB_HOST_PORT=DB 호스트 주소
DB_APP_PORT=DB 앱 주소
DB_USER=DB 유저 ID
DB_PASSWORD=DB 유저 비밀번호
DB_NAME=사용할 스키마 이름 # SCHEMA NAME
DB_OPTIONS=DB 설정
DB_DRIVER_CLASS_NAME=Driver 클래스 이름 설정
DB_DIALECT=DB 방언 설정

# MySQL
MYSQL_TYPE=mysql
MYSQL_DOMAIN_NAME=MySQL 도메인 이름
MYSQL_HOST_PORT=MySQL 호스트 포트 주소
MYSQL_APP_PORT=MySQL 앱 포트 주소
MYSQL_USER=MySQL 유저 ID
MYSQL_PASSWORD=MySQL 유저 Password
MYSQL_DATABASE=MySQL DB 이름
MYSQL_SCHEMA_NAME=스키마 이름
MYSQL_OPTIONS=MySQL 설정
MYSQL_DRIVER_CLASS_NAME=com.mysql.cj.jdbc.Driver # Driver
설정
MYSQL_DIALECT=org.hibernate.dialect.MySQLDialect # 방언 설정

### REDIS  ###
REDIS_DOMAIN_NAME=Redis 도메인 이름
REDIS_HOST_PORT=Redis 호스트 포트 주소
```

```
REDIS_APP_PORT=Redis 앱 포트 주소
REDIS_VOLUMES=볼륨 설정
REDIS_PASSWORD=Redis 비밀번호

### Mongo DB ###
MONGODB_USERNAME=MongoDB 유저 이름
MONGODB_PASSWORD=MognoDB 유저 비밀번호
MONGODB_CLUSTER=클러스터 설정
MONGODB_APP_NAME=Mongo DB APP 이름
MONGODB_DATABASE=Mongo DB DB 이름

### SonarQube ###
SONAR_PROJECT_KEY=소나 큐브 설정
SONAR_PROJECT_NAME=소나 큐브 이름

### OpenAI ###
AI_API_KEY=AI API 키

### STT ###
STT_API_URI=STT API URI 설정
STT_API_MODEL=모델 설정

### Chat ###
CHAT_API_URI=CHAT API URI 설정
CHAT_API_MODEL=MODEL 설정
CHAT_MAX_COMPLETION_TOKEN=최대 토큰 설정
CHAT_TEMPERATURE=TEMPERTURE 설정
CHAT_PROMPT=PROMPT 설정

### Gan ###
GAN_API_URL=GAN_API URL 경로

### Facer ###
FACER_API_URL=FACER_API URL 경로

### HASH_SALT ###
HASH_SALT=HASH_SALT 설정
```

```
### QR ###
QR_BASE_URL=QR URL 설정
QR_REDIS_TTL=QR Reids 캐싱 시간 설정

### Style 캐싱 ###
STYLE_REDIS_TTL=Redis 캐싱 시간 설정
```

Front-End 환경 변수

- X

Back-End 환경 변수

- application.properties

```
spring.config.import=optional:file:.env[.properties]
spring.application.name=backend
spring.profiles.active=prod
```

- application-prod.properties

```
server.port=${SPRING_BOOT_HOST_PORT}
server.servlet.context-path=${SPRING_BOOT_CONTEXT_PATH}
# database
spring.datasource.url=jdbc:${DB_TYPE}://${DB_DOMAIN_NAM
E}:${DB_PORT}/${DB_NAME}${DB_OPTIONS}
spring.datasource.username=${DB_USER}
spring.datasource.password=${DB_PASSWORD}
spring.datasource.driver-class-name=${DB_DRIVER_CLASS_NAME}
spring.jpa.hibernate.ddl-auto=none
spring.jpa.properties.hibernate.dialect=${DB_DIALECT}
# mongodb
spring.data.mongodb.uri=mongodb+srv://${MONGODB_USERNAM
E}:${MONGODB_PASSWORD}@${MONGODB_CLUSTER}/?retryWrites=true
&w=majority&appName=${MONGODB_APP_NAME}
spring.data.mongodb.database=${MONGODB_DATABASE}
# JVM ?? ???? KST? ??
spring.jackson.time-zone=Asia/Seoul
# hibernate ?? ??
```

```
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
logging.level.org.hibernate.type.descriptor.sql=trace
#logging ??? WARN ?? ??
logging.level.com.sixback.backend=DEBUG
logging.level.org.springframework.orm.jpa=DEBUG
# Redis
spring.data.redis.host=${REDIS_DOMAIN_NAME}
spring.data.redis.port=${REDIS_HOST_PORT}
spring.data.redis.password=${REDIS_PASSWORD}
# OpenAI
spring.data.api.key=${AI_API_KEY}
spring.data.api.url=${AI_API_URL}
# STT
spring.data.stt.uri=${STT_API_URI}
spring.data.stt.model=${STT_API_MODEL}
# Chat
spring.data.chat.uri=${CHAT_API_URI}
spring.data.chat.model=${CHAT_API_MODEL}
spring.data.chat.max.completion.token=${CHAT_MAX_COMPLETION
_TOKEN}
spring.data.chat.temperature=${CHAT_TEMPERATURE}
spring.data.chat.prompt=${CHAT_PROMPT}
# Gan
spring.data.gan.url=${GAN_API_URL}
# Facer
spring.data.facer.url=${FACER_API_URL}
# HASH_SALT
spring.data.hash.salt=${HASH_SALT}
# QR Base url
spring.data.qr.base.url=${QR_BASE_URL}
spring.data.qr.redis.ttl.seconds=${QR_REDIS_TTL}
spring.data.style.redis.ttl.seconds=${STYLE_REDIS_TTL}
```

Nginx 설정 파일

- default

```
server {
        root /var/www/html;

        index index.html index.htm index.nginx-debian.html;

        server_name {Service_name};

        autoindex_localtime on;

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

        location / {
                return 404;
        }

        location /smartMirrorApi {
                proxy_pass $service_url;
                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;
        }


    listen 443 ssl; # managed by Certbot
    ssl_certificate {SSL_root}; # managed by Certbot
    ssl_certificate_key {privkey.pem_root}; # managed by Ce
rtbot
    include {options-ssl-nginx.conf_root}; # managed by Cer
tbot
    ssl_dhparam {ssl-dhparams.pem_root}; # managed by Certb
ot
}

server {
        if ($host = {Domain}) {
                return 301 https://$host$request_uri;
```

```
        } # managed by Certbot


        listen 80;
        server_name {Domain};
        location / {
                return 301 https://$host$request_uri;
        }
        return 404; # managed by Certbot
}
```

- service-url.inc

```
set $service_url http://127.0.0.1:{Blue or Green};
```

Front-End Build

- Visual Studio에서 실행

Back-End Build

1. 프로젝트 빌드 `./gradlew build`
2. 빌드된 JAR 파일 실행 `java -jar build/libs/{project_name}.jar`

2. 배포

deploy.sh 파일

- {Green}, {Blue}, {After}, {Before}, {TestPath}는 원하는 값 적용

```
# deploy.sh

# .env 파일 로드
if [ -f .env ]; then
  export $(cat .env | grep -v '#' | awk '/=/ {print $1}')
else
  echo ".env 파일을 찾을 수 없습니다."
  exit 1
fi
```

```
#0
# 이미지 갱신
sudo docker compose -p {Blue} -f /home/ubuntu/docker-compos
e.{Blue}.yml pull
sudo docker compose -p {Green} -f /home/ubuntu/docker-compo
se.{Green}.yml pull

#1
EXIST_GITCHAN=$(sudo docker compose -p {Blue} -f docker-com
pose.{Blue}.yml ps | grep Up)

if [ -z "$EXIST_GITCHAN" ]; then
        echo "{Blue} 컨테이너 실행"
        sudo docker compose -p {Blue} -f /home/ubuntu/docke
r-compose.{Blue}.yml up -d --force-recreate
        BEFORE_COLOR="{Green}"
        AFTER_COLOR="{Blue}"
        BEFORE_PORT={Green}
        AFTER_PORT={Blue}
else
        echo "{Green} 컨테이너 실행"
        sudo docker compose -p {Green} -f /home/ubuntu/dock
er-compose.{Green}.yml up -d --force-recreate
        BEFORE_COLOR="{Blue}"
        AFTER_COLOR="{Green}"
        BEFORE_PORT={Blue}
        AFTER_PORT={Green}
fi

echo "${AFTER_COLOR} server up(port:${AFTER_PORT})"



# 2
for cnt in `seq 1 10`;
do
    echo "서버 응답 확인하는중~(${cnt}/10)";
```

```
    UP=$(curl -s http://127.0.0.1:${AFTER_PORT}/{TestPath})
    echo "http://127.0.0.1:${AFTER_PORT}/{TestPath}"
    echo "port : $AFTER_PORT Ans: $UP"
    if [ "${UP}" != "OK" ]; then
        sleep 7
        continue
    else
        break
    fi
done

if [ $cnt -eq 10 ]; then
    echo "서버에 문제가 있어요..."
    # sudo docker stop {After}${AFTER_PORT}
    # sudo docker rm {After}${AFTER_PORT}
    exit 1
fi

# 3
sudo sed -i "s/${BEFORE_PORT}/${AFTER_PORT}/" /etc/nginx/co
nf.d/service-url.inc
sudo nginx -s reload
echo "Deploy Completed!!"

# 4
echo "$BEFORE_COLOR server down(port:${BEFORE_PORT})"
sudo docker compose -p ${BEFORE_COLOR} -f docker-compos
e.${BEFORE_COLOR}.yml down
```

Deploy-default.sh 파일

```
# Docker Compose 파일 경로
COMPOSE_FILE="{파일 경로}/docker-compose.yml"
MYSQL_PROJECT_NAME="mysql"
REDIS_PROJECT_NAME="redis"

# MySQL 컨테이너 상태 확인 및 실행
if sudo docker compose -p $MYSQL_PROJECT_NAME -f $COMPOSE_FIL
```

```
    echo "MySQL 컨테이너가 이미 실행 중입니다."
else
    echo "MySQL 컨테이너가 실행되고 있지 않습니다. 새로운 컨테이너를 시작

    # MySQL 이미지 풀
    sudo docker compose -p $MYSQL_PROJECT_NAME -f $COMPOSE_FI

    # MySQL 컨테이너 시작
    sudo docker compose -p $MYSQL_PROJECT_NAME -f $COMPOSE_FI

    echo "MySQL 컨테이너가 시작되었습니다."
fi

# Redis 컨테이너 상태 확인 및 실행
if sudo docker compose -p $REDIS_PROJECT_NAME -f $COMPOSE_FIL
    echo "Redis 컨테이너가 이미 실행 중입니다."
else
    echo "Redis 컨테이너가 실행되고 있지 않습니다. 새로운 컨테이너를 시작

    # Redis 이미지 풀
    sudo docker compose -p $REDIS_PROJECT_NAME -f $COMPOSE_FI

    # Redis 컨테이너 시작
    sudo docker compose -p $REDIS_PROJECT_NAME -f $COMPOSE_FI

    echo "Redis 컨테이너가 시작되었습니다."
fi
```

Jenkins 파이프 라인(빌드 및 배포)

```
pipeline {
    agent any

    environment {

        // GITLAB
        GITLAB_BRANCH = 브랜치 명
        GITLAB_CREDENTIALS_ID = 인증 ID
```

```
        GITLAB_URL = GitLab URL

        // MATTERMOST
        MATTERMOST_ENDPOINT = credentials('MATTERMOST_ENDPOIN

        // DOCKER
        DOCKER_COMPOSE_DIR = DockerCompose 파일 경로  // docker

        GITLAB_FILE = credentials("GITLAB_ENV_FILE")

        // SONARQUBE
        SONAR_PROJECT_KEY = credentials('SONAR_PROJECT_KEY')
      SONAR_PROJECT_NAME = credentials('SONAR_PROJECT_NAME')
      SONAR_HOST_URL = credentials('SONAR_HOST_URL')
      SONAR_TOKEN = credentials('SONAR_TOKEN')
    } // environment End

    stages {
        stage('Git Checkout') {
            steps {
                script {
                    git branch: "${GITLAB_BRANCH}", credentia
                }
            }
            post {
                failure {
                    notifyBuildResult('failure', 'Git Checkou
                }
                success {
                    echo 'Repository clone success!'
                }
            }
        } // Git Checkout End

        stage('Load Environment Variables') {
            steps {
                script {
                    dir('backend') {
```

```
                    withCredentials([file(credentialsId:
                        if (fileExists('.env')) { // .env
                            sh 'chmod 600 .env'
                        }
                        sh 'cp $ENV_FILE .env'
                        // sh 'rm .env' // 파일 내용 삭제
                    }
                }
            }
        }
    } // Load Environment Variables End

    stage('Build') {
        steps {
            dir('backend') {
                sh 'chmod +x ./gradlew' // 프로젝트 권한 변경
                sh './gradlew clean build' // 프로젝트 빌드
            }
            dir('mysql') {
                sh 'docker build -t mysql:8.0 .'
            }
        }
    } // Build End

    stage('SonarQube analysis') {
        steps{
            dir('backend') {
                withSonarQubeEnv('SonarQubeToken') { // s
                    sh '''
                    ./gradlew sonar \
                      -Dsonar.projectKey=$SONAR_PROJECT_K
                      -Dsonar.projectName=$SONAR_PROJECT_
                      -Dsonar.host.url=$SONAR_HOST_URL \
                      -Dsonar.token=$SONAR_TOKEN
                    '''
                }
            }
        }
```

```
        } // SonarQube Analysis End

        stage('Test') {
            steps {
                script {
                    dir ('backend') {
                        sh './gradlew test' // Gradle 테스트를
                    }
                }
            }
        } // Test End

        stage('Docker Hub Login'){
            steps {
                withCredentials([usernamePassword(credentials
                    sh 'echo "$DOCKER_PASSWORD" | docker logi
                }
            }
        } // Docker Hub Login End

        stage('Docker Build and Push') {
            steps {
                withCredentials([usernamePassword(credentials
                    dir('backend') {
                        sh 'docker build -f Dockerfile -t $DO
                        sh 'docker push $DOCKER_REPO/$DOCKER_
                        echo 'docker push Success!!'
                    }

                }
                echo 'docker push Success!!'
            }
        } // Docker Build and Push End

        stage('Deploy') {
            steps {
                sshagent(credentials: ['MY_SSH_CREDENTIALS'])
                    withCredentials([string(credentialsId: 'E
```

```
                        withCredentials([file(credentialsId:
                            sh '''
                                scp -o StrictHostKeyChecking=
                                ssh -o StrictHostKeyChecking=
                            '''
                        }
                        sh 'ssh -o StrictHostKeyChecking=no u
                        sh 'ssh -o StrictHostKeyChecking=no u
                    }
                }
            }
        } // Deploy End
    } // stages End

    post {
        always {
            echo '빌드 완료 후 도커 이미지 정리 중...'
            sh "docker image prune -f"
        } // always End

        success {
            script {
                def Author_ID = sh(script: "git show -s --pre
                def Author_Name = sh(script: "git show -s --p
                mattermostSend(color: 'good',
                    message: "빌드 성공: ${env.JOB_NAME} #${env
                    endpoint: MATTERMOST_ENDPOINT,
                    channel: Mattermost 채널 이름
                )
            }
        } // success End

        failure {
            script {
                def Author_ID = sh(script: "git show -s --pre
                def Author_Name = sh(script: "git show -s --p
                mattermostSend(color: 'danger',
                    message: "빌드 실패: ${env.JOB_NAME} #${env
```

```
                    endpoint: MATTERMOST_ENDPOINT,
                    channel: Mattermost 채널 이름
                )
            }
        } // failure End
    } // post End
} // pipeline End
```