포팅 메뉴얼

서버 버전 설정

백엔드

의존성 (Group:Artifact)	버전
JDK	17
lombok	1.18.36
spring-boot-starter-web	3.4.3
spring-boot-starter-data-redis	3.4.3
spring-cloud-starter-openfeign	4.2.0
feign-form	3.8.0
feign-form-spring	3.8.0
spring-boot-starter-data-elasticsearch	3.4.4
spring-cloud-starter-netflix-eureka-client	4.2.0
spring-cloud-starter-aws-secrets-manager-config	2.4.4
spring-kafka	4.0.0-M1
mybatis-spring-boot-starter	3.0.4
spring-boot-starter-webflux	3.4.3
jjwt	0.12.6
spring-boot-starter-security	3.4.3
spring-boot-starter-data-jpa	3.4.3
mysql-connector-j	9.1.0
spring-boot-starter-websocket	3.4.3
spring-cloud-starter-gateway	4.2.0
spring-boot-starter-actuator	3.4.3
jjwt-api	0.11.5

jjwt-impl	0.11.5
jjwt-jackson	0.11.5
spring-boot-starter-data-mongodb	3.4.3

Flutter(모바일)

패키지명	버전
flutter	sdk: flutter
cupertino_icons	1.0.2
go_router	13.2.0
flutter_riverpod	2.4.9
equatable	2.0.5
dio	5.4.1
shared_preferences	2.2.2
flutter_secure_storage	9.0.0
google_fonts	6.1.0
flutter_svg	2.0.15
google_sign_in	6.3.0
lottie	3.1.0
logger	2.0.2+1
jwt_decoder	2.0.1
http	1.1.0
intl	0.19.0
cached_network_image	3.4.1
shimmer	3.0.0
carousel_slider	5.0.0
url_launcher	6.3.1
stomp_dart_client	2.1.3

React(웹)

패키지명	버전
@emotion/react	11.14.0

@emotion/styled	**11.14.0**
@hookform/resolvers	**4.1.3**
@react-oauth/google	**0.12.1**
@types/antd	**0.12.32**
@types/axios	**0.9.36**
@types/react-router-dom	**5.3.3**
@types/react-toastify	**4.0.2**
antd	**5.24.5**
axios	**1.8.4**
jwt-decode	**4.0.0**
react	**19.0.0**
react-dom	**19.0.0**
react-hook-form	**7.55.0**
react-icons	**5.5.0**
react-router-dom	**7.4.0**
react-toastify	**11.0.5**
recharts	**2.15.1**
zod	**3.24.2**
zustand	**5.0.3**

Ubuntu - Docker 설치

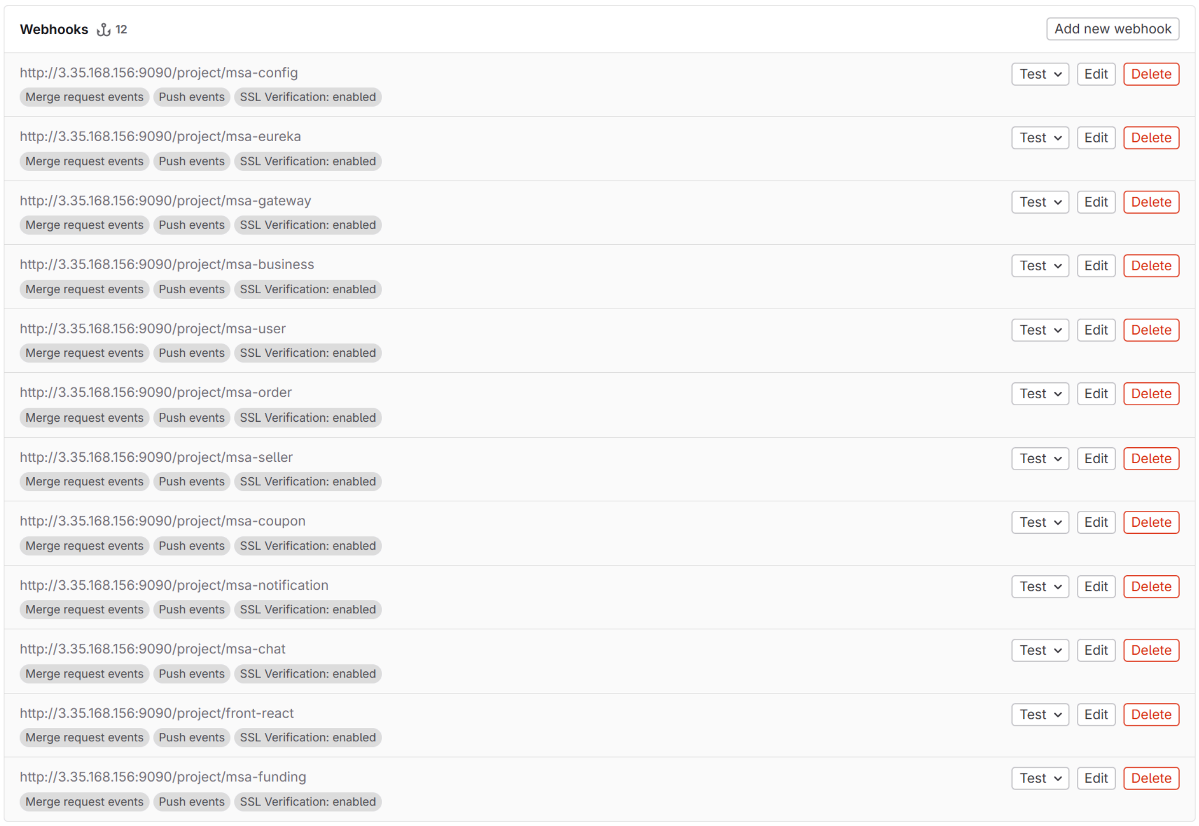
```
# 0. 해당 EC2 접속
ssh -i {KEY_PATH} {USER}@{SERVER_IP}

# 1. 기존 패키지 업데이트
sudo apt update && sudo apt upgrade -y

# 2. 필수 패키지 설치
sudo apt install apt-transport-https ca-certificates curl software-properties-c

# 3. Docker 공식 GPG 키 추가
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearm
```

```
# 4. Docker 저장소 추가
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

# 5. 패키지 업데이트 & Docker 설치
sudo apt update
sudo apt install docker-ce docker-ce-cli containerd.io -y

# 6. Docker 서비스 시작 및 부팅 시 자동 실행 설정
sudo systemctl start docker
sudo systemctl enable docker

# 7. 현재 사용자에게 docker 그룹 권한 부여
sudo usermod -aG docker $USER
```

네트워크 생성

```
docker network create msa-network
```

Gitlab Connection, Webhook 설정


Webhooks 12

Add new webhook

http://3.35.168.156:9090/project/msa-config
Merge request events | Push events | SSL Verification: enabled

http://3.35.168.156:9090/project/msa-eureka
Merge request events | Push events | SSL Verification: enabled

http://3.35.168.156:9090/project/msa-gateway
Merge request events | Push events | SSL Verification: enabled

http://3.35.168.156:9090/project/msa-business
Merge request events | Push events | SSL Verification: enabled

http://3.35.168.156:9090/project/msa-user
Merge request events | Push events | SSL Verification: enabled

http://3.35.168.156:9090/project/msa-order
Merge request events | Push events | SSL Verification: enabled

http://3.35.168.156:9090/project/msa-seller
Merge request events | Push events | SSL Verification: enabled

http://3.35.168.156:9090/project/msa-coupon
Merge request events | Push events | SSL Verification: enabled

http://3.35.168.156:9090/project/msa-notification
Merge request events | Push events | SSL Verification: enabled

http://3.35.168.156:9090/project/msa-chat
Merge request events | Push events | SSL Verification: enabled

http://3.35.168.156:9090/project/front-react
Merge request events | Push events | SSL Verification: enabled

http://3.35.168.156:9090/project/msa-funding
Merge request events | Push events | SSL Verification: enabled

Jenkins 실행

```
docker run -d \
  --name jenkins-server \
  -p 9090:8080 \
  -p 50000:50000 \
  --network msa-network \
  jenkins/jenkins
```

Jenkins 초기 패스워드 확인

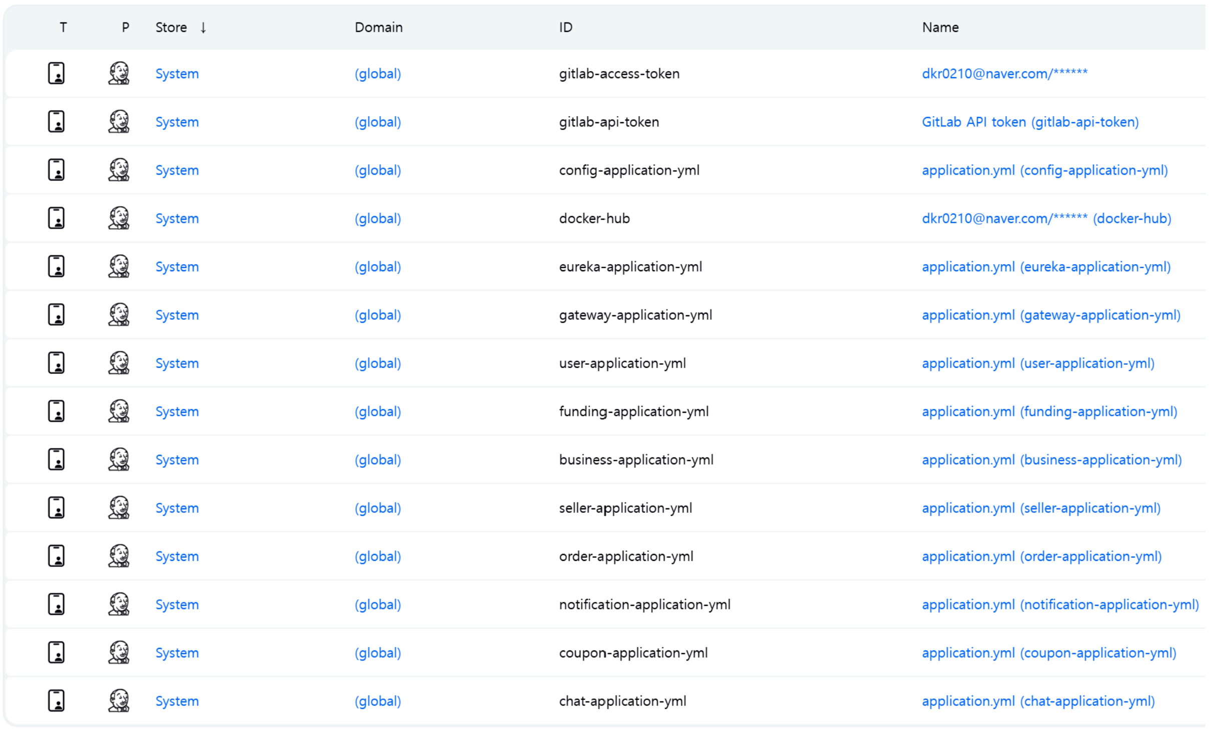
```
docker exec jenkins-server cat /var/jenkins_home/secrets/initialAdminPasswc
```

Jenkins 파이프라인 플러그인 설치

- **GitLab**
- **Docker**

- **GitLab Authentication**
- **Generic WebHook Trigger**
- **SSH**

환경 변수 및 Credential 설정



T	P	Store ↓	Domain	ID	Name
		System	(global)	gitlab-access-token	dkr0210@naver.com/******
		System	(global)	gitlab-api-token	GitLab API token (gitlab-api-token)
		System	(global)	config-application-yml	application.yml (config-application-yml)
		System	(global)	docker-hub	dkr0210@naver.com/****** (docker-hub)
		System	(global)	eureka-application-yml	application.yml (eureka-application-yml)
		System	(global)	gateway-application-yml	application.yml (gateway-application-yml)
		System	(global)	user-application-yml	application.yml (user-application-yml)
		System	(global)	funding-application-yml	application.yml (funding-application-yml)
		System	(global)	business-application-yml	application.yml (business-application-yml)
		System	(global)	seller-application-yml	application.yml (seller-application-yml)
		System	(global)	order-application-yml	application.yml (order-application-yml)
		System	(global)	notification-application-yml	application.yml (notification-application-yml)
		System	(global)	coupon-application-yml	application.yml (coupon-application-yml)
		System	(global)	chat-application-yml	application.yml (chat-application-yml)

서비스별 포트 매핑

서비스 이름	컨테이너 이미지	컨테이너 포트	호스트 포트
Gateway	`msa-gateway`	8080	8080
Eureka Server	`msa-eureka`	8761	8761
Config Server	`msa-config`	9000	9000
User Server	`msa-user`	8080	8082
Business Server	`msa-business`	8080	8081
Seller Server	`msa-seller`	8080	8083
Funding Server	`msa-funding`	8080	8084
Order Server	`msa-order`	8080	8085

Coupon Server	`msa-coupon`	8080	8086
Chat Server	`msa-chat`	8080	8087
Notification	`msa-notification`	8080	8088
Elasticsearch	`elasticsearch`	9200	9200
MySQL	`mysql`	3306	3306
Kafka	`confluentinc/cp-kafka`	9092	9092
Zookeeper	`confluentinc/cp-zookeeper`	2181	2181
Redis	`redis`	6379	6379
Nginx	`nginx`	80 / 443	80 / 443
Jenkins	`jenkins/jenkins`	8080	9090
Front Server	`front-server`	3000	3000

컨테이너 세팅

My-SQL

```
docker run -d --name mysql-container --network msa-network -e MYSQL_RC
```

Elasticsearch docker-compose.yml

```
services:
  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:8.15.5
    container_name: elasticsearch
    environment:
      - discovery.type=single-node
      - ES_JAVA_OPTS=-Xms2g -Xmx2g
      - xpack.security.enabled=false
      - network.host=0.0.0.0
    ports:
      - "9200:9200"
      - "9300:9300"
    volumes:
      - esdata:/usr/share/elasticsearch/data
    networks:
      - msa-network

volumes:
  esdata:
    driver: local

networks:
  msa-network:
    external: true
```

Kafka docker-compose.yml

```
services:
  zookeeper:
    image: confluentinc/cp-zookeeper:latest
    container_name: zookeeper-server
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000
    ports:
      - "2181:2181"
    networks:
      - msa-network

  kafka:
    image: confluentinc/cp-kafka:latest
    container_name: kafka-server
    depends_on:
      - zookeeper
    ports:
      - "9092:9092"
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://3.35.168.156:9092
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
    networks:
      - msa-network

networks:
  msa-network:
    external: true
```

Business Dockerfile

```
FROM openjdk:17-jdk-slim
ARG JAR_FILE=build/libs/business-0.0.1-SNAPSHOT.jar
COPY ${JAR_FILE} /app
ENTRYPOINT ["java", "-jar", "/app.jar"]
```

Chat Dockerfile

```
FROM openjdk:17-jdk-slim
ARG JAR_FILE=build/libs/chat-0.0.1-SNAPSHOT.jar
COPY ${JAR_FILE} /app.jar
ENTRYPOINT ["java", "-jar", "/app.jar"]
```

Coupon Dockerfile

```
FROM openjdk:17-jdk-slim
ARG JAR_FILE=build/libs/coupon-0.0.1-SNAPSHOT.jar
COPY ${JAR_FILE} /app.jar
ENTRYPOINT ["java", "-jar", "/app.jar"]
```

Eureka Dockerfile

```
FROM openjdk:17-jdk-slim
ARG JAR_FILE=build/libs/eureka-0.0.1-SNAPSHOT.jar
```

```
COPY ${JAR_FILE} /app.jar
ENTRYPOINT ["java", "-jar", "/app.jar"]
```

Funding Dockerfile

```
FROM openjdk:17
ARG JAR_FILE=build/libs/funding-0.0.1-SNAPSHOT.jar
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java","-jar","/app.jar"]
```

Gateway Dockerfile

```
FROM openjdk:17-jdk-slim
ARG JAR_FILE=build/libs/gateway-0.0.1-SNAPSHOT.jar
COPY ${JAR_FILE} /app.jar
ENTRYPOINT ["java", "-jar", "/app.jar"]
```

Notification Dockerfile

```
FROM openjdk:17-jdk-slim
ARG JAR_FILE=build/libs/notification-0.0.1-SNAPSHOT.jar
COPY ${JAR_FILE} /app.jar
ENTRYPOINT ["java", "-jar", "/app.jar"]
```

Order Dockerfile

```
FROM openjdk:17-jdk-slim
ARG JAR_FILE=build/libs/order-0.0.1-SNAPSHOT.jar
COPY ${JAR_FILE} /app.jar
ENTRYPOINT ["java", "-jar", "/app.jar"]
```

Seller Dockerfile

```
FROM openjdk:17-jdk-slim
ARG JAR_FILE=build/libs/seller-0.0.1-SNAPSHOT.jar
COPY ${JAR_FILE} /app.jar
ENTRYPOINT ["java", "-jar", "/app.jar"]
```

User Dockerfile

```
FROM openjdk:17-jdk-slim
ARG JAR_FILE=build/libs/user-0.0.1-SNAPSHOT.jar
COPY ${JAR_FILE} /app.jar
ENTRYPOINT ["java", "-jar", "/app.jar"]
```

Front Dockerfile

```
FROM node:21-alpine AS build

WORKDIR /app
COPY package.json ./
COPY package-lock.json ./

RUN npm i
COPY . ./
RUN npm run build

FROM node:21-alpine

WORKDIR /app
COPY --from=build /app/build /app/build

EXPOSE 3000

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

NGINX 설정 (/etc/nginx/conf.d/default.conf)

```
server {
    listen 80;
    server_name j12e206.p.ssafy.io;
    client_max_body_size 10M;
    return 301 https://$host$request_uri;
}

server {
    listen 443 ssl;
    server_name j12e206.p.ssafy.io;

    ssl_certificate /etc/nginx/ssl/fullchain.pem;
    ssl_certificate_key /etc/nginx/ssl/privkey.pem;

    root /usr/share/nginx/html;
    index index.html index.htm;

    client_max_body_size 10M;

    location / {
        proxy_pass http://3.36.67.192:3000;
        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 https;

    }

    location /api/ {
        proxy_pass http://3.35.168.156:8080/api/;
        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 https;
    }

    location /ws-stomp {
        proxy_pass http://3.35.168.156:8088/ws-stomp;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_set_header Host $host;
    }
}
```

파이프라인

Pipeline(Front)

```
pipeline {
    agent any

    environment {
        DOCKER_IMAGE_NAME = 'Docker 이미지 이름'
        DOCKER_REGISTRY = 'Docker Hub 사용자/레포'
        CONTAINER_NAME = '컨테이너 이름'
    }

    stages {
        stage('Checkout') {
            steps {
                script {
                    git credentialsId: 'gitlab-access-token',
                        url: 'Gitlab',
                        branch: '브랜치 이름'

                    def GIT_BRANCH = sh(script: "git rev-parse --abbrev-ref HEAD", r
                def GIT_COMMIT_MESSAGE = sh(script: "git log -1 --pretty=%B", 

                if (GIT_BRANCH != "브랜치 이름") {
                    echo "현재 브랜치가 브랜치 이름이 아님. 빌드 중단."
                    currentBuild.result = 'ABORTED'
                    error("Stopping pipeline")
                }

                if (!GIT_COMMIT_MESSAGE.contains("Merge branch '브랜치이름'")
                    echo "브랜치에서의 머지 커밋이 아님. 빌드 중단."
                    currentBuild.result = 'ABORTED'
                    error("Stopping pipeline")
                } else {
                    echo "✅ Merge commit 감지됨. 빌드 진행."
                }
            }
        }
    }

    stage('Install Dependencies & Build React') {
        steps {
            script {
                sh 'cd front_react && npm ci && npm run build'
            }
        }
    }

    stage('Build Docker Image') {
        steps {
            script {
                sh 'cd front_react && docker build -t $DOCKER_REGISTRY:$DOCK
            }
        }
    }

    stage('Push Docker Image') {
        steps {
            script {
                withCredentials([usernamePassword(credentialsId: 'docker-hub', 
                    sh "docker login -u ${DOCKER_USERNAME} -p ${DOCKER_PAS
                }
                sh "docker push $DOCKER_REGISTRY:$DOCKER_IMAGE_NAME"
            }
        }
    }

    stage('Deploy Container') {
        steps {
            script {
                sh "docker container rm -f $CONTAINER_NAME || true"
                sh "docker image rm -f $DOCKER_REGISTRY/$DOCKER_IMAGE_N
                sh "docker pull $DOCKER_REGISTRY:$DOCKER_IMAGE_NAME"
                sh "docker run -d --name $CONTAINER_NAME -p 3000:3000 --n
            }
        }
    }
  }

  post {
    success {
        echo '✅ React Frontend Build & Deploy Success!'
    }
    failure {
        echo '❌ Build or Deploy Failed!'
    }
  }
}
```

Pipeline(Back)

```
pipeline {
  agent any

  environment {
    DOCKER_IMAGE_NAME = 'Docker 이미지 이름'
    DOCKER_REGISTRY = 'Docker 레지스트리 (필요시)'
    CONTAINER_NAME = '컨테이너 이름'
}

stages {
    stage('Checkout') {
        steps {
            script {
                git credentialsId: 'gitlab-access-token',
                    url: 'Gitlab',
                    branch: '브랜치 이름'

                // 현재 브랜치 확인
                def GIT_BRANCH = sh(script: "git rev-parse --abbrev-ref HEAD", r

                // 마지막 커밋 메시지 확인
                def GIT_COMMIT_MESSAGE = sh(script: "git log -1 --pretty=%B",

                // develop/BE 브랜치가 아닐 경우 빌드 중단
                if (GIT_BRANCH != "브랜치 이름") {
                    echo "현재 브랜치: ${GIT_BRANCH}"
                    echo "현재 브랜치가 '브랜치 이름'가 아님. 빌드 중단."
                    currentBuild.result = 'ABORTED'
                    error("Stopping pipeline")
                }

                // feature/BE/config에서 머지된 커밋인지 확인
                if (!GIT_COMMIT_MESSAGE.contains("Merge branch '브랜치 이름'"
                    echo "브랜치 이름 브랜치에서의 머지 커밋이 아님. 빌드 중단."
                    currentBuild.result = 'ABORTED'
                    error("Stopping pipeline")
                } else {
                    echo "✅ Merge commit 감지됨. 빌드 진행."
                }
            }
        }
    }
```

```
stage('Copy Config File') {
    steps {
        script {
            withCredentials([file(credentialsId: 'yml 파일', variable: 'APP_YML')
                sh 'mkdir -p back/프로젝트/src/main/resources'
                sh 'chmod -R 777 back/프로젝트/src/main/resources'
                // Jenkins 환경에 있는 `application.yml`을 프로젝트 폴더로 복사
                sh 'cp $APP_YML back/프로젝트/src/main/resources/application
            }
        }
    }
}

stage('Build JAR') {
    steps {
        script {
            // Gradle 빌드 실행 (JAR 파일 생성)
            sh 'cd back/프로젝트 && chmod +x ./gradlew && ./gradlew clean bu
        }
    }
}

stage('Build Docker Image') {
    steps {
        script {
            // Dockerfile 위치와 이미지 이름을 설정하여 이미지를 빌드
            sh "cd back/프로젝트 && docker build -t $DOCKER_REGISTRY:$DO
        }
    }
}

stage('Push Docker Image') {
    steps {
        script {
            // Docker Hub 로그인인
            withCredentials([usernamePassword(credentialsId: 'docker-hub', 
                sh "docker login -u ${DOCKER_USERNAME} -p ${DOCKER_PAS
                }
                // Docker 이미지가 필요한 경우 레지스트리로 푸시
                sh "docker push $DOCKER_REGISTRY:$DOCKER_IMAGE_NAME"
            }
        }
    }

    stage('Deploy Container') {
        steps {
            script {
                // 기존 컨테이너 제거
                sh "docker container rm -f $CONTAINER_NAME || true"
                // 기존 이미지 제거
                sh "docker image rm -f $DOCKER_REGISTRY/$DOCKER_IMAGE_N
                // 남아있는 이미지 출력
                sh "docker image ls -a"
                // 새 버전 이미지 받아오기
                sh "docker pull $DOCKER_REGISTRY:$DOCKER_IMAGE_NAME"
                // 컨테이너로 이미지 실행
                sh "docker run -d --name $CONTAINER_NAME -p 포트번호:8080 -
            }
        }
    }


  }

  post {
    success {
        echo 'Build and Deploy Success!'
    }
    failure {
        echo 'Build or Deploy Failed!'
    }
  }
}
```